

TiDB Documentation

PingCAP Inc.

20211223

Table of Contents

1	About TiDB	25
1.1	TiDB Introduction	25
1.1.1	Key features	25
1.1.2	Use cases	26
1.1.3	See also	27
1.2	TiDB 5.3 Release Notes	27
1.2.1	Compatibility changes	28
1.2.2	New features	40
1.2.3	Telemetry	46
1.2.4	Removed feature	46
1.2.5	Improvements	46
1.2.6	Bug Fixes	48
1.3	TiDB Features	51
1.3.1	Data types, functions, and operators	51
1.3.2	Indexing and constraints	52
1.3.3	SQL statements	52
1.3.4	Advanced SQL features	52
1.3.5	Data definition language (DDL)	53
1.3.6	Transactions	53
1.3.7	Partitioning	53
1.3.8	Statistics	54

1.3.9	Security	54
1.3.10	Data import and export	54
1.3.11	Management, observability, and tools	55
1.4	TiDB Experimental Features	55
1.4.1	Stability	55
1.4.2	Scheduling	55
1.4.3	SQL	56
1.4.4	Configuration management	56
1.4.5	Data sharing and subscription	56
1.4.6	Storage	56
1.4.7	Data migration	57
1.4.8	Backup and restoration	57
1.4.9	Garbage collection	57
1.4.10	Diagnostics	57
1.5	Benchmarks	57
1.5.1	TiDB Sysbench Performance Test Report – v5.3.0 vs. v5.2.2	57
1.5.2	TiDB TPC-C Performance Test Report – v5.3.0 vs. v5.2.2	64
1.5.3	TiDB TPC-H 100GB Performance Test Report – TiDB v5.3 MPP mode vs. Greenplum 6.15.0 and Apache Spark 3.1.1	68
1.6	MySQL Compatibility	71
1.6.1	Unsupported features	72
1.6.2	Features that are different from MySQL	72
1.7	TiDB Limitations	77
1.7.1	Limitations on identifier length	77
1.7.2	Limitations on the total number of databases, tables, views, and connections	78
1.7.3	Limitations on a single database	78
1.7.4	Limitations on a single table	78
1.7.5	Limitation on a single row	78
1.7.6	Limitation on a single column	79
1.7.7	Limitations on string types	79
1.7.8	Limitations on SQL statements	79

1.8	TiDB Adopters	80
1.9	Credits	82
1.9.1	TiDB developers	82
1.9.2	Writers and translators for TiDB documentation	83
2	Quick Start	83
2.1	Quick Start Guide for the TiDB Database Platform	83
2.1.1	Deploy a local test environment on Mac OS	84
2.1.2	Deploy a local test environment on Linux OS	86
2.1.3	Set up a test environment on a single machine using TiUP cluster	88
2.1.4	What's next	94
2.2	Quick Start Guide for TiDB HTAP	94
2.2.1	Basic concepts	94
2.2.2	Steps	95
2.2.3	What's next	99
2.3	Explore SQL with TiDB	99
2.3.1	Category	99
2.3.2	Show, create and drop a database	100
2.3.3	Create, show, and drop a table	100
2.3.4	Create, show, and drop an index	101
2.3.5	Insert, update, and delete data	102
2.3.6	Query data	102
2.3.7	Create, authorize, and delete a user	103
2.4	Explore HTAP	103
2.4.1	Use cases	103
2.4.2	Architecture	104
2.4.3	Environment preparation	104
2.4.4	Data preparation	105
2.4.5	Data processing	105
2.4.6	Performance monitoring	106
2.4.7	Troubleshooting	106
2.4.8	What's next	106

2.5	Import Example Database	107
2.5.1	Download all data files	107
2.5.2	Load data into TiDB	107
3	Deploy	108
3.1	Software and Hardware Recommendations.....	108
3.1.1	Linux OS version requirements	108
3.1.2	Software recommendations	109
3.1.3	Server recommendations	110
3.1.4	Network requirements	112
3.1.5	Web browser requirements	118
3.2	TiDB Environment and System Configuration Check	118
3.2.1	Mount the data disk ext4 filesystem with options on the target machines that deploy TiKV	118
3.2.2	Check and disable system swap	120
3.2.3	Check and stop the firewall service of target machines	121
3.2.4	Check and install the NTP service	121
3.2.5	Check and configure the optimal parameters of the operating system ..	124
3.2.6	Manually configure the SSH mutual trust and sudo without password ..	129
3.2.7	Install the numactl tool	130
3.3	Plan Cluster Topology.....	131
3.3.1	Minimal Deployment Topology	131
3.3.2	TiFlash Deployment Topology	133
3.3.3	TiCDC Deployment Topology	136
3.3.4	TiDB Binlog Deployment Topology	139
3.3.5	TiSpark Deployment Topology	143
3.3.6	Geo-Distributed Deployment Topology	146
3.3.7	Hybrid Deployment Topology	150
3.4	Install and Start	155
3.4.1	Deploy a TiDB Cluster Using TiUP	155
3.4.2	Deploy a TiDB Cluster in Kubernetes	164

3.5	Check Cluster Status	164
3.5.1	Check the TiDB cluster status	164
3.5.2	Log in to the database and perform simple operations	167
3.6	Test Cluster Performance	169
3.6.1	How to Test TiDB Using Sysbench	169
3.6.2	How to Run TPC-C Test on TiDB	174
4	Migrate	176
4.1	Migration Overview	176
4.1.1	Migrate from Aurora to TiDB	176
4.1.2	Migrate from MySQL to TiDB	177
4.1.3	Migrate data from files to TiDB	177
4.2	Migrate from MySQL	178
4.2.1	Migrate from Amazon Aurora MySQL Using TiDB Lightning	178
4.2.2	Migrate from MySQL SQL Files Using TiDB Lightning	181
4.2.3	Migrate from Amazon Aurora MySQL Using DM	183
4.3	Migrate from CSV Files	183
4.3.1	TiDB Lightning CSV Support and Restrictions	183
4.3.2	LOAD DATA	189
4.4	Migrate from MySQL SQL Files Using TiDB Lightning	192
4.4.1	Step 1: Deploy TiDB Lightning	192
4.4.2	Step 2: Configure data source of TiDB Lightning	192
4.4.3	Step 3: Run TiDB Lightning to import data	193
4.5	Replicate Incremental Data between TiDB Clusters in Real Time	194
4.5.1	Implementation principles	194
4.5.2	Replication process	194
5	Maintain	196
5.1	Upgrade	196
5.1.1	Upgrade TiDB Using TiUP	196
5.1.2	Use TiDB Operator	202

5.2	Scale	202
5.2.1	Scale the TiDB Cluster Using TiUP	202
5.2.2	Use TiDB Operator	213
5.3	Backup and Restore	213
5.3.1	Use BR Tool (Recommended)	213
5.4	Time Zone Support	278
5.5	Daily Check	281
5.5.1	Key indicators of TiDB Dashboard	281
5.6	Maintain a TiFlash Cluster	287
5.6.1	Check the TiFlash version	287
5.6.2	TiFlash critical logs	287
5.6.3	TiFlash system table	288
5.7	TiUP Common Operations	288
5.7.1	View the cluster list	289
5.7.2	Start the cluster	289
5.7.3	View the cluster status	290
5.7.4	Modify the configuration	290
5.7.5	Replace with a hotfix package	291
5.7.6	Rename the cluster	292
5.7.7	Stop the cluster	292
5.7.8	Clean up cluster data	293
5.7.9	Destroy the cluster	293
5.8	Modify Configuration Online	294
5.8.1	Common Operations	294
5.9	Online Unsafe Recovery	344
5.9.1	Feature description	344
5.9.2	User scenarios	345
5.9.3	Usage	345
6	Monitor and Alert	347
6.1	TiDB Monitoring Framework Overview	347
6.1.1	About Prometheus in TiDB	347
6.1.2	About Grafana in TiDB	348

6.2	TiDB Monitoring API	350
6.2.1	Use the status interface	350
6.2.2	Use the metrics interface	352
6.3	Deploy Monitoring Services for the TiDB Cluster	352
6.3.1	Deploy Prometheus and Grafana	352
6.3.2	Configure Grafana	356
6.3.3	View component metrics	357
6.4	Export Grafana Snapshots	358
6.4.1	Usage	358
6.4.2	FAQs	359
6.5	TiDB Cluster Alert Rules	360
6.5.1	TiDB alert rules	362
6.5.2	PD alert rules	366
6.5.3	TiKV alert rules	371
6.5.4	TiFlash alert rules	381
6.5.5	TiDB Binlog alert rules	381
6.5.6	TiCDC Alert rules	381
6.5.7	Node_exporter host alert rules	381
6.5.8	Blackbox_exporter TCP, ICMP, and HTTP alert rules	385
6.6	TiFlash Alert Rules	390
6.6.1	TiFlash_schema_error	390
6.6.2	TiFlash_schema_apply_duration	390
6.6.3	TiFlash_raft_read_index_duration	390
6.6.4	TiFlash_raft_wait_index_duration	391
7	Troubleshoot	391
7.1	TiDB Troubleshooting Map	391
7.1.1	1. Service Unavailable	391
7.1.2	2. Latency increases significantly	392
7.1.3	3. TiDB issues	393
7.1.4	4. TiKV issues	397
7.1.5	5. PD issues	401
7.1.6	6. Ecosystem tools	404
7.1.7	7. Common log analysis	409

7.2	Identify Slow Queries	412
7.2.1	Usage example	412
7.2.2	Fields description	413
7.2.3	Related system variables	415
7.2.4	Memory mapping in slow log	417
7.2.5	<code>SLOW_QUERY / CLUSTER_SLOW_QUERY</code> usage examples	418
7.2.6	Query slow queries with pseudo <code>stats</code>	420
7.2.7	Identify problematic SQL statements	425
7.3	Analyze Slow Queries	430
7.3.1	Identify the performance bottleneck of the query	430
7.3.2	Analyze system issues	431
7.3.3	Analyze optimizer issues	438
7.4	SQL Diagnostics	438
7.4.1	Overview	439
7.4.2	Cluster information tables	439
7.4.3	Cluster monitoring tables	440
7.4.4	Automatic diagnostics	441
7.5	Identify Expensive Queries	441
7.5.1	Expensive query log example	441
7.5.2	Fields description	442
7.6	Statement Summary Tables	443
7.6.1	<code>statements_summary</code>	443
7.6.2	<code>statements_summary_history</code>	445
7.6.3	<code>statements_summary_evicted</code>	445
7.6.4	The <code>cluster</code> tables for statement summary	445
7.6.5	Parameter configuration	446
7.6.6	Limitation	448
7.6.7	Troubleshooting examples	448
7.6.8	Fields description	450

7.7	Troubleshoot Hotspot Issues	453
7.7.1	Common hotspots	453
7.7.2	Identify hotspot issues	454
7.7.3	Use <code>SHARD_ROW_ID_BITS</code> to process hotspots	458
7.7.4	Handle auto-increment primary key hotspot tables using <code>AUTO_RANDOM</code>	461
7.7.5	Optimization of small table hotspots	464
7.8	Troubleshoot Increased Read and Write Latency	464
7.8.1	Common causes	464
7.8.2	Other causes	468
7.9	Use <code>PLAN REPLAYER</code> to Save and Restore the On-Site Information of a Cluster	469
7.9.1	Use <code>PLAN REPLAER</code> to export cluster information	469
7.9.2	Use <code>PLAN REPLAYER</code> to import cluster information	471
7.10	TiDB Cluster Troubleshooting Guide	471
7.10.1	Cannot connect to the database	472
7.10.2	Cannot start <code>tidb-server</code>	472
7.10.3	Cannot start <code>tikv-server</code>	473
7.10.4	Cannot start <code>pd-server</code>	473
7.10.5	The TiDB/TiKV/PD process aborts unexpectedly	473
7.10.6	TiDB panic	473
7.10.7	The connection is rejected	474
7.10.8	Open too many files	474
7.10.9	Database access times out and the system load is too high	474
7.11	Troubleshoot High Disk I/O Usage in TiDB	474
7.11.1	Check the current I/O metrics	475
7.11.2	Handle I/O issues	478
7.12	Troubleshoot Lock Conflicts	478
7.12.1	Optimistic transaction mode	478
7.12.2	Pessimistic transaction mode	485

7.13	Troubleshoot a TiFlash Cluster	494
7.13.1	TiFlash fails to start	494
7.13.2	TiFlash replica is always unavailable.....	495
7.13.3	TiFlash query time is unstable, and the error log prints many <code>Lock Exception</code> messages	496
7.13.4	Some queries return the <code>Region Unavailable</code> error	496
7.13.5	Data file corruption	496
7.14	Troubleshoot Write Conflicts in Optimistic Transactions.....	497
7.14.1	The reason of write conflicts.....	497
7.14.2	Detect write conflicts.....	497
7.14.3	Resolve write conflicts.....	499
8	Performance Tuning	500
8.1	System Tuning	500
8.1.1	Operating System Tuning.....	500
8.2	Software Tuning	505
8.2.1	Configuration	505
8.2.2	Coprocessor Cache.....	525
8.3	SQL Tuning	528
8.3.1	SQL Tuning Overview.....	528
8.3.2	Understanding the Query Execution Plan	528
8.3.3	SQL Optimization Process	593
8.3.4	Control Execution Plans	660
9	Tutorials	693
9.1	Multiple Data Centers in One City Deployment	693
9.1.1	Raft protocol	693
9.1.2	Three DCs in one city deployment	694
9.2	Three Data Centers in Two Cities Deployment	700
9.2.1	Overview	700
9.2.2	Deployment architecture	700
9.2.3	Configuration	703

9.3	Two Data Centers in One City Deployment	707
9.3.1	Deployment architecture	707
9.3.2	Configuration	709
9.4	Read Historical Data	714
9.4.1	Use Stale Read (Recommended)	714
9.4.2	Read Historical Data Using the System Variable <code>tidb_snapshot</code>	721
9.5	Best Practices	725
9.5.1	TiDB Best Practices	725
9.5.2	Best Practices for Developing Java Applications with TiDB	733
9.5.3	Best Practices for Using HAProxy in TiDB	745
9.5.4	Highly Concurrent Write Best Practices	758
9.5.5	Best Practices for Monitoring TiDB Using Grafana	769
9.5.6	PD Scheduling Best Practices	780
9.5.7	Best Practices for TiKV Performance Tuning with Massive Regions ..	789
9.5.8	Best Practices for Three-Node Hybrid Deployment	795
9.5.9	Local Read under Three Data Centers Deployment	800
9.6	Placement Rules	801
9.6.1	Rule system	802
9.6.2	Configure rules	804
9.6.3	Typical usage scenarios	809
9.7	Load Base Split	812
9.7.1	Scenarios	813
9.7.2	Implementation principles	813
9.7.3	Usage	813
9.8	Store Limit	814
9.8.1	Implementation principles	814
9.8.2	Usage	815
10	TiDB Tools	816
10.1	TiDB Tools Overview	816
10.1.1	Deployment and operation Tools	816
10.1.2	Data management tools	817
10.1.3	OLAP Query tool	820

10.2	TiDB Tools Use Cases	820
10.2.1	Deploy and operate TiDB on physical or virtual machines	820
10.2.2	Deploy and operate TiDB in Kubernetes	820
10.2.3	Import data from CSV to TiDB	820
10.2.4	Import full data from MySQL/Aurora	820
10.2.5	Migrate data from MySQL/Aurora	820
10.2.6	Back up and restore TiDB cluster	821
10.2.7	Migrate data to TiDB	821
10.2.8	TiDB incremental data subscription	821
10.3	Download TiDB Tools	821
10.3.1	TiUP	821
10.3.2	TiDB Operator	821
10.3.3	TiDB Binlog	821
10.3.4	TiDB Lightning	822
10.3.5	BR (backup and restore)	823
10.3.6	TiDB DM (Data Migration)	824
10.3.7	Dumpling	825
10.3.8	sync-diff-inspector	826
10.3.9	TiCDC	827
10.4	TiUP	827
10.4.1	TiUP Documentation Map	827
10.4.2	TiUP Overview	827
10.4.3	TiUP Terminology and Concepts	831
10.4.4	Manage TiUP Components with TiUP Commands	832
10.4.5	TiUP FAQ	836
10.4.6	TiUP Troubleshooting Guide	837
10.4.7	TiUP Reference	839
10.4.8	Topology Configuration File for TiDB Deployment Using TiUP	842
10.4.9	TiUP Mirror Reference Guide	865
10.4.10	TiUP Components	875

10.5	TiDB Operator	901
10.6	Backup & Restore (BR)	902
10.6.1	BR Tool Overview	902
10.6.2	Use BR Command-line for Backup and Restoration	919
10.6.3	BR Use Cases	933
10.6.4	External Storages	953
10.6.5	Backup & Restore FAQ	963
10.7	TiDB Binlog	968
10.7.1	TiDB Binlog Cluster Overview	968
10.7.2	TiDB Binlog Tutorial	970
10.7.3	TiDB Binlog Cluster Deployment	980
10.7.4	TiDB Binlog Cluster Operations	992
10.7.5	TiDB Binlog Configuration File	996
10.7.6	Upgrade TiDB Binlog	1014
10.7.7	TiDB Binlog Monitoring	1016
10.7.8	Reparo User Guide	1033
10.7.9	binlogctl	1037
10.7.10	Binlog Consumer Client User Guide	1040
10.7.11	TiDB Binlog Relay Log	1043
10.7.12	Bidirectional Replication between TiDB Clusters	1045
10.7.13	TiDB Binlog Glossary	1050
10.7.14	Troubleshoot	1050
10.7.15	TiDB Binlog FAQ	1052
10.8	TiDB Lightning	1060
10.8.1	TiDB Lightning Overview	1060
10.8.2	TiDB Lightning Tutorial	1063
10.8.3	TiDB Lightning Deployment	1066
10.8.4	TiDB Lightning Prechecks	1070
10.8.5	TiDB Lightning Configuration	1077
10.8.6	Key Features	1097
10.8.7	TiDB Lightning Monitoring	1135
10.8.8	TiDB Lightning FAQs	1153
10.8.9	TiDB Lightning Glossary	1163

10.9	TiDB Data Migration	1167
10.9.1	DM versions	1167
10.9.2	Basic features	1168
10.9.3	Advanced features	1169
10.9.4	Usage restrictions	1170
10.10	TiCDC	1171
10.10.1	TiCDC Overview	1171
10.10.2	Deploy TiCDC	1180
10.10.3	Manage TiCDC Cluster and Replication Tasks	1183
10.10.4	Troubleshoot TiCDC	1203
10.10.5	Key Monitoring Metrics of TiCDC	1219
10.10.6	TiCDC Alert Rules	1225
10.10.7	TiCDC OpenAPI	1229
10.10.8	TiCDC Open Protocol	1242
10.10.9	Quick Start Guide on Integrating TiDB with Confluent Platform	1254
10.10.10	TiCDC Glossary	1257
10.11	Dumpling Overview	1258
10.11.1	Improvements of Dumpling compared with Mydumper	1258
10.11.2	Dumpling introduction	1258
10.11.3	Export data from TiDB/MySQL	1259
10.11.4	Option list of Dumpling	1266
10.12	sync-diff-inspector	1271
10.12.1	sync-diff-inspector User Guide	1271
10.12.2	Data Check for Tables with Different Schema or Table Names	1279
10.12.3	Data Check in the Sharding Scenario	1281
10.12.4	Data Check for TiDB Upstream and Downstream Clusters	1285
10.12.5	Data Check in the DM Replication Scenario	1286
10.13	TiSpark	1287
10.13.1	TiSpark Quick Start Guide	1287
10.13.2	TiSpark User Guide	1291

11.1	Cluster Architecture	1301
11.1.1	TiDB Architecture	1301
11.1.2	TiDB Storage	1302
11.1.3	TiDB Computing	1308
11.1.4	TiDB Scheduling	1315
11.2	Key Monitoring Metrics	1319
11.2.1	Key Metrics	1319
11.2.2	TiDB Monitoring Metrics	1325
11.2.3	Key Monitoring Metrics of PD	1331
11.2.4	Key Monitoring Metrics of TiKV	1341
11.2.5	Monitor the TiFlash Cluster	1362
11.3	Secure	1366
11.3.1	Enable TLS between TiDB Clients and Servers	1366
11.3.2	Enable TLS Between TiDB Components	1371
11.3.3	Generate Self-Signed Certificates	1375
11.3.4	Encryption at Rest	1378
11.3.5	Enable Encryption for Disk Spill	1384
11.3.6	Log Redaction	1385
11.4	Privileges	1386
11.4.1	Security Compatibility with MySQL	1386
11.4.2	Privilege Management	1387
11.4.3	TiDB User Account Management	1397
11.4.4	Role-Based Access Control	1401
11.4.5	Certificate-Based Authentication for Login	1408
11.5	SQL	1418
11.5.1	SQL Language Structure and Syntax	1418
11.5.2	SQL Statements	1458
11.5.3	Data Types	1788
11.5.4	Functions and Operators	1806
11.5.5	Clustered Indexes	1849
11.5.6	Constraints	1855
11.5.7	Generated Columns	1861

11.5.8	SQL Mode	1865
11.5.9	Table Attributes	1891
11.5.10	Transactions	1894
11.5.11	Garbage Collection (GC)	1917
11.5.12	Views	1921
11.5.13	Partitioning	1926
11.5.14	Temporary Tables	1957
11.5.15	Character Set and Collation	1964
11.5.16	Placement Rules in SQL	1974
11.5.17	System Tables	1980
11.6	UI	2098
11.6.1	TiDB Dashboard	2098
11.7	CLI	2233
11.7.1	TiKV Control User Guide	2233
11.7.2	PD Control User Guide	2248
11.7.3	TiDB Control User Guide	2273
11.7.4	PD Recover User Guide	2280
11.8	Command Line Flags	2284
11.8.1	Configuration Options	2284
11.8.2	TiKV Configuration Flags	2289
11.8.3	TiFlash Command-Line Flags	2292
11.8.4	PD Configuration Flags	2292
11.9	Configuration File Parameters	2295
11.9.1	TiDB Configuration File	2295
11.9.2	TiKV Configuration File	2315
11.9.3	Configure TiFlash	2354
11.9.4	PD Configuration File	2363
11.10	System Variables	2373
11.10.1	Variable Reference	2374
11.11	Storage Engines	2426
11.11.1	TiKV	2426
11.11.2	TiFlash	2443

11.12 Telemetry	2457
11.12.1 What is shared?	2457
11.12.2 Disable telemetry	2459
11.12.3 Check telemetry status	2461
11.12.4 Compliance	2462
11.13 Error Codes and Troubleshooting	2462
11.13.1 Error codes	2462
11.13.2 Troubleshooting	2471
11.14 Table Filter	2471
11.14.1 Usage	2471
11.14.2 Syntax	2472
11.14.3 Multiple rules	2475
11.15 Schedule Replicas by Topology Labels	2476
11.15.1 Configure <code>labels</code> based on the cluster topology	2477
11.15.2 PD schedules based on topology label	2480
12 FAQs	2481
12.1 TiDB FAQ	2481
12.1.1 About TiDB	2481
12.1.2 Deployment on the cloud	2483
12.1.3 Troubleshoot	2483
12.2 SQL FAQs	2486
12.2.1 What are the MySQL variables that TiDB is compatible with?	2486
12.2.2 The order of results is different from MySQL when <code>ORDER BY</code> is omitted	2486
12.2.3 Does TiDB support <code>SELECT FOR UPDATE</code> ?	2487
12.2.4 Can the codec of TiDB guarantee that the UTF-8 string is memcomparable? Is there any coding suggestion if our key needs to support UTF-8?	2488
12.2.5 What is the maximum number of statements in a transaction?	2488
12.2.6 Why does the auto-increment ID of the later inserted data is smaller than that of the earlier inserted data in TiDB?	2488
12.2.7 How do I modify the <code>sql_mode</code> in TiDB?	2488

12.2.8	Error: <code>java.sql.BatchUpdateException:statement count 5001 exceeds the transaction limitation while using Sqoop to write data into TiDB in batches</code>	2488
12.2.9	Does TiDB have a function like the Flashback Query in Oracle? Does it support DDL?.....	2489
12.2.10	Does TiDB release space immediately after deleting data?	2489
12.2.11	Does TiDB support the <code>REPLACE INTO</code> syntax?	2489
12.2.12	Why does the query speed get slow after data is deleted?	2489
12.2.13	What should I do if it is slow to reclaim storage space after deleting data?	2489
12.2.14	Does <code>SHOW PROCESSLIST</code> display the system process ID?.....	2490
12.2.15	How to control or change the execution priority of SQL commits?.....	2490
12.2.16	What's the trigger strategy for <code>auto analyze</code> in TiDB?.....	2491
12.2.17	Can I use hints to override the optimizer behavior?	2491
12.2.18	Why the <code>Information schema is changed</code> error is reported?	2491
12.2.19	What are the causes of the “ <code>Information schema is out of date</code> ” error?..	2492
12.2.20	Error is reported when executing DDL statements under high concurrency?	2492
12.2.21	SQL optimization.....	2492
12.2.22	Database optimization.....	2494
12.3	Deployment, Operations and Maintenance FAQs	2495
12.3.1	Operating system requirements	2495
12.3.2	Server requirements	2495
12.3.3	Installation and deployment	2497
12.3.4	Cluster management	2504
12.3.5	Monitoring	2517
12.4	Upgrade and After Upgrade FAQs	2518
12.4.1	Upgrade FAQs	2518
12.4.2	After upgrade FAQs.....	2518
12.5	High Availability FAQs	2523
12.5.1	How is TiDB strongly consistent?	2524
12.5.2	What's the recommended solution for the deployment of three geo-distributed data centers?.....	2524

12.6	High Reliability FAQs	2524
12.6.1	Does TiDB support modifying the MySQL version string of the server to a specific one that is required by the security vulnerability scanning tool?.....	2524
12.6.2	What authentication protocols does TiDB support? What's the process?.....	2524
12.6.3	How to modify the user password and privilege?.....	2525
12.7	Migration FAQs	2525
12.7.1	Full data export and import.....	2525
12.7.2	Migrate the data online.....	2528
12.7.3	Migrate the traffic	2528
13	Glossary	2530
13.1	A	2530
13.1.1	ACID	2530
13.2	L	2531
13.2.1	leader/follower/learner	2531
13.3	O	2531
13.3.1	Old value	2531
13.3.2	Operator	2531
13.3.3	Operator step	2531
13.4	P	2531
13.4.1	pending/down	2531
13.5	R	2532
13.5.1	Region/peer/Raft group	2532
13.5.2	Region split	2532
13.5.3	restore	2532
13.6	S	2532
13.6.1	scheduler	2532
13.6.2	Store	2532
13.7	T	2533
13.7.1	TSO	2533
14	Release Notes	2533

14.1	TiDB Release Notes	2533
14.1.1	5.3	2533
14.1.2	5.2	2533
14.1.3	5.1	2533
14.1.4	5.0	2533
14.1.5	4.0	2534
14.1.6	3.1	2534
14.1.7	3.0	2534
14.1.8	2.1	2535
14.1.9	2.0	2536
14.1.10	1.0	2536
14.2	TiDB Release Timeline	2537
14.3	v5.3	2540
14.3.1	TiDB 5.3 Release Notes	2540
14.4	v5.2	2565
14.4.1	TiDB 5.2.3 Release Note	2565
14.4.2	TiDB 5.2.2 Release Notes	2565
14.4.3	TiDB 5.2.1 Release Notes	2569
14.4.4	TiDB 5.2 Release Notes	2569
14.5	v5.1	2586
14.5.1	TiDB 5.1.3 Release Note	2586
14.5.2	TiDB 5.1.2 Release Notes	2586
14.5.3	TiDB 5.1.1 Release Notes	2590
14.5.4	TiDB 5.1 Release Notes	2595
14.6	v5.0	2613
14.6.1	TiDB 5.0.5 Release Note	2613
14.6.2	TiDB 5.0.4 Release Notes	2613
14.6.3	TiDB 5.0.3 Release Notes	2619
14.6.4	TiDB 5.0.2 Release Notes	2623
14.6.5	TiDB 5.0.1 Release Notes	2627
14.6.6	What's New in TiDB 5.0	2629
14.6.7	TiDB 5.0 RC Release Notes	2646

14.7	v4.0	2653
14.7.1	TiDB 4.0.16 Release Notes	2653
14.7.2	TiDB 4.0.15 Release Notes	2656
14.7.3	TiDB 4.0.14 Release Notes	2660
14.7.4	TiDB 4.0.13 Release Notes	2665
14.7.5	TiDB 4.0.12 Release Notes	2669
14.7.6	TiDB 4.0.11 Release Notes	2673
14.7.7	TiDB 4.0.10 Release Notes	2677
14.7.8	TiDB 4.0.9 Release Notes	2680
14.7.9	TiDB 4.0.8 Release Notes	2687
14.7.10	TiDB 4.0.7 Release Notes	2691
14.7.11	TiDB 4.0.6 Release Notes	2693
14.7.12	TiDB 4.0.5 Release Notes	2698
14.7.13	TiDB 4.0.4 Release Notes	2703
14.7.14	TiDB 4.0.3 Release Notes	2703
14.7.15	TiDB 4.0.2 Release Notes	2708
14.7.16	TiDB 4.0.1 Release Notes	2713
14.7.17	TiDB 4.0 GA Release Notes	2714
14.7.18	TiDB 4.0 RC.2 Release Notes	2718
14.7.19	TiDB 4.0 RC.1 Release Notes	2724
14.7.20	TiDB 4.0 RC Release Notes	2728
14.7.21	TiDB 4.0.0 Beta.2 Release Notes	2731
14.7.22	TiDB 4.0.0 Beta.1 Release Notes	2733
14.7.23	TiDB 4.0 Beta Release Notes	2736
14.8	v3.1	2740
14.8.1	TiDB 3.1.2 Release Notes	2740
14.8.2	TiDB 3.1.1 Release Notes	2740
14.8.3	TiDB 3.1.0 GA Release Notes	2742
14.8.4	TiDB 3.1 RC Release Notes	2744
14.8.5	TiDB 3.1 Beta.2 Release Notes	2747
14.8.6	TiDB 3.1 Beta.1 Release Notes	2749
14.8.7	TiDB 3.1 Beta Release Notes	2750

14.9 v3.0	2751
14.9.1 TiDB 3.0.20 Release Notes	2751
14.9.2 TiDB 3.0.19 Release Notes	2753
14.9.3 TiDB 3.0.18 Release Notes	2754
14.9.4 TiDB 3.0.17 Release Notes	2755
14.9.5 TiDB 3.0.16 Release Notes	2756
14.9.6 TiDB 3.0.15 Release Notes	2757
14.9.7 TiDB 3.0.14 Release Notes	2758
14.9.8 TiDB 3.0.13 Release Notes	2762
14.9.9 TiDB 3.0.12 Release Notes	2763
14.9.10 TiDB 3.0.11 Release Notes	2764
14.9.11 TiDB 3.0.10 Release Notes	2766
14.9.12 TiDB 3.0.9 Release Notes	2768
14.9.13 TiDB 3.0.8 Release Notes	2770
14.9.14 TiDB 3.0.7 Release Notes	2775
14.9.15 TiDB 3.0.6 Release Notes	2775
14.9.16 TiDB 3.0.5 Release Notes	2779
14.9.17 TiDB 3.0.4 Release Notes	2782
14.9.18 TiDB 3.0.3 Release Notes	2787
14.9.19 TiDB 3.0.2 Release Notes	2790
14.9.20 TiDB 3.0.1 Release Notes	2795
14.9.21 TiDB 3.0 GA Release Notes	2799
14.9.22 TiDB 3.0.0-rc.3 Release Notes	2806
14.9.23 TiDB 3.0.0-rc.2 Release Notes	2810
14.9.24 TiDB 3.0.0-rc.1 Release Notes	2813
14.9.25 TiDB 3.0.0 Beta.1 Release Notes	2818
14.9.26 TiDB 3.0 Beta Release Notes	2821
14.10 v2.1	2825
14.10.1 TiDB 2.1.19 Release Notes	2825
14.10.2 TiDB 2.1.18 Release Notes	2828
14.10.3 TiDB 2.1.17 Release Notes	2831
14.10.4 TiDB 2.1.16 Release Notes	2834

14.10.5 TiDB 2.1.15 Release Notes	2836
14.10.6 TiDB 2.1.14 Release Notes	2838
14.10.7 TiDB 2.1.13 Release Notes	2840
14.10.8 TiDB 2.1.12 Release Notes	2841
14.10.9 TiDB 2.1.11 Release Notes	2842
14.10.10TiDB 2.1.10 Release Notes	2843
14.10.11TiDB 2.1.9 Release Notes	2844
14.10.12TiDB 2.1.8 Release Notes	2846
14.10.13TiDB 2.1.7 Release Notes	2848
14.10.14TiDB 2.1.6 Release Notes	2849
14.10.15TiDB 2.1.5 Release Notes	2850
14.10.16TiDB 2.1.4 Release Notes	2852
14.10.17TiDB 2.1.3 Release Notes	2853
14.10.18TiDB 2.1.2 Release Notes	2855
14.10.19TiDB 2.1.1 Release Notes	2856
14.10.20TiDB 2.1 GA Release Notes	2857
14.10.21TiDB 2.1 RC5 Release Notes	2863
14.10.22TiDB 2.1 RC4 Release Notes	2865
14.10.23TiDB 2.1 RC3 Release Notes	2866
14.10.24TiDB 2.1 RC2 Release Notes	2868
14.10.25TiDB 2.1 RC1 Release Notes	2872
14.10.26TiDB 2.1 Beta Release Notes	2877
14.11 v2.0	2879
14.11.1 TiDB 2.0.11 Release Notes	2879
14.11.2 TiDB 2.0.10 Release Notes	2880
14.11.3 TiDB 2.0.9 Release Notes	2881
14.11.4 TiDB 2.0.8 Release Notes	2882
14.11.5 TiDB 2.0.7 Release Notes	2883
14.11.6 TiDB 2.0.6 Release Notes	2884
14.11.7 TiDB 2.0.5 Release Notes	2886
14.11.8 TiDB 2.0.4 Release Notes	2887
14.11.9 TiDB 2.0.3 Release Notes	2888

14.11.10TiDB 2.0.2 Release Notes	2889
14.11.11TiDB 2.0.1 Release Notes	2889
14.11.12TiDB 2.0 Release Notes	2890
14.11.13TiDB 2.0 RC5 Release Notes	2895
14.11.14TiDB 2.0 RC4 Release Notes	2896
14.11.15TiDB 2.0 RC3 Release Notes	2897
14.11.16TiDB 2.0 RC1 Release Notes	2898
14.11.17TiDB 1.1 Beta Release Notes	2899
14.11.18TiDB 1.1 Alpha Release Notes	2901
14.12 v1.0	2902
14.12.1 TiDB 1.0.8 Release Notes	2902
14.12.2 TiDB 1.0.7 Release Notes	2903
14.12.3 TiDB 1.0.6 Release Notes	2904
14.12.4 TiDB 1.0.5 Release Notes	2904
14.12.5 TiDB 1.0.4 Release Notes	2905
14.12.6 TiDB 1.0.3 Release Notes	2905
14.12.7 TiDB 1.0.2 Release Notes	2906
14.12.8 TiDB 1.0.1 Release Notes	2907
14.12.9 TiDB 1.0 Release Notes	2907
14.12.10Pre-GA Release Notes	2913
14.12.11TiDB RC4 Release Notes	2914
14.12.12TiDB RC3 Release Notes	2915
14.12.13TiDB RC2 Release Notes	2917
14.12.14TiDB RC1 Release Notes	2918

1 About TiDB

1.1 TiDB Introduction

TiDB (/’ta di bi:/, “Ti” stands for Titanium) is an open-source NewSQL database that supports Hybrid Transactional and Analytical Processing (HTAP) workloads. It is MySQL compatible and features horizontal scalability, strong consistency, and high availability. The goal of TiDB is to provide users with a one-stop database solution that covers OLTP (Online Transactional Processing), OLAP (Online Analytical Processing), and HTAP services. TiDB is suitable for various use cases that require high availability and strong consistency with large-scale data.

1.1.1 Key features

- **Horizontally scaling out or scaling in easily**

The TiDB architecture design of separating computing from storage enables you to separately scale out or scale in the computing or storage capacity online as needed. The scaling process is transparent to application operations and maintenance staff.

- **Financial-grade high availability**

The data is stored in multiple replicas. Data replicas obtain the transaction log using the Multi-Raft protocol. A transaction can be committed only when data has been successfully written into the majority of replicas. This can guarantee strong consistency, and availability when a minority of replicas go down. To meet the requirements of different disaster tolerance levels, you can configure the geographic location and number of replicas as needed.

- **Real-time HTAP**

TiDB provides two storage engines: **TiKV**, a row-based storage engine, and **TiFlash**, a columnar storage engine. TiFlash uses the Multi-Raft Learner protocol to replicate data from TiKV in real time, ensuring that the data between the TiKV row-based storage engine and the TiFlash columnar storage engine are consistent. TiKV and TiFlash can be deployed on different machines as needed to solve the problem of HTAP resource isolation.

- **Cloud-native distributed database**

TiDB is a distributed database designed for the cloud, providing flexible scalability, reliability and security on the cloud platform. Users can elastically scale TiDB to meet the requirements of their changing workloads. In TiDB, each piece of data has 3 replicas at least, which can be scheduled in different cloud availability zones to tolerate the outage of a whole data center. **TiDB Operator** helps manage TiDB on Kubernetes and automates tasks related to operating the TiDB cluster, which makes TiDB easier to deploy on any cloud that provides managed Kubernetes. **TiDB Cloud** (Beta), the fully-managed TiDB service, is the easiest, most economical, and most resilient way

to unlock the full power of TiDB in the cloud, allowing you to deploy and run TiDB clusters with just a few clicks.

- **Compatible with the MySQL 5.7 protocol and MySQL ecosystem**

TiDB is compatible with the MySQL 5.7 protocol, common features of MySQL, and the MySQL ecosystem. To migrate your applications to TiDB, you do not need to change a single line of code in many cases or only need to modify a small amount of code. In addition, TiDB provides a series of [data migration tools](#) to help easily migrate application data into TiDB.

1.1.2 Use cases

- **Financial industry scenarios with high requirements for data consistency, reliability, availability, scalability, and disaster tolerance**

As we all know, the financial industry has high requirements for data consistency, reliability, availability, scalability, and disaster tolerance. The traditional solution is to provide services in two data centers in the same city, and provide data disaster recovery but no services in a third data center located in another city. This solution has the disadvantages of low resource utilization, high maintenance cost, and the fact that RTO (Recovery Time Objective) and RPO (Recovery Point Objective) cannot meet expectations. TiDB uses multiple replicas and the Multi-Raft protocol to schedule data to different data centers, racks, and machines. When some machines fail, the system can automatically switch to ensure that the system RTO = 30s and RPO = 0.

- **Massive data and high concurrency scenarios with high requirements for storage capacity, scalability, and concurrency**

As applications grow rapidly, the data surges. Traditional standalone databases cannot meet the data capacity requirements. The solution is to use sharding middleware or a NewSQL database (like TiDB), and the latter is more cost-effective. TiDB adopts a separate computing and storage architecture, which enables you to scale out or scale in the computing or storage capacity separately. The computing layer supports a maximum of 512 nodes, each node supports a maximum of 1,000 concurrencies, and the maximum cluster capacity is at the PB (petabytes) level.

- **Real-time HTAP scenarios**

With the fast growth of 5G, Internet of Things, and artificial intelligence, the data generated by a company keeps increasing tremendously, reaching a scale of hundreds of TB (terabytes) or even the PB level. The traditional solution is to process online transactional applications using an OLTP database and use an ETL (Extract, Transform, Load) tool to replicate the data into an OLAP database for data analysis. This solution has multiple disadvantages such as high storage costs and poor real-time performance. TiDB introduces the TiFlash columnar storage engine in v4.0, which combines with the TiKV row-based storage engine to build TiDB as a true HTAP database. With a small amount of extra storage cost, you can handle both online transactional processing and real-time data analysis in the same system, which greatly saves the cost.

- **Data aggregation and secondary processing scenarios**

The application data of most companies are scattered in different systems. As the application grows, the decision-making leaders need to understand the business status of the entire company to make decisions in time. In this case, the company needs to aggregate the scattered data into the same system and execute secondary processing to generate a T+0 or T+1 report. The traditional solution is to use ETL and Hadoop, but the Hadoop system is complicated, with high operations and maintenance cost and storage cost. Compared with Hadoop, TiDB is much simpler. You can replicate data into TiDB using ETL tools or data migration tools provided by TiDB. Reports can be directly generated using SQL statements.

1.1.3 See also

- [TiDB Architecture](#)
- [TiDB Storage](#)
- [TiDB Computing](#)
- [TiDB Scheduling](#)

1.2 TiDB 5.3 Release Notes

Release date: November 30, 2021

TiDB version: 5.3.0

In v5.3, the key new features or improvements are as follows:

- Introduce temporary tables to simplify your application logic and improve performance
- Support setting attributes for tables and partitions
- Support creating users with the least privileges on TiDB Dashboard to enhance system security
- Optimize the timestamp processing flow in TiDB to improve the overall performance
- Enhance the performance of TiDB Data Migration (DM) so that data is migrated from MySQL to TiDB with lower latency
- Support parallel import using multiple TiDB Lightning instances to improve the efficiency of full data migration
- Support saving and restoring the on-site information of a cluster with a single SQL statement, which helps improve the efficiency of troubleshooting issues relating to execution plans
- Support the continuous profiling experimental feature to improve the observability of database performance
- Continue optimizing the storage and computing engines to improve the system performance and stability

1.2.1 Compatibility changes

Note:

When upgrading from an earlier TiDB version to v5.3.0, if you want to know the compatibility change notes of all intermediate versions, you can check the [Release Notes](#) of the corresponding version.

1.2.1.1 System variables

Variable	name	Change type	Description
<code>tidb_enable_noop_functions</code>	Temporary tables are now supported by TiDB so CREATE → TEMPORARY → TABLE and DROP → TEMPORARY → TABLE no longer require enabling <code>tidb_enable_noop_functions</code> → .	Modifications	Temporary tables are now supported by TiDB so CREATE → TEMPORARY → TABLE and DROP → TEMPORARY → TABLE no longer require enabling <code>tidb_enable_noop_functions</code> → .

Variable		
name	Change type	Description
<code>tidb_enable_pseudo_estimate_ratio</code>	Newly added →	<p>Controls the behavior of the optimizer when the statistics on a table expire. The default value is <code>ON</code>. When the number of modified rows in the table is greater than 80% of the total rows (This ratio can be adjusted by the configuration <code>pseudo-estimate</code> → <code>ratio</code>), the optimizer considers that the statistics other than the total number of rows are no longer reliable and use pseudo statistics instead.</p> <p>When you set the value as <code>OFF</code>, even if the statistics expire, the optimizer still uses them.</p>

Variable name	Change type	Description
<code>tidb_enable_tso_follower_proxy</code>	Newly added proxy →	<p>Determines whether to enable or disable the TSO Follower Proxy feature. The default value is OFF, which means the TSO Follower Proxy feature is disabled.</p> <p>At this time, TiDB only gets TSO from PD leader. When this feature is enabled, TiDB evenly sends the requests to all PD nodes when acquiring TSO. The PD follower then forwards the TSO requests to reduce the CPU pressure of PD leader.</p>

Variable			
name	Change type	Description	
<code>tidb_tso_client_new_batched_max_set_time</code>	New by adding ↪	maximum waiting time for a batch saving operation when TiDB requests TSO from PD. The default value is 0, which means no additional waiting.	
<code>tidb_tmp_table_new_max_size</code>	New by adding ↪	Limits the maximum size of a single temporary table. If the temporary table exceeds this size, an error will occur.	

1.2.1.2 Configuration file parameters

Configuration file	Configuration item	Change type	Description
TiDB	<code>prepared-plan-cache.</code> ↪ ↪ ↪ ↪ ↪	Modified	Controls the number of cached statements. The default value is changed from 100 to 1000.

file	item	Change type	Description
TiKV	<code>storage.</code> ↳ <code>reserve</code> ↳ <code>-space</code>	Modified	Controls space reserved for disk protection when TiKV is started. Starting from v5.3.0, 80% of the reserved space is used as the extra disk space required for operations and maintenance when the disk space is insufficient, and the other 20% is used to store temporary files.
TiKV	<code>memory-</code> ↳ <code>usage-</code> ↳ <code>limit</code>	Modified	This configuration item is new in TiDB v5.3.0 and its value is calculated based on storage.block-cache.capacity.

file	item	Change type	Description
TiKV	raftstore. ↵ store- ↵ io-pool ↵ -size	Newly added	The allowable number of threads that process Raft I/O tasks, which is the size of the StoreWriter thread pool. When you modify the size of this thread pool, refer to Performance tuning for TiKV thread pools .

file	item	Change type	Description
TiKV	raftstore. ↳ raft- ↳ write- ↳ size- ↳ limit	Newly added threshold at which Raft data is written into the disk. If the data size is larger than the value of this configuration item, the data is written to the disk. When the value of raftstore. ↳ store- ↳ io-pool ↳ -size is 0, this configuration item does not take effect.	Determines the threshold at which Raft data is written into the disk. If the data size is larger than the value of this configuration item, the data is written to the disk. When the value of raftstore. ↳ store- ↳ io-pool ↳ -size is 0, this configuration item does not take effect.

file	item	Change type	Description
TiKV	raftstore. ↳ raft- ↳ msg- ↳ flush- ↳ interval ↳	Newly added	<p>Determines the interval at which Raft messages are sent in batches.</p> <p>The Raft messages in batches are sent at every interval specified by this configuration item.</p> <p>When the value of raftstore. ↳ store- ↳ io-pool ↳ -size is 0, this configuration item does not take effect.</p>
TiKV	raftstore. ↳ raft- ↳ reject- ↳ transfer ↳ -leader ↳ - ↳ duration ↳	Deleted	<p>Determines the smallest duration that a Leader is transferred to a newly added node.</p>

Configuration file	Configuration item	Change type	Description
PD	<code>log.file.</code> ↳ <code>max-</code> ↳ <code>days</code>	Modified	Controls the maximum number of days that logs are retained for. The default value is changed from 1 to 0.
PD	<code>log.file.</code> ↳ <code>max-</code> ↳ <code>backups</code>	Modified	Controls the maximum number of logs that are retained for. The default value is changed from 7 to 0.

file	item	Change type	Description
PD	patrol- ↵ region- ↵ interval ↵	Modified	<p>Controls the running frequency at which replicaChecker checks the health state of a Region. The smaller this value is, the faster replicaChecker runs.</p> <p>Normally, you do not need to adjust this parameter. The default value is changed from 100ms to 10ms.</p>

file	item	Change type	Description
PD	max- ↵ snapshot ↵ -count	Modified	<p>Controls the maximum number of snapshots that a single store receives or sends at the same time.</p> <p>PD schedulers depend on this configuration to prevent the resources used for normal traffic from being preempted. The default value is changed from 3 to 64.</p>

file	item	Change type	Description
PD	max- ↵ pending ↵ -peer- ↵ count	Modified	Controls the maximum number of pending peers in a single store. PD schedulers depend on this configuration to prevent too many Regions with outdated logs from being generated on some nodes. The default value is changed from 16 to 64.

1.2.1.3 Others

- Temporary tables:
 - If you have created local temporary tables in a TiDB cluster earlier than v5.3.0, these tables are actually ordinary tables, and handled as ordinary tables after the cluster is upgraded to v5.3.0 or a later version. If you have created global temporary tables in a TiDB cluster of v5.3.0 or a later version, when the cluster is downgraded to a version earlier than v5.3.0, these tables are handled as ordinary tables and cause a data error.
 - Since v5.3.0, TiCDC and BR support **global temporary tables**. If you use TiCDC and BR of a version earlier than v5.3.0 to replicate global temporary tables to the downstream, a table definition error occurs.
 - The following clusters are expected to be v5.3.0 or later; otherwise, data error is reported when you create a global temporary table:

- * the cluster to be imported using TiDB ecosystem tools
 - * the cluster restored using TiDB ecosystem tools
 - * the downstream cluster in a replication task using TiDB ecosystem tools
- For the compatibility information of temporary tables, refer to [Compatibility with MySQL temporary tables](#) and [Compatibility restrictions with other TiDB features](#).
- For releases earlier than v5.3.0, TiDB reports an error when a system variable is set to an illegal value. For v5.3.0 and later releases, TiDB returns success with a warning such as “|Warning | 1292 | Truncated incorrect xxx: ‘xx’” when a system variable is set to an illegal value.
 - Fix the issue that the `SHOW VIEW` permission is not required to execute `SHOW CREATE → VIEW`. Now you are expected to have the `SHOW VIEW` permission to execute the `SHOW CREATE VIEW` statement.
 - The system variable `sql_auto_is_null` is added to the noop functions. When `tidb_enable_noop_functions = 0/OFF`, modifying this variable value causes an error.
 - The `GRANT ALL ON performance_schema.*` syntax is no longer permitted. If you execute this statement in TiDB, an error occurs.
 - Fix the issue that auto-analyze is unexpectedly triggered outside the specified time period when new indexes are added before v5.3.0. In v5.3.0, after you set the time period through the `tidb_auto_analyze_start_time` and `tidb_auto_analyze_end_time` variables, auto-analyze is triggered only during this time period.
 - The default storage directory for plugins is changed from “” to `/data/deploy/plugin`.
 - The DM code is migrated to [the folder “dm” in TiCDC code repository](#). Now DM follows TiDB in version numbers. Next to v2.0.x, the new DM version is v5.3.0, and you can upgrade from v2.0.x to v5.3.0 without any risk.

1.2.2 New features

1.2.2.1 SQL

- **Use SQL interface to set placement rules for data (experimental feature)**

Support the `[CREATE | ALTER] PLACEMENT POLICY` syntax that provides a SQL interface to set placement rules for data. Using this feature, you can specify tables and partitions to be scheduled to specific regions, data centers, racks, hosts, or replica count rules. This meets your application demands for lower cost and higher flexibility. The typical user scenarios are as follows:

- Merge multiple databases of different applications to reduce the cost on database maintenance, and achieve application resource isolation through the rule configuration
- Increase replica count for important data to improve the application availability and data reliability
- Store new data into SSDs and store old data into HDDs to lower the cost on data archiving and storage
- Schedule the leaders of hotspot data to high-performance TiKV instances
- Separate cold data to lower-cost storage mediums to improve cost efficiency

[User document, #18030](#)

- **Temporary tables**

Support the `CREATE [GLOBAL] TEMPORARY TABLE` statement to create temporary tables. Using this feature, you can easily manage the temporary data generated in the calculation process of an application. Temporary data is stored in memory and you can use the `tidb_tmp_table_max_size` variable to limit the size of a temporary table. TiDB supports the following types of temporary tables:

- Global temporary tables
 - * Visible to all sessions in the cluster, and table schemas are persistent.
 - * Provides transaction-level data isolation. The temporary data is effective only in the transaction. After the transaction finishes, the data is automatically dropped.
- Local temporary tables
 - * Visible only to the current session, and tables schemas are not persistent.
 - * Supports duplicated table names. You do not need to design complicated naming rules for your application.
 - * Provides session-level data isolation, which enables you to design a simpler application logic. After the transaction finishes, the temporary tables are dropped.

[User document, #24169](#)

- **Support the FOR UPDATE OF TABLES syntax**

For a SQL statement that joins multiple tables, TiDB supports acquiring pessimistic locks on the rows correlated to the tables that are included in `OF TABLES`.

[User document, #28689](#)

- **Table attributes**

Support the `ALTER TABLE [PARTITION] ATTRIBUTES` statement that allows you to set attributes for a table or partition. Currently, TiDB only supports setting the `merge_option` attribute. By adding this attribute, you can explicitly control the Region merge behavior.

User scenarios: When you perform the `SPLIT TABLE` operation, if no data is inserted after a certain period of time, the empty Regions are automatically merged by default. In this case, you can set the table attribute to `merge_option=deny` to avoid the automatic merging of Regions.

[User document](#), #3839

1.2.2.2 Security

- **Support creating users with the least privileges on TiDB Dashboard**

The account system of TiDB Dashboard is consistent with that of TiDB SQL. Users accessing TiDB Dashboard are authenticated and authorized based on TiDB SQL users' privileges. Therefore, TiDB Dashboard requires limited privileges, or merely the read-only privilege. You can configure users to access TiDB Dashboard based on the principle of least privilege, thus avoiding access of high-privileged users.

It is recommended that you create a least-privileged SQL user to access and sign in with TiDB Dashboard. This avoids access of high-privileged users and improves security.

[User document](#)

1.2.2.3 Performance

- **Optimize the timestamp processing flow of PD**

TiDB optimizes its timestamp processing flow and reduces the timestamp processing load of PD by enabling PD Follower Proxy and modifying the batch waiting time required when the PD client requests TSO in batches. This helps improve the overall scalability of the system.

- Support enabling or disabling PD Follower Proxy through the system variable `tidb_enable_tso_follower_proxy`. Suppose that the TSO requests load of PD is too high. In this case, enabling PD follower proxy can batch forward the TSO requests collected during the request cycle on followers to the leader nodes. This solution can effectively reduce the number of direct interactions between clients and leaders, reduce the pressure of the load on leaders, and improve the overall performance of TiDB.

Note:

When the number of clients is small and the PD leader CPU load is not full, it is NOT recommended to enable PD Follower Proxy.

- Support using the system variable `tidb_tso_client_batch_max_wait_time` to set the maximum waiting time needed for the PD client to batch request TSO. The unit of this time is milliseconds. In case that PD has a high TSO requests load, you can reduce the load and improve the throughput by increasing the waiting time to get a larger batch size.

Note:

When the TSO request load is not high, it is NOT recommended to modify this variable value.

[User document, #3149](#)

1.2.2.4 Stability

- **Support Online Unsafe Recovery after some stores are permanently damaged (experimental feature)**

Support the `unsafe remove-failed-stores` command that performs online data unsafe recovery. Suppose that the majority of data replicas encounter issues like permanent damage (such as disk damage), and these issues cause the data ranges in an application to be unreadable or unwritable. In this case, you can use the Online Unsafe Recovery feature implemented in PD to recover the data, so that the data is readable or writable again.

It is recommended to perform the feature-related operations with the support of the TiDB team.

[User document, #10483](#)

1.2.2.5 Data migration

- **DM replication performance enhanced**

Supports the following features to ensure lower-latency data replication from MySQL to TiDB:

- Compact multiple updates on a single row into one statement
- Merge batch updates of multiple rows into one statement

- **Add DM OpenAPI to better maintain DM clusters (experimental feature)**

DM provides the OpenAPI feature for querying and operating the DM cluster. It is similar to the feature of `dmctl tools`.

Currently, DM OpenAPI is an experimental feature and disabled by default. It is not recommended to use it in a production environment.

[User document](#)

- **TiDB Lightning Parallel Import**

TiDB Lightning provides parallel import capability to extend the original feature. It allows you to deploy multiple Lightning instances at the same time to import single tables or multiple tables to downstream TiDB in parallel. Without changing the way customers use it, it greatly improves the data migration ability, allowing you to migrate data in a more real-time way to further process, integrate and analyze them. It improves the efficiency of enterprise data management.

In our test, using 10 TiDB Lightning instances, a total of 20 TiB MySQL data can be imported to TiDB within 8 hours. The performance of multiple table import is also improved. A single TiDB Lightning instance can support importing at 250 GiB/h, and the overall migration is 8 times faster than the original performance.

[User document](#)

- **TiDB Lightning Prechecks**

TiDB Lightning provides the ability to check the configuration before running a migration task. It is enabled by default. This feature automatically performs some routine checks for disk space and execution configuration. The main purpose is to ensure that the whole subsequent import process goes smoothly.

[User document](#)

- **TiDB Lightning supports importing files of GBK character set**

You can specify the character set of the source data file. TiDB Lightning will convert the source file from the specified character set to UTF-8 encoding during the import process.

[User document](#)

- **Sync-diff-inspector improvement**

- Improve the comparison speed from 375 MB/s to 700 MB/s
- Reduce the memory consumption of TiDB nodes by nearly half during comparison
- Optimize the user interface and display the progress bar during comparison

[User document](#)

1.2.2.6 Diagnostic efficiency

- **Save and restore the on-site information of a cluster**

When you locate and troubleshoot the issues of a TiDB cluster, you often need to provide information on the system and the query plan. To help you get the information and troubleshoot cluster issues in a more convenient and efficient way, the PLAN REPLAY command is introduced in TiDB v5.3.0. This command enables you to easily save and restore the on-site information of a cluster, improves the efficiency of troubleshooting, and helps you more easily archive the issues for management.

The features of PLAN REPLAYER are as follows:

- Exports the information of a TiDB cluster at an on-site troubleshooting to a ZIP-formatted file for storage.
- Imports into a cluster the ZIP-formatted file exported from another TiDB cluster. This file contains the information of the latter TiDB cluster at an on-site troubleshooting.

[User document](#), #26325

1.2.2.7 TiDB data share subscription

- **TiCDC Eventually Consistent Replication**

TiCDC provides the eventually consistent replication capability in disaster scenarios. When a disaster occurs in the primary TiDB cluster and the service cannot be resumed in a short period of time, TiCDC needs to provide the ability to ensure the consistency of data in the secondary cluster. Meanwhile, TiCDC needs to allow the business to quickly switch the traffic to the secondary cluster to avoid the database being unavailable for a long time and affecting the business.

This feature supports TiCDC to replicate incremental data from a TiDB cluster to the secondary relational database TiDB/Aurora/MySQL/MariaDB. In case the primary cluster crashes, TiCDC can recover the secondary cluster to a certain snapshot in the primary cluster within 5 minutes, given the condition that before disaster the replication status of TiCDC is normal and replication lag is small. It allows data loss of less than 30 minutes, that is, RTO <= 5min, and RPO <= 30min.

[User document](#)

- **TiCDC supports the HTTP protocol OpenAPI for managing TiCDC tasks**

Since TiDB v5.3.0, TiCDC OpenAPI becomes an General Availability (GA) feature. You can query and operate TiCDC clusters using OpenAPI in the production environment.

1.2.2.8 Deployment and maintenance

- **Continuous Profiling (experimental feature)**

TiDB Dashboard supports the Continuous Profiling feature, which stores instance performance analysis results automatically in real time when TiDB clusters are running. You can check the performance analysis result in a flame graph, which is more observable and shortens troubleshooting time.

This feature is disabled by default and needs to be enabled on the **Continuous Profile** page of TiDB Dashboard.

This feature is only available for clusters upgraded or installed using TiUP v1.7.0 or above.

[User document](#)

1.2.3 Telemetry

TiDB adds the information to the telemetry report about whether or not the TEMPORARY TABLE feature is used. This does not include table names or table data.

To learn more about telemetry and how to disable this behavior, refer to [Telemetry](#).

1.2.4 Removed feature

Starting from TiCDC v5.3.0, the cyclic replication feature between TiDB clusters (an experimental feature in v5.0.0) has been removed. If you have already used this feature to replicate data before upgrading TiCDC, the related data is not affected after the upgrade.

1.2.5 Improvements

- TiDB

- Show the affected SQL statements in the debug log when the coprocessor encounters a lock, which is helpful in diagnosing problems [#27718](#)
- Support showing the size of the backup and restore data when backing up and restoring data in the SQL logical layer [#27247](#)
- Improve the default collection logic of ANALYZE when `tidb_analyze_version` is 2, which accelerates collection and reduces resource overhead
- Introduce the `ANALYZE TABLE table_name COLUMNS col_1, col_2, ... , → col_n` syntax. The syntax allows collecting statistics only on a portion of the columns in wide tables, which improves the speed of statistics collection

- TiKV

- Enhance disk space protection to improve storage stability

To solve the issue that TiKV might panic in case of a disk fully-written error, TiKV introduces a two-level threshold defense mechanism to protect the disk remaining space from being exhausted by excess traffic. Additionally, the mechanism provides the ability to reclaim space when the threshold is triggered. When the remaining space threshold is triggered, some write operations will fail and TiKV will return a disk full error as well as a list of disk full nodes. In this case, to recover the space and restore the service, you can execute Drop/Truncate → Table or scale out the nodes.

- Simplify the algorithm of L0 flow control [#10879](#)
- Improve the error log report in the raft client module [#10944](#)
- Improve logging threads to avoid them becoming a performance bottleneck [#10841](#)
- Add more statistics types of write queries [#10507](#)

- PD
 - Add more types of write queries to QPS dimensions in the hotspot scheduler [#3869](#)
 - Support dynamically adjusting the retry limit of the balance Region scheduler to improve the performance of the scheduler [#3744](#)
 - Update TiDB Dashboard to v2021.10.08.1 [#4070](#)
 - Support that the evict leader scheduler can schedule Regions with unhealthy peers [#4093](#)
 - Speed up the exit process of schedulers [#4146](#)
- TiFlash
 - Improve the execution efficiency of the TableScan operator greatly
 - Improve the execution efficiency of the Exchange operator
 - Reduce write amplification and memory usage during GC of the storage engine (experimental feature)
 - Improve the stability and availability of TiFlash when TiFlash restarts, which reduces possible query failures following the restart
 - Support pushing down multiple new String and Time functions to the MPP engine
 - * String functions: LIKE pattern, FORMAT(), LOWER(), LTRIM(), RTRIM(), SUBSTRING_INDEX(), TRIM(), UCASE(), UPPER()
 - * Mathematical functions: ROUND (decimal, int)
 - * Date and time functions: HOUR(), MICROSECOND(), MINUTE(), SECOND(), SYSDATE()
 - * Type conversion function: CAST(time, real)
 - * Aggregation functions: GROUP_CONCAT(), SUM(enum)
 - Support 512-bit SIMD
 - Enhance the cleanup algorithm for outdated data to reduce disk usage and read files more efficiently
 - Fix the issue that dashboard does not display memory or CPU information in some non-Linux systems
 - Unify the naming style of TiFlash log files (keep the naming style consistent with that of TiKV) and support dynamic modification of logger.count and logger.size
 - Improve the data validation capability of column-based files (checksums, experimental feature)
- Tools
 - TiCDC
 - * Reduce the default value of the Kafka sink configuration item MaxMessageBytes ↴ from 64 MB to 1 MB to fix the issue that large messages are rejected by the Kafka Broker [#3104](#)

- * Reduce memory usage in the replication pipeline [#2553](#) [#3037](#) [#2726](#)
- * Optimize monitoring items and alert rules to improve observability of synchronous links, memory GC, and stock data scanning processes [#2735](#) [#1606](#) [#3000](#) [#2985](#) [#2156](#)
- * When the sync task status is normal, no more historical error messages are displayed to avoid misleading users [#2242](#)

1.2.6 Bug Fixes

- TiDB
 - Fix an error that occurs during execution caused by the wrong execution plan. The wrong execution plan is caused by the shallow copy of schema columns when pushing down the aggregation operators on partitioned tables [#27797](#) [#26554](#)
 - Fix the issue that plan-cache cannot detect changes of unsigned flags [#28254](#)
 - Fix the wrong partition pruning when the partition function is out of range [#28233](#)
 - Fix the issue that planner might cache invalid plans for `join` in some cases [#28087](#)
 - Fix wrong index hash join when hash column type is enum [#27893](#)
 - Fix a batch client bug that recycling idle connection might block sending requests in some rare cases [#27688](#)
 - Fix the TiDB Lightning panic issue when it fails to perform checksum on a target cluster [#27686](#)
 - Fix wrong results of the `date_add` and `date_sub` functions in some cases [#27232](#)
 - Fix wrong results of the `hour` function in vectorized expression [#28643](#)
 - Fix the authenticating issue when connecting to MySQL 5.1 or an older client version [#27855](#)
 - Fix the issue that auto analyze might be triggered out of the specified time when a new index is added [#28698](#)
 - Fix a bug that setting any session variable invalidates `tidb_snapshot` [#28683](#)
 - Fix a bug that BR is not working for clusters with many missing-peer Regions [#27534](#)
 - Fix the unexpected error like `tidb_cast to Int32 is not supported` when the unsupported `cast` is pushed down to TiFlash [#23907](#)
 - Fix the issue that DECIMAL overflow is missing in the `%s` value is out of ↪ `range in '%s'` error message [#27964](#)
 - Fix a bug that the availability detection of MPP node does not work in some corner cases [#3118](#)
 - Fix the DATA RACE issue when assigning MPP task ID [#27952](#)
 - Fix the INDEX OUT OF RANGE error for a MPP query after deleting an empty dual table [#28250](#)
 - Fix the issue of false positive error log `invalid cop task execution summaries ↪ length` for MPP queries [#1791](#)
 - Fix the issue of error log `cannot found column in Schema column` for MPP queries [#28149](#)

- Fix the issue that TiDB might panic when TiFlash is shutting down [#28096](#)
- Remove the support for insecure 3DES (Triple Data Encryption Algorithm) based TLS cipher suites [#27859](#)
- Fix the issue that Lightning connects to offline TiKV nodes during pre-check and causes import failures [#27826](#)
- Fix the issue that pre-check cost too much time when importing many files to tables [#27605](#)
- Fix the issue that rewriting expressions makes `between` infer wrong collation [#27146](#)
- Fix the issue that `group_concat` function did not consider the collation [#27429](#)
- Fix the result wrong that occurs when the argument of the `extract` function is a negative duration [#27236](#)
- Fix the issue that creating partition fails if `NO_UNSIGNED_SUBTRACTION` is set [#26765](#)
- Avoid expressions with side effects in column pruning and aggregation pushdown [#27106](#)
- Remove useless gRPC logs [#24190](#)
- Limit the valid decimal length to fix precision-related issues [#3091](#)
- Fix the issue of a wrong way to check for overflow in `plus` expression [#26977](#)
- Fix the issue of `data too long` error when dumping statistics from the table with `new collation` data [#27024](#)
- Fix the issue that the retried transactions' statements are not included in `TIDB_TRX` [#28670](#)
- Fix the wrong default value of the `plugin_dir` configuration [#28084](#)
- TiKV
 - Fix the issue of unavailable TiKV caused by Raftstore deadlock when migrating Regions. The workaround is to disable the scheduling and restart the unavailable TiKV [#10909](#)
 - Fix the issue that CDC adds scan retries frequently due to the Congest error [#11082](#)
 - Fix the issue that the Raft connection is broken when the channel is full [#11047](#)
 - Fix the issue that batch messages are too large in Raft client implementation [#9714](#)
 - Fix the issue that some coroutines leak in `resolved_ts` [#10965](#)
 - Fix a panic issue that occurs to the coprocessor when the size of response exceeds 4 GiB [#9012](#)
 - Fix the issue that snapshot Garbage Collection (GC) misses GC snapshot files when snapshot files cannot be garbage collected [#10813](#)
 - Fix a panic issue caused by timeout when processing Coprocessor requests [#10852](#)
 - Fix a memory leak caused by monitoring data of statistics threads [#11195](#)
 - Fix a panic issue caused by getting the cgroup information from some platforms [#10980](#)

- Fix the issue of poor scan performance because MVCC Deletion versions are not dropped by compaction filter GC [#11248](#)
- PD
 - Fix the issue that PD incorrectly delete the peers with data and in pending status because the number of peers exceeds the number of configured peers [#4045](#)
 - Fix the issue that PD does not fix down peers in time [#4077](#)
 - Fix the issue that the scatter range scheduler cannot schedule empty Regions [#4118](#)
 - Fix the issue that the key manager cost too much CPU [#4071](#)
 - Fix the data race issue that might occur when setting configurations of hot Region scheduler [#4159](#)
 - Fix slow leader election caused by stucked Region syncer [#3936](#)
- TiFlash
 - Fix the issue of inaccurate TiFlash Store Size statistics
 - Fix the issue that TiFlash fails to start up on some platforms due to the absence of library `ns1`
 - Block the infinite wait of `wait index` when writing pressure is heavy (a default timeout of 5 minutes is added), which prevents TiFlash from waiting too long for data replication to provide services
 - Fix the slow and no result issues of the log search when the log volume is large
 - Fix the issue that only the most recent logs can be searched when searching old historical logs
 - Fix the possible wrong result when a new collation is enabled
 - Fix the possible parsing errors when an SQL statement contains extremely long nested expressions
 - Fix the possible `Block schema mismatch` error of the Exchange operator
 - Fix the possible `Can't compare` error when comparing Decimal types
 - Fix the `3rd arguments of function substringUTF8 must be constants` error of the `left/substring` function
- Tools
 - TiCDC
 - * Fix the issue that TiCDC replication task might terminate when the upstream TiDB instance unexpectedly exits [#3061](#)
 - * Fix the issue that TiCDC process might panic when TiKV sends duplicate requests to the same Region [#2386](#)
 - * Fix unnecessary CPU consumption when verifying downstream TiDB/MySQL availability [#3073](#)
 - * Fix the issue that the volume of Kafka messages generated by TiCDC is not constrained by `max-message-size` [#2962](#)

- * Fix the issue that TiCDC sync task might pause when an error occurs during writing a Kafka message [#2978](#)
- * Fix the issue that some partitioned tables without valid indexes might be ignored when `force-replicate` is enabled [#2834](#)
- * Fix the issue that scanning stock data might fail due to TiKV performing GC when scanning stock data takes too long [#2470](#)
- * Fix a possible panic issue when encoding some types of columns into Open Protocol format [#2758](#)
- * Fix a possible panic issue when encoding some types of columns into Avro format [#2648](#)
- TiDB Binlog
 - * Fix the issue that when most tables are filtered out, checkpoint can not be updated under some special load [#1075](#)

1.3 TiDB Features

This document lists the features supported in each TiDB version. Note that supports for experimental features might change before the final release.

1.3.1 Data types, functions, and operators

Data types, functions, and operators	5.3	5.2	5.1	5.0	
Numeric types	Y	Y	Y	Y	
Date and time types	Y	Y	Y	Y	
String types	Y	Y	Y	Y	
JSON type	Experimental	Experimental	Experimental	Experimental	E
Control flow functions	Y	Y	Y	Y	
String functions	Y	Y	Y	Y	
Numeric functions and operators	Y	Y	Y	Y	
Date and time functions	Y	Y	Y	Y	
Bit functions and operators	Y	Y	Y	Y	
Cast functions and operators	Y	Y	Y	Y	
Encryption and compression functions	Y	Y	Y	Y	
Information functions	Y	Y	Y	Y	
JSON functions	Experimental	Experimental	Experimental	Experimental	E
Aggregation functions	Y	Y	Y	Y	
Window functions	Y	Y	Y	Y	
Miscellaneous functions	Y	Y	Y	Y	
Operators	Y	Y	Y	Y	
Character sets and collations ¹	Y	Y	Y	Y	

¹TiDB incorrectly treats latin1 as a subset of utf8. See [TiDB #18955](#) for more details.

1.3.2 Indexing and constraints

Indexing and constraints	5.3	5.2	5.1	5.0
Expression indexes	Experimental	Experimental	Experimental	Experimental
Columnar storage (TiFlash)	Y	Y	Y	Y
RocksDB engine	Y	Y	Y	Y
Titan plugin	Y	Y	Y	Y
Invisible indexes	Y	Y	Y	Y
Composite PRIMARY KEY	Y	Y	Y	Y
Unique indexes	Y	Y	Y	Y
Clustered index on integer PRIMARY KEY	Y	Y	Y	Y
Clustered index on composite or non-integer key	Y	Y	Y	Y

1.3.3 SQL statements

SQL statements ²	5.3	5.2	5.1	5.0	4.0
Basic SELECT, INSERT, UPDATE, DELETE, REPLACE	Y	Y	Y	Y	Y
INSERT ON DUPLICATE KEY UPDATE	Y	Y	Y	Y	Y
LOAD DATA INFILE	Y	Y	Y	Y	Y
SELECT INTO OUTFILE	Y	Y	Y	Y	Y
INNER JOIN, LEFT\ RIGHT [OUTER] JOIN	Y	Y	Y	Y	Y
UNION, UNION ALL	Y	Y	Y	Y	Y
EXCEPT and INTERSECT operators	Y	Y	Y	Y	N
GROUP BY, ORDER BY	Y	Y	Y	Y	Y
Window Functions	Y	Y	Y	Y	Y
Common Table Expressions (CTE)	Y	Y	Y	N	N
START TRANSACTION, COMMIT, ROLLBACK	Y	Y	Y	Y	Y
EXPLAIN	Y	Y	Y	Y	Y
EXPLAIN ANALYZE	Y	Y	Y	Y	Y
User-defined variables	Experimental	Experimental	Experimental	Experimental	Experimental

1.3.4 Advanced SQL features

Advanced SQL features	5.3	5.2	5.1	5.0	Experiments
Prepared statement cache	Y	Experimental	Experimental	Experimental	Experimental
SQL plan management (SPM)	Y	Y	Y	Y	Y
Coprocessor cache	Y	Y	Y	Y	Y
Stale Read	Y	Y	Y	N	Experiments

Advanced SQL features	5.3	5.2	5.1	5.0
Follower reads	Y	Y	Y	Y
Read historical data (tidb_snapshot)	Y	Y	Y	Y
Optimizer hints	Y	Y	Y	Y
MPP Execution Engine	Y	Y	Y	Y
Index Merge Join	Experimental	Experimental	Experimental	Experimental
Placement Rules in SQL	Experimental	N	N	N

1.3.5 Data definition language (DDL)

Data definition language (DDL)	5.3	5.2	5.1	5.0	4.0
Basic CREATE, DROP, ALTER, RENAME, TRUNCATE	Y	Y	Y	Y	Y
Generated columns	Experimental	Experimental	Experimental	Experimental	Experimental
Views	Y	Y	Y	Y	Y
Sequences	Y	Y	Y	Y	Y
Auto increment	Y	Y	Y	Y	Y
Auto random	Y	Y	Y	Y	Y
DDL algorithm assertions	Y	Y	Y	Y	Y
Multi schema change: add column(s)	Experimental	Experimental	Experimental	Experimental	Experimental
Change column type	Y	Y	Y	N	N
Temporary tables	Y	N	N	N	N

1.3.6 Transactions

Transactions	5.3	5.2	5.1	5.0	4.0
Async commit	Y	Y	Y	Y	N
1PC	Y	Y	Y	Y	N
Large transactions (10GB)	Y	Y	Y	Y	Y
Pessimistic transactions	Y	Y	Y	Y	Y
Optimistic transactions	Y	Y	Y	Y	Y
Repeatable-read isolation (snapshot isolation)	Y	Y	Y	Y	Y
Read-committed isolation	Y	Y	Y	Y	Y

1.3.7 Partitioning

Partitioning	5.3	5.2	5.1	5.0	4.0
Range partitioning	Y	Y	Y	Y	Y
Hash partitioning	Y	Y	Y	Y	Y
List partitioning	Experimental	Experimental	Experimental	Experimental	N

Partitioning	5.3	5.2	5.1	5.0	4.0
List COLUMNS partitioning	Experimental	Experimental	Experimental	Experimental	N
EXCHANGE PARTITION	Experimental	Experimental	Experimental	Experimental	N
Dynamic Pruning	Experimental	Experimental	Experimental	N	N

1.3.8 Statistics

Statistics	5.3	5.2	5.1	5.0	4.0
CMSketch	Deprecated	Deprecated	Deprecated	Deprecated	Y
Histograms	Y	Y	Y	Y	Y
Extended statistics (multiple columns)	Experimental	Experimental	Experimental	Experimental	N
Statistics Feedback	Experimental	Experimental	Experimental	Experimental	Experimental
Fast Analyze	Experimental	Experimental	Experimental	Experimental	Experimental

1.3.9 Security

Security	5.3	5.2	5.1	5.0	4.0
Transparent layer security (TLS)	Y	Y	Y	Y	Y
Encryption at rest (TDE)	Y	Y	Y	Y	Y
Role-based authentication (RBAC)	Y	Y	Y	Y	Y
Certificate-based authentication	Y	Y	Y	Y	Y
caching_sha2_password authentication	Y	Y	N	N	N
MySQL compatible GRANT system	Y	Y	Y	Y	Y
Dynamic Privileges	Y	Y	Y	N	N
Security Enhanced Mode	Y	Y	Y	N	N
Redacted Log Files	Y	Y	Y	Y	N

1.3.10 Data import and export

Data import and export	5.3	5.2	5.1	5.0	4.0
Fast Importer (TiDB Lightning)	Y	Y	Y	Y	Y
mydumper logical dumper	Deprecated	Deprecated	Deprecated	Deprecated	Deprecated
Dumpling logical dumper	Y	Y	Y	Y	Y
Transactional LOAD DATA	Y	Y	Y	Y	N ³
Database migration toolkit (DM)	Y	Y	Y	Y	Y
TiDB Binlog	Y	Y	Y	Y	Y
Change data capture (CDC)	Y	Y	Y	Y	Y

³For TiDB v4.0, the LOAD DATA transaction does not guarantee atomicity.

1.3.11 Management, observability, and tools

Management, observability, and tools	5.3	5.2	5.1	5.0
TiDB Dashboard	Y	Y	Y	Y
SQL diagnostics	Experimental	Experimental	Experimental	Experimental
Information schema	Y	Y	Y	Y
Metrics schema	Y	Y	Y	Y
Statements summary tables	Y	Y	Y	Y
Slow query log	Y	Y	Y	Y
TiUP deployment	Y	Y	Y	Y
Ansible deployment	N	N	N	N
Kubernetes operator	Y	Y	Y	Y
Built-in physical backup	Y	Y	Y	Y
Top SQL	Y	Y	N	N
Global Kill	Experimental	Experimental	Experimental	Experimental
Lock View	Y	Y	Experimental	Experimental
SHOW CONFIG	Experimental	Experimental	Experimental	Experimental
SET CONFIG	Experimental	Experimental	Experimental	Experimental
Continuous Profiling	Experimental	N	N	N

1.4 TiDB Experimental Features

This document introduces the experimental features of TiDB in different versions. It is **NOT** recommended to use these features in the production environment.

1.4.1 Stability

- TiFlash limits the use of I/O resources by compressing or sorting data, mitigating the contention for I/O resources between background tasks and front-end data reading and writing (Introduced in v5.0)
- Improve the stability of the optimizer's choice of indexes (Introduced in v5.0)
 - Extend the statistics feature by collecting the multi-column order dependency information.
 - Refactor the statistics module, including deleting the TopN value from CMSKetch and the histogram, and adding NDV information for histogram buckets of each table index.

1.4.2 Scheduling

- Cascading Placement Rules feature. It is a replica rule system that guides PD to generate corresponding schedules for different types of data. By combining different scheduling rules, you can finely control the attributes of any continuous data range,

such as the number of replicas, the storage location, the host type, whether to participate in Raft election, and whether to act as the Raft leader. See [Cascading Placement Rules](#) for details. (Introduced in v4.0)

- Elastic scheduling feature. It enables the TiDB cluster to dynamically scale out and in on Kubernetes based on real-time workloads, which effectively reduces the stress during your application's peak hours and saves overheads. See [Enable TidbCluster Auto-scaling](#) for details. (Introduced in v4.0)

1.4.3 SQL

- [Use SQL interface to set placement rules for data] (/placement-rules-in-sql.md) (Introduced in v5.3)
- List Partition (Introduced in v5.0)
- List COLUMNS Partition (Introduced in v5.0)
- [Dynamic Pruning Mode for Partitioned Tables](#). (Introduced in v5.1)
- The expression index feature. The expression index is also called the function-based index. When you create an index, the index fields do not have to be a specific column but can be an expression calculated from one or more columns. This feature is useful for quickly accessing the calculation-based tables. See [Expression index](#) for details. (Introduced in v4.0)
- [Generated Columns](#) (Introduced in v2.1)
- [User-Defined Variables](#) (Introduced in v2.1)
- [JSON data type](#) and [JSON functions](#) (Introduced in v2.1)
- [View](#) (Introduced in v2.1)

1.4.4 Configuration management

- Persistently store configuration parameters in PD, and support dynamically modifying configuration items. (Introduced in v4.0)
- [SHOW CONFIG](#) (Introduced in v4.0)

1.4.5 Data sharing and subscription

- [Integrate TiCDC with Kafka Connect \(Confluent Platform\)](#) (Introduced in v5.0)

1.4.6 Storage

- [Disable Titan](#) (Introduced in v4.0)
- [Titan Level Merge](#) (Introduced in v4.0)
- TiFlash supports distributing the new data of the storage engine on multiple hard drives to share the I/O pressure. (Introduced in v4.0)

1.4.7 Data migration

- DM OpenAPI (Introduced in v5.3)

1.4.8 Backup and restoration

- Back up Raw KV (Introduced in v3.1)

1.4.9 Garbage collection

- Green GC (Introduced in v5.0)

1.4.10 Diagnostics

- SQL diagnostics (Introduced in v4.0)
- Cluster diagnostics (Introduced in v4.0)
- Continuous profiling (Introduced in v5.3)
- Online Unsafe Recovery (Introduced in v5.3)

1.5 Benchmarks

1.5.1 TiDB Sysbench Performance Test Report – v5.3.0 vs. v5.2.2

1.5.1.1 Test overview

This test aims at comparing the Sysbench performance of TiDB v5.3.0 and TiDB v5.2.2 in the Online Transactional Processing (OLTP) scenario. The results show that the performance of v5.3.0 is nearly the same as that of v5.2.2.

1.5.1.2 Test environment (AWS EC2)

1.5.1.2.1 Hardware configuration

Service type	EC2 type	Instance count
PD	m5.xlarge	3
TiKV	i3.4xlarge	3
TiDB	c5.4xlarge	3
Sysbench	c5.9xlarge	1

1.5.1.2.2 Software version

Service type	Software version
PD	v5.2.2 and v5.3.0
TiDB	v5.2.2 and v5.3.0
TiKV	v5.2.2 and v5.3.0
Sysbench	1.1.0-ead2689

1.5.1.2.3 Parameter configuration

TiDB v5.3.0 and TiDB v5.2.2 use the same configuration.

TiDB parameter configuration

```
log.level: "error"
performance.max-procs: 20
prepared-plan-cache.enabled: true
tikv-client.max-batch-wait-time: 2000000
```

TiKV parameter configuration

```
storage.scheduler-worker-pool-size: 5
raftstore.store-pool-size: 3
raftstore.apply-pool-size: 3
rocksdb.max-background-jobs: 8
raftdb.max-background-jobs: 4
raftdb.allow-concurrent-memtable-write: true
server.grpc-concurrency: 6
readpool.unified.min-thread-count: 5
readpool.unified.max-thread-count: 20
readpool.storage.normal-concurrency: 10
pessimistic-txn.pipelined: true
```

TiDB global variable configuration

```
set global tidb_hashagg_final_concurrency=1;
set global tidb_hashagg_partial_concurrency=1;
set global tidb_enable_async_commit = 1;
set global tidb_enable_1pc = 1;
set global tidb_guarantee_linearizability = 0;
set global tidb_enable_clustered_index = 1;
```

HAProxy configuration - haproxy.cfg

For more details about how to use HAProxy on TiDB, see [Best Practices for Using HAProxy in TiDB](#).

global	# Global configuration.
--------	-------------------------

```

chroot      /var/lib/haproxy          # Changes the current directory and
    ↪ sets superuser privileges for the startup process to improve
    ↪ security.
pidfile     /var/run/haproxy.pid    # Writes the PIDs of HAProxy processes
    ↪ into this file.
maxconn     4000                   # The maximum number of concurrent
    ↪ connections for a single HAProxy process.
user        haproxy                # Same with the UID parameter.
group       haproxy                # Same with the GID parameter. A
    ↪ dedicated user group is recommended.
nbproc      64                     # The number of processes created when
    ↪ going daemon. When starting multiple processes to forward requests
    ↪ , ensure that the value is large enough so that HAProxy does not
    ↪ block processes.
daemon      # Makes the process fork into
    ↪ background. It is equivalent to the command line "-D" argument. It
    ↪ can be disabled by the command line "-db" argument.

defaults          # Default configuration.
log global        # Inherits the settings of the global
    ↪ configuration.
retries 2          # The maximum number of retries to
    ↪ connect to an upstream server. If the number of connection attempts
    ↪ exceeds the value, the backend server is considered unavailable.
timeout connect 2s # The maximum time to wait for a
    ↪ connection attempt to a backend server to succeed. It should be set
    ↪ to a shorter time if the server is located on the same LAN as
    ↪ HAProxy.
timeout client 30000s      # The maximum inactivity time on the
    ↪ client side.
timeout server 30000s      # The maximum inactivity time on the
    ↪ server side.

listen tidb-cluster        # Database load balancing.
bind 0.0.0.0:3390          # The Floating IP address and
    ↪ listening port.
mode tcp                 # HAProxy uses layer 4, the transport
    ↪ layer.
balance roundrobin        # The server with the fewest
    ↪ connections receives the connection. "leastconn" is recommended
    ↪ where long sessions are expected, such as LDAP, SQL and TSE, rather
    ↪ than protocols using short sessions, such as HTTP. The algorithm
    ↪ is dynamic, which means that server weights might be adjusted on
    ↪ the fly for slow starts for instance.
server tidb-1 10.9.18.229:4000 check inter 2000 rise 2 fall 3 # Detects

```

```

→ port 4000 at a frequency of once every 2000 milliseconds. If it is
→ detected as successful twice, the server is considered available;
→ if it is detected as failed three times, the server is considered
→ unavailable.
server tidb-2 10.9.39.208:4000 check inter 2000 rise 2 fall 3
server tidb-3 10.9.64.166:4000 check inter 2000 rise 2 fall 3

```

1.5.1.3 Test plan

1. Deploy TiDB v5.3.0 and v5.2.2 using TiUP.
2. Use Sysbench to import 16 tables, each table with 10 million rows of data.
3. Execute the `analyze table` statement on each table.
4. Back up the data used for restore before different concurrency tests, which ensures data consistency for each test.
5. Start the Sysbench client to perform the `point_select`, `read_write`, `update_index`, and `update_non_index` tests. Perform stress tests on TiDB via HAProxy. For each concurrency under each workload, the test takes 20 minutes.
6. After each type of test is completed, stop the cluster, overwrite the cluster with the backup data in step 4, and restart the cluster.

1.5.1.3.1 Prepare test data

Run the following command to prepare the test data:

```
sysbench oltp_common \
--threads=16 \
--rand-type=uniform \
--db-driver=mysql \
--mysql-db=sbtest \
--mysql-host=$aws_nlb_host \
--mysql-port=$aws_nlb_port \
--mysql-user=root \
--mysql-password=password \
prepare --tables=16 --table-size=10000000
```

1.5.1.3.2 Perform the test

Run the following command to perform the test:

```
sysbench $testname \
--threads=$threads \
--time=1200 \
--report-interval=1 \
--rand-type=uniform \
```

```
--db-driver=mysql \
--mysql-db=sbtest \
--mysql-host=$aws_nlb_host \
--mysql-port=$aws_nlb_port \
run --tables=16 --table-size=10000000
```

1.5.1.4 Test results

1.5.1.4.1 Point Select performance

Threads	v5.2.2	v5.3.0	v5.2.2 95% latency	v5.3.0 95% latency	TPS improvement (%)
	TPS	TPS	(ms)	(ms)	(%)
300	267673.17	267516.77	1.76	1.67	-0.06
600	369820.29	361672.56	2.91	2.97	-2.20
900	417143.31	416479.47	4.1	4.18	-0.16

Compared with v5.2.2, the Point Select performance of v5.3.0 is reduced slightly by 0.81%.

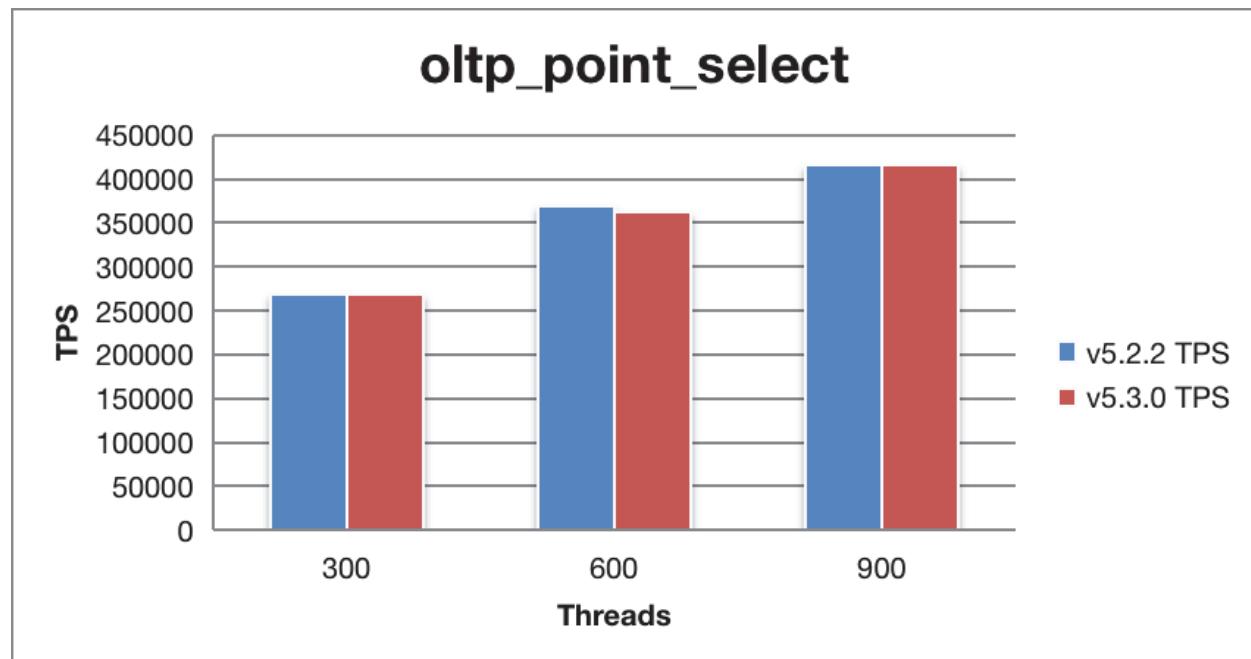


Figure 1: Point Select

1.5.1.4.2 Update Non-index performance

Threads	v5.2.2	v5.3.0	v5.2.2 95% latency	v5.3.0 95% latency	TPS improvement (%)
	TPS	TPS	(ms)	(ms)	
300	39715.31	40041.03	11.87	12.08	0.82
600	50239.42	51110.04	20.74	20.37	1.73
900	57073.97	57252.74	28.16	27.66	0.31

Compared with v5.2.2, the Update Non-index performance of v5.3.0 is improved slightly by 0.95%.

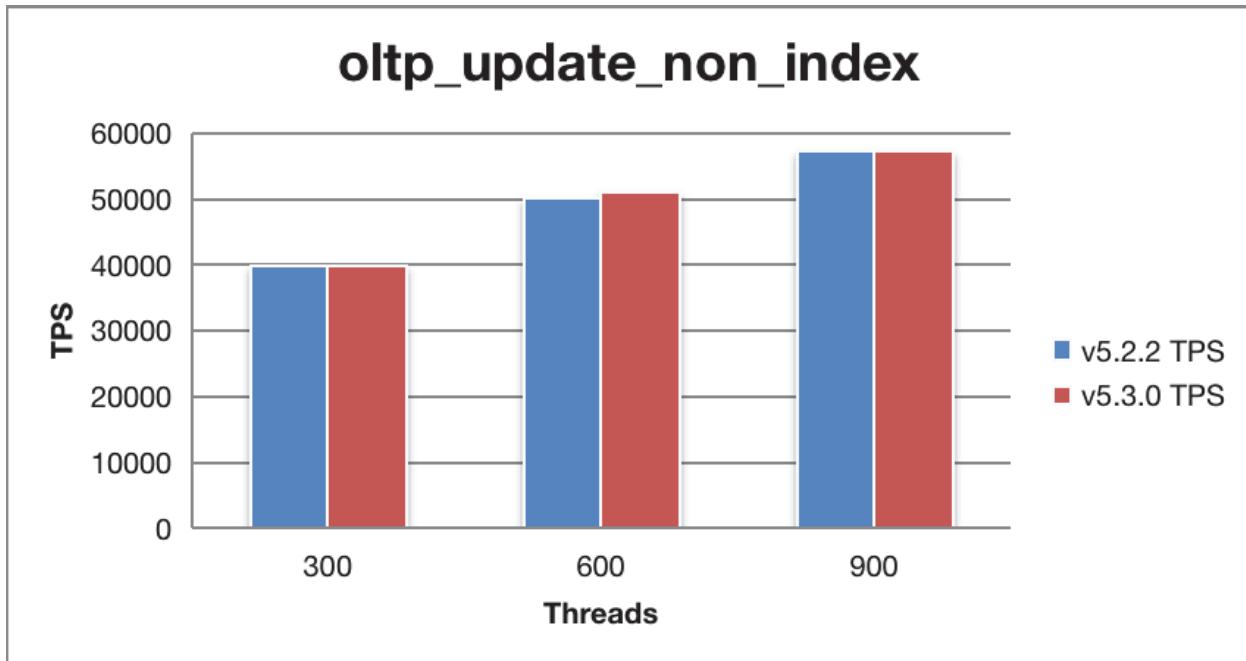


Figure 2: Update Non-index

1.5.1.4.3 Update Index performance

Threads	v5.2.2	v5.3.0	v5.2.2 95% latency	v5.3.0 95% latency	TPS improvement (%)
	TPS	TPS	(ms)	(ms)	
300	17634.03	17821.1	25.74	25.74	1.06
600	20998.59	21534.13	46.63	45.79	2.55
900	23420.75	23859.64	64.47	62.19	1.87

Compared with v5.2.2, the Update Index performance of v5.3.0 is improved slightly by 1.83%.

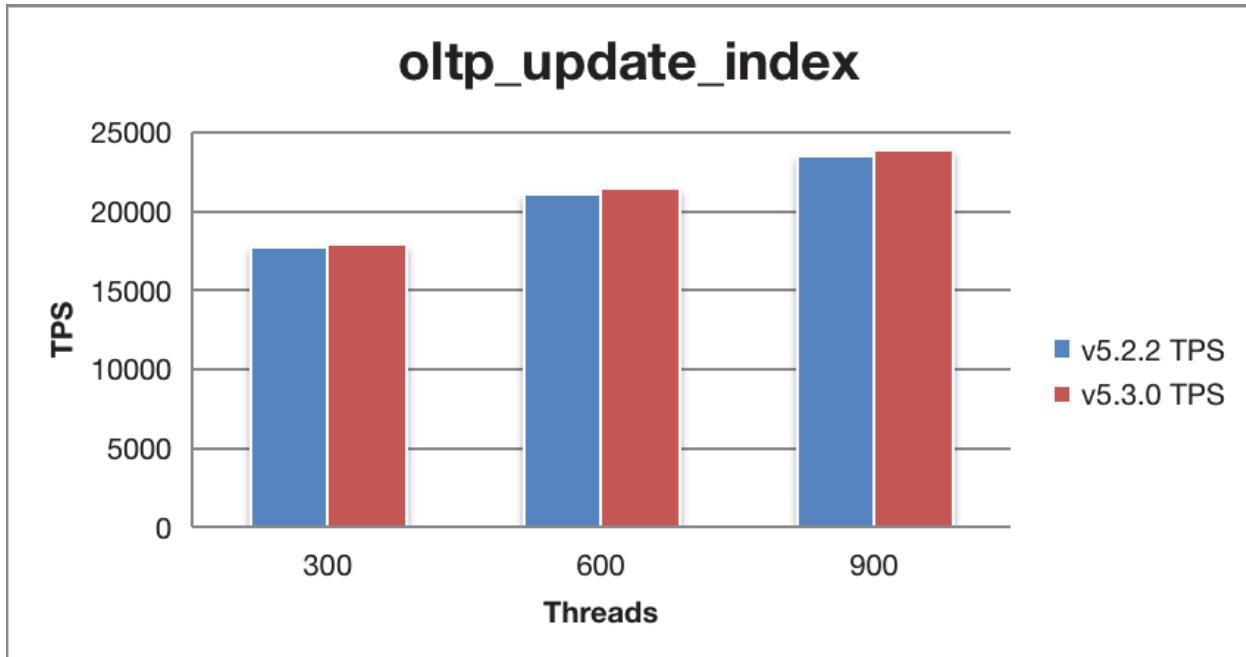


Figure 3: Update Index

1.5.1.4.4 Read Write performance

Threads	v5.2.2 TPS	v5.3.0 TPS	v5.2.2 95% latency (ms)	v5.3.0 95% latency (ms)	TPS im- provement (%)
300	3872.01	3848.63	106.75	106.75	-0.60
600	4514.17	4471.77	200.47	196.89	-0.94
900	4877.05	4861.45	287.38	282.25	-0.32

Compared with v5.2.2, the Read Write performance of v5.3.0 is reduced slightly by 0.62%.

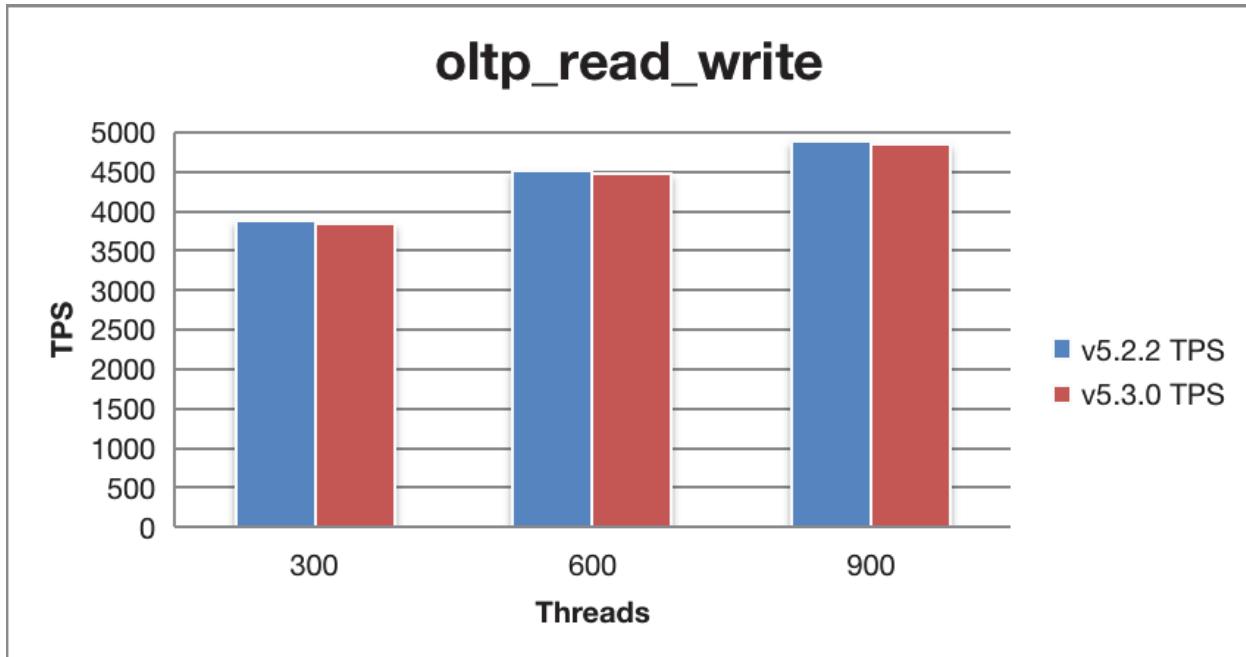


Figure 4: Read Write

1.5.2 TiDB TPC-C Performance Test Report – v5.3.0 vs. v5.2.2

1.5.2.1 Test overview

This test aims at comparing the TPC-C performance of TiDB v5.3.0 and TiDB v5.2.2 in the online transactional processing (OLTP) scenario. The result shows that compared with v5.2.2, the TPC-C performance of v5.3.0 is reduced by 2.99%.

1.5.2.2 Test environment (AWS EC2)

1.5.2.3 Hardware configuration

Service type	EC2 type	Instance count
PD	m5.xlarge	3
TiKV	i3.4xlarge	3
TiDB	c5.4xlarge	3
TPC-C	c5.9xlarge	1

1.5.2.3.1 Software version

Service type	Software version
PD	v5.2.2 and v5.3.0

Service type	Software version
TiDB	v5.2.2 and v5.3.0
TiKV	v5.2.2 and v5.3.0
TiUP	1.5.1

1.5.2.3.2 Parameter configuration

TiDB v5.3.0 and TiDB v5.2.2 use the same configuration.

TiDB parameter configuration

```
log.level: "error"
performance.max-procs: 20
prepared-plan-cache.enabled: true
tikv-client.max-batch-wait-time: 2000000
```

TiKV parameter configuration

```
pessimistic-txn.pipelined: true
raftdb.allow-concurrent-memtable-write: true
raftdb.max-background-jobs: 4
raftstore.apply-max-batch-size: 2048
raftstore.apply-pool-size: 3
raftstore.store-max-batch-size: 2048
raftstore.store-pool-size: 3
readpool.storage.normal-concurrency: 10
readpool.unified.max-thread-count: 20
readpool.unified.min-thread-count: 5
rocksdb.max-background-jobs: 8
server.grpc-concurrency: 6
storage.scheduler-worker-pool-size: 20
```

TiDB global variable configuration

```
set global tidb_hashagg_final_concurrency=1;
set global tidb_hashagg_partial_concurrency=1;
set global tidb_enable_async_commit = 1;
set global tidb_enable_1pc = 1;
set global tidb_guarantee_linearizability = 0;
set global tidb_enable_clustered_index = 1;
```

HAProxy configuration - haproxy.cfg

For more details about how to use HAProxy on TiDB, see [Best Practices for Using HAProxy in TiDB](#).

global	# Global configuration.
--------	-------------------------

```

chroot      /var/lib/haproxy          # Changes the current directory and
    ↪ sets superuser privileges for the startup process to improve
    ↪ security.
pidfile     /var/run/haproxy.pid    # Writes the PIDs of HAProxy processes
    ↪ into this file.
maxconn     4000                   # The maximum number of concurrent
    ↪ connections for a single HAProxy process.
user        haproxy                # Same with the UID parameter.
group       haproxy                # Same with the GID parameter. A
    ↪ dedicated user group is recommended.
nbproc      64                     # The number of processes created when
    ↪ going daemon. When starting multiple processes to forward requests
    ↪ , ensure that the value is large enough so that HAProxy does not
    ↪ block processes.
daemon      # Makes the process fork into
    ↪ background. It is equivalent to the command line "-D" argument. It
    ↪ can be disabled by the command line "-db" argument.

defaults
log global           # Default configuration.
    ↪ configuration.
retries 2            # The maximum number of retries to
    ↪ connect to an upstream server. If the number of connection attempts
    ↪ exceeds the value, the backend server is considered unavailable.
timeout connect 2s   # The maximum time to wait for a
    ↪ connection attempt to a backend server to succeed. It should be set
    ↪ to a shorter time if the server is located on the same LAN as
    ↪ HAProxy.
timeout client 30000s # The maximum inactivity time on the
    ↪ client side.
timeout server 30000s # The maximum inactivity time on the
    ↪ server side.

listen tidb-cluster          # Database load balancing.
    bind 0.0.0.0:3390          # The Floating IP address and
    ↪ listening port.
mode tcp                 # HAProxy uses layer 4, the transport
    ↪ layer.
balance roundrobin         # The server with the fewest
    ↪ connections receives the connection. "leastconn" is recommended
    ↪ where long sessions are expected, such as LDAP, SQL and TSE, rather
    ↪ than protocols using short sessions, such as HTTP. The algorithm
    ↪ is dynamic, which means that server weights might be adjusted on
    ↪ the fly for slow starts for instance.
server tidb-1 10.9.18.229:4000 check inter 2000 rise 2 fall 3 # Detects

```

```

→ port 4000 at a frequency of once every 2000 milliseconds. If it is
→ detected as successful twice, the server is considered available;
→ if it is detected as failed three times, the server is considered
→ unavailable.

server tidb-2 10.9.39.208:4000 check inter 2000 rise 2 fall 3
server tidb-3 10.9.64.166:4000 check inter 2000 rise 2 fall 3

```

1.5.2.4 Test plan

1. Deploy TiDB v5.3.0 and v5.2.2 using TiUP.
2. Create a database named tpcc: `create database tpcc;`.
3. Use BenchmarkSQL to import the TPC-C 5000 Warehouse data: `tiup bench tpcc --prepare --warehouse 5000 --db tpcc -H 127.0.0.1 -p 4000`.
4. Run the `tiup bench tpcc run -U root --db tpcc --host 127.0.0.1 --port 4000 --time 1800s --warehouses 5000 --threads {{thread}}` command to perform stress tests on TiDB via HAProxy. For each concurrency, the test takes 30 minutes.
5. Extract the tpmC data of New Order from the result.

1.5.2.5 Test result

Compared with v5.2.2, the TPC-C performance of v5.3.0 is **reduced slightly by 2.99%**.

Threads	v5.2.2 tpmC	v5.3.0 tpmC	tpmC improvement (%)
50	42228.8	41580	-1.54
100	49400	48248.2	-2.33
200	54436.6	52809.4	-2.99
400	57026.7	54117.1	-5.10

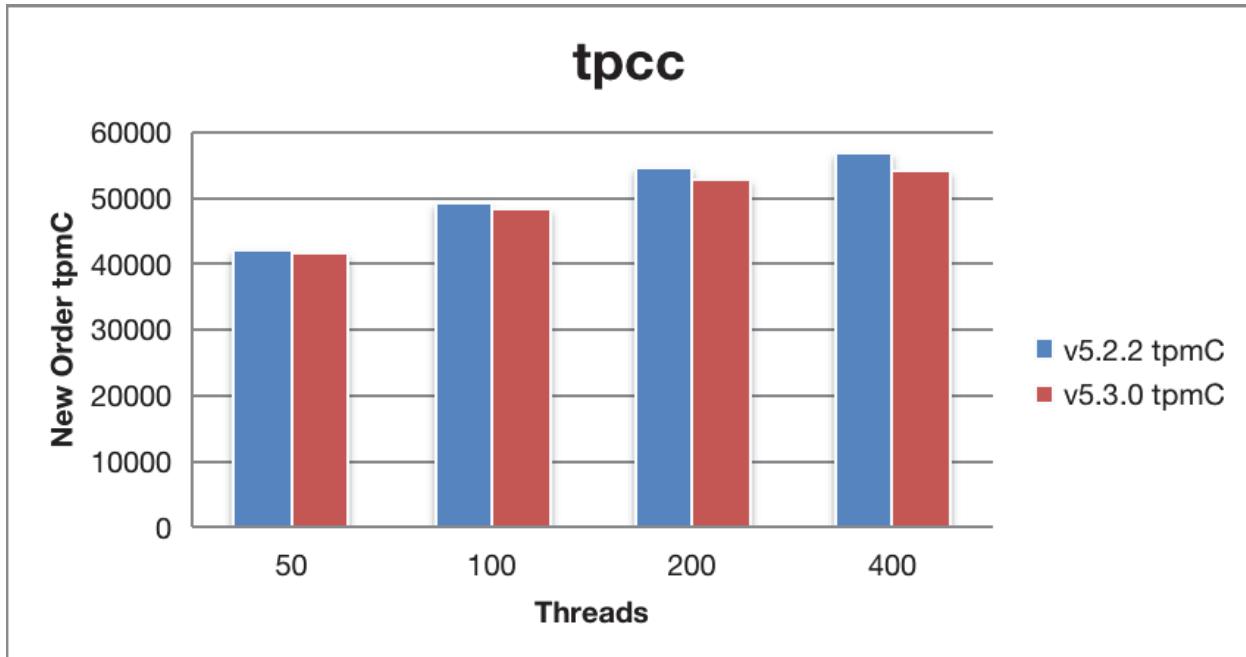


Figure 5: TPC-C

1.5.3 TiDB TPC-H 100GB Performance Test Report – TiDB v5.3 MPP mode vs. Greenplum 6.15.0 and Apache Spark 3.1.1

1.5.3.1 Test overview

This test aims at comparing the TPC-H 100GB performance of TiDB v5.3 in the MPP mode with that of Greenplum and Apache Spark, two mainstream analytics engines, in their latest versions. The test result shows that the performance of TiDB v5.3 in the MPP mode is two to three times faster than that of the other two solutions under TPC-H workload.

In v5.0, TiDB introduces the MPP mode for [TiFlash](#), which significantly enhances TiDB's Hybrid Transactional and Analytical Processing (HTAP) capabilities. Test objects in this report are as follows:

- TiDB v5.3 columnar storage in the MPP mode
- Greenplum 6.15.0
- Apache Spark 3.1.1 + Parquet

1.5.3.2 Test environment

1.5.3.2.1 Hardware prerequisite

- Node count: 3

- CPU: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 40 cores
- Memory: 189 GB
- Disks: NVMe 3TB * 2

1.5.3.2.2 Software version

Service type	Software version
TiDB	5.3
Greenplum	6.15.0
Apache Spark	3.1.1

1.5.3.2.3 Parameter configuration

TiDB v5.3

For the v5.3 cluster, TiDB uses the default parameter configuration except for the following configuration items.

In the configuration file `users.toml` of TiFlash, configure `max_memory_usage` as follows:

```
[profiles.default]
max_memory_usage = 1000000000000000
```

Set session variables with the following SQL statements:

```
set @@tidb_isolation_read_engines='tiflash';
set @@tidb_allow_mpp=1;
set @@tidb_mem_quota_query = 10 << 30;
```

All TPC-H test tables are replicated to TiFlash in columnar format, with no additional partitions or indexes.

Greenplum

Except for the initial 3 nodes, the Greenplum cluster is deployed using an additional master node. Each segment server contains 8 segments, which means 4 segments per NVMe SSD. So there are 24 segments in total. The storage format is append-only/column-oriented storage and partition keys are used as primary keys.

```
log_statement = all
gp_autostats_mode = none
statement_mem = 2048MB
gp_vmem_protect_limit = 16384
```

Apache Spark

The test of Apache Spark uses Apache Parquet as the storage format and stores the data on HDFS. The HDFS system consists of three nodes. Each node has two assigned NVMe

SSD disks as the data disks. The Spark cluster is deployed in standalone mode, using NVMe SSD disks as the local directory of `spark.local.dir` to speed up the shuffle spill, with no additional partitions or indexes.

```
--driver-memory 20G
--total-executor-cores 120
--executor-cores 5
--executor-memory 15G
```

1.5.3.3 Test result

Note:

The following test results are the average data of three tests. All numbers are in seconds.

Query ID	TiDB v5.3	Greenplum 6.15.0	Apache Spark 3.1.1 + Parquet
1	8.08	64.1307	52.64
2	2.53	4.76612	11.83
3	4.84	15.62898	13.39
4	10.94	12.88318	8.54
5	12.27	23.35449	25.23
6	1.32	6.033	2.21
7	5.91	12.31266	25.45
8	6.71	11.82444	23.12
9	44.19	22.40144	35.2
10	7.13	12.51071	12.18
11	2.18	2.6221	10.99
12	2.88	7.97906	6.99
13	6.84	10.15873	12.26
14	1.69	4.79394	3.89
15	3.29	10.48785	9.82
16	5.04	4.64262	6.76
17	11.7	74.65243	44.65
18	12.87	64.87646	30.27
19	4.75	8.08625	4.7
20	8.89	15.47016	8.4
21	24.44	39.08594	34.83
22	1.23	7.67476	4.59

TPC-H 100GB on 3 Nodes

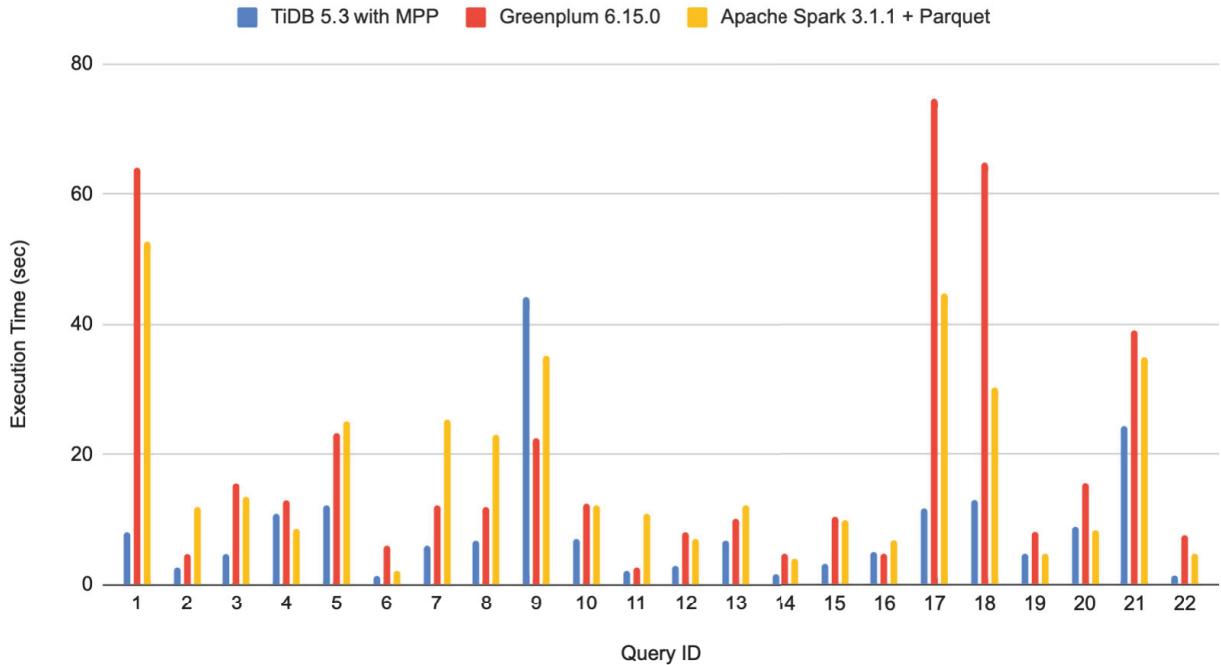


Figure 6: TPC-H

In the performance diagram above:

- Blue lines represent TiDB v5.3;
- Red lines represent Greenplum 6.15.0;
- Yellow lines represent Apache Spark 3.1.1.
- The y-axis represents the execution time of the query. The less the time is, the better the performance is.

1.6 MySQL Compatibility

TiDB is highly compatible with the MySQL 5.7 protocol and the common features and syntax of MySQL 5.7. The ecosystem tools for MySQL 5.7 (PHPMyAdmin, Navicat, MySQL Workbench, mysqldump, and Mydumper/myloader) and the MySQL client can be used for TiDB.

However, some features of MySQL are not supported. This could be because there is now a better way to solve the problem (such as XML functions superseded by JSON), or a lack of current demand versus effort required (such as stored procedures and functions). Some features might also be difficult to implement as a distributed system.

- In addition, TiDB does not support the MySQL replication protocol, but provides specific tools to replicate data with MySQL.
 - Replicate data from MySQL: [TiDB Data Migration \(DM\)](#) is a tool that supports the full data migration and the incremental data replication from MySQL/MariaDB into TiDB.
 - Replicate data to MySQL: [TiCDC](#) is a tool for replicating the incremental data of TiDB by pulling TiKV change logs. TiCDC uses the [MySQL sink](#) to replicate the incremental data of TiDB to MySQL.

Note:

This page refers to general differences between MySQL and TiDB. Refer to the dedicated pages for [Security](#) and [Pessimistic Transaction Mode](#) compatibility.

1.6.1 Unsupported features

- Stored procedures and functions
- Triggers
- Events
- User-defined functions
- FOREIGN KEY constraints [#18209](#)
- FULLTEXT/SPATIAL functions and indexes [#1793](#)
- Character sets other than utf8, utf8mb4, ascii, latin1 and binary
- SYS schema
- Optimizer trace
- XML Functions
- X-Protocol [#1109](#)
- Savepoints [#6840](#)
- Column-level privileges [#9766](#)
- XA syntax (TiDB uses a two-phase commit internally, but this is not exposed via an SQL interface)
- CREATE TABLE tblName AS SELECT stmt syntax [#4754](#)
- CHECK TABLE syntax [#4673](#)
- CHECKSUM TABLE syntax [#1895](#)
- GET_LOCK and RELEASE_LOCK functions [#14994](#)
- LOAD DATA with the REPLACE keyword [#24515](#)

1.6.2 Features that are different from MySQL

1.6.2.1 Auto-increment ID

- In TiDB, auto-increment columns are only guaranteed to be unique and incremental on a single TiDB server, but they are *not* guaranteed to be incremental among multiple TiDB servers or allocated sequentially. It is recommended that you do not mix default values and custom values. Otherwise, you might encounter the **Duplicated Error** error message.
- You can use the `tidb_allow_remove_auto_inc` system variable to allow or forbid removing the `AUTO_INCREMENT` column attribute. The syntax of removing the column attribute is `alter table modify` or `alter table change`.
- TiDB does not support adding the `AUTO_INCREMENT` column attribute, and this attribute cannot be recovered once it is removed.
- See [AUTO_INCREMENT](#) for more details.

Note:

- If you have not specified the primary key when creating a table, TiDB uses `_tidb_rowid` to identify the row. The allocation of this value shares an allocator with the auto-increment column (if such a column exists). If you specify an auto-increment column as the primary key, TiDB uses this column to identify the row. In this situation, the following situation might happen:

```
mysql> create table t(id int unique key AUTO_INCREMENT);
Query OK, 0 rows affected (0.05 sec)

mysql> insert into t values(),(),();
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> select _tidb_rowid, id from t;
+-----+-----+
| _tidb_rowid | id |
+-----+-----+
|          4 |   1 |
|          5 |   2 |
|          6 |   3 |
+-----+-----+
3 rows in set (0.01 sec)
```

1.6.2.2 Performance schema

TiDB uses a combination of [Prometheus](#) and [Grafana](#) to store and query the performance monitoring metrics. Performance schema tables return empty results in TiDB.

1.6.2.3 Query Execution Plan

The output format, output content, and the privilege setting of Query Execution Plan (`EXPLAIN/EXPLAIN FOR`) in TiDB is greatly different from those in MySQL.

The MySQL system variable `optimizer_switch` is read-only in TiDB and has no effect on query plans. You can also use `optimizer hints` in similar syntax to MySQL, but the available hints and implementation might differ.

See [Understand the Query Execution Plan](#) for more details.

1.6.2.4 Built-in functions

TiDB supports most of the MySQL built-in functions, but not all. The statement `SHOW BUILTINS` provides a list of functions that are available.

See also: [TiDB SQL Grammar](#).

1.6.2.5 DDL

In TiDB, all supported DDL changes are performed online. Compared with DDL operations in MySQL, the DDL operations in TiDB have the following major restrictions:

- Multiple operations cannot be completed in a single `ALTER TABLE` statement. For example, it is not possible to add multiple columns or indexes in a single statement. Otherwise, the `Unsupported multi schema change` error might be output.
- `ALTER TABLE` in TiDB does not support the changes of some data types. For example, TiDB does not support the change from the `DECIMAL` type to the `DATE` type. If a data type change is unsupported, TiDB reports the `Unsupported modify column: type %d not match origin %d` error. Refer to [ALTER TABLE](#) for more details.
- The `ALGORITHM={INSTANT,INPLACE,COPY}` syntax functions only as an assertion in TiDB, and does not modify the `ALTER` algorithm. See [ALTER TABLE](#) for further details.
- Adding/Dropping the primary key of the `CLUSTERED` type is unsupported. For more details about the primary key of the `CLUSTERED` type, refer to [clustered index](#).
- Different types of indexes (`HASH|BTREE|RTREE|FULLTEXT`) are not supported, and will be parsed and ignored when specified.
- Table Partitioning supports `HASH`, `RANGE`, and `LIST` partitioning types. Table Partitioning also supports `ADD`, `DROP`, and `TRUNCATE` operations. The other partition operations are ignored. The `Warning: Unsupported partition type %s, treat as normal table` error might be output, where `%s` is a specific partition type. The following Table Partition syntaxes are not supported:
 - `PARTITION BY KEY`

- SUBPARTITION
- {CHECK|TRUNCATE|OPTIMIZE|REPAIR|IMPORT|DISCARD|REBUILD|REORGANIZE|
 ↪ COALESCE} PARTITION

For more details, see [Partitioning](#).

1.6.2.6 Analyze table

[Statistics Collection](#) works differently in TiDB than in MySQL, in that it is a relatively lightweight and short-lived operation in MySQL/InnoDB, while in TiDB it completely rebuilds the statistics for a table and can take much longer to complete.

These differences are documented further in [ANALYZE TABLE](#).

1.6.2.7 Limitations of SELECT syntax

- The syntax `SELECT ... INTO @variable` is not supported.
- The syntax `SELECT ... GROUP BY ... WITH ROLLUP` is not supported.
- The syntax `SELECT ... GROUP BY expr` does not imply `GROUP BY expr ORDER BY
 ↪ expr` as it does in MySQL 5.7.

For details, see the [SELECT](#) statement reference.

1.6.2.8 Views

Views in TiDB are not updatable. They do not support write operations such as `UPDATE`, `INSERT`, and `DELETE`.

1.6.2.9 Temporary tables

For details, see [Compatibility between TiDB local temporary tables and MySQL temporary tables](#).

1.6.2.10 Storage engines

For compatibility reasons, TiDB supports the syntax to create tables with alternative storage engines. In implementation, TiDB describes the metadata as the InnoDB storage engine.

TiDB supports storage engine abstraction similar to MySQL, but you need to specify the storage engine using the `--store` option when you start the TiDB server.

1.6.2.11 SQL modes

TiDB supports most **SQL modes**:

- The compatibility modes, such as `ORACLE` and `POSTGRESQL` are parsed but ignored. Compatibility modes are deprecated in MySQL 5.7 and removed in MySQL 8.0.
- The `ONLY_FULL_GROUP_BY` mode has minor **semantic differences** from MySQL 5.7.
- The `NO_DIR_IN_CREATE` and `NO_ENGINE_SUBSTITUTION` SQL modes in MySQL are accepted for compatibility, but are not applicable to TiDB.

1.6.2.12 Default differences

- Default character set:
 - The default value in TiDB is `utf8mb4`.
 - The default value in MySQL 5.7 is `latin1`.
 - The default value in MySQL 8.0 is `utf8mb4`.
- Default collation:
 - The default collation of `utf8mb4` in TiDB is `utf8mb4_bin`.
 - The default collation of `utf8mb4` in MySQL 5.7 is `utf8mb4_general_ci`.
 - The default collation of `utf8mb4` in MySQL 8.0 is `utf8mb4_0900_ai_ci`.
- Default value of `foreign_key_checks`:
 - The default value in TiDB is `OFF` and currently TiDB only supports `OFF`.
 - The default value in MySQL 5.7 is `ON`.
- Default SQL mode:
 - The default SQL mode in TiDB includes these modes: `ONLY_FULL_GROUP_BY`,
 ↳ `STRICT_TRANS_TABLES`,`NO_ZERO_IN_DATE`,`NO_ZERO_DATE`,`ERROR_FOR_DIVISION_BY_ZERO`
 ↳ ,`NO_AUTO_CREATE_USER`,`NO_ENGINE_SUBSTITUTION`.
 - The default SQL mode in MySQL:
 - * The default SQL mode in MySQL 5.7 is the same as TiDB.
 - * The default SQL mode in MySQL 8.0 includes these modes: `ONLY_FULL_GROUP_BY`
 ↳ ,`STRICT_TRANS_TABLES`,`NO_ZERO_IN_DATE`,`NO_ZERO_DATE`,`ERROR_FOR_DIVISION_BY_ZERO`
 ↳ ,`NO_ENGINE_SUBSTITUTION`.
- Default value of `lower_case_table_names`:
 - The default value in TiDB is 2 and currently TiDB only supports 2.
 - The default value in MySQL:
 - * On Linux: 0
 - * On Windows: 1
 - * On macOS: 2
- Default value of `explicit_defaults_for_timestamp`:

- The default value in TiDB is `ON` and currently TiDB only supports `ON`.
- The default value in MySQL:
 - * For MySQL 5.7: `OFF`.
 - * For MySQL 8.0: `ON`.

1.6.2.13 Date and Time

1.6.2.13.1 Named timezone

- TiDB uses all time zone rules currently installed in the system for calculation (usually the `tzdata` package). You can use all time zone names without importing the time zone table data. You cannot modify the calculation rules by importing the time zone table data.
- MySQL uses the local time zone by default and relies on the current time zone rules built into the system (such as when to start daylight saving time) for calculation; and the time zone cannot be specified by the time zone name without [importing the time zone table data](#).

1.6.2.14 Type system differences

The following column types are supported by MySQL, but **NOT** by TiDB:

- `FLOAT4/FLOAT8`
- `SQL_TSI_*` (including `SQL_TSI_MONTH`, `SQL_TSI_WEEK`, `SQL_TSI_DAY`, `SQL_TSI_HOUR`, `SQL_TSI_MINUTE` and `SQL_TSI_SECOND`, excluding `SQL_TSI_YEAR`)

1.6.2.15 Incompatibility caused by deprecated features

TiDB does not implement certain features that have been marked as deprecated in MySQL, including:

- Specifying precision for floating point types. MySQL 8.0 [deprecates](#) this feature, and it is recommended to use the `DECIMAL` type instead.
- The `ZEROFILL` attribute. MySQL 8.0 [deprecates](#) this feature, and it is recommended to instead pad numeric values in your application.

1.7 TiDB Limitations

This document describes the common usage limitations of TiDB, including the maximum identifier length and the maximum number of supported databases, tables, indexes, partitioned tables, and sequences.

1.7.1 Limitations on identifier length

Identifier type	Maximum length (number of characters allowed)
Database	64
Table	64
Column	64
Index	64
View	64
Sequence	64

1.7.2 Limitations on the total number of databases, tables, views, and connections

Identifier type	Maximum number
Databases	unlimited
Tables	unlimited
Views	unlimited
Connections	unlimited

1.7.3 Limitations on a single database

Type	Upper limit
Tables	unlimited

1.7.4 Limitations on a single table

Type	Upper limit (default value)
Columns	Defaults to 1071 and can be adjusted up to 4096
Indexes	Defaults to 64 and can be adjusted up to 512
Rows	unlimited
Size	unlimited
Partitions	1024

- The upper limit of `Columns` can be modified via [table-column-count-limit](#).
- The upper limit of `Indexes` can be modified via [index-limit](#).

1.7.5 Limitation on a single row

Type	Upper limit
Size	6 MB by default. You can adjust the size limit via the <code>txn-entry-size-limit</code> configuration item.

1.7.6 Limitation on a single column

Type	Upper limit
Size	6 MB

1.7.7 Limitations on string types

Type	Upper limit
CHAR	256 characters
BINARY	256 characters
VARBINARY	65535 characters
VARCHAR	16383 characters
TEXT	6 MB
BLOB	6 MB

1.7.8 Limitations on SQL statements

Type	Upper limit
The maximum number of SQL statements in a single transaction	When the optimistic transaction is used and the transaction retry is enabled, the default upper limit is 5000, which can be modified using <code>stmt-count ↪ -limit.</code>

1.8 TiDB Adopters

This is a list of TiDB adopters in various industries.

Company	Industry	Case studies
SHAREit Group	Internet	English ; Chinese
China Telecom Bestpay	Mobile Payment	English ; Chinese
VNG	Mobile Payment	English #1, #2
Ping++	Mobile Payment	
LianLian Tech	Mobile Payment	
U-Next	Media and Entertainment	English
Dailymotion	Media and Entertainment	
iQIYI	Media and Entertainment	English #1, #2; Chinese
BookMyShow	Media and Entertainment	English
Yiguo.com	E-commerce	English
Shopee	E-commerce	English #1, #2
Zhuan Zhuan	E-commerce	English #1, #2
Xiaohongshu	E-commerce	English ; Chinese
Meituan	E-commerce	English #1, #2; Chinese
Happigo.com	E-commerce	
Yimutian	E-commerce	
Maizuo	E-commerce	
Mogujie	E-commerce	
Zhihu	Knowledge Sharing	English #1, #2, #3
PatSnap	Artificial Intelligence	English ; Chinese

Company	Industry	Case studies
JD Cloud	Cloud Computing	English
Mobike	Ridesharing	English
Autohome	Automobile	English; Chinese
Chehaoduo	Automobile	English
Xiaomi	Consumer Electronics	English #1, #2
LY.com	Travel	
Qunar.com	Travel	
Hulu	Entertainment	
PalFish	EdTech	English; Chinese
VIPKid	EdTech	English; Chinese
Yuanfudao.com	EdTech	English
Bank of Beijing	Banking	English; Chinese
WeBank	Banking	English; Chinese
Bank of China	Banking	English; Chinese
China Zheshang Bank	Banking	English; Chinese
Industrial and Commercial Bank of China	Banking	
Ping An Life Insurance	Insurance	English; Chinese
Yimian Data	Big Data	
CAASDATA	Big Data	
58.com	Advertising	English; Chinese
Mobikok	AdTech	
Ninja Van	Logistics	English
ZTO Express	Logistics	English; Chinese
JD.com	Logistics	English; Chinese
G7 Networks	Logistics	
Hive-Box	Logistics	
NetEase Games	Gaming	English; Chinese
GAEA	Gaming	English
Kunlun	Gaming	English
YOOZOO Games	Gaming	
Seasun Games	Gaming	
FUNYOURS JAPAN	Gaming	
Hoodinn	Gaming	
SEA group	Gaming	
Zhaopin.com	Recruiting	
BIGO	Live Streaming	English; Chinese
Huya Live	Live Streaming	English; Chinese
Panda.tv	Live Streaming	
Phoenix New Media	Media	
Tencent OMG	Media	
Terren	Media	
LeCloud	Media	
Miaopai	Media	

Company	Industry	Case studies
Meizu	Media	
Sogou	MediaTech	
Gengmei	Plastic Surgery	
Keruyun	SaaS	
LinkDoc Technology	HealthTech	
Chunyu Yisheng	HealthTech	
Qutoutiao	Social Network	
Jimri Toutiao	Mobile News Platform	
360 Finance	FinTech	Chinese
Tongdun Technology	FinTech	
Wacai	FinTech	
Tree Finance	FinTech	
Mashang Consumer Finance	FinTech	
Snowball Finance	FinTech	
QuantGroup	FinTech	
FINUP	FinTech	
Meili Finance	FinTech	
Guolian Securities	Financial Services	
Founder Securities	Financial Services	
China Telecom Shanghai	Telecom	
State Administration of Taxation	Finance	
Hainan eKing Technology	Enterprise Technology	
Wuhan Antian Information Technology	Enterprise Technology	
Lenovo	Enterprise Technology	
2Dfire.com	FoodTech	
Acewill	FoodTech	
Ausnutria Dairy	FoodTech	
Qingdao Telaidian	Electric Car Charger	

1.9 Credits

Each contributor has played an important role in promoting the robust development of TiDB. We sincerely appreciate all contributors who have submitted code, written and translated documents for TiDB.

1.9.1 TiDB developers

TiDB developers contribute to new feature development, performance improvement, stability guarantee, and bug fixes. The following is the list of contributors in TiDB related repos:

- pingcap/tidb

- [tikv/tikv](#)
- [pingcap/parser](#)
- [tikv/pd](#)
- [pingcap/tidb-operator](#)
- [pingcap/tiup](#)
- [pingcap/br](#)
- [pingcap/dm](#)
- [pingcap/tidb-binlog](#)
- [pingcap/tidb-dashboard](#)
- [pingcap/tiflow](#)
- [pingcap/tidb-tools](#)
- [pingcap/tidb-lightning](#)
- [pingcap/tispark](#)
- [pingcap/dumpling](#)
- [tikv/client-java](#)
- [tidb-incubator/TiBigData](#)
- [ti-community-infra](#)

For the full list of contributors, see [SIG | TiDB DevGroup](#).

1.9.2 Writers and translators for TiDB documentation

Writers and translators write and translate documents for TiDB and the related projects. The following is the list of contributors in TiDB documentation related repos:

- [pingcap/docs-cn](#)
- [pingcap/docs](#)
- [pingcap/docs-tidb-operator](#)
- [pingcap/docs-dm](#)
- [tikv/website](#)

2 Quick Start

2.1 Quick Start Guide for the TiDB Database Platform

This guide walks you through the quickest way to get started with TiDB. You will be using TiUP, a package manager in the TiDB ecosystem, to help you run any TiDB cluster component with only a single line of command.

Note:

TiDB, TiUP and TiDB Dashboard share usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see [Telemetry](#).

Note:

The deployment method provided in this guide is **ONLY FOR** quick start, **NOT FOR** production.

- To deploy an on-premises production cluster, see [production installation guide](#).
- To deploy TiDB in Kubernetes, see [Get Started with TiDB in Kubernetes](#).
- To manage TiDB in the cloud, see [TiDB Cloud Quick Start](#).

2.1.1 Deploy a local test environment on Mac OS

As a distributed system, a basic TiDB test cluster usually consists of 2 TiDB instances, 3 TiKV instances, 3 PD instances, and optional TiFlash instances. With TiUP Playground, you can quickly build the test cluster by taking the following steps:

1. Download and install TiUP:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
↪ install.sh | sh
```

2. Declare the global environment variable:

Note:

After the installation, TiUP displays the absolute path of the corresponding `profile` file. You need to modify the following `source` command according to the path.

```
source .bash_profile
```

3. Start the cluster in the current session:

- If you want to start a TiDB cluster of the latest version with 1 TiDB instance, 1 TiKV instance, 1 PD instance, and 1 TiFlash instance, run the following command:

```
tiup playground
```

- If you want to specify the TiDB version and the number of the instances of each component, run a command like this:

```
tiup playground v5.3.0 --db 2 --pd 3 --kv 3
```

The command downloads a version cluster to the local machine and starts it, such as v5.3.0. To view the latest version, run `tiup list tidb`.

This command returns the access methods of the cluster:

```
CLUSTER START SUCCESSFULLY, Enjoy it ^-^
To connect TiDB: mysql --comments --host 127.0.0.1 --port 4001 -u
    ↪ root -p (no password)
To connect TiDB: mysql --comments --host 127.0.0.1 --port 4000 -u
    ↪ root -p (no password)
To view the dashboard: http://127.0.0.1:2379/dashboard
PD client endpoints: [127.0.0.1:2379 127.0.0.1:2382 127.0.0.1:2384]
To view Prometheus: http://127.0.0.1:9090
To view Grafana: http://127.0.0.1:3000
```

Note:

- Since v5.2.0, TiDB supports running `tiup playground` on the machine that uses the Apple M1 chip.
- For the playground operated in this way, after the test deployment is finished, TiUP will clean up the original cluster data. You will get a new cluster after re-running the command.
- If you want the data to be persisted on storage, run `tiup --tag ↪ <your-tag> playground`. For details, refer to [TiUP Reference Guide](#).

4. Start a new session to access TiDB:

- Use the TiUP client to connect to TiDB.

```
tiup client
```

- You can also use the MySQL client to connect to TiDB.

```
mysql --host 127.0.0.1 --port 4000 -u root
```

5. Access the Prometheus dashboard of TiDB at <http://127.0.0.1:9090>.

6. Access the **TiDB Dashboard** at <http://127.0.0.1:2379/dashboard>. The default user-name is `root`, with an empty password.
7. (Optional) **Load data to TiFlash** for analysis.
8. Clean up the cluster after the test deployment:
 1. Stop the process by pressing `ctrl-c`.
 2. Run the following command:

```
tiup clean --all
```

Note:

TiUP Playground listens on `127.0.0.1` by default, and the service is only locally accessible. If you want the service to be externally accessible, specify the listening address using the `--host` parameter to bind the network interface card (NIC) to an externally accessible IP address.

2.1.2 Deploy a local test environment on Linux OS

As a distributed system, a basic TiDB test cluster usually consists of 2 TiDB instances, 3 TiKV instances, 3 PD instances, and optional TiFlash instances. With TiUP Playground, you can quickly build the test cluster by taking the following steps:

1. Download and install TiUP:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
↪ install.sh | sh
```

2. Declare the global environment variable:

Note:

After the installation, TiUP displays the absolute path of the corresponding `profile` file. You need to modify the following `source` command according to the path.

```
source .bash_profile
```

3. Start the cluster in the current session:

- If you want to start a TiDB cluster of the latest version with 1 TiDB instance, 1 TiKV instance, 1 PD instance, and 1 TiFlash instance, run the following command:

```
tiup playground
```

- If you want to specify the TiDB version and the number of the instances of each component, run a command like this:

```
tiup playground v5.3.0 --db 2 --pd 3 --kv 3
```

The command downloads a version cluster to the local machine and starts it, such as v5.3.0. To view the latest version, run `tiup list tidb`.

This command returns the access methods of the cluster:

```
CLUSTER START SUCCESSFULLY, Enjoy it ^-^
To connect TiDB: mysql --host 127.0.0.1 --port 4000 -u root -p (no
    ↪ password) --comments
To view the dashboard: http://127.0.0.1:2379/dashboard
PD client endpoints: [127.0.0.1:2379]
To view the Prometheus: http://127.0.0.1:9090
To view the Grafana: http://127.0.0.1:3000
```

Note:

For the playground operated in this way, after the test deployment is finished, TiUP will clean up the original cluster data. You will get a new cluster after re-running the command. If you want the data to be persisted on storage, run `tiup --tag <your-tag> playground`
↪ For details, refer to [TiUP Reference Guide](#).

4. Start a new session to access TiDB:

- Use the TiUP client to connect to TiDB.

```
tiup client
```

- You can also use the MySQL client to connect to TiDB.

```
mysql --host 127.0.0.1 --port 4000 -u root
```

5. Access the Prometheus dashboard of TiDB at <http://127.0.0.1:9090>.

6. Access the [TiDB Dashboard](http://127.0.0.1:2379/dashboard) at <http://127.0.0.1:2379/dashboard>. The default user-name is `root`, with an empty password.

7. Access the Grafana dashboard of TiDB through <http://127.0.0.1:3000>. Both the default username and password are `admin`.

8. (Optional) [Load data to TiFlash](#) for analysis.
9. Clean up the cluster after the test deployment:
 1. Stop the process by pressing `ctrl-c`.
 2. Run the following command:

```
tiup clean --all
```

Note:

TiUP Playground listens on 127.0.0.1 by default, and the service is only locally accessible. If you want the service to be externally accessible, specify the listening address using the `--host` parameter to bind the network interface card (NIC) to an externally accessible IP address.

2.1.3 Set up a test environment on a single machine using TiUP cluster

- Scenario: Experience a smallest TiDB cluster with the complete topology and simulate the production deployment steps on a single Linux server.
- Time required: 10 minutes

This section describes how to deploy a TiDB cluster using a YAML file of the smallest topology in TiUP.

2.1.3.1 Prepare

Prepare a target machine that meets the following requirements:

- CentOS 7.3 or a later version is installed
- The Linux OS has access to the Internet, which is required to download TiDB and related software installation packages

The smallest TiDB cluster topology is as follows:

Note: > > The IP address of the following instances only serves as an example IP. In your actual deployment, you need to replace the IP with your actual IP.

Instance	Count	IP	Configuration
TiKV	3	10.0.1	Avoid 10.0.1 don- 10.0.1 flict be- tween the port and the di- rec- tory
TiDB	1	10.0.1	The de- fault port Global di- rec- tory con- fig- u- ra- tion
PD	1	10.0.1	The de- fault port Global di- rec- tory con- fig- u- ra- tion

Instance	Count	IP	Configuration
TiFlash	1	10.0.1.1	The de- fault port Global di- rec- tory con- fig- u- ra- tion
Monit	1	10.0.1.1	The de- fault port Global di- rec- tory con- fig- u- ra- tion

Other requirements for the target machine:

- The `root` user and its password is required
- **Stop the firewall service of the target machine**, or open the port needed by the TiDB cluster nodes
- Currently, TiUP supports deploying TiDB on the x86_64 (AMD64 and ARM) architectures:
 - It is recommended to use CentOS 7.3 or later versions on AMD64
 - It is recommended to use CentOS 7.6 1810 on ARM

2.1.3.2 Deploy

Note:

You can log in to the target machine as a regular user or the `root` user. The following steps use the `root` user as an example.

1. Download and install TiUP:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
↪ install.sh | sh
```

2. Install the cluster component of TiUP:

```
tiup cluster
```

3. If the TiUP cluster is already installed on the machine, update the software version:

```
tiup update --self && tiup update cluster
```

4. Use the `root` user privilege to increase the connection limit of the `sshd` service. This is because TiUP needs to simulate deployment on multiple machines.

1. Modify `/etc/ssh/sshd_config`, and set `MaxSessions` to 20.

2. Restart the `sshd` service:

```
service sshd restart
```

5. Create and start the cluster:

Edit the configuration file according to the following template, and name it as `topo.yaml`:

```
# # Global variables are applied to all deployments and used as the
# default value of
# # the deployments if a specific deployment value is missing.
global:
  user: "tidb"
  ssh_port: 22
  deploy_dir: "/tidb-deploy"
  data_dir: "/tidb-data"

# # Monitored variables are applied to all the machines.
monitored:
  node_exporter_port: 9100
```

```

blackbox_exporter_port: 9115

server_configs:
tidb:
  log.slow-threshold: 300
tikv:
  readpool.storage.use-unified-pool: false
  readpool.coprocessor.use-unified-pool: true
pd:
  replication.enable-placement-rules: true
  replication.location-labels: ["host"]
tiflash:
  logger.level: "info"

pd_servers:
- host: 10.0.1.1

tidb_servers:
- host: 10.0.1.1

tikv_servers:
- host: 10.0.1.1
  port: 20160
  status_port: 20180
  config:
    server.labels: { host: "logic-host-1" }

- host: 10.0.1.1
  port: 20161
  status_port: 20181
  config:
    server.labels: { host: "logic-host-2" }

- host: 10.0.1.1
  port: 20162
  status_port: 20182
  config:
    server.labels: { host: "logic-host-3" }

tiflash_servers:
- host: 10.0.1.1

monitoring_servers:
- host: 10.0.1.1

```

```
grafana_servers:
  - host: 10.0.1.1
```

- **user: "tidb"**: Use the `tidb` system user (automatically created during deployment) to perform the internal management of the cluster. By default, use port 22 to log in to the target machine via SSH.
- **replication.enable-placement-rules**: This PD parameter is set to ensure that TiFlash runs normally.
- **host**: The IP of the target machine.

6. Execute the cluster deployment command:

```
tiup cluster deploy <cluster-name> <tidb-version> ./topo.yaml --user
  ↳ root -p
```

- **<cluster-name>**: Set the cluster name
- **<tidb-version>**: Set the TiDB cluster version. You can see all the supported TiDB versions by running the `tiup list tidb` command

Enter “y” and the `root` user’s password to complete the deployment:

```
Do you want to continue? [y/N]: y
Input SSH password:
```

7. Start the cluster:

```
tiup cluster start <cluster-name>
```

8. Access the cluster:

- Install the MySQL client. If it is already installed, skip this step.

```
yum -y install mysql
```

- Access TiDB. The password is empty:

```
mysql -h 10.0.1.1 -P 4000 -u root
```

- Access the Grafana monitoring dashboard at <http://%7Bgrafana-ip%7D:3000>. The default username and password are both `admin`.
- Access the **TiDB Dashboard** at <http://%7Bpd-ip%7D:2379/dashboard>. The default username is `root`, and the password is empty.
- To view the currently deployed cluster list:

```
tiup cluster list
```

- To view the cluster topology and status:

```
tiup cluster display <cluster-name>
```

2.1.4 What's next

- If you have just deployed a TiDB cluster for the local test environment:
 - Learn [Basic SQL operations in TiDB](#)
 - [Migrate data to TiDB](#)
- If you are ready to deploy a TiDB cluster for the production environment:
 - [Deploy TiDB using TiUP](#)
 - [Deploy TiDB on Cloud using TiDB Operator](#)
- If you're looking for analytics solution with TiFlash:
 - [Use TiFlash](#)
 - [TiFlash Overview](#)

2.2 Quick Start Guide for TiDB HTAP

This guide walks you through the quickest way to get started with TiDB's one-stop solution of Hybrid Transactional and Analytical Processing (HTAP).

Note:

The steps provided in this guide is ONLY for quick start in the test environment. For production environments, [explore HTAP](#) is recommended.

2.2.1 Basic concepts

Before using TiDB HTAP, you need to have some basic knowledge about [TiKV](#), a row-based storage engine for TiDB Online Transactional Processing (OLTP), and [TiFlash](#), a columnar storage engine for TiDB Online Analytical Processing (OLAP).

- Storage engines of HTAP: The row-based storage engine and the columnar storage engine co-exist for HTAP. Both storage engines can replicate data automatically and keep strong consistency. The row-based storage engine optimizes OLTP performance, and the columnar storage engine optimizes OLAP performance.
- Data consistency of HTAP: As a distributed and transactional key-value database, TiKV provides transactional interfaces with ACID compliance, and guarantees data consistency between multiple replicas and high availability with the implementation of the [Raft consensus algorithm](#). As a columnar storage extension of TiKV, TiFlash replicates data from TiKV in real time according to the Raft Learner consensus algorithm, which ensures that data is strongly consistent between TiKV and TiFlash.

- Data isolation of HTAP: TiKV and TiFlash can be deployed on different machines as needed to solve the problem of HTAP resource isolation.
- MPP computing engine: **MPP** is a distributed computing framework provided by the TiFlash engine since TiDB 5.0, which allows data exchange between nodes and provides high-performance, high-throughput SQL algorithms. In the MPP mode, the run time of the analytic queries can be significantly reduced.

2.2.2 Steps

In this document, you can experience the convenience and high performance of TiDB HTAP by querying an example table in a popular [TPC-H](#) dataset.

2.2.2.1 Step 1. Deploy a local test environment

Before using TiDB HTAP, follow the steps in the [Quick Start Guide for the TiDB Database Platform](#) to prepare a local test environment, and run the following command to deploy a TiDB cluster:

```
tiup playground
```

Note:

`tiup playground` command is ONLY for quick start, NOT for production.

2.2.2.2 Step 2. Prepare test data

In the following steps, you can create a [TPC-H](#) dataset as the test data to use TiDB HTAP. If you are interested in TPC-H, see [General Implementation Guidelines](#).

Note:

If you want to use your existing data for analytic queries, you can [migrate your data to TiDB](#). If you want to design and create your own test data, you can create it by executing SQL statements or using related tools.

1. Install the test data generation tool by running the following command:

```
tiup install bench
```

2. Generate the test data by running the following command:

```
tiup bench tpch --sf=1 prepare
```

If the output of this command shows **Finished**, it indicates that the data is created.

3. Execute the following SQL statement to view the generated data:

```
SELECT
    CONCAT(table_schema,'.',table_name) AS 'Table Name',
    table_rows AS 'Number of Rows',
    FORMAT_BYTES(data_length) AS 'Data Size',
    FORMAT_BYTES(index_length) AS 'Index Size',
    FORMAT_BYTES(data_length+index_length) AS 'Total'
FROM
    information_schema.TABLES
WHERE
    table_schema='test';
```

As you can see from the output, eight tables are created in total, and the largest table has 6.5 million rows (the number of rows created by the tool depends on the actual SQL query result because the data is randomly generated).

```
sql +-----+-----+-----+-----+
→ | Table Name | Number of Rows | Data Size | Index Size | Total |
→ +-----+-----+-----+-----+
→ | test.nation | 25 | 2.44 KiB | 0 bytes | 2.44 KiB | | test.region
→ | 5 | 416 bytes | 0 bytes | 416 bytes | | test.part | 200000
→ | 25.07 MiB | 0 bytes | 25.07 MiB | | test.supplier | 10000 |
→ | 1.45 MiB | 0 bytes | 1.45 MiB | | test.partsupp | 800000 |
→ | 120.17 MiB| 12.21 MiB | 132.38 MiB| | test.customer | 150000 | 24.77
→ | MiB | 0 bytes | 24.77 MiB | | test.orders | 1527648 | 174.40
→ | MiB | 0 bytes | 174.40 MiB| | test.lineitem | 6491711 | 849.07 MiB|
→ | 99.06 MiB | 948.13 MiB| +-----+-----+
→ | 8 rows in set (0.06 sec)
```

This is a database of a commercial ordering system. In which, the **test.nation** table indicates the information about countries, the **test.region** table indicates the information about regions, the **test.part** table indicates the information about parts, the **test.supplier** table indicates the information about suppliers, the **test.partsupp** table indicates the information about parts of suppliers, the **test.customer** table indicates the information about customers, the **test.orders** table indicates the information about orders, and the **test.lineitem** table indicates the information about online items.

2.2.2.3 Step 3. Query data with the row-based storage engine

To know the performance of TiDB with only the row-based storage engine, execute the following SQL statements:

```
SELECT
    l_orderkey,
    SUM(
        l_extendedprice * (1 - l_discount)
    ) AS revenue,
    o_orderdate,
    o_shipppriority
FROM
    customer,
    orders,
    lineitem
WHERE
    c_mktsegment = 'BUILDING'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < DATE '1996-01-01'
AND l_shipdate > DATE '1996-02-01'
GROUP BY
    l_orderkey,
    o_orderdate,
    o_shipppriority
ORDER BY
    revenue DESC,
    o_orderdate
limit 10;
```

This is a shipping priority query, which provides the priority and potential revenue of the highest-revenue order that has not been shipped before a specified date. The potential revenue is defined as the sum of `l_extendedprice * (1-l_discount)`. The orders are listed in the descending order of revenue. In this example, this query lists the unshipped orders with potential query revenue in the top 10.

2.2.2.4 Step 4. Replicate the test data to the columnar storage engine

After TiFlash is deployed, TiKV does not replicate data to TiFlash immediately. You need to execute the following DDL statements in a MySQL client of TiDB to specify which tables need to be replicated. After that, TiDB will create the specified replicas in TiFlash accordingly.

```
ALTER TABLE test.customer SET TIFLASH REPLICAS 1;
ALTER TABLE test.orders SET TIFLASH REPLICAS 1;
ALTER TABLE test.lineitem SET TIFLASH REPLICAS 1;
```

To check the replication status of the specific tables, execute the following statements:

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = 'test
    ↪ ' and TABLE_NAME = 'customer';
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = 'test
    ↪ ' and TABLE_NAME = 'orders';
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA = 'test
    ↪ ' and TABLE_NAME = 'lineitem';
```

In the result of the above statements:

- AVAILABLE indicates whether the TiFlash replica of a specific table is available or not. 1 means available and 0 means unavailable. Once a replica becomes available, this status does not change any more. If you use DDL statements to modify the number of replicas, the replication status will be recalculated.
- PROGRESS means the progress of the replication. The value is between 0.0 and 1.0. 1 means at least one replica is replicated.

2.2.2.5 Step 5. Analyze data faster using HTAP

Execute the SQL statements in [Step 3](#) again, and you can see the performance of TiDB HTAP.

For tables with TiFlash replicas, the TiDB optimizer automatically determines whether to use TiFlash replicas based on the cost estimation. To check whether or not a TiFlash replica is selected, you can use the `desc` or `explain analyze` statement. For example:

```
explain analyze SELECT
    l_orderkey,
    SUM(
        l_extendedprice * (1 - l_discount)
    ) AS revenue,
    o_orderdate,
    o_shippriority
FROM
    customer,
    orders,
    lineitem
WHERE
    c_mktsegment = 'BUILDING'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < DATE '1996-01-01'
AND l_shipdate > DATE '1996-02-01'
GROUP BY
    l_orderkey,
    o_orderdate,
```

```

    o_shipppriority
ORDER BY
    revenue DESC,
    o_orderdate
limit 10;

```

If the result of the EXPLAIN statement shows ExchangeSender and ExchangeReceiver operators, it indicates that the MPP mode has taken effect.

In addition, you can specify that each part of the entire query is computed using only the TiFlash engine. For detailed information, see [Use TiDB to read TiFlash replicas](#).

You can compare query results and query performance of these two methods.

2.2.3 What's next

- [Architecture of TiDB HTAP](#)
- [Explore HTAP](#)
- [Use TiFlash](#)

2.3 Explore SQL with TiDB

TiDB is compatible with MySQL, you can use MySQL statements directly in most of the cases. For unsupported features, see [Compatibility with MySQL](#).

To experiment with SQL and test out TiDB compatibility with MySQL queries, you can run [TiDB directly in your web browser without installing it](#). You can also first deploy a TiDB cluster and then run SQL statements in it.

This page walks you through the basic TiDB SQL statements such as DDL, DML and CRUD operations. For a complete list of TiDB statements, see [TiDB SQL Syntax Diagram](#).

2.3.1 Category

SQL is divided into the following 4 types according to their functions:

- DDL (Data Definition Language): It is used to define database objects, including databases, tables, views, and indexes.
- DML (Data Manipulation Language): It is used to manipulate application related records.
- DQL (Data Query Language): It is used to query the records after conditional filtering.
- DCL (Data Control Language): It is used to define access privileges and security levels.

Common DDL features are creating, modifying, and deleting objects (such as tables and indexes). The corresponding commands are CREATE, ALTER, and DROP.

2.3.2 Show, create and drop a database

A database in TiDB can be considered as a collection of objects such as tables and indexes.

To show the list of databases, use the `SHOW DATABASES` statement:

```
SHOW DATABASES;
```

To use the database named `mysql`, use the following statement:

```
USE mysql;
```

To show all the tables in a database, use the `SHOW TABLES` statement:

```
SHOW TABLES FROM mysql;
```

To create a database, use the `CREATE DATABASE` statement:

```
CREATE DATABASE db_name [options];
```

To create a database named `samp_db`, use the following statement:

```
CREATE DATABASE IF NOT EXISTS samp_db;
```

Add `IF NOT EXISTS` to prevent an error if the database exists.

To delete a database, use the `DROP DATABASE` statement:

```
DROP DATABASE samp_db;
```

2.3.3 Create, show, and drop a table

To create a table, use the `CREATE TABLE` statement:

```
CREATE TABLE table_name column_name data_type constraint;
```

For example, to create a table named `person` which includes fields such as number, name, and birthday, use the following statement:

```
CREATE TABLE person (
    id INT(11),
    name VARCHAR(255),
    birthday DATE
);
```

To view the statement that creates the table (DDL), use the `SHOW CREATE` statement:

```
SHOW CREATE table person;
```

To delete a table, use the `DROP TABLE` statement:

```
DROP TABLE person;
```

2.3.4 Create, show, and drop an index

Indexes are used to speed up queries on indexed columns. To create an index for the column whose value is not unique, use the `CREATE INDEX` statement:

```
CREATE INDEX person_id ON person (id);
```

Or use the `ALTER TABLE` statement:

```
ALTER TABLE person ADD INDEX person_id (id);
```

To create a unique index for the column whose value is unique, use the `CREATE UNIQUE INDEX` statement:

```
CREATE UNIQUE INDEX person_unique_id ON person (id);
```

Or use the `ALTER TABLE` statement:

```
ALTER TABLE person ADD UNIQUE person_unique_id (id);
```

To show all the indexes in a table, use the `SHOW INDEX` statement:

```
SHOW INDEX FROM person;
```

To delete an index, use the `DROP INDEX` or `ALTER TABLE` statement. `DROP INDEX` can be nested in `ALTER TABLE`:

```
DROP INDEX person_id ON person;
```

```
ALTER TABLE person DROP INDEX person_unique_id;
```

Note:

DDL operations are not transactions. You don't need to run a `COMMIT` statement when executing DDL operations.

2.3.5 Insert, update, and delete data

Common DML features are adding, modifying, and deleting table records. The corresponding commands are `INSERT`, `UPDATE`, and `DELETE`.

To insert data into a table, use the `INSERT` statement:

```
INSERT INTO person VALUES(1, 'tom', '20170912');
```

To insert a record containing data of some fields into a table, use the `INSERT` statement:

```
INSERT INTO person(id, name) VALUES('2', 'bob');
```

To update some fields of a record in a table, use the `UPDATE` statement:

```
UPDATE person SET birthday='20180808' WHERE id=2;
```

To delete the data in a table, use the `DELETE` statement:

```
DELETE FROM person WHERE id=2;
```

Note:

The `UPDATE` and `DELETE` statements without the `WHERE` clause as a filter operate on the entire table.

2.3.6 Query data

DQL is used to retrieve the desired data rows from a table or multiple tables.

To view the data in a table, use the `SELECT` statement:

```
SELECT * FROM person;
```

To query a specific column, add the column name after the `SELECT` keyword:

```
SELECT name FROM person;
```

```
+-----+
| name |
+-----+
| tom |
+-----+
1 rows in set (0.00 sec)
```

Use the WHERE clause to filter all records that match the conditions and then return the result:

```
SELECT * FROM person where id<5;
```

2.3.7 Create, authorize, and delete a user

DCL are usually used to create or delete users, and manage user privileges.

To create a user, use the CREATE USER statement. The following example creates a user named `tiuser` with the password `123456`:

```
CREATE USER 'tiuser'@'localhost' IDENTIFIED BY '123456';
```

To grant `tiuser` the privilege to retrieve the tables in the `samp_db` database:

```
GRANT SELECT ON samp_db.* TO 'tiuser'@'localhost';
```

To check the privileges of `tiuser`:

```
SHOW GRANTS for tiuser@localhost;
```

To delete `tiuser`:

```
DROP USER 'tiuser'@'localhost';
```

2.4 Explore HTAP

This guide describes how to explore and use the features of TiDB Hybrid Transactional and Analytical Processing (HTAP).

Note:

If you are new to TiDB HTAP and want to start using it quickly, see [Quick start with HTAP](#).

2.4.1 Use cases

TiDB HTAP can handle the massive data that increases rapidly, reduce the cost of DevOps, and be deployed in either on-premises or cloud environments easily, which brings the value of data assets in real time.

The following are the typical use cases of HTAP:

- Hybrid workload

When using TiDB for real-time Online Analytical Processing (OLAP) in hybrid load scenarios, you only need to provide an entry point of TiDB to your data. TiDB automatically selects different processing engines based on the specific business.

- Real-time stream processing

When using TiDB in real-time stream processing scenarios, TiDB ensures that all the data flowed in constantly can be queried in real time. At the same time, TiDB also can handle highly concurrent data workloads and Business Intelligence (BI) queries.

- Data hub

When using TiDB as a data hub, TiDB can meet specific business needs by seamlessly connecting the data for the application and the data warehouse.

For more information about use cases of TiDB HTAP, see [blogs about HTAP](#) on the PingCAP website.

2.4.2 Architecture

In TiDB, a row-based storage engine [TiKV](#) for Online Transactional Processing (OLTP) and a columnar storage engine [TiFlash](#) for Online Analytical Processing (OLAP) co-exist, replicate data automatically, and keep strong consistency.

For more information about the architecture, see [architecture of TiDB HTAP](#).

2.4.3 Environment preparation

Before exploring the features of TiDB HTAP, you need to deploy TiDB and the corresponding storage engines according to the data volume. If the data volume is large (for example, 100 T), it is recommended to use TiFlash Massively Parallel Processing (MPP) as the primary solution and TiSpark as the supplementary solution.

- TiFlash

- If you have deployed a TiDB cluster with no TiFlash node, add the TiFlash nodes in the current TiDB cluster. For detailed information, see [Scale out a TiFlash cluster](#).
- If you have not deployed a TiDB cluster, see [Deploy a TiDB Cluster using TiUP](#). Based on the minimal TiDB topology, you also need to deploy the [topology of TiFlash](#).
- When deciding how to choose the number of TiFlash nodes, consider the following scenarios:

- * If your use case requires OLTP with small-scale analytical processing and Ad-Hoc queries, deploy one or several TiFlash nodes. They can dramatically increase the speed of analytic queries.
- * If the OLTP throughput does not cause significant pressure to I/O usage rate of the TiFlash nodes, each TiFlash node uses more resources for computation, and thus the TiFlash cluster can have near-linear scalability. The number of TiFlash nodes should be tuned based on expected performance and response time.
- * If the OLTP throughput is relatively high (for example, the write or update throughput is higher than 10 million lines/hours), due to the limited write capacity of network and physical disks, the I/O between TiKV and TiFlash becomes a bottleneck and is also prone to read and write hotspots. In this case, the number of TiFlash nodes has a complex non-linear relationship with the computation volume of analytical processing, so you need to tune the number of TiFlash nodes based on the actual status of the system.

- TiSpark

- If your data needs to be analyzed with Spark, deploy TiSpark (Spark 3.x is not currently supported). For specific process, see [TiSpark User Guide](#).

2.4.4 Data preparation

After TiFlash is deployed, TiKV does not replicate data to TiFlash automatically. You need to manually specify which tables need to be replicated to TiFlash. After that, TiDB creates the corresponding TiFlash replicas.

- If there is no data in the TiDB Cluster, migrate the data to TiDB first. For detailed information, see [data migration](#).
- If the TiDB cluster already has the replicated data from upstream, after TiFlash is deployed, data replication does not automatically begin. You need to manually specify the tables to be replicated to TiFlash. For detailed information, see [Use TiFlash](#).

2.4.5 Data processing

With TiDB, you can simply enter SQL statements for query or write requests. For the tables with TiFlash replicas, TiDB uses the front-end optimizer to automatically choose the optimal execution plan.

Note:

The MPP mode of TiFlash is enabled by default. When an SQL statement is executed, TiDB automatically determines whether to run in the MPP mode through the optimizer.

- To disable the MPP mode of TiFlash, set the value of the `tidb_allow_mpp` system variable to `OFF`.
- To forcibly enable MPP mode of TiFlash for query execution, set the values of `tidb_allow_mpp` and `tidb_enforce_mpp` to `ON`.
- To check whether TiDB chooses the MPP mode to execute a specific query, see [Explain Statements in the MPP Mode](#). If the output of `EXPLAIN` statement includes the `ExchangeSender` and `ExchangeReceiver` operators, the MPP mode is in use.

2.4.6 Performance monitoring

When using TiDB, you can monitor the TiDB cluster status and performance metrics in either of the following ways:

- **TiDB Dashboard**: you can see the overall running status of the TiDB cluster, analyse distribution and trends of read and write traffic, and learn the detailed execution information of slow queries.
- **Monitoring system (Prometheus & Grafana)**: you can see the monitoring parameters of TiDB cluster-related components including PD, TiDB, TiKV, TiFlash, TiCDC, and Node_exporter.

To see the alert rules of TiDB cluster and TiFlash cluster, see [TiDB cluster alert rules](#) and [TiFlash alert rules](#).

2.4.7 Troubleshooting

If any issue occurs during using TiDB, refer to the following documents:

- [Analyze slow queries](#)
- [Identify expensive queries](#)
- [Troubleshoot hotspot issues](#)
- [TiDB cluster troubleshooting guide](#)
- [Troubleshoot a TiFlash Cluster](#)

You are also welcome to create [Github Issues](#) or submit your questions on [AskTUG](#).

2.4.8 What's next

- To check the TiFlash version, critical logs, system tables, see [Maintain a TiFlash cluster](#).
- To remove a specific TiFlash node, see [Scale out a TiFlash cluster](#).

2.5 Import Example Database

Examples used in the TiDB manual use [System Data](#) from Capital Bikeshare, released under the [Capital Bikeshare Data License Agreement](#).

2.5.1 Download all data files

The system data is available [for download in .zip files](#) organized per year. Downloading and extracting all files requires approximately 3GB of disk space. To download all files for years 2010-2017 using a bash script:

```
mkdir -p bikeshare-data && cd bikeshare-data

curl -L --remote-name-all https://s3.amazonaws.com/capitalbikeshare-data
    ↳ /{2010..2017}-capitalbikeshare-tripdata.zip
unzip \*-tripdata.zip
```

2.5.2 Load data into TiDB

The system data can be imported into TiDB using the following schema:

```
CREATE DATABASE bikeshare;
USE bikeshare;

CREATE TABLE trips (
    trip_id bigint NOT NULL PRIMARY KEY AUTO_INCREMENT,
    duration integer not null,
    start_date datetime,
    end_date datetime,
    start_station_number integer,
    start_station varchar(255),
    end_station_number integer,
    end_station varchar(255),
    bike_number varchar(255),
    member_type varchar(255)
);
```

You can import files individually using the example `LOAD DATA` command here, or import all files using the bash loop below:

```
SET tidb_dml_batch_size = 20000;
LOAD DATA LOCAL INFILE '2017Q1-capitalbikeshare-tripdata.csv' INTO TABLE
    ↳ trips
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
```

```
IGNORE 1 LINES
(duration, start_date, end_date, start_station_number, start_station,
end_station_number, end_station, bike_number, member_type);
```

2.5.2.1 Import all files

Note:

When you start the MySQL client, use the `--local-infile=1` option.

To import all `*.csv` files into TiDB in a bash loop:

```
for FILE in *.csv; do
  echo "== $FILE =="
  mysql bikeshare --local-infile=1 -e "SET tidb_dml_batch_size = 20000; LOAD
    ↪ DATA LOCAL INFILE '${FILE}' INTO TABLE trips FIELDS TERMINATED BY
    ↪ ',' ENCLOSED BY '\"' LINES TERMINATED BY '\r\n' IGNORE 1 LINES (
    ↪ duration, start_date, end_date, start_station_number, start_station,
    ↪ end_station_number, end_station, bike_number, member_type);"
done;
```

3 Deploy

3.1 Software and Hardware Recommendations

As an open source distributed NewSQL database with high performance, TiDB can be deployed in the Intel architecture server, ARM architecture server, and major virtualization environments and runs well. TiDB supports most of the major hardware networks and Linux operating systems.

3.1.1 Linux OS version requirements

Linux OS Platform	Version
Red Hat Enterprise Linux	7.3 or later 7.x releases
CentOS	7.3 or later 7.x releases
Oracle Enterprise Linux	7.3 or later 7.x releases
Ubuntu LTS	16.04 or later

Note:

- For Oracle Enterprise Linux, TiDB supports the Red Hat Compatible Kernel (RHCK) and does not support the Unbreakable Enterprise Kernel provided by Oracle Enterprise Linux.
- A large number of TiDB tests have been run on the CentOS 7.3 system, and in our community there are a lot of best practices in which TiDB is deployed on the Linux operating system. Therefore, it is recommended to deploy TiDB on CentOS 7.3 or later.
- The support for the Linux operating systems above includes the deployment and operation in physical servers as well as in major virtualized environments like VMware, KVM and XEN.
- Red Hat Enterprise Linux 8.0, CentOS 8 Stream, and Oracle Enterprise Linux 8.0 are not supported yet as the testing of these platforms is in progress.
- Support for CentOS 8 Linux is not planned because its upstream support ends on December 31, 2021.
- Support for Ubuntu 16.04 will be removed in future versions of TiDB. Upgrading to Ubuntu 18.04 or later is strongly recommended.

Other Linux OS versions such as Debian Linux and Fedora Linux might work but are not officially supported.

3.1.2 Software recommendations

3.1.2.1 Control machine

Software	Version
sshpass	1.06 or later
TiUP	1.5.0 or later

Note:

It is required that you **deploy TiUP on the control machine** to operate and manage TiDB clusters.

3.1.2.2 Target machines

Software	Version
sshpss	1.06 or later
numa	2.0.12 or later
tar	any

3.1.3 Server recommendations

You can deploy and run TiDB on the 64-bit generic hardware server platform in the Intel x86-64 architecture or on the hardware server platform in the ARM architecture. The requirements and recommendations about server hardware configuration (ignoring the resources occupied by the operating system itself) for development, test, and production environments are as follows:

3.1.3.1 Development and test environments

Component	CPU	Memory	Local Storage	Network	Instance Number (Minimum Requirement)
TiDB	8 core+	16 GB+	No special requirements	Gigabit network card	1 (can be deployed on the same machine with PD)
PD	4 core+	8 GB+	SAS, 200 GB+	Gigabit network card	1 (can be deployed on the same machine with TiDB)
TiKV	8 core+	32 GB+	SAS, 200 GB+	Gigabit network card	3
TiFlash	32 core+	64 GB+	SSD, 200 GB+	Gigabit network card	1
TiCDC	8 core+	16 GB+	SAS, 200 GB+	Gigabit network card	1

Note:

- In the test environment, the TiDB and PD instances can be deployed on the same server.

- For performance-related test, do not use low-performance storage and network hardware configuration, in order to guarantee the correctness of the test result.
- For the TiKV server, it is recommended to use NVMe SSDs to ensure faster reads and writes.
- If you only want to test and verify the features, follow [Quick Start Guide for TiDB](#) to deploy TiDB on a single machine.
- The TiDB server uses the disk to store server logs, so there are no special requirements for the disk type and capacity in the test environment.

3.1.3.2 Production environment

Component	CPU	Memory	Hard Disk Type	Network	Instance N
TiDB	16 core+	32 GB+	SAS	10 Gigabit network card (2 preferred)	
PD	4 core+	8 GB+	SSD	10 Gigabit network card (2 preferred)	
TiKV	16 core+	32 GB+	SSD	10 Gigabit network card (2 preferred)	
TiFlash	48 core+	128 GB+	1 or more SSDs	10 Gigabit network card (2 preferred)	
TiCDC	16 core+	64 GB+	SSD	10 Gigabit network card (2 preferred)	
Monitor	8 core+	16 GB+	SAS	Gigabit network card	

Note:

- In the production environment, the TiDB and PD instances can be deployed on the same server. If you have a higher requirement for performance and reliability, try to deploy them separately.
- It is strongly recommended to use higher configuration in the production environment.
- It is recommended to keep the size of TiKV hard disk within 2 TB if you are using PCIe SSDs or within 1.5 TB if you are using regular SSDs.

Before you deploy TiFlash, note the following items:

- TiFlash can be [deployed on multiple disks](#).
- It is recommended to use a high-performance SSD as the first disk of the TiFlash data directory to buffer the real-time replication of TiKV data. The performance of this disk should not be lower than that of TiKV, such as PCI-E SSD. The disk capacity should be no less than 10% of the total capacity; otherwise, it might become the bottleneck

of this node. You can deploy ordinary SSDs for other disks, but note that a better PCI-E SSD brings better performance.

- It is recommended to deploy TiFlash on different nodes from TiKV. If you must deploy TiFlash and TiKV on the same node, increase the number of CPU cores and memory, and try to deploy TiFlash and TiKV on different disks to avoid interfering each other.
- The total capacity of the TiFlash disks is calculated in this way: $\frac{\text{the data volume of the entire TiKV cluster to be replicated}}{\text{the number of TiKV replicas}} * \text{the number of TiFlash replicas}$. For example, if the overall planned capacity of TiKV is 1 TB, the number of TiKV replicas is 3, and the number of TiFlash replicas is 2, then the recommended total capacity of TiFlash is $1024 \text{ GB} / 3 * 2$. You can replicate only the data of some tables. In such case, determine the TiFlash capacity according to the data volume of the tables to be replicated.

Before you deploy TiCDC, note that it is recommended to deploy TiCDC on PCIe-SSD disks larger than 1 TB.

3.1.4 Network requirements

As an open source distributed NewSQL database, TiDB requires the following network port configuration to run. Based on the TiDB deployment in actual environments, the administrator can open relevant ports in the network side and host side.

Component	Port	Description	Default
TiDB	4000	the communication port for the application and DBA tools	the communication port for the application and DBA tools

Component	Port	Description
TiDB	10080	the communication port to re-port TiDB status
TiKV	20160	the TiKV communication port
TiKV	20180	the communication port to re-port TiKV status
PD	2379	the communication port between TiDB and PD

Component	Port	Description
PD	2380	the inter-node communication port within the PD cluster
TiFlash	9000	the Ti-Flash TCP service port
TiFlash	8123	the Ti-Flash HTTP service port
TiFlash	3930	the Ti-Flash RAFT and Co-processor service port

Component	Port	Description
TiFlash	20170	the Ti- Flash Proxy ser- vice port
TiFlash	20292	the port for Prometheus to pull Ti- Flash Proxy met- rics
TiFlash	8234	the port for Prometheus to pull Ti- Flash met- rics
Pump	8250	the Pump com- mu- nica- tion port
Drainer	8249	the Drainer com- mu- nica- tion port

Component	Port	Description
TiCDC	8300	the communication port for the Prometheus service
Prometheus	9090	the communication port for the Prometheus service
Node_exporter	9100	the communication port for the system information of every TiDB cluster node

Component	Port	Description
Blackbox Exporter	9115	Blackbox_exporter communicates with the TiDB cluster to monitor the ports in the Grafana3000 port for the external Web monitoring service and client (Browser) access.
Grafana	3000	the Grafana component provides a web-based interface for monitoring the TiDB cluster.

Component	Port	Description
Alertmanager	9093	the port for the alert web service
Alertmanager	9094	the alert communication port

3.1.5 Web browser requirements

TiDB relies on [Grafana](#) to provide visualization of database metrics. A recent version of Internet Explorer, Chrome or Firefox with Javascript enabled is sufficient.

3.2 TiDB Environment and System Configuration Check

This document describes the environment check operations before deploying TiDB. The following steps are ordered by priorities.

3.2.1 Mount the data disk ext4 filesystem with options on the target machines that deploy TiKV

For production deployments, it is recommended to use NVMe SSD of EXT4 filesystem to store TiKV data. This configuration is the best practice, whose reliability, security, and stability have been proven in a large number of online scenarios.

Log in to the target machines using the `root` user account.

Format your data disks to the ext4 filesystem and add the `nodelalloc` and `noatime` mount options to the filesystem. It is required to add the `nodelalloc` option, or else the TiUP deployment cannot pass the precheck. The `noatime` option is optional.

Note:

If your data disks have been formatted to ext4 and have added the mount options, you can uninstall it by running the `umount /dev/nvme0n1p1` command, skip directly to the fifth step below to edit the `/etc/fstab` file, and add the options again to the filesystem.

Take the `/dev/nvme0n1` data disk as an example:

1. View the data disk.

```
fdisk -l
```

```
Disk /dev/nvme0n1: 1000 GB
```

2. Create the partition.

```
parted -s -a optimal /dev/nvme0n1 mklabel gpt -- mkpart primary ext4 1
→ -1
```

Note:

Use the `lsblk` command to view the device number of the partition: for a NVMe disk, the generated device number is usually `nvme0n1p1`; for a regular disk (for example, `/dev/sdb`), the generated device number is usually `sdb1`.

3. Format the data disk to the ext4 filesystem.

```
mkfs.ext4 /dev/nvme0n1p1
```

4. View the partition UUID of the data disk.

In this example, the UUID of `nvme0n1p1` is `c51eb23b-195c-4061-92a9-3fad812cc12f`.

```
lsblk -f
```

NAME	FSTYPE	LABEL	UUID	MOUNTPOINT
sda				
-sda1	ext4		237b634b-a565-477b-8371-6dff0c41f5ab	/boot
-sda2	swap		f414c5c0-f823-4bb1-8fdf-e531173a72ed	
-sda3	ext4		547909c1-398d-4696-94c6-03e43e317b60	/

```

sr0
nvme0n1
└─ nvme0n1p1 ext4      c51eb23b-195c-4061-92a9-3fad812cc12f

```

5. Edit the `/etc/fstab` file and add the `nodelalloc` mount options.

```
vi /etc/fstab
```

```

UUID=c51eb23b-195c-4061-92a9-3fad812cc12f /data1 ext4 defaults,
    ↪ nodelalloc,noatime 0 2

```

6. Mount the data disk.

```
mkdir /data1 && \
mount -a
```

7. Check using the following command.

```
mount -t ext4
```

```
/dev/nvme0n1p1 on /data1 type ext4 (rw,noatime,nodelalloc,data=ordered)
```

If the filesystem is ext4 and `nodelalloc` is included in the mount options, you have successfully mounted the data disk ext4 filesystem with options on the target machines.

3.2.2 Check and disable system swap

TiDB needs sufficient memory space for operation. When memory is insufficient, using swap as a buffer might degrade performance. Therefore, it is recommended to disable the system swap permanently by executing the following commands:

```
echo "vm.swappiness = 0">>> /etc/sysctl.conf
swapoff -a && swapon -a
sysctl -p
```

Note:

- Executing `swapoff -a` and then `swapon -a` is to refresh swap by dumping data to memory and cleaning up swap. If you drop the `swappiness` change and execute only `swapoff -a`, swap will be enabled again after you restart the system.
- `sysctl -p` is to make the configuration effective without restarting the system.

3.2.3 Check and stop the firewall service of target machines

In TiDB clusters, the access ports between nodes must be open to ensure the transmission of information such as read and write requests and data heartbeats. In common online scenarios, the data interaction between the database and the application service and between the database nodes are all made within a secure network. Therefore, if there are no special security requirements, it is recommended to stop the firewall of the target machine. Otherwise, refer to [the port usage](#) and add the needed port information to the allowlist of the firewall service.

The rest of this section describes how to stop the firewall service of a target machine.

1. Check the firewall status. Take CentOS Linux release 7.7.1908 (Core) as an example.

```
sudo firewall-cmd --state  
sudo systemctl status firewalld.service
```

2. Stop the firewall service.

```
sudo systemctl stop firewalld.service
```

3. Disable automatic start of the firewall service.

```
sudo systemctl disable firewalld.service
```

4. Check the firewall status.

```
sudo systemctl status firewalld.service
```

3.2.4 Check and install the NTP service

TiDB is a distributed database system that requires clock synchronization between nodes to guarantee linear consistency of transactions in the ACID model.

At present, the common solution to clock synchronization is to use the Network Time Protocol (NTP) services. You can use the pool.ntp.org timing service on the Internet, or build your own NTP service in an offline environment.

To check whether the NTP service is installed and whether it synchronizes with the NTP server normally, take the following steps:

1. Run the following command. If it returns `running`, then the NTP service is running.

```
sudo systemctl status ntpd.service
```

```
ntp.service - Network Time Service
Loaded: loaded (/usr/lib/systemd/system/ntp.service; disabled; vendor
        → preset: disabled)
Active: active (running) since — 2017-12-18 13:13:19 CST; 3s ago
```

- If it returns `Unit ntp.service could not be found.`, then try the following command to see whether your system is configured to use `chronyd` instead of `ntpd` to perform clock synchronization with NTP:

```
sudo systemctl status chronyd.service
```

```
chronyd.service - NTP client/server
Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled;
        → vendor preset: enabled)
Active: active (running) since Mon 2021-04-05 09:55:29 EDT; 3 days
        → ago
```

If the result shows that neither `chronyd` nor `ntpd` is configured, it means that neither of them is installed in your system. You should first install `chronyd` or `ntpd` and ensure that it can be automatically started. By default, `ntpd` is used.

If your system is configured to use `chronyd`, proceed to step 3.

2. Run the `ntpstat` command to check whether the NTP service synchronizes with the NTP server.

Note:

For the Ubuntu system, you need to install the `ntpstat` package.

```
ntpstat
```

- If it returns `synchronised to NTP server` (synchronizing with the NTP server), then the synchronization process is normal.

```
synchronised to NTP server (85.199.214.101) at stratum 2
time correct to within 91 ms
polling server every 1024 s
```

- The following situation indicates the NTP service is not synchronizing normally:
`unsynchronised`
- The following situation indicates the NTP service is not running normally:

Unable to talk to NTP daemon. Is it running?

3. Run the `chronyc tracking` command to check whether the Chrony service synchronizes with the NTP server.

Note:

This only applies to systems that use Chrony instead of NTPd.

`chronyc tracking`

- If the command returns `Leap status : Normal`, the synchronization process is normal.

```
Reference ID   : 5EC69F0A (ntp1.time.nl)
Stratum        : 2
Ref time (UTC) : Thu May 20 15:19:08 2021
System time    : 0.000022151 seconds slow of NTP time
Last offset    : -0.000041040 seconds
RMS offset     : 0.000053422 seconds
Frequency      : 2.286 ppm slow
Residual freq  : -0.000 ppm
Skew           : 0.012 ppm
Root delay     : 0.012706812 seconds
Root dispersion : 0.000430042 seconds
Update interval : 1029.8 seconds
Leap status    : Normal
```

- If the command returns the following result, an error occurs in the synchronization:

`Leap status : Not synchronised`

- If the command returns the following result, the `chronyd` service is not running normally:

`506 Cannot talk to daemon`

To make the NTP service start synchronizing as soon as possible, run the following command. Replace `pool.ntp.org` with your NTP server.

```
sudo systemctl stop ntpd.service && \
sudo ntpdate pool.ntp.org && \
sudo systemctl start ntpd.service
```

To install the NTP service manually on the CentOS 7 system, run the following command:

```
sudo yum install ntp ntpdate && \
sudo systemctl start ntpd.service && \
sudo systemctl enable ntpd.service
```

3.2.5 Check and configure the optimal parameters of the operating system

For TiDB in the production environment, it is recommended to optimize the operating system configuration in the following ways:

1. Disable THP (Transparent Huge Pages). The memory access pattern of databases tends to be sparse rather than consecutive. If the high-level memory fragmentation is serious, higher latency will occur when THP pages are allocated.
2. Set the I/O Scheduler of the storage media to `noop`. For the high-speed SSD storage media, the kernel's I/O scheduling operations can cause performance loss. After the Scheduler is set to `noop`, the performance is better because the kernel directly sends I/O requests to the hardware without other operations. Also, the `noop` Scheduler is better applicable.
3. Choose the `performance` mode for the `cpufreq` module which controls the CPU frequency. The performance is maximized when the CPU frequency is fixed at its highest supported operating frequency without dynamic adjustment.

Take the following steps to check the current operating system configuration and configure optimal parameters:

1. Execute the following command to see whether THP is enabled or disabled:

```
cat /sys/kernel/mm/transparent_hugepage/enabled
```

```
[always] madvise never
```

Note:

If `[always] madvise never` is output, THP is enabled. You need to disable it.

2. Execute the following command to see the I/O Scheduler of the disk where the data directory is located. Assume that you create data directories on both sdb and sdc disks:

```
cat /sys/block/sd[bc]/queue/scheduler
```

```
noop [deadline] cfq
noop [deadline] cfq
```

Note:

If `noop [deadline] cfq` is output, the I/O Scheduler for the disk is in the `deadline` mode. You need to change it to `noop`.

3. Execute the following command to see the `ID_SERIAL` of the disk:

```
udevadm info --name=/dev/sdb | grep ID_SERIAL
```

```
E: ID_SERIAL=36d0946606d79f90025f3e09a0c1f9e81
E: ID_SERIAL_SHORT=6d0946606d79f90025f3e09a0c1f9e81
```

Note:

If multiple disks are allocated with data directories, you need to execute the above command several times to record the `ID_SERIAL` of each disk.

4. Execute the following command to see the power policy of the cpufreq module:

```
cpupower frequency-info --policy
```

```
analyzing CPU 0:
current policy: frequency should be within 1.20 GHz and 3.10 GHz.
      The governor "powersave" may decide which speed to use
      ↪ within this range.
```

Note:

If The governor "powersave" is output, the power policy of the cpufreq module is powersave. You need to modify it to performance. If you use a virtual machine or a cloud host, the output is usually `Unable to determine current policy`, and you do not need to change anything.

5. Configure optimal parameters of the operating system:

- Method one: Use tuned (Recommended)

1. Execute the `tuned-adm list` command to see the tuned profile of the current operating system:

```
tuned-adm list
```

Available profiles: - balanced ↳ profile - desktop - hpc-compute - latency-performance ↳ performance at the cost of increased power consumption - network-latency ↳ performance at the cost of increased power consumption, ↳ focused on low latency network performance - network-throughput ↳ throughput, generally only necessary on older CPUs or 40G ↳ + networks - powersave - throughput-performance ↳ provides excellent performance across a variety of common ↳ server workloads - virtual-guest ↳ virtual guest - virtual-host Current active profile: balanced	- General non-specialized tuned - Optimize for the desktop use-case - Optimize for HPC compute workloads - Optimize for deterministic - Optimize for deterministic - Optimize for streaming network - Optimize for low power consumption - Broadly applicable tuning that - Optimizes for running inside a - Optimizes for running KVM guests
---	--

The output `Current active profile: balanced` means that the tuned profile of the current operating system is **balanced**. It is recommended to optimize the configuration of the operating system based on the current profile.

2. Create a new tuned profile:

```
mkdir /etc/tuned/balanced-tidb-optimal/
vi /etc/tuned/balanced-tidb-optimal/tuned.conf
```

```
[main]
include=balanced

[cpu]
governor=performance

[vm]
transparent_hugepages=never

[disk]
devices_udev_regex=(ID_SERIAL=36d0946606d79f90025f3e09a0c1fc035
                   ↳ )|(ID_SERIAL=36d0946606d79f90025f3e09a0c1f9e81)
```

```
elevator=noop
```

The output `include=balanced` means to add the optimization configuration of the operating system to the current `balanced` profile.

3. Apply the new tuned profile:

```
tuned-adm profile balanced-tidb-optimal
```

- Method two: Configure using scripts. Skip this method if you already use method one.

1. Execute the `grubby` command to see the default kernel version:

Note:

Install the `grubby` package first before you execute `grubby`.

```
grubby --default-kernel
```

```
/boot/vmlinuz-3.10.0-957.el7.x86_64
```

2. Execute `grubby --update-kernel` to modify the kernel configuration:

```
grubby --args="transparent_hugepage=never" --update-kernel /  
→ boot/vmlinuz-3.10.0-957.el7.x86_64
```

Note:

`--update-kernel` is followed by the actual default kernel version.

3. Execute `grubby --info` to see the modified default kernel configuration:

```
grubby --info /boot/vmlinuz-3.10.0-957.el7.x86_64
```

Note:

`--info` is followed by the actual default kernel version.

```
index=0  
kernel=/boot/vmlinuz-3.10.0-957.el7.x86_64  
args="ro crashkernel=auto rd.lvm.lv=centos/root rd.lvm.lv=  
→ centos/swap rhgb quiet LANG=en_US.UTF-8  
→ transparent_hugepage=never"  
root=/dev/mapper/centos-root  
initrd=/boot/initramfs-3.10.0-957.el7.x86_64.img  
title=CentOS Linux (3.10.0-957.el7.x86_64) 7 (Core)
```

4. Modify the current kernel configuration to immediately disable THP:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

5. Configure the I/O Scheduler in the udev script:

```
vi /etc/udev/rules.d/60-tidb-schedulers.rules
```

```
ACTION=="add|change", SUBSYSTEM=="block", ENV{ID_SERIAL}=="36
    ↪ d0946606d79f90025f3e09a0c1fc035", ATTR{queue/scheduler}="
        ↪ noop"
ACTION=="add|change", SUBSYSTEM=="block", ENV{ID_SERIAL}=="36
    ↪ d0946606d79f90025f3e09a0c1f9e81", ATTR{queue/scheduler}="
        ↪ noop"
```

6. Apply the udev script:

```
udevadm control --reload-rules
udevadm trigger --type=devices --action=change
```

7. Create a service to configure the CPU power policy:

```
cat >> /etc/systemd/system/cpupower.service << EOF
[Unit]
Description=CPU performance
[Service]
Type=oneshot
ExecStart=/usr/bin/cpupower frequency-set --governor
    ↪ performance
[Install]
WantedBy=multi-user.target
EOF
```

8. Apply the CPU power policy configuration service:

```
systemctl daemon-reload
systemctl enable cpupower.service
systemctl start cpupower.service
```

6. Execute the following command to verify the THP status:

```
cat /sys/kernel/mm/transparent_hugepage/enabled
```

```
always madvise [never]
```

7. Execute the following command to verify the I/O Scheduler of the disk where the data directory is located:

```
cat /sys/block/sd[bc]/queue/scheduler
```

```
[noop] deadline cfq
[noop] deadline cfq
```

8. Execute the following command to see the power policy of the cpufreq module:

```
cpupower frequency-info --policy
````
```

analyzing CPU 0: current policy: frequency should be within 1.20 GHz and 3.10 GHz.  
The governor “performance” may decide which speed to use within this range. ““

9. Execute the following commands to modify the `sysctl` parameters:

```
echo "fs.file-max = 1000000">>> /etc/sysctl.conf
echo "net.core.somaxconn = 32768">>> /etc/sysctl.conf
echo "net.ipv4.tcp_tw_recycle = 0">>> /etc/sysctl.conf
echo "net.ipv4.tcp_syncookies = 0">>> /etc/sysctl.conf
echo "vm.overcommit_memory = 1">>> /etc/sysctl.conf
sysctl -p
```

10. Execute the following command to configure the user’s `limits.conf` file:

```
cat << EOF >>/etc/security/limits.conf
tidb soft nofile 1000000
tidb hard nofile 1000000
tidb soft stack 32768
tidb hard stack 32768
EOF
```

### 3.2.6 Manually configure the SSH mutual trust and sudo without password

This section describes how to manually configure the SSH mutual trust and sudo without password. It is recommended to use TiUP for deployment, which automatically configure SSH mutual trust and login without password. If you deploy TiDB clusters using TiUP, ignore this section.

1. Log in to the target machine respectively using the `root` user account, create the `tidb` user and set the login password.

```
useradd tidb && \
passwd tidb
```

2. To configure sudo without password, run the following command, and add `tidb ALL`  
 $\hookrightarrow$  `=ALL NOPASSWD: ALL` to the end of the file:

```
visudo
```

```
tidb ALL=(ALL) NOPASSWD: ALL
```

3. Use the `tidb` user to log in to the control machine, and run the following command. Replace `10.0.1.1` with the IP of your target machine, and enter the `tidb` user password of the target machine as prompted. After the command is executed, SSH mutual trust is already created. This applies to other machines as well. Newly created `tidb` users do not have the `.ssh` directory. To create such a directory, execute the command that generates the RSA key. To deploy TiDB components on the control machine, configure mutual trust for the control machine and the target machine itself.

```
ssh-keygen -t rsa
ssh-copy-id -i ~/.ssh/id_rsa.pub 10.0.1.1
```

4. Log in to the control machine using the `tidb` user account, and log in to the IP of the target machine using `ssh`. If you do not need to enter the password and can successfully log in, then the SSH mutual trust is successfully configured.

```
ssh 10.0.1.1
```

```
[tidb@10.0.1.1 ~]$
```

5. After you log in to the target machine using the `tidb` user, run the following command. If you do not need to enter the password and can switch to the `root` user, then sudo without password of the `tidb` user is successfully configured.

```
sudo -su root
```

```
[root@10.0.1.1 tidb]#
```

### 3.2.7 Install the `numactl` tool

This section describes how to install the NUMA tool. In online environments, because the hardware configuration is usually higher than required, to better plan the hardware resources, multiple instances of TiDB or TiKV can be deployed on a single machine. In such scenarios, you can use NUMA tools to prevent the competition for CPU resources which might cause reduced performance.

**Note:**

- Binding cores using NUMA is a method to isolate CPU resources and is suitable for deploying multiple instances on highly configured physical machines.
- After completing deployment using `tiup cluster deploy`, you can use the `exec` command to perform cluster level management operations.

1. Log in to the target node to install. Take CentOS Linux release 7.7.1908 (Core) as an example.

```
sudo yum -y install numactl
```

2. Run the `exec` command using `tiup cluster` to install in batches.

```
tiup cluster exec --help
```

```
Run shell command on host in the tidb cluster
Usage:
cluster exec <cluster-name> [flags]
Flags:
 --command string the command run on cluster host (default "ls")
 -h, --help help for exec
 --sudo use root permissions (default false)
```

To use the sudo privilege to execute the installation command for all the target machines in the `tidb-test` cluster, run the following command:

```
tiup cluster exec tidb-test --sudo --command "yum -y install numactl"
```

### 3.3 Plan Cluster Topology

#### 3.3.1 Minimal Deployment Topology

This document describes the minimal deployment topology of TiDB clusters.

##### 3.3.1.1 Topology information

---

|      | Instanc | Count | IP       | Configuration                                              |
|------|---------|-------|----------|------------------------------------------------------------|
| TiDB | 3       | 16    | 10.0.1.1 | Default<br>VCores 1<br>32GB<br>* 1                         |
| PD   | 3       | 4     | 10.0.1.1 | Default<br>VCores 1<br>8GB<br>* 1                          |
| TiKV | 3       | 16    | 10.0.1.1 | Default<br>VCores 1<br>32GB<br>2TB<br>(nvme<br>ssd)<br>* 1 |

Physical  
ma-  
chine  
con-  
fig-  
u-  
ra-

|  | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce count<br>Monit<br>oring &<br>Grafana | IP<br>10.0.1.10<br>VCores<br>8GB<br>* 1<br>500GB<br>(ssd) | Configura<br>tion<br>Default<br>port<br>Global<br>di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion |
|--|-------------------------------------------------------|---------------------------------------------------|-----------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
|  |                                                       |                                                   |                                                           |                                                                                                            |

### 3.3.1.1 Topology templates

- The simple template for the minimal topology
- The complex template for the minimal topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see [Topology Configuration File for Deploying TiDB Using TiUP](#).

#### Note:

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

### 3.3.2 TiFlash Deployment Topology

This document describes the deployment topology of [TiFlash](#) based on the minimal TiDB topology.

TiFlash is a columnar storage engine, and gradually becomes the standard cluster topology. It is suitable for real-time HTAP applications.

### 3.3.2.1 Topology information

|      |   | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan-<br>ce<br>Count     | IP                        | Configuration                                            |
|------|---|-------------------------------------------------------|----------------------------|---------------------------|----------------------------------------------------------|
| TiDB | 3 | 16<br>VCores<br>32GB                                  | 10.0.1<br>10.0.1<br>10.0.1 | Default<br>port<br>Global | di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion |
| PD   | 3 | 4<br>VCores<br>8GB                                    | 10.0.1<br>10.0.1<br>10.0.1 | Default<br>port<br>Global | di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion |

---

|                       | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce | Coun<br>t | IP    | Configura<br>tion                                   |
|-----------------------|-------------------------------------------------------|--------------|-----------|-------|-----------------------------------------------------|
| TiKV3                 | 16                                                    | 10.0.1.1     | Default   | VCore | 10.0.1.1<br>32GB<br>2TB<br>(nvme<br>ssd)<br>* 1     |
| TiFlash               | 32                                                    | 10.0.1.1     | Default   | VCore | 10.0.1.1<br>64<br>GB<br>2TB<br>(nvme<br>ssd)<br>* 1 |
| Monit<br>&<br>Grafana | 4                                                     | 10.0.1.1     | Default   | VCore | 10.0.1.1<br>8GB<br>* 1<br>500GB<br>(ssd)            |

---

### 3.3.2.1.1 Topology templates

- The simple template for the TiFlash topology
- The complex template for the TiFlash topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see [Topology Configuration File for Deploying TiDB Using TiUP](#).

### 3.3.2.1.2 Key parameters

- To enable the [Placement Rules](#) feature of PD, set the value of `replication.enable-placement-rules` in the configuration template to `true`.
- The instance level "`-host`" configuration in `tiflash_servers` only supports IP, not domain name.
- For detailed TiFlash parameter description, see [TiFlash Configuration](#).

#### Note:

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

### 3.3.3 TiCDC Deployment Topology

#### Note:

TiCDC is a feature for general availability (GA) since v4.0.6. You can use it in the production environment.

This document describes the deployment topology of [TiCDC](#) based on the minimal cluster topology.

TiCDC is a tool for replicating the incremental data of TiDB, introduced in TiDB 4.0. It supports multiple downstream platforms, such as TiDB, MySQL, and MQ. Compared with TiDB Binlog, TiCDC has lower latency and native high availability.

#### 3.3.3.1 Topology information

| Instance Type | Count | IP                                                  | Configuration                                                                                                 |
|---------------|-------|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| TiDB          | 3     | 16<br>VCores<br>32GB<br>* 1                         | 10.0.1.1<br>Default<br>Port<br>10.0.1.3<br>di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion           |
| PD            | 3     | 4<br>VCores<br>8GB<br>* 1                           | 10.0.1.1<br>Default<br>Port<br>10.0.1.5<br>di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion           |
| TiKV          | 3     | 16<br>VCores<br>32GB<br>2TB<br>(nvme<br>ssd)<br>* 1 | 10.0.1.1<br>Default<br>Port<br>10.0.1.9<br>Global<br>di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion |

|            | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce count | IP       | Configuration                                            |
|------------|-------------------------------------------------------|--------------------|----------|----------------------------------------------------------|
| CDC        | 3                                                     | 8                  | 10.0.1.1 | Default                                                  |
|            |                                                       |                    | VCore    | 10.0.1.1 port                                            |
|            |                                                       |                    | 16GB     | 10.0.1.1 Global                                          |
|            |                                                       | * 1                |          | di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion |
| Monitoring | 4                                                     |                    | 10.0.1.1 | Default                                                  |
| &          |                                                       |                    | VCore    | port                                                     |
| Grafana    |                                                       | 8GB                |          | Global                                                   |
|            |                                                       | * 1                |          | di-<br>rec-<br>tory<br>con-<br>fig-<br>u-<br>ra-<br>tion |
|            |                                                       |                    | 500GB    |                                                          |
|            |                                                       |                    | (ssd)    |                                                          |

### 3.3.3.1.1 Topology templates

- The simple template for the TiCDC topology
- The complex template for the TiCDC topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see [Topology Configuration File for Deploying TiDB Using TiUP](#).

**Note:**

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

### 3.3.4 TiDB Binlog Deployment Topology

This document describes the deployment topology of [TiDB Binlog](#) based on the minimal TiDB topology.

TiDB Binlog is the widely used component for replicating incremental data. It provides near real-time backup and replication.

#### 3.3.4.1 Topology information

|        | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan-<br>ce<br>Count                                                                                                                                                                              | IP                                                                                                                                                                                  | Configuration |
|--------|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| TiDB 3 | 16<br>VCores<br>32<br>GB                              | 10.0.1.1<br>10.0.1.2<br>10.0.1.3<br>10.0.1.4<br>10.0.1.5<br>10.0.1.6<br>10.0.1.7<br>10.0.1.8<br>10.0.1.9<br>10.0.1.10<br>10.0.1.11<br>10.0.1.12<br>10.0.1.13<br>10.0.1.14<br>10.0.1.15<br>10.0.1.16 | Default<br>port<br>3306<br>fig-<br>u-<br>ra-<br>tion;<br>En-<br>able<br><code>enable_binlog</code><br>→ ;<br>En-<br>able<br><code>ignore</code><br>→ -<br>→ <code>error</code><br>→ |               |
|        |                                                       |                                                                                                                                                                                                     |                                                                                                                                                                                     |               |

---

|       |   | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce count                           | IP                                                                                            | Configuration |
|-------|---|-------------------------------------------------------|----------------------------------------------|-----------------------------------------------------------------------------------------------|---------------|
| PD    | 3 | 4<br>VCores<br>8<br>GB                                | 10.0.1.1<br>10.0.1.2<br>10.0.1.3<br>10.0.1.4 | Default<br>Port<br>Con-<br>fig-<br>u-<br>ra-<br>tion                                          |               |
| TiKV3 |   | 16<br>VCores<br>32<br>GB                              | 10.0.1.1<br>10.0.1.2<br>10.0.1.3<br>10.0.1.4 | Default<br>Port<br>Con-<br>fig-<br>u-<br>ra-<br>tion                                          |               |
| Pump3 |   | 8<br>VCores<br>16GB                                   | 10.0.1.1<br>10.0.1.2<br>10.0.1.3<br>10.0.1.4 | Default<br>Port<br>Con-<br>fig-<br>u-<br>ra-<br>tion;<br>Set<br>GC<br>time<br>to<br>7<br>days |               |

---

|       | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce | Coun<br>t | IP                                                                                                                                                                                                                                                                                              | Configura<br>tion |
|-------|-------------------------------------------------------|--------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| Drain | dr                                                    | 8            | 10.0.1.12 | Default<br>port<br>con-<br>fig-<br>u-<br>ra-<br>tion;<br>Set<br>the<br>de-<br>fault<br>ini-<br>tial-<br>iza-<br>tion<br>com-<br>mitTS<br>-1<br>as<br>the<br>lat-<br>est<br>times-<br>tamp;<br>Con-<br>fig-<br>ure<br>the<br>down-<br>stream<br>tar-<br>get<br>TiDB<br>as<br>10.0.1.12:4000<br>→ |                   |

---

### 3.3.4.1.1 Topology templates

- The simple template for the TiDB Binlog topology (with `mysql` as the downstream type)
- The simple template for the TiDB Binlog topology (with `file` as the downstream type)
- The complex template for the TiDB Binlog topology

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see [Topology Configuration File for Deploying TiDB Using TiUP](#).

### 3.3.4.1.2 Key parameters

The key parameters in the topology configuration templates are as follows:

- `server_configs.tidb.binlog.enable: true`
  - Enables the binlog service.
  - Default value: `false`.
- `server_configs.tidb.binlog.ignore-error: true`
  - It is recommended to enable this configuration in high availability scenarios.
  - If set to `true`, when an error occurs, TiDB stops writing data into binlog, and adds 1 to the value of the `tidb_server_critical_error_total` monitoring metric.
  - If set to `false`, when TiDB fails to write data into binlog, the whole TiDB service is stopped.

- `drainer_servers.config.syncer.db-type`

The downstream type of TiDB Binlog. Currently, `mysql`, `tidb`, `kafka`, and `file` are supported.

- `drainer_servers.config.syncer.to`

The downstream configuration of TiDB Binlog. Depending on different `db-types`, you can use this configuration item to configure the connection parameters of the downstream database, the connection parameters of Kafka, and the file save path. For details, refer to [TiDB Binlog Configuration File](#).

#### Note:

- When editing the configuration file template, if you do not need custom ports or directories, modify the IP only.

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

### 3.3.5 TiSpark Deployment Topology

#### Warning:

TiSpark support in the TiUP cluster is still an experimental feature. It is **NOT** recommended to use it in the production environment.

This document introduces the TiSpark deployment topology and how to deploy TiSpark based on the minimum cluster topology.

TiSpark is a component built for running Apache Spark on top of TiDB/TiKV to answer complex OLAP queries. It brings benefits of both the Spark platform and the distributed TiKV cluster to TiDB and makes TiDB a one-stop solution for both online transactions and analytics.

For more information about the TiSpark architecture and how to use it, see [TiSpark User Guide](#).

#### 3.3.5.1 Topology information

| Instance | Count | IP                                                  | Configuration                                                                                                 |
|----------|-------|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| TiDB     | 3     | 16<br>VCores<br>32GB<br>* 1                         | 10.0.1.1<br>Default<br>Port<br>10.0.1.3<br>Global<br>di-<br>recto-<br>ry<br>con-<br>fig-<br>u-<br>ra-<br>tion |
| PD       | 3     | 4<br>VCores<br>8GB<br>* 1                           | 10.0.1.1<br>Default<br>Port<br>10.0.1.5<br>Global<br>di-<br>recto-<br>ry<br>con-<br>fig-<br>u-<br>ra-<br>tion |
| TiKV     | 3     | 16<br>VCores<br>32GB<br>2TB<br>(nvme<br>ssd)<br>* 1 | 10.0.1.1<br>Default<br>Port<br>10.0.1.9<br>Global<br>di-<br>recto-<br>ry<br>con-<br>fig-<br>u-<br>ra-<br>tion |

---

|            | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce count | IP                 | Configuration             |
|------------|-------------------------------------------------------|--------------------|--------------------|---------------------------|
| TiSpark    | 8                                                     | 10.0.1.1           | Default            |                           |
|            |                                                       | VCore              | (mas-<br>16GB ter) | port<br>Global            |
|            | * 1                                                   | 10.0.1.2           | (worker1)          | 10.0.1.23<br>(worker2)    |
|            |                                                       |                    |                    | fig-<br>u-<br>ra-<br>tion |
| Monitoring | 4                                                     | 10.0.1.1           | Default            |                           |
| &          |                                                       | VCore              |                    | port                      |
| Grafana    | 8GB                                                   |                    |                    | Global                    |
|            | * 1                                                   |                    |                    | di-                       |
|            | 500GB                                                 |                    |                    | rec-                      |
|            | (ssd)                                                 |                    |                    | tory                      |
|            |                                                       |                    |                    | con-                      |
|            |                                                       |                    |                    | fig-<br>u-<br>ra-<br>tion |

---

### 3.3.5.2 Topology templates

- Simple TiSpark topology template
- Complex TiSpark topology template

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see [Topology Configuration File for Deploying TiDB Using TiUP](#).

**Note:**

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

### 3.3.5.3 Prerequisites

TiSpark is based on the Apache Spark cluster, so before you start the TiDB cluster that contains TiSpark, you must ensure that Java Runtime Environment (JRE) 8 is installed on the server that deploys TiSpark. Otherwise, TiSpark cannot be started.

TiUP does not support installing JRE automatically. You need to install it on your own. For detailed installation instruction, see [How to download and install prebuilt OpenJDK packages](#).

If JRE 8 has already been installed on the deployment server but is not in the path of the system's default package management tool, you can specify the path of the JRE environment to be used by setting the `java_home` parameter in the topology configuration. This parameter corresponds to the `JAVA_HOME` system environment variable.

## 3.3.6 Geo-Distributed Deployment Topology

This document takes the typical architecture of three data centers (DC) in two cities as an example, and introduces the geo-distributed deployment architecture and the key configuration. The cities used in this example are Shanghai (referred to as `sha`) and Beijing (referred to as `bja` and `bjb`).

### 3.3.6.1 Topology information

---

|        |       | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | BJ        | SH        | Configuration                                            |
|--------|-------|-------------------------------------------------------|-----------|-----------|----------------------------------------------------------|
| Instan | Count | IP                                                    | IP        |           |                                                          |
| TiDB   | 5     | 16                                                    | 10.0.1.10 | 10.0.1.15 | Default<br>VCores<br>32GB<br>* 1                         |
| PD     | 5     | 4                                                     | 10.0.1.10 | 10.0.1.10 | Default<br>VCores<br>8GB<br>* 1                          |
| TiKV   | 5     | 16                                                    | 10.0.1.10 | 10.0.1.10 | Default<br>VCores<br>32GB<br>2TB<br>(nvme<br>ssd)<br>* 1 |

|         | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | BJ        | SH |               |
|---------|-------------------------------------------------------|-----------|----|---------------|
| Instan  | Count                                                 | IP        | IP | Configuration |
| Monit   | 4                                                     | 10.0.1.16 |    | Default       |
| ring &  |                                                       | VCore     |    | port          |
| Grafana | 8GB                                                   |           |    | Global        |
|         | * 1                                                   |           |    | di-           |
|         | 500GB                                                 |           |    | rec-          |
|         | (ssd)                                                 |           |    | tory          |
|         |                                                       |           |    | con-          |
|         |                                                       |           |    | fig-          |
|         |                                                       |           |    | u-            |
|         |                                                       |           |    | ra-           |
|         |                                                       |           |    | tion          |

### 3.3.6.1.1 Topology templates

- The geo-distributed topology template

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see [Topology Configuration File for Deploying TiDB Using TiUP](#).

### 3.3.6.1.2 Key parameters

This section describes the key parameter configuration of the TiDB geo-distributed deployment.

#### TiKV parameters

- The gRPC compression format (`none` by default):

To increase the transmission speed of gRPC packages between geo-distributed target nodes, set this parameter to `gzip`.

```
server.grpc-compression-type: gzip
```

- The label configuration:

Since TiKV is deployed across different data centers, if the physical machines go down, the Raft Group might lose three of the default five replicas, which causes the cluster

unavailability. To address this issue, you can configure the labels to enable the smart scheduling of PD, which ensures that the Raft Group does not allow three replicas to be located in TiKV instances on the same machine in the same cabinet of the same data center.

- The TiKV configuration:

The same host-level label information is configured for the same physical machine.

```
config:
 server.labels:
 zone: bj
 dc: bja
 rack: rack1
 host: host2
```

- To prevent remote TiKV nodes from launching unnecessary Raft elections, it is required to increase the minimum and maximum number of ticks that the remote TiKV nodes need to launch an election. The two parameters are set to 0 by default.

```
raftstore.raft-min-election-timeout-ticks: 1000
raftstore.raft-max-election-timeout-ticks: 1020
```

## PD parameters

- The PD metadata information records the topology of the TiKV cluster. PD schedules the Raft Group replicas on the following four dimensions:

```
replication.location-labels: ["zone", "dc", "rack", "host"]
```

- To ensure high availability of the cluster, adjust the number of Raft Group replicas to be 5:

```
replication.max-replicas: 5
```

- Forbid the remote TiKV Raft replica being elected as Leader:

```
label-property:
 reject-leader:
 - key: "dc"
 value: "sha"
```

### Note:

Since TiDB 5.2, the `label-property` configuration is not supported by default. To set the replica policy, use the [placement rules](#).

For the further information about labels and the number of Raft Group replicas, see [Schedule Replicas by Topology Labels](#).

**Note:**

- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

### 3.3.7 Hybrid Deployment Topology

This document describes the topology and key parameters of the TiKV and TiDB hybrid deployment.

The hybrid deployment is usually used in the following scenario:

The deployment machine has multiple CPU processors with sufficient memory. To improve the utilization rate of the physical machine resources, multiple instances can be deployed on a single machine, that is, TiDB and TiKV's CPU resources are isolated through NUMA node bindings. PD and Prometheus are deployed together, but their data directories need to use separate file systems.

#### 3.3.7.1 Topology information

---

| Instance | Count | IP        | Configuration                                       |
|----------|-------|-----------|-----------------------------------------------------|
| TiDB     | 6     | 10.0.1.10 | Configure<br>VCores<br>64GB<br>bind<br>CPU<br>cores |

---

|    | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce count | IP       | Configuration                                           |
|----|-------------------------------------------------------|--------------------|----------|---------------------------------------------------------|
| PD | 3                                                     | 16                 | 10.0.1.1 | Configure<br>VCores<br>32<br>GB                         |
|    |                                                       |                    | 10.0.1.1 | the<br>location_labels<br>→<br>pa-<br>ram-<br>e-<br>ter |

---

|        | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Count    | IP            | Configuration |
|--------|-------------------------------------------------------|----------|---------------|---------------|
| Instan | Ce                                                    | tion     |               |               |
| TiKV6  | 32                                                    | 10.0.1.7 | VCore10.0.1.8 | 64GB10.0.1.9  |

rate  
 the  
 instance-  
 level  
 port  
 and  
 sta-  
 tus\_port;  
 2.  
 Con-  
 fig-  
 ure  
 the  
 global  
 pa-  
 ram-  
 e-  
 ters  
**readpool**  
 ↪ ,  
**storage**  
 ↪  
 and  
**raftstore**  
 ↪ ;  
 3.  
 Con-  
 fig-  
 ure  
 la-  
 bels  
 of  
 the  
 instance-  
 level  
 host;  
 4.  
 Con-  
 f

|                       | Physical<br>ma-<br>chine<br>con-<br>fig-<br>u-<br>ra- | Instan<br>ce count                                        | IP | Configuration |
|-----------------------|-------------------------------------------------------|-----------------------------------------------------------|----|---------------|
| Monit<br>&<br>Grafana | 4<br>VCores<br>8GB<br>* 1<br>500GB<br>(ssd)           | 10.0.1.10<br>Default<br>con-<br>fig-<br>u-<br>ra-<br>tion |    |               |
|                       |                                                       |                                                           |    |               |

### 3.3.7.1.1 Topology templates

- The simple template for the hybrid deployment
- The complex template for the hybrid deployment

For detailed descriptions of the configuration items in the above TiDB cluster topology file, see [Topology Configuration File for Deploying TiDB Using TiUP](#).

### 3.3.7.1.2 Key parameters

This section introduces the key parameters when you deploy multiple instances on a single machine, which is mainly used in scenarios when multiple instances of TiDB and TiKV are deployed on a single machine. You need to fill in the results into the configuration template according to the calculation methods provided below.

- Optimize the configuration of TiKV
  - To configure `readpool` to be self-adaptive to the thread pool. By configuring the `readpool.unified.max-thread-count` parameter, you can make `readpool.storage` and `readpool.coprocessor` share a unified thread pool, and set the self-adaptive switch respectively.

\* Enable `readpool.storage` and `readpool.coprocessor`:

```
readpool.storage.use-unified-pool: true
readpool.coprocessor.use-unified-pool: true
```

\* The calculation method:

```
readpool.unified.max-thread-count = cores * 0.8 / the number of
 ↪ TiKV instances
```

- To configure the storage CF (all RocksDB column families) to be self-adaptive to memory. By configuring the `storage.block-cache.capacity` parameter, you can make CF automatically balance the memory usage.

- \* `storage.block-cache` enables the CF self-adaptation by default. You do not need to modify it.

```
storage.block-cache.shared: true
```

- \* The calculation method:

```
storage.block-cache.capacity = (MEM_TOTAL * 0.5 / the number of
 ↪ TiKV instances)
```

- If multiple TiKV instances are deployed on the same physical disk, add the `capacity` parameter in the TiKV configuration:

```
raftstore.capacity = disk total capacity / the number of TiKV
 ↪ instances
```

- The label scheduling configuration

Since multiple instances of TiKV are deployed on a single machine, if the physical machines go down, the Raft Group might lose two of the default three replicas, which causes the cluster unavailability. To address this issue, you can use the label to enable the smart scheduling of PD, which ensures that the Raft Group has more than two replicas in multiple TiKV instances on the same machine.

- The TiKV configuration

The same host-level label information is configured for the same physical machine:

```
config:
 server.labels:
 host: tikv1
```

- The PD configuration

To enable PD to identify and scheduling Regions, configure the `labels` type for PD:

```
pd:
 replication.location-labels: ["host"]
```

- `numa_node` core binding

- In the instance parameter module, configure the corresponding `numa_node` parameter and add the number of CPU cores.
- Before using NUMA to bind cores, make sure that the numactl tool is installed, and confirm the information of CPUs in the physical machines. After that, configure the parameters.
- The `numa_node` parameter corresponds to the `numactl --membind` configuration.

**Note:**

- When editing the configuration file template, modify the required parameter, IP, port, and directory.
- Each component uses the global `<deploy_dir>/<components_name>-<→ port>` as their `deploy_dir` by default. For example, if TiDB specifies the 4001 port, its `deploy_dir` is `/tidb-deploy/tidb-4001` by default. Therefore, in multi-instance scenarios, when specifying a non-default port, you do not need to specify the directory again.
- You do not need to manually create the `tidb` user in the configuration file. The TiUP cluster component automatically creates the `tidb` user on the target machines. You can customize the user, or keep the user consistent with the control machine.
- If you configure the deployment directory as a relative path, the cluster will be deployed in the home directory of the user.

## 3.4 Install and Start

### 3.4.1 Deploy a TiDB Cluster Using TiUP

TiUP is a cluster operation and maintenance tool introduced in TiDB 4.0. TiUP provides [TiUP cluster](#), a cluster management component written in Golang. By using TiUP cluster, you can easily perform daily database operations, including deploying, starting, stopping, destroying, scaling, and upgrading a TiDB cluster, and manage TiDB cluster parameters.

TiUP supports deploying TiDB, TiFlash, TiDB Binlog, TiCDC, and the monitoring system. This document introduces how to deploy TiDB clusters of different topologies.

**Note:**

TiDB, TiUP and TiDB Dashboard share usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see [Telemetry](#).

### 3.4.1.1 Step 1: Prerequisites and precheck

Make sure that you have read the following documents:

- [Hardware and software requirements](#)
- [Environment and system configuration check](#)

### 3.4.1.2 Step 2: Install TiUP on the control machine

You can install TiUP on the control machine in either of the two ways: online deployment and offline deployment.

#### 3.4.1.2.1 Method 1: Deploy TiUP online

Log in to the control machine using a regular user account (take the `tidb` user as an example). All the following TiUP installation and cluster management operations can be performed by the `tidb` user.

1. Install TiUP by executing the following command:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
↪ install.sh | sh
```

2. Set the TiUP environment variables:

Redeclare the global environment variables:

```
source .bash_profile
```

Confirm whether TiUP is installed:

```
which tiup
```

3. Install the TiUP cluster component:

```
tiup cluster
```

4. If TiUP is already installed, update the TiUP cluster component to the latest version:

```
tiup update --self && tiup update cluster
```

Expected output includes “Update successfully!”.

5. Verify the current version of your TiUP cluster:

```
tiup --binary cluster
```

### 3.4.1.2.2 Method 2: Deploy TiUP offline

Perform the following steps in this section to deploy a TiDB cluster offline using TiUP:

Step 1: Prepare the TiUP offline component package

To prepare the TiUP offline component package, manually pack an offline component package using `tiup mirror clone`.

1. Install the TiUP package manager online.

1. Install the TiUP tool:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.
 ↪ com/install.sh | sh
```

2. Redeclare the global environment variables:

```
source .bash_profile
```

3. Confirm whether TiUP is installed:

```
which tiup
```

2. Pull the mirror using TiUP.

1. Pull the needed components on a machine that has access to the Internet:

```
tiup mirror clone tidb-community-server-{$version}-linux-amd64 ${
 ↪ version} --os=linux --arch=amd64
```

The command above creates a directory named `tidb-community-server-{$version}-linux-amd64` in the current directory, which contains the component package necessary for starting a cluster.

2. Pack the component package by using the `tar` command and send the package to the control machine in the isolated environment:

```
tar czvf tidb-community-server-{$version}-linux-amd64.tar.gz tidb-
 ↪ community-server-{$version}-linux-amd64
```

`tidb-community-server-{$version}-linux-amd64.tar.gz` is an independent offline environment package.

3. Customize the offline mirror, or adjust the contents of an existing offline mirror.

If you want to adjust an existing offline mirror (such as adding a new version of a component), take the following steps:

- When pulling an offline mirror, you can get an incomplete offline mirror by specifying specific information via parameters, such as the component and version information. For example, you can pull an offline mirror that includes only the offline mirror of TiUP v1.7.0 and TiUP Cluster v1.7.0 by running the following command:

```
tiup mirror clone tiup-custom-mirror-v1.7.0 --tiup v1.7.0 --cluster
→ v1.7.0
```

If you only need the components for a particular platform, you can specify them using the `--os` or `--arch` parameters.

- Refer to the step 2 of “Pull the mirror using TiUP”, and send this incomplete offline mirror to the control machine in the isolated environment.
- Check the path of the current offline mirror on the control machine in the isolated environment. If your TiUP tool is of a recent version, you can get the current mirror address by running the following command:

```
tiup mirror show
```

If the output of the above command indicates that the `show` command does not exist, you might be using an older version of TiUP. In this case, you can get the current mirror address from `$HOME/.tiup/tiup.toml`. Record this mirror address. In the following steps,  `${base_mirror}` is used to refer to this address.

- Merge an incomplete offline mirror into an existing offline mirror:

First, copy the `keys` directory in the current offline mirror to the `$HOME/.tiup` directory:

```
cp -r ${base_mirror}/keys $HOME/.tiup/
```

Then use the TiUP command to merge the incomplete offline mirror into the mirror in use:

```
tiup mirror merge tiup-custom-mirror-v1.7.0
```

- When the above steps are completed, check the result by running the `tiup list` command. In this document’s example, the outputs of both `tiup list tiup` and `tiup list cluster` show that the corresponding components of v1.7.0 are available.

## Step 2: Deploy the offline TiUP component

After sending the package to the control machine of the target cluster, install the TiUP component by running the following commands:

```
tar xzvf tidb-community-server-${version}-linux-amd64.tar.gz && \
sh tidb-community-server-${version}-linux-amd64/local_install.sh && \
source /home/tidb/.bash_profile
```

The `local_install.sh` script automatically executes the `tiup mirror set tidb → -community-server-${version}-linux-amd64` command to set the current mirror address to `tidb-community-server-${version}-linux-amd64`.

To switch the mirror to another directory, you can manually execute the `tiup mirror → set <mirror-dir>` command. To switch the mirror to the online environment, you can execute the `tiup mirror set https://tiup-mirrors.pingcap.com` command.

### 3.4.1.3 Step 3: Initialize cluster topology file

According to the intended cluster topology, you need to manually create and edit the cluster initialization configuration file.

To create the cluster initialization configuration file, you can create a YAML-formatted configuration file on the control machine using TiUP:

```
tiup cluster template > topology.yaml
```

#### Note:

For the hybrid deployment scenarios, you can also execute `tiup cluster → template --full > topology.yaml` to create the recommended topology template. For the geo-distributed deployment scenarios, you can execute `tiup cluster template --multi-dc > topology.yaml` to create the recommended topology template.

Execute `vi topology.yaml` to see the configuration file content:

```
global:
 user: "tidb"
 ssh_port: 22
 deploy_dir: "/tidb-deploy"
 data_dir: "/tidb-data"
server_configs: {}
pd_servers:
 - host: 10.0.1.4
 - host: 10.0.1.5
 - host: 10.0.1.6
tidb_servers:
 - host: 10.0.1.7
 - host: 10.0.1.8
 - host: 10.0.1.9
tikv_servers:
 - host: 10.0.1.1
```

```

- host: 10.0.1.2
- host: 10.0.1.3
monitoring_servers:
- host: 10.0.1.4
grafana_servers:
- host: 10.0.1.4
alertmanager_servers:
- host: 10.0.1.4

```

The following examples cover six common scenarios. You need to modify the configuration file (named `topology.yaml`) according to the topology description and templates in the corresponding links. For other scenarios, edit the configuration template accordingly.

- **Minimal deployment topology**

This is the basic cluster topology, including tidb-server, tikv-server, and pd-server. It is suitable for OLTP applications.

- **TiFlash deployment topology**

This is to deploy TiFlash along with the minimal cluster topology. TiFlash is a columnar storage engine, and gradually becomes a standard cluster topology. It is suitable for real-time HTAP applications.

- **TiCDC deployment topology**

This is to deploy TiCDC along with the minimal cluster topology. TiCDC is a tool for replicating the incremental data of TiDB, introduced in TiDB 4.0. It supports multiple downstream platforms, such as TiDB, MySQL, and MQ. Compared with TiDB Binlog, TiCDC has lower latency and native high availability. After the deployment, start TiCDC and [create the replication task using cdc cli](#).

- **TiDB Binlog deployment topology**

This is to deploy TiDB Binlog along with the minimal cluster topology. TiDB Binlog is the widely used component for replicating incremental data. It provides near real-time backup and replication.

- **TiSpark deployment topology**

This is to deploy TiSpark along with the minimal cluster topology. TiSpark is a component built for running Apache Spark on top of TiDB/TiKV to answer the OLAP queries. Currently, TiUP cluster's support for TiSpark is still **experimental**.

- **Hybrid deployment topology**

This is to deploy multiple instances on a single machine. You need to add extra configurations for the directory, port, resource ratio, and label.

- **Geo-distributed deployment topology**

This topology takes the typical architecture of three data centers in two cities as an example. It introduces the geo-distributed deployment architecture and the key configuration that requires attention.

**Note:**

- For parameters that should be globally effective, configure these parameters of corresponding components in the `server_configs` section of the configuration file.
- For parameters that should be effective on a specific node, configure these parameters in the `config` of this node.
- Use `.` to indicate the subcategory of the configuration, such as `log.slow ↴ -threshold`. For more formats, see [TiUP configuration template](#).
- For more parameter description, see [TiDB config.toml.example](#), [TiKV config.toml.example](#), [PD config.toml.example](#), and [TiFlash configuration](#).

#### 3.4.1.4 Step 4: Execute the deployment command

**Note:**

You can use secret keys or interactive passwords for security authentication when you deploy TiDB using TiUP:

- If you use secret keys, you can specify the path of the keys through `-i` or `--identity_file`;
- If you use passwords, add the `-p` flag to enter the password interaction window;
- If password-free login to the target machine has been configured, no authentication is required.

In general, TiUP creates the user and group specified in the `topology.yaml` file on the target machine, with the following exceptions:

- The user name configured in `topology.yaml` already exists on the target machine.
- You have used the `--skip-create-user` option in the command line to explicitly skip the step of creating the user.

Before you execute the `deploy` command, use the `check` and `check --apply` commands to detect and automatically repair the potential risks in the cluster:

```
tiup cluster check ./topology.yaml --user root [-p] [-i /home/root/.ssh/
→ gcp_rsa]
tiup cluster check ./topology.yaml --apply --user root [-p] [-i /home/root/..
→ ssh/gcp_rsa]
```

Then execute the `deploy` command to deploy the TiDB cluster:

```
tiup cluster deploy tidb-test v5.3.0 ./topology.yaml --user root [-p] [-i /
→ home/root/.ssh/gcp_rsa]
```

In the above command:

- The name of the deployed TiDB cluster is `tidb-test`.
- You can see the latest supported versions by running `tiup list tidb`. This document takes `v5.3.0` as an example.
- The initialization configuration file is `topology.yaml`.
- `--user root`: Log in to the target machine through the `root` key to complete the cluster deployment, or you can use other users with `ssh` and `sudo` privileges to complete the deployment.
- `[-i]` and `[-p]`: optional. If you have configured login to the target machine without password, these parameters are not required. If not, choose one of the two parameters. `[-i]` is the private key of the `root` user (or other users specified by `--user`) that has access to the target machine. `[-p]` is used to input the user password interactively.
- If you need to specify the user group name to be created on the target machine, see [this example](#).

At the end of the output log, you will see `Deployed cluster `tidb-test``  
`successfully`. This indicates that the deployment is successful.

### 3.4.1.5 Step 5: Check the clusters managed by TiUP

```
tiup cluster list
```

TiUP supports managing multiple TiDB clusters. The command above outputs information of all the clusters currently managed by TiUP, including the name, deployment user, version, and secret key information:

| Starting /home/tidb/.tiup/components/cluster/v1.5.0/cluster list |       |         |            |
|------------------------------------------------------------------|-------|---------|------------|
| Name                                                             | User  | Version | Path       |
| →                                                                |       |         | PrivateKey |
| ---                                                              | ----- | -----   | -----      |
| →                                                                |       |         | -----      |

```
tidb-test tidb v5.3.0 /home/tidb/.tiup/storage/cluster/clusters/
 ↳ tidb-test /home/tidb/.tiup/storage/cluster/clusters/tidb-test/ssh/
 ↳ id_rsa
```

#### 3.4.1.6 Step 6: Check the status of the deployed TiDB cluster

For example, execute the following command to check the status of the `tidb-test` cluster:

```
tiup cluster display tidb-test
```

Expected output includes the instance ID, role, host, listening port, and status (because the cluster is not started yet, so the status is `Down/inactive`), and directory information.

#### 3.4.1.7 Step 7: Start the TiDB cluster

```
tiup cluster start tidb-test
```

If the output log includes `Started cluster `tidb-test` successfully`, the start is successful.

#### 3.4.1.8 Step 8: Verify the running status of the TiDB cluster

For the specific operations, see [Verify Cluster Status](#).

#### 3.4.1.9 What's next

If you have deployed [TiFlash](#) along with the TiDB cluster, see the following documents:

- [Use TiFlash](#)
- [Maintain a TiFlash Cluster](#)
- [TiFlash Alert Rules and Solutions](#)
- [Troubleshoot TiFlash](#)

If you have deployed [TiCDC](#) along with the TiDB cluster, see the following documents:

- [Manage TiCDC Cluster and Replication Tasks](#)
- [Troubleshoot TiCDC](#)

### 3.4.2 Deploy a TiDB Cluster in Kubernetes

You can use [TiDB Operator](#) to deploy TiDB clusters in Kubernetes. TiDB Operator is an automatic operation system for TiDB clusters in Kubernetes. It provides full life-cycle management for TiDB including deployment, upgrades, scaling, backup, fail-over, and configuration changes. With TiDB Operator, TiDB can run seamlessly in the Kubernetes clusters deployed on a public or private cloud.

For the current TiDB release, the compatible version of TiDB Operator is v1.2.

Currently, the TiDB in Kubernetes documentation is independent of the TiDB documentation. For detailed steps on how to deploy TiDB clusters in Kubernetes using TiDB Operator, see [TiDB in Kubernetes documentation](#).

## 3.5 Check Cluster Status

After a TiDB cluster is deployed, you need to check whether the cluster runs normally. This document introduces how to check the cluster status using TiUP commands, [TiDB Dashboard](#) and Grafana, and how to log into the TiDB database to perform simple SQL operations.

### 3.5.1 Check the TiDB cluster status

This section describes how to check the TiDB cluster status using TiUP commands, [TiDB Dashboard](#), and Grafana.

#### 3.5.1.1 Use TiUP

Use the `tiup cluster display <cluster-name>` command to check the cluster status. For example:

```
tiup cluster display tidb-test
```

Expected output: If the `Status` information of each node is `Up`, the cluster runs normally.

#### 3.5.1.2 Use TiDB Dashboard

1. Log in to TiDB Dashboard at  `${pd-ip} : ${pd-port} / dashboard`. The username and password is the same as that of the TiDB `root` user. If you have modified the `root` password, enter the modified password. The password is empty by default.

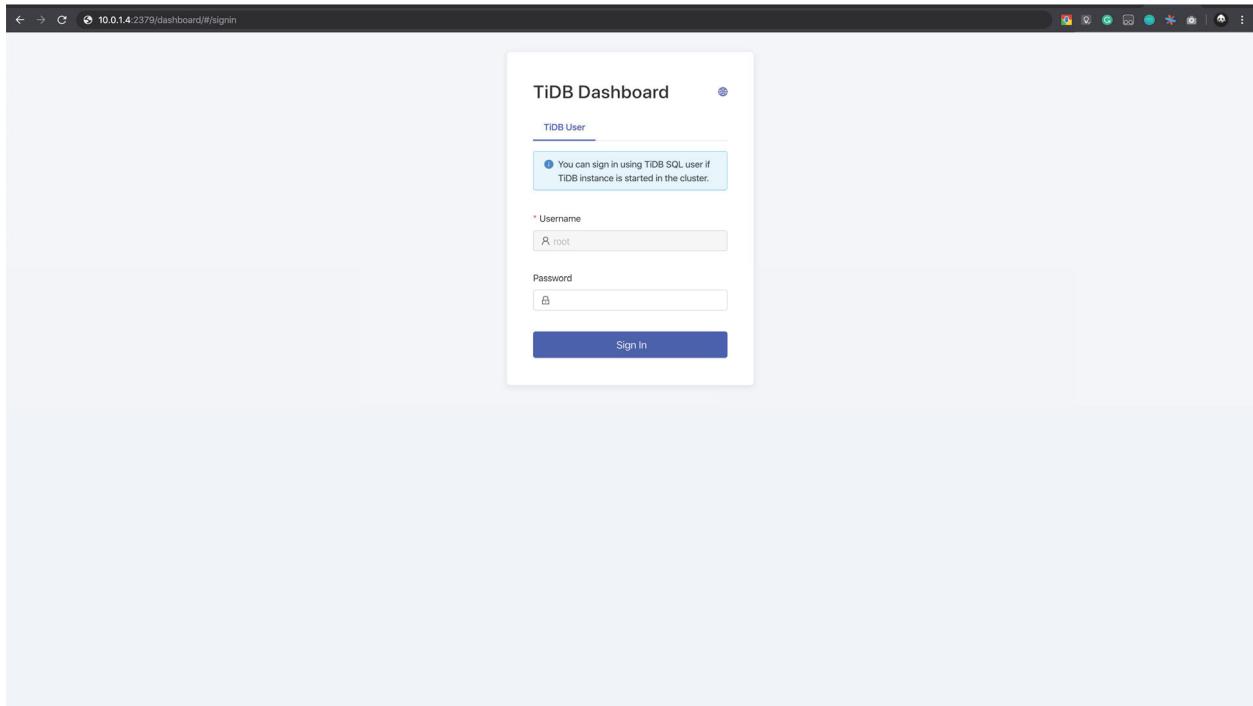
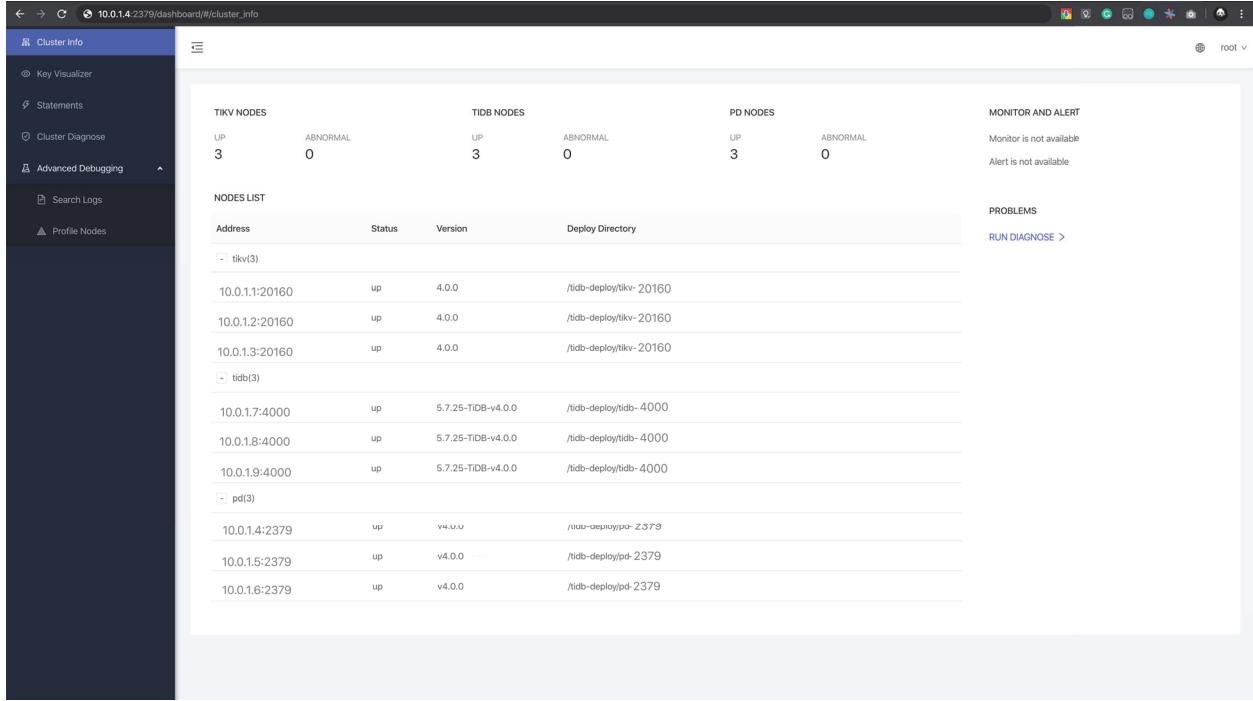


Figure 7: TiDB-Dashboard

2. The home page displays the node information in the TiDB cluster.



| TIKV NODES |          | TiDB NODES |          | PD NODES |          | MONITOR AND ALERT        |                        |
|------------|----------|------------|----------|----------|----------|--------------------------|------------------------|
| UP         | ABNORMAL | UP         | ABNORMAL | UP       | ABNORMAL | Monitor is not available | Alert is not available |
| 3          | 0        | 3          | 0        | 3        | 0        |                          |                        |

| NODES LIST     |        |                    |                         | PROBLEMS       |  |
|----------------|--------|--------------------|-------------------------|----------------|--|
| Address        | Status | Version            | Deploy Directory        | RUN DIAGNOSE > |  |
| - tikv(3)      |        |                    |                         |                |  |
| 10.0.1.1:20160 | up     | 4.0.0              | /tidb-deploy/tikv-20160 |                |  |
| 10.0.1.2:20160 | up     | 4.0.0              | /tidb-deploy/tikv-20160 |                |  |
| 10.0.1.3:20160 | up     | 4.0.0              | /tidb-deploy/tikv-20160 |                |  |
| - tidb(3)      |        |                    |                         |                |  |
| 10.0.1.7:4000  | up     | 5.7.25-TiDB-v4.0.0 | /tidb-deploy/tidb-4000  |                |  |
| 10.0.1.8:4000  | up     | 5.7.25-TiDB-v4.0.0 | /tidb-deploy/tidb-4000  |                |  |
| 10.0.1.9:4000  | up     | 5.7.25-TiDB-v4.0.0 | /tidb-deploy/tidb-4000  |                |  |
| - pd(3)        |        |                    |                         |                |  |
| 10.0.1.4:2379  | up     | v4.0.0             | /tidb-deploy/pd-2379    |                |  |
| 10.0.1.5:2379  | up     | v4.0.0             | /tidb-deploy/pd-2379    |                |  |
| 10.0.1.6:2379  | up     | v4.0.0             | /tidb-deploy/pd-2379    |                |  |

Figure 8: TiDB-Dashboard-status

### 3.5.1.3 Use Grafana

1. Log in to the Grafana monitoring at \${Grafana-ip}:3000. The default username and password are both **admin**.
2. To check the TiDB port status and load monitoring information, click **Overview**.

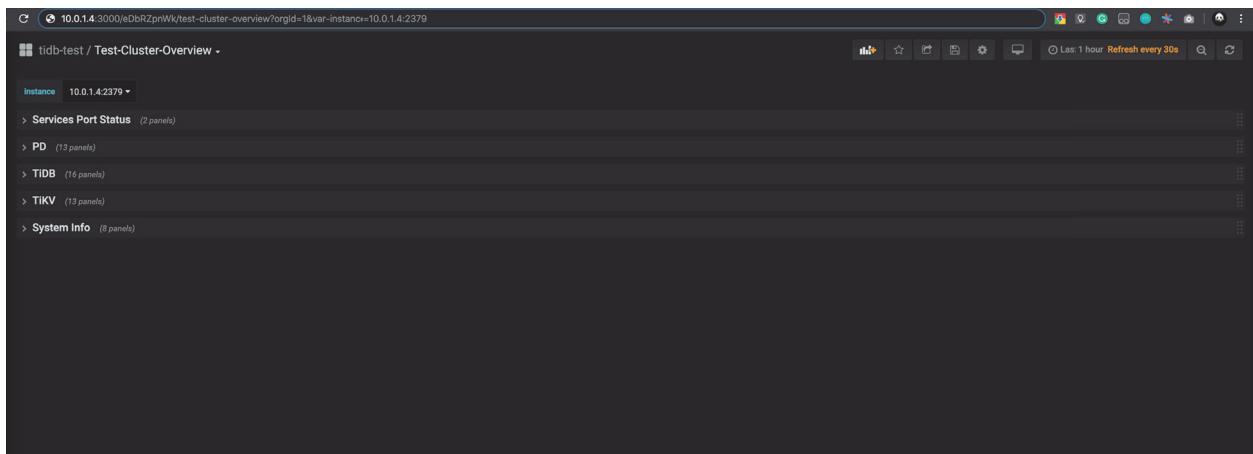


Figure 9: Grafana-overview

### 3.5.2 Log in to the database and perform simple operations

#### Note:

Install the MySQL client before you log in to the database.

Log in to the database by running the following command:

```
mysql -u root -h ${tidb_server_host_IP_address} -P 4000
```

`${tidb_server_host_IP_address}` is one of the IP addresses set for `tidb_servers` when you **initialize the cluster topology file**, such as `10.0.1.7`.

The following information indicates successful login:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.25-TiDB-v5.0.0 TiDB Server (Apache License 2.0)
 → Community Edition, MySQL 5.7 compatible
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved
 → .
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
 → statement.
```

#### 3.5.2.1 Database operations

- Check the version of TiDB:

```
select tidb_version()\G
```

Expected output:

```
***** 1. row *****
tidb_version(): Release Version: v5.0.0
Edition: Community
Git Commit Hash: 689a6b6439ae7835947fcaccf329a3fc303986cb
Git Branch: HEAD
UTC Build Time: 2020-05-28 11:09:45
GoVersion: go1.13.4
```

```
Race Enabled: false
TiKV Min Version: v3.0.0-60965b006877ca7234adaced7890d7b029ed1306
Check Table Before Drop: false
1 row in set (0.00 sec)
```

- Create a database named pingcap:

```
create database pingcap;
```

Expected output:

```
Query OK, 0 rows affected (0.10 sec)
```

Switch to the pingcap database:

```
use pingcap;
```

Expected output:

```
Database changed
```

- Create a table named tab\_tidb:

```
CREATE TABLE `tab_tidb` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `name` varchar(20) NOT NULL DEFAULT '',
 `age` int(11) NOT NULL DEFAULT 0,
 `version` varchar(20) NOT NULL DEFAULT '',
 PRIMARY KEY (`id`),
 KEY `idx_age` (`age`));
```

Expected output:

```
Query OK, 0 rows affected (0.11 sec)
```

- Insert data:

```
insert into `tab_tidb` values (1, 'TiDB', 5, 'TiDB-v5.0.0');
```

Expected output:

```
Query OK, 1 row affected (0.03 sec)
```

- View the entries in tab\_tidb:

```
select * from tab_tidb;
```

Expected output:

```
+----+----+----+
| id | name | age | version |
+----+----+----+
| 1 | TiDB | 5 | TiDB-v5.0.0 |
+----+----+----+
1 row in set (0.00 sec)
```

- View the store state, `store_id`, capacity, and uptime of TiKV:

```
select STORE_ID, ADDRESS, STORE_STATE, STORE_STATE_NAME, CAPACITY, AVAILABLE
 , UPTIME from INFORMATION_SCHEMA.TIKV_STORE_STATUS;
```

Expected output:

```
+--+
→ -----+-----+-----+-----+
→
| STORE_ID | ADDRESS | STORE_STATE | STORE_STATE_NAME | CAPACITY
→ | AVAILABLE | UPTIME |
+--+
→ -----+-----+-----+-----+
→
| 1 | 10.0.1.1:20160 | 0 | Up | 49.98GiB | 46.3
→ GiB | 5h21m52.474864026s |
| 4 | 10.0.1.2:20160 | 0 | Up | 49.98GiB |
→ 46.32GiB | 5h21m52.522669177s |
| 5 | 10.0.1.3:20160 | 0 | Up | 49.98GiB |
→ 45.44GiB | 5h21m52.713660541s |
+--+
→ -----+-----+-----+-----+
→
3 rows in set (0.00 sec)
```

- Exit TiDB:

```
exit
```

Expected output:

```
Bye
```

## 3.6 Test Cluster Performance

### 3.6.1 How to Test TiDB Using Sysbench

It is recommended to use Sysbench 1.0 or later, which can be [downloaded here](#).

### 3.6.1.1 Test plan

#### 3.6.1.1.1 TiDB configuration

Higher log level means fewer logs to be printed and thus positively influences TiDB performance. Enable `prepared plan cache` in the TiDB configuration to lower the cost of optimizing execution plan. Specifically, you can add the following command in the TiUP configuration file:

```
server_configs:
 tidb:
 log.level: "error"
 prepared-plan-cache.enabled: true
```

#### 3.6.1.1.2 TiKV configuration

Higher log level also means better performance for TiKV.

There are multiple Column Families on TiKV cluster which are mainly used to store different types of data, including Default CF, Write CF, and Lock CF. For the Sysbench test, you only need to focus on Default CF and Write CF. The Column Family that is used to import data has a constant proportion among TiDB clusters:

Default CF : Write CF = 4 : 1

Configuring the block cache of RocksDB on TiKV should be based on the machine's memory size, in order to make full use of the memory. To deploy a TiKV cluster on a 40GB virtual machine, it is recommended to configure the block cache as follows:

```
server_configs:
 tikv:
 log-level: "error"
 rocksdb.defaultcf.block-cache-size: "24GB"
 rocksdb.writecf.block-cache-size: "6GB"
```

You can also configure TiKV to share block cache:

```
server_configs:
 tikv:
 storage.block-cache.capacity: "30GB"
```

For more detailed information on TiKV performance tuning, see [Tune TiKV Performance](#).

### 3.6.1.2 Test process

**Note:**

The test in this document was performed without load balancing tools such as HAProxy. We run the Sysbench test on individual TiDB node and added the results up. The load balancing tools and the parameters of different versions might also impact the performance.

### 3.6.1.2.1 Sysbench configuration

This is an example of the Sysbench configuration file:

```
mysql-host={TIDB_HOST}
mysql-port=4000
mysql-user=root
mysql-password=password
mysql-db=sbtest
time=600
threads={8, 16, 32, 64, 128, 256}
report-interval=10
db-driver=mysql
```

The above parameters can be adjusted according to actual needs. Among them, `TIDB_HOST` is the IP address of the TiDB server (because we cannot include multiple addresses in the configuration file), `threads` is the number of concurrent connections in the test, which can be adjusted in “8, 16, 32, 64, 128, 256”. When importing data, it is recommended to set `threads = 8` or `16`. After adjusting `threads`, save the file named **config**.

See the following as a sample **config** file:

```
mysql-host=172.16.30.33
mysql-port=4000
mysql-user=root
mysql-password=password
mysql-db=sbtest
time=600
threads=16
report-interval=10
db-driver=mysql
```

### 3.6.1.2.2 Data import

**Note:**

If you enable the optimistic transaction model (TiDB uses the pessimistic transaction mode by default), TiDB rolls back transactions when a concurrency conflict is found. Setting `tidb_disable_txn_auto_retry` to `off` turns on the automatic retry mechanism after meeting a transaction conflict, which can prevent Sysbench from quitting because of the transaction conflict error.

Before importing the data, it is necessary to make some settings to TiDB. Execute the following command in MySQL client:

```
set global tidb_disable_txn_auto_retry = off;
```

Then exit the client.

Restart MySQL client and execute the following SQL statement to create a database `sbtest`:

```
create database sbtest;
```

Adjust the order in which Sysbench scripts create indexes. Sysbench imports data in the order of “Build Table -> Insert Data -> Create Index”, which takes more time for TiDB to import data. Users can adjust the order to speed up the import of data. Suppose that you use the Sysbench version 1.0.14. You can adjust the order in either of the following two ways:

- Download the modified `oltp_common.lua` file for TiDB and overwrite the `/usr/share` ↵ `/sysbench/oltp_common.lua` file with it.
- In `/usr/share/sysbench/oltp_common.lua`, move the lines 235-240 to be right behind the line 198.

**Note:**

This operation is optional and is only to save the time consumed by data import.

At the command line, enter the following command to start importing data. The config file is the one configured in the previous step:

```
sysbench --config-file=config oltp_point_select --tables=32 --table-size
↪ =10000000 prepare
```

### 3.6.1.2.3 Warming data and collecting statistics

To warm data, we load data from disk into the block cache of memory. The warmed data has significantly improved the overall performance of the system. It is recommended to warm data once after restarting the cluster.

Sysbench 1.0.14 does not provide data warming, so it must be done manually. If you are using a later version of Sysbench, you can use the data warming feature included in the tool itself.

Take a table sbtest7 in Sysbench as an example. Execute the following SQL to warming up data:

```
SELECT COUNT(pad) FROM sbtest7 USE INDEX (k_7);
```

Collecting statistics helps the optimizer choose a more accurate execution plan. The `analyze` command can be used to collect statistics on the table sbtest. Each table needs statistics.

```
ANALYZE TABLE sbtest7;
```

### 3.6.1.2.4 Point select test command

```
sysbench --config-file=config oltp_point_select --tables=32 --table-size
↪ =10000000 run
```

### 3.6.1.2.5 Update index test command

```
sysbench --config-file=config oltp_update_index --tables=32 --table-size
↪ =10000000 run
```

### 3.6.1.2.6 Read-only test command

```
sysbench --config-file=config oltp_read_only --tables=32 --table-size
↪ =10000000 run
```

## 3.6.1.3 Common issues

### 3.6.1.3.1 TiDB and TiKV are both properly configured under high concurrency, why is the overall performance still low?

This issue often has things to do with the use of a proxy. You can add pressure on single TiDB server, sum each result up and compare the summed result with the result with proxy.

Take HAProxy as an example. The parameter `nbproc` can increase the number of processes it can start at most. Later versions of HAProxy also support `nbthread` and `cpu-map`. All of these can mitigate the negative impact of proxy use on performance.

### 3.6.1.3.2 Under high concurrency, why is the CPU utilization rate of TiKV still low?

Although the overall CPU utilization rate is low for TiKV, the CPU utilization rate of some modules in the cluster might be high.

The maximum concurrency limits for other modules on TiKV, such as storage readpool, coprocessor, and gRPC, can be adjusted through the TiKV configuration file.

The actual CPU usage can be observed through Grafana's TiKV Thread CPU monitor panel. If there is a bottleneck on the modules, it can be adjusted by increasing the concurrency of the modules.

### 3.6.1.3.3 Given that TiKV has not yet reached the CPU usage bottleneck under high concurrency, why is TiDB's CPU utilization rate still low?

CPU of NUMA architecture is used on some high-end equipment where cross-CPU access to remote memory will greatly reduce performance. By default, TiDB will use all CPUs of the server, and goroutine scheduling will inevitably lead to cross-CPU memory access.

Therefore, it is recommended to deploy  $n$  TiDBs ( $n$  is the number of NUMA CPUs) on the server of NUMA architecture, and meanwhile set the TiDB parameter `max-procs` to a value that is the same as the number of NUMA CPU cores.

## 3.6.2 How to Run TPC-C Test on TiDB

This document describes how to test TiDB using [TPC-C](#).

TPC-C is an online transaction processing (OLTP) benchmark. It tests the OLTP system by using a commodity sales model that involves the following five transactions of different types:

- NewOrder
- Payment
- OrderStatus
- Delivery
- StockLevel

### 3.6.2.1 Prepare

Before testing, TPC-C Benchmark specifies the initial state of the database, which is the rule for data generation in the database. The `ITEM` table contains a fixed number of 100,000 items, while the number of warehouses can be adjusted. If there are  $W$  records in the `WAREHOUSE` table, then:

- The `STOCK` table has  $W * 100,000$  records (Each warehouse corresponds to the stock data of 100,000 items)

- The DISTRICT table has  $W * 10$  records (Each warehouse provides services to 10 districts)
- The CUSTOMER table has  $W * 10 * 3,000$  records (Each district has 3,000 customers)
- The HISTORY table has  $W * 10 * 3,000$  records (Each customer has one transaction history)
- The ORDER table has  $W * 10 * 3,000$  records (Each district has 3,000 orders and the last 900 orders generated are added to the NEW-ORDER table. Each order randomly generates 5 ~ 15 ORDER-LINE records.)

In this document, the testing uses 1,000 warehouses as an example to test TiDB.

TPC-C uses tpmC (transactions per minute) to measure the maximum qualified throughput (MQTh, Max Qualified Throughput). The transactions are the NewOrder transactions and the final unit of measure is the number of new orders processed per minute.

The test in this document is implemented based on [go-tpc](#). You can download the test program using [TiUP](#) commands.

```
tiup install bench
```

For detailed usage of the TiUP Bench component, see [TiUP Bench](#).

### 3.6.2.2 Load data

**Loading data is usually the most time-consuming and problematic stage of the entire TPC-C test.** This section provides the following command to load data.

Execute the following TiUP command in Shell:

```
tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 1000 prepare
```

Based on different machine configurations, this loading process might take a few hours. If the cluster size is small, you can use a smaller WAREHOUSE value for the test.

After the data is loaded, you can execute the `tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 4 check` command to validate the data correctness.

### 3.6.2.3 Run the test

Execute the following command to run the test:

```
tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 1000 run
```

During the test, test results are continuously printed on the console:

```
[Current] NEW_ORDER - Takes(s): 4.6, Count: 5, TPM: 65.5, Sum(ms): 4604, Avg
 ↪ (ms): 920, 90th(ms): 1500, 99th(ms): 1500, 99.9th(ms): 1500
[Current] ORDER_STATUS - Takes(s): 1.4, Count: 1, TPM: 42.2, Sum(ms): 256,
 ↪ Avg(ms): 256, 90th(ms): 256, 99th(ms): 256, 99.9th(ms): 256
```

```
[Current] PAYMENT - Takes(s): 6.9, Count: 5, TPM: 43.7, Sum(ms): 2208, Avg(
 ↪ ms): 441, 90th(ms): 512, 99th(ms): 512, 99.9th(ms): 512
[Current] STOCK_LEVEL - Takes(s): 4.4, Count: 1, TPM: 13.8, Sum(ms): 224,
 ↪ Avg(ms): 224, 90th(ms): 256, 99th(ms): 256, 99.9th(ms): 256
...
```

After the test is finished, the test summary results are printed:

```
[Summary] DELIVERY - Takes(s): 455.2, Count: 32, TPM: 4.2, Sum(ms): 44376,
 ↪ Avg(ms): 1386, 90th(ms): 2000, 99th(ms): 4000, 99.9th(ms): 4000
[Summary] DELIVERY_ERR - Takes(s): 455.2, Count: 1, TPM: 0.1, Sum(ms): 953,
 ↪ Avg(ms): 953, 90th(ms): 1000, 99th(ms): 1000, 99.9th(ms): 1000
[Summary] NEW_ORDER - Takes(s): 487.8, Count: 314, TPM: 38.6, Sum(ms):
 ↪ 282377, Avg(ms): 899, 90th(ms): 1500, 99th(ms): 1500, 99.9th(ms):
 ↪ 1500
[Summary] ORDER_STATUS - Takes(s): 484.6, Count: 29, TPM: 3.6, Sum(ms):
 ↪ 8423, Avg(ms): 290, 90th(ms): 512, 99th(ms): 1500, 99.9th(ms): 1500
[Summary] PAYMENT - Takes(s): 490.1, Count: 321, TPM: 39.3, Sum(ms): 144708,
 ↪ Avg(ms): 450, 90th(ms): 512, 99th(ms): 1000, 99.9th(ms): 1500
[Summary] STOCK_LEVEL - Takes(s): 487.6, Count: 41, TPM: 5.0, Sum(ms): 9318,
 ↪ Avg(ms): 227, 90th(ms): 512, 99th(ms): 1000, 99.9th(ms): 1000
```

After the test is finished, you can execute the `tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 4 check` command to validate the data correctness.

### 3.6.2.4 Clean up test data

Execute the following command to clean up the test data:

```
tiup bench tpcc -H 172.16.5.140 -P 4000 -D tpcc --warehouses 4 cleanup
```

## 4 Migrate

### 4.1 Migration Overview

This document describes how to migrate data to TiDB, including migrating data from MySQL and from CSV/SQL files.

#### 4.1.1 Migrate from Aurora to TiDB

In a cloud environment, you can directly migrate full data to TiDB by exporting snapshot from Aurora. For details, see [Migrate from Amazon Aurora MySQL Using TiDB Lightning](#).

## 4.1.2 Migrate from MySQL to TiDB

To migrate data from MySQL to TiDB, it is recommended to use one of the following methods:

- Use [Dumpling and TiDB Lightning](#) to migrate full data.
- Use [TiDB Data Migration \(DM\)](#) to migrate full and incremental data.

### 4.1.2.1 Use Dumpling and TiDB Lightning (full data)

#### 4.1.2.1.1 Scenarios

You can use Dumpling and TiDB Lightning to migrate full data when the data size is greater than 1 TB. If you need to replicate incremental data, it is recommended to [use DM](#) to create an incremental replication task.

#### 4.1.2.1.2 Migration method

1. Use Dumpling to export the full MySQL data.
2. Use TiDB Lightning to import the full data to TiDB. For details, refer to [Migrate data using Dumpling and TiDB Lightning](#).

### 4.1.2.2 Use DM

#### 4.1.2.2.1 Scenarios

You can use DM to migrate full MySQL data and to replicate incremental data. It is suggested that the size of the full data is less than 1 TB. Otherwise, it is recommended to use Dumpling and TiDB Lightning to import the full data, and then use DM to replicate the incremental data.

#### 4.1.2.2.2 Migration method

For details, refer to [Migrate from MySQL \(Amazon Aurora\)](#).

## 4.1.3 Migrate data from files to TiDB

You can migrate data from CSV/SQL files to TiDB.

### 4.1.3.1 Migrate data from CSV files to TiDB

#### 4.1.3.1.1 Scenarios

You can migrate data from heterogeneous databases that are not compatible with the MySQL protocol to TiDB.

#### 4.1.3.1.2 Migration method

1. Export full data to CSV files.
2. Import CSV files to TiDB using one of the following methods:

- Use TiDB Lightning.

Its import speed is fast. It is recommended to use TiDB Lightning in the case of large amounts of data in CSV files. For details, refer to [TiDB Lightning CSV Support](#).

- Use the `LOAD DATA` statement.

Execute the `LOAD DATA` statement in TiDB to import CSV files. This is more convenient, but if an error or interruption occurs during the import, manual intervention is required to check the consistency and integrity of the data. Therefore, it is **not recommended** to use this method in the production environment. For details, refer to [LOAD DATA](#).

#### 4.1.3.2 Migrate data from SQL files to TiDB

Use Mydumper and TiDB Lightning to migrate data from SQL files to TiDB. For details, refer to [Use Dumpling and TiDB Lightning](#).

## 4.2 Migrate from MySQL

### 4.2.1 Migrate from Amazon Aurora MySQL Using TiDB Lightning

This document introduces how to migrate full data from Amazon Aurora MySQL to TiDB using TiDB Lightning.

#### 4.2.1.1 Step 1: Export full data from Aurora to Amazon S3

Refer to [AWS Documentation - Exporting DB snapshot data to Amazon S3](#) to export the snapshot data of Aurora to Amazon S3.

#### 4.2.1.2 Step 2: Deploy TiDB Lightning

For detailed deployment methods, see [Deploy TiDB Lightning](#).

#### 4.2.1.3 Step 3: Configure the data source of TiDB Lightning

Based on different deployment methods, edit the `tidb-lightning.toml` configuration file as follows:

1. Configure `data-source-dir` under `[mydumper]` as the S3 Bucket path of exported data in [step 1](#).

```
[mydumper]
Data source directory
data-source-dir = "s3://bucket-name/data-path"
```

2. Configure the target TiDB cluster as follows:

```
[tidb]
The target cluster information. Fill in one address of tidb-server.
host = "172.16.31.1"
port = 4000
user = "root"
password = ""
The PD address of the cluster.
pd-addr = "127.0.0.1:2379"
```

3. Configure the backend mode:

```
[tikv-importer]
Uses Local-backend.
backend = "local"
The storage path of local temporary files. Ensure that the
corresponding directory does not exist or is empty and that the
disk capacity is large enough for storage.
sorted-kv-dir = "/path/to/local-temp-dir"
```

4. Configure the file routing.

```
[mydumper]
no-schema = true

[[mydumper.files]]
Uses single quoted strings to avoid escaping.
pattern = '(?i)^(?:[/]*)([a-z0-9_]+)\.([a-z0-9_]+)/(?:[^/]*)*(?:[a-
 z0-9\-.]+)\.(parquet)$'
schema = '$1'
table = '$2'
type = '$3'
```

#### Note:

- The above example uses the Local-backend for best performance. You can also choose TiDB-backend or Importer-backend according to your

need. For detailed introduction of the three backend modes, see [TiDB Lightning Backends](#).

- Because the path for exporting snapshot data from Aurora is different from the default file naming format supported by TiDB Lightning, you need to set additional file routing configuration.
- If TLS is enabled in the target TiDB cluster, you also need to configure TLS.

For other configurations, see [TiDB Lightning Configuration](#).

#### 4.2.1.4 Step 4: Create table schema

Because the snapshot data exported from Aurora to S3 does not contain the SQL statement file used to create database tables, you need to manually export and import the table creation statements corresponding to the database tables into TiDB. You can use Dumpling and TiDB Lightning to create all table schemas:

1. Use Dumpling to export table schema files:

```
./dumpling --host database-1.cedtf9htlae.us-west-2.rds.amazonaws.com
 ↪ --port 3306 --user root --password password --consistency none --
 ↪ no-data --output ./schema --filter "mydb.*"
```

#### Note:

- Set the parameters of the data source address and the path of output files according to your actual situation. For example, `database-1.cedtf9htlae.us-west-2.rds.amazonaws.com` is the address of Aurora MySQL.
- If you need to export all database tables, you do not need to set the `--filter` parameter. If you only need to export some of the database tables, configure `--filter` according to `table-filter`.

2. Use TiDB Lightning to create table schemas:

```
./tidb-lightning -config tidb-lightning.toml -d ./schema -no-schema=
 ↪ false
```

In this example, TiDB Lightning is only used to create table schemas, so you need to execute the above command quickly. At a regular speed, ten table creation statements can be executed in one second.

**Note:**

If the number of database tables to create is relatively small, you can manually create the corresponding databases and tables in TiDB directly, or use other tools such as mysqldump to export the schema and then import it into TiDB.

#### 4.2.1.5 Step 5: Run TiDB Lightning to import data

Run TiDB Lightning to start the import operation. If you start TiDB Lightning by using nohup directly in the command line, the program might exit because of the SIGHUP signal. Therefore, it is recommended to write nohup in a script. For example:

```
###!/bin/bash
export AWS_ACCESS_KEY_ID=${AccessKey}
export AWS_SECRET_ACCESS_KEY=${SecretKey}
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

When the import operation is started, view the progress by the following two ways:

- grep the keyword `progress` in logs, which is updated every 5 minutes by default.
- Access the monitoring dashboard. See [Monitor TiDB Lightning](#) for details.

#### 4.2.2 Migrate from MySQL SQL Files Using TiDB Lightning

This document describes how to migrate data from MySQL SQL files to TiDB using TiDB Lightning. For details on how to generate MySQL SQL files, refer to [Dumpling](#).

The data migration process described in this document uses TiDB Lightning. The steps are as follows.

##### 4.2.2.1 Step 1: Deploy TiDB Lightning

Before you start the migration, [deploy TiDB Lightning](#).

**Note:**

- If you choose the Local-backend to import data, the TiDB cluster cannot provide services during the import process. This mode imports data quickly, which is suitable for importing a large amount of data (above the TB level).

- If you choose the TiDB-backend, the cluster can provide services normally during the import, at a slower import speed.
- For detailed differences between the two backend modes, see [TiDB Lightning Backends](#).

#### 4.2.2.2 Step 2: Configure data source of TiDB Lightning

This document takes the TiDB-backend as an example. Create the `tidb-lightning.toml` configuration file and add the following major configurations in the file:

1. Set the `data-source-dir` under `[mydumper]` to the path of the MySQL SQL file.

```
[mydumper]
Data source directory
data-source-dir = "/data/export"
```

**Note:**

If a corresponding schema already exists in the downstream, set `no-schema=true` to skip the creation of the schema.

2. Add the configuration of the target TiDB cluster.

```
[tidb]
The target cluster information. Fill in one address of tidb-server.
host = "172.16.31.1"
port = 4000
user = "root"
password = ""
```

3. Add the necessary parameter for the backend. This document uses the TiDB-backend mode. Here, “backend” can also be set to “local” or “importer” according to your actual application scenario. For details, refer to [Backend Mode](#).

```
[tikv-importer]
backend = "tidb"
```

4. Add necessary parameters for importing the TiDB cluster.

```
[tidb]
host = "{{tidb-host}}"
port = {{tidb-port}}
user = "{{tidb-user}}"
password = "{{tidb-password}}"
```

For other configurations, see [TiDB Lightning Configuration](#).

#### 4.2.2.3 Step 3: Run TiDB Lightning to import data

Run TiDB Lightning to start the import operation. If you start TiDB Lightning by using `nohup` directly in the command line, the program might exit because of the `SIGHUP` signal. Therefore, it is recommended to write `nohup` in a script. For example:

```
###!/bin/bash
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

When the import operation is started, view the progress by the following two ways:

- grep the keyword `progress` in logs, which is updated every 5 minutes by default.
- Access the monitoring dashboard. See [Monitor TiDB Lightning](#) for details.

#### 4.2.3 Migrate from Amazon Aurora MySQL Using DM

To migrate data from MySQL (Amazon Aurora) to TiDB, refer to [Migrate from MySQL \(Amazon Aurora\)](#).

### 4.3 Migrate from CSV Files

#### 4.3.1 TiDB Lightning CSV Support and Restrictions

This document describes how to migrate data from CSV files to TiDB using TiDB Lightning. For information about how to generate CSV files from MySQL, see [Export to CSV files using Dumpling](#).

TiDB Lightning supports reading CSV (comma-separated values) data source, as well as other delimited format such as TSV (tab-separated values).

##### 4.3.1.1 File name

A CSV file representing a whole table must be named as `db_name.table_name.csv`. This will be restored as a table `table_name` inside the database `db_name`.

If a table spans multiple CSV files, they should be named like `db_name.table_name → .003.csv`. The number part do not need to be continuous, but must be increasing and zero-padded.

The file extension must be `*.csv`, even if the content is not separated by commas.

#### 4.3.1.2 Schema

CSV files are schema-less. To import them into TiDB, a table schema must be provided. This could be done either by:

- Providing a file named `db_name.table_name-schema.sql` containing the CREATE TABLE DDL statement, and also a file named `db_name-schema-create.sql` containing the CREATE DATABASE DDL statement.
- Creating the empty tables directly in TiDB in the first place, and then setting [`mydumper`] `no-schema = true` in `tidb-lightning.toml`.

#### 4.3.1.3 Configuration

The CSV format can be configured in `tidb-lightning.toml` under the `[mydumper.csv]` section. Most settings have a corresponding option in the MySQL `LOAD DATA` statement.

```
[mydumper.csv]
Separator between fields. Must be ASCII characters. It is not
 ↪ recommended to use the default ','. It is recommended to use '\|+\|'
 ↪ or other uncommon character combinations.
separator = ','

Quoting delimiter. Empty value means no quoting.
delimiter = ""

Line terminator. Empty value means both "\n" (LF) and "\r\n" (CRLF) are
 ↪ line terminators.
terminator = ''

Whether the CSV files contain a header.
If `header` is true, the first line will be skipped.
header = true

Whether the CSV contains any NULL value.
If `not-null` is true, all columns from CSV cannot be NULL.
not-null = false

When `not-null` is false (that is, CSV can contain NULL),
fields equal to this value will be treated as NULL.
null = '\N'

Whether to interpret backslash escapes inside fields.
backslash-escape = true

If a line ends with a separator, remove it.
trim-last-separator = false
```

In all string fields such as `separator`, `delimiter` and `terminator`, if the input involves special characters, you can use backslash escape sequence to represent them in a *double-quoted* string ("..."). For example, `separator = "\u001f"` means using the ASCII character 0x1F as separator.

Additionally, you can use *single-quoted* strings ('...') to suppress backslash escaping. For example, `terminator = '\n'` means using the two-character string: a backslash followed

by the letter “n”, as the terminator.

See the [TOML v1.0.0 specification](#) for details.

#### 4.3.1.3.1 `separator`

- Defines the field separator.
- Can be multiple characters, but must not be empty.
- Common values:
  - ‘,’ for CSV (comma-separated values)
  - “\t” for TSV (tab-separated values)
  - “\u0001” to use the ASCII character 0x01 as separator
- Corresponds to the FIELDS TERMINATED BY option in the LOAD DATA statement.

#### 4.3.1.3.2 `delimiter`

- Defines the delimiter used for quoting.
- If `delimiter` is empty, all fields are unquoted.
- Common values:
  - ‘”’ quote fields with double-quote, same as [RFC 4180](#)
  - ‘’’ disable quoting
- Corresponds to the FIELDS ENCLOSED BY option in the LOAD DATA statement.

#### 4.3.1.3.3 `terminator`

- Defines the line terminator.
- If `terminator` is empty, both “\r” (U+000D Carriage Return) and “\n” (U+000A Line Feed) are used as terminator.
- Corresponds to the LINES TERMINATED BY option in the LOAD DATA statement.

#### 4.3.1.3.4 `header`

- Whether *all* CSV files contain a header row.
- If `header` is true, the first row will be used as the *column names*. If `header` is false, the first row is not special and treated as an ordinary data row.

#### 4.3.1.3.5 `not-null` and `null`

- The `not-null` setting controls whether all fields are non-nullable.
- If `not-null` is false, the string specified by `null` will be transformed to the SQL NULL instead of a concrete value.
- Quoting will not affect whether a field is null.

For example, with the CSV file:

```
A,B,C
\N,"\"N",
```

In the default settings (`not-null = false; null = '\N'`), the columns A and B are both converted to NULL after importing to TiDB. The column C is simply the empty string '' but not NULL.

#### 4.3.1.3.6 `backslash-escape`

- Whether to interpret backslash escapes inside fields.
- If `backslash-escape` is true, the following sequences are recognized and transformed:

| Sequence | Converted to             |
|----------|--------------------------|
| \0       | Null character (U+0000)  |
| \b       | Backspace (U+0008)       |
| \n       | Line feed (U+000A)       |
| \r       | Carriage return (U+000D) |
| \t       | Tab (U+0009)             |
| \Z       | Windows EOF (U+001A)     |

In all other cases (for example, \"") the backslash is simply stripped, leaving the next character ("") in the field. The character left has no special roles (for example, delimiters) and is just an ordinary character.

- Quoting will not affect whether backslash escapes are interpreted.
- Corresponds to the `FIELDS ESCAPED BY '\'` option in the `LOAD DATA` statement.

#### 4.3.1.3.7 `trim-last-separator`

- Treats the field `separator` as a terminator, and removes all trailing separators.

For example, with the CSV file:

```
A,,B,,
```

- When `trim-last-separator = false`, this is interpreted as a row of 5 fields ('A', ↪ '', 'B', '', '').
- When `trim-last-separator = true`, this is interpreted as a row of 3 fields ('A', ↪ '', 'B').
- This option is deprecated, because the behavior with multiple trailing separators is not intuitive. Use the `terminator` option instead. If your old configuration was

```
separator = ','
trim-last-separator = true
```

we recommend changing this to

```
separator = ','
terminator = ",\n"
```

#### 4.3.1.3.8 Non-configurable options

TiDB Lightning does not support every option supported by the `LOAD DATA` statement. Some examples:

- There cannot be line prefixes (`LINES STARTING BY`).
- The header cannot be simply skipped (`IGNORE n LINES`). It must be valid column names if present.

#### 4.3.1.4 Strict format

Lightning works the best when the input files have uniform size around 256 MB. When the input is a single huge CSV file, Lightning can only use one thread to process it, which slows down import speed a lot.

This can be fixed by splitting the CSV into multiple files first. For the generic CSV format, there is no way to quickly identify when a row starts and ends without reading the whole file. Therefore, Lightning by default does *not* automatically split a CSV file. However, if you are certain that the CSV input adheres to certain restrictions, you can enable the `strict-format` setting to allow Lightning to split the file into multiple 256 MB-sized chunks for parallel processing.

```
[mydumper]
strict-format = true
```

Currently, a strict CSV file means every field occupies only a single line. In other words, one of the following must be true:

- Delimiter is empty, or
- Every field does not contain the terminator itself. In the default configuration, this means every field does not contain CR (\r) or LF (\n).

If a CSV file is not strict, but `strict-format` was wrongly set to `true`, a field spanning multiple lines may be cut in half into two chunks, causing parse failure, or even worse, quietly importing corrupted data.

#### 4.3.1.5 Common configurations

##### 4.3.1.5.1 CSV

The default setting is already tuned for CSV following RFC 4180.

```
[mydumper.csv]
separator = ',' # It is not recommended to use the default ',' . It is
 ↪ recommended to use '\|+\| ' or other uncommon character combinations
 ↪ .
delimiter = """
header = true
not-null = false
null = '\N'
backslash-escape = true
```

Example content:

```
ID,Region,Count
1,"East",32
2,"South",\N
3,"West",10
4,"North",39
```

##### 4.3.1.5.2 TSV

```
[mydumper.csv]
separator = "\t"
delimiter = ''
header = true
not-null = false
null = 'NULL'
backslash-escape = false
```

Example content:

| ID | Region | Count |
|----|--------|-------|
| 1  | East   | 32    |
| 2  | South  | NULL  |
| 3  | West   | 10    |
| 4  | North  | 39    |

#### 4.3.1.5.3 TPC-H DBGEN

```
[mydumper.csv]
separator = '|'
delimiter = ''
terminator = "|\\n"
header = false
not-null = true
backslash-escape = false
```

Example content:

```
1|East|32|
2|South|0|
3|West|10|
4|North|39|
```

### 4.3.2 LOAD DATA

The LOAD DATA statement batch loads data into a TiDB table.

#### 4.3.2.1 Synopsis

```
LoadDataStmt ::=
 'LOAD' 'DATA' LocalOpt 'INFILE' stringLit DuplicateOpt 'INTO' 'TABLE'
 ↪ TableName CharsetOpt Fields Lines IgnoreLines
 ↪ ColumnNameOrUserVarListOptWithBrackets LoadDataSetSpecOpt
```

#### 4.3.2.2 Parameters

##### 4.3.2.2.1 LocalOpt

You can specify that the imported data file is located on the client or on the server by configuring the LocalOpt parameter. Currently, TiDB only supports data import from the client. Therefore, when importing data, set the value of LocalOpt to Local.

#### 4.3.2.2.2 Fields and Lines

You can specify how to process the data format by configuring the `Fields` and `Lines` parameters.

- `FIELDS TERMINATED BY`: Specifies the separating character of each data.
- `FIELDS ENCLOSED BY`: Specifies the enclosing character of each data.
- `LINES TERMINATED BY`: Specifies the line terminator, if you want to end a line with a certain character.

Take the following data format as an example:

```
"bob","20","street 1"\r\n
"alice","33","street 1"\r\n
```

If you want to extract `bob`, `20`, and `street 1`, specify the separating character as `' , '`, and the enclosing character as `'\"'`:

```
FIELDS TERMINATED BY ' , ' ENCLOSED BY '\"' LINES TERMINATED BY '\r\n'
```

If you do not specify the parameters above, the imported data is processed in the following way by default:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY ''
LINES TERMINATED BY '\n'
```

#### 4.3.2.2.3 IGNORE number LINES

You can ignore the first `number` lines of a file by configuring the `IGNORE number LINES` parameter. For example, if you configure `IGNORE 1 LINES`, the first line of a file is ignored.

In addition, TiDB currently only supports parsing the syntax of the `DuplicateOpt`, `CharsetOpt`, and `LoadDataSetSpecOpt` parameters.

#### 4.3.2.3 Examples

```
CREATE TABLE trips (
 trip_id bigint NOT NULL PRIMARY KEY AUTO_INCREMENT,
 duration integer not null,
 start_date datetime,
 end_date datetime,
 start_station_number integer,
 start_station varchar(255),
 end_station_number integer,
 end_station varchar(255),
 bike_number varchar(255),
 member_type varchar(255)
);
```

```
Query OK, 0 rows affected (0.14 sec)
```

The following example imports data using `LOAD DATA`. Comma is specified as the separating character. The double quotation marks that enclose the data is ignored. The first line of the file is ignored.

If you see the error message `ERROR 1148 (42000): the used command is not allowed with this TiDB version`, refer to [ERROR 1148 \(42000\): the used command is not allowed with this TiDB version](#).

```
LOAD DATA LOCAL INFILE '/mnt/evo970/data-sets/bikeshare-data/2017Q4-
→ capitalbikeshare-tripdata.csv' INTO TABLE trips FIELDS TERMINATED BY
→ ',' ENCLOSED BY '\"' LINES TERMINATED BY '\r\n' IGNORE 1 LINES (
→ duration, start_date, end_date, start_station_number, start_station,
→ end_station_number, end_station, bike_number, member_type);
```

```
Query OK, 815264 rows affected (39.63 sec)
```

```
Records: 815264 Deleted: 0 Skipped: 0 Warnings: 0
```

`LOAD DATA` also supports using hexadecimal ASCII character expressions or binary ASCII character expressions as the parameters for `FIELDS ENCLOSED BY` and `FIELDS TERMINATED BY`. See the following example:

```
LOAD DATA LOCAL INFILE '/mnt/evo970/data-sets/bikeshare-data/2017Q4-
→ capitalbikeshare-tripdata.csv' INTO TABLE trips FIELDS TERMINATED BY
→ x'2c' ENCLOSED BY b'100010' LINES TERMINATED BY '\r\n' IGNORE 1 LINES
→ (duration, start_date, end_date, start_station_number, start_station
→ , end_station_number, end_station, bike_number, member_type);
```

In the above example, `x'2c'` is the hexadecimal representation of the `,` character and `b'100010'` is the binary representation of the `"` character.

#### 4.3.2.4 MySQL compatibility

This statement is understood to be fully compatible with MySQL except for the `LOAD DATA...REPLACE INTO` syntax [#24515](#). Any other compatibility differences should be reported via an issue on GitHub.

##### Note:

In earlier releases of TiDB, `LOAD DATA` committed every 20000 rows. By default, TiDB now commits all rows in one transaction. This can result in the error `ERROR 8004 (HY000) at line 1: Transaction is too large`, `size: 100000058` after upgrading from TiDB 4.0 or earlier versions.

The recommended way to resolve this error is to increase the `txn-total-size-limit` value in your `tidb.toml` file. If you are unable to increase this limit, you can also restore the previous behavior by setting `tidb_dml_batch_size` to 20000.

#### 4.3.2.5 See also

- [INSERT](#)
- [Import Example Database](#)
- [TiDB Lightning](#)

## 4.4 Migrate from MySQL SQL Files Using TiDB Lightning

This document describes how to migrate data from MySQL SQL files to TiDB using TiDB Lightning. For details on how to generate MySQL SQL files, refer to [Dumpling](#).

The data migration process described in this document uses TiDB Lightning. The steps are as follows.

### 4.4.1 Step 1: Deploy TiDB Lightning

Before you start the migration, [deploy TiDB Lightning](#).

#### Note:

- If you choose the Local-backend to import data, the TiDB cluster cannot provide services during the import process. This mode imports data quickly, which is suitable for importing a large amount of data (above the TB level).
- If you choose the TiDB-backend, the cluster can provide services normally during the import, at a slower import speed.
- For detailed differences between the two backend modes, see [TiDB Lightning Backends](#).

### 4.4.2 Step 2: Configure data source of TiDB Lightning

This document takes the TiDB-backend as an example. Create the `tidb-lightning.toml` configuration file and add the following major configurations in the file:

- Set the `data-source-dir` under `[mydumper]` to the path of the MySQL SQL file.

```
[mydumper]
Data source directory
data-source-dir = "/data/export"
```

**Note:**

If a corresponding schema already exists in the downstream, set `no-schema=true` to skip the creation of the schema.

- Add the configuration of the target TiDB cluster.

```
[tidb]
The target cluster information. Fill in one address of tidb-server.
host = "172.16.31.1"
port = 4000
user = "root"
password = ""
```

- Add the necessary parameter for the backend. This document uses the TiDB-backend mode. Here, “backend” can also be set to “local” or “importer” according to your actual application scenario. For details, refer to [Backend Mode](#).

```
[tikv-importer]
backend = "tidb"
```

- Add necessary parameters for importing the TiDB cluster.

```
[tidb]
host = "{{tidb-host}}"
port = {{tidb-port}}
user = "{{tidb-user}}"
password = "{{tidb-password}}"
```

For other configurations, see [TiDB Lightning Configuration](#).

#### 4.4.3 Step 3: Run TiDB Lightning to import data

Run TiDB Lightning to start the import operation. If you start TiDB Lightning by using `nohup` directly in the command line, the program might exit because of the `SIGHUP` signal. Therefore, it is recommended to write `nohup` in a script. For example:

```
##!/bin/bash
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

When the import operation is started, view the progress by the following two ways:

- grep the keyword `progress` in logs, which is updated every 5 minutes by default.
- Access the monitoring dashboard. See [Monitor TiDB Lightning](#) for details.

## 4.5 Replicate Incremental Data between TiDB Clusters in Real Time

This document describes how to configure a TiDB cluster and its secondary MySQL or TiDB cluster, and how to replicate the incremental data from the primary cluster to the secondary cluster in real time.

If you need to configure a running TiDB cluster and its secondary cluster for replicating incremental data in real time, use the [Backup & Restore \(BR\)](#), [Dumpling](#) and [TiDB Binlog](#).

### 4.5.1 Implementation principles

Each transaction written to TiDB is allocated a unique commit timestamp (commit TS). Through this TS, you can get the global consistency status of a TiDB cluster.

The cluster data is exported using BR or Dumpling at a globally consistent point in time. Then starting from this point in time, TiDB Binlog is used to replicate incremental data. That is, the replication process is divided into two stages: full replication and incremental replication.

1. Perform a full backup and get the commit TS of the backup data.
2. Perform incremental replication. Make sure that the start time of incremental replication is the commit TS of the backup data.

#### Note:

The commit TS obtained after exporting the backup data is a closed interval. The initial-commit-ts obtained after starting the replication process using TiDB Binlog is an open interval.

### 4.5.2 Replication process

Suppose that the existing cluster A works properly. First, you need to create a new cluster B as the secondary cluster of cluster A and then replicate the incremental data in cluster A to cluster B in real time. See the following steps for instruction.

#### 4.5.2.1 Step 1. Enable TiDB Binlog

Make sure that TiDB Binlog has been deployed and enabled in cluster A.

#### 4.5.2.2 Step 2. Export all cluster data

1. Export the data in cluster A (with global consistency ensured) to the specified path by using any of the following tools:

- Use BR for full backup
- Use Dumpling to import full data

2. Obtain a globally consistent timestamp COMMIT\_TS:

- Use the BR validate command to obtain the backup timestamp. For example:

```
COMMIT_TS=`br validate decode --field="end-version" -s local:///`
↪ home/tidb/backupdata | tail -n1`
```

- Or check the Dumpling metadata and obtain Pos (COMMIT\_TS).

```
cat metadata
```

```
Started dump at: 2020-11-10 10:40:19
SHOW MASTER STATUS:
Log: tidb-binlog
Pos: 420747102018863124
```

```
Finished dump at: 2020-11-10 10:40:20
```

3. Export the data in cluster A to cluster B.

#### 4.5.2.3 Step 3. Replicate incremental data

Modify the `drainer.toml` configuration file of TiDB Binlog by adding the following configuration to specify the COMMIT\_TS from which TiDB Binlog starts replicating data to cluster B.

```
initial-commit-ts = COMMIT_TS
[syncer.to]
host = {the IP address of cluster B}
port = 3306
```

## 5 Maintain

### 5.1 Upgrade

#### 5.1.1 Upgrade TiDB Using TiUP

This document is targeted for the following upgrade paths:

- Upgrade from TiDB 4.0 versions to TiDB 5.3 versions.
- Upgrade from TiDB 5.0 versions to TiDB 5.3 versions.
- Upgrade from TiDB 5.1 versions to TiDB 5.3 versions.
- Upgrade from TiDB 5.2 versions to TiDB 5.3 versions.

#### Note:

If your cluster to be upgraded is v3.1 or an earlier version (v3.0 or v2.1), the direct upgrade to v5.3 or its patch versions is not supported. You need to upgrade your cluster first to v4.0 and then to v5.3.

##### 5.1.1.1 Upgrade caveat

- TiDB currently does not support version downgrade or rolling back to an earlier version after the upgrade.
- For the v4.0 cluster managed using TiDB Ansible, you need to import the cluster to TiUP (`tiup cluster`) for new management according to [Upgrade TiDB Using TiUP \(v4.0\)](#). Then you can upgrade the cluster to v5.3 or its patch versions according to this document.
- To update versions earlier than 3.0 to 5.3:
  1. Update this version to 3.0 using [TiDB Ansible](#).
  2. Use TiUP (`tiup cluster`) to import the TiDB Ansible configuration.
  3. Update the 3.0 version to 4.0 according to [Upgrade TiDB Using TiUP \(v4.0\)](#).
  4. Upgrade the cluster to v5.3 according to this document.
- Support upgrading the versions of TiDB Binlog, TiCDC, TiFlash, and other components.
- For detailed compatibility changes of different versions, see the [Release Notes](#) of each version. Modify your cluster configuration according to the “Compatibility Changes” section of the corresponding release notes.

**Note:**

Do not execute any DDL request during the upgrade, otherwise an undefined behavior issue might occur.

### 5.1.1.2 Preparations

This section introduces the preparation works needed before upgrading your TiDB cluster, including upgrading TiUP and the TiUP Cluster component.

#### 5.1.1.2.1 Step 1: Upgrade TiUP or TiUP offline mirror

Before upgrading your TiDB cluster, you first need to upgrade TiUP or TiUP mirror.

Upgrade TiUP and TiUP Cluster

**Note:**

If the control machine of the cluster to upgrade cannot access `https://tiup` ↪ `-mirrors.pingcap.com`, skip this section and see [Upgrade TiUP offline mirror](#).

1. Upgrade the TiUP version. It is recommended that the TiUP version is 1.7.0 or later.

```
tiup update --self
tiup --version
```

2. Upgrade the TiUP Cluster version. It is recommended that the TiUP Cluster version is 1.7.0 or later.

```
tiup update cluster
tiup cluster --version
```

Upgrade TiUP offline mirror

**Note:**

If the cluster to upgrade was deployed not using the offline method, skip this step.

Refer to [Deploy a TiDB Cluster Using TiUP - Deploy TiUP offline](#) to download the TiUP mirror of the new version and upload it to the control machine. After executing `local_install.sh`, TiUP will complete the overwrite upgrade.

```
tar xzvf tidb-community-server-{$version}-linux-amd64.tar.gz
sh tidb-community-server-{$version}-linux-amd64/local_install.sh
source /home/tidb/.bash_profile
```

After the overwrite upgrade, execute the following command to upgrade the TiUP Cluster component.

```
tiup update cluster
```

Now, the offline mirror has been upgraded successfully. If an error occurs during TiUP operation after the overwriting, it might be that the `manifest` is not updated. You can try `rm -rf ~/.tiup/manifests/*` before running TiUP again.

#### 5.1.1.2.2 Step 2: Edit TiUP topology configuration file

##### Note:

Skip this step if one of the following situations applies:

- You have not modified the configuration parameters of the original cluster. Or you have modified the configuration parameters using `tiup → cluster` but no more modification is needed.
- After the upgrade, you want to use v5.3's default parameter values for the unmodified configuration items.

1. Enter the `vi` editing mode to edit the topology file:

```
tiup cluster edit-config <cluster-name>
```

2. Refer to the format of `topology` configuration template and fill the parameters you want to modify in the `server_configs` section of the topology file.
3. After the modification, enter : + w + q to save the change and exit the editing mode. Enter Y to confirm the change.

**Note:**

Before you upgrade the cluster to v5.3, make sure that the parameters you have modified in v4.0 are compatible in v5.3. For details, see [TiKV Configuration File](#).

The following three TiKV parameters are obsolete in TiDB v5.3. If the following parameters have been configured in your original cluster, you need to delete these parameters through `edit-config`:

- `pessimistic-txn.enabled`
- `server.request-batch-enable-cross-command`
- `server.request-batch-wait-duration`

#### 5.1.1.2.3 Step 3: Check the health status of the current cluster

To avoid the undefined behaviors or other issues during the upgrade, it is recommended to check the health status of Regions of the current cluster before the upgrade. To do that, you can use the `check` sub-command.

```
tiup cluster check <cluster-name> --cluster
```

After the command is executed, the “Region status” check result will be output.

- If the result is “All Regions are healthy”, all Regions in the current cluster are healthy and you can continue the upgrade.
- If the result is “Regions are not fully healthy: m miss-peer, n pending-peer” with the “Please fix unhealthy regions before other operations.” prompt, some Regions in the current cluster are abnormal. You need to troubleshoot the anomalies until the check result becomes “All Regions are healthy”. Then you can continue the upgrade.

#### 5.1.1.3 Perform a rolling upgrade to the TiDB cluster

This section describes how to perform a rolling upgrade to the TiDB cluster and how to verify the version after the upgrade.

##### 5.1.1.3.1 Upgrade the TiDB cluster to a specified version

You can upgrade your cluster in one of the two ways: online upgrade and offline upgrade.

By default, TiUP Cluster upgrades the TiDB cluster using the online method, which means that the TiDB cluster can still provide services during the upgrade process. With the online method, the leaders are migrated one by one on each node before the upgrade and restart. Therefore, for a large-scale cluster, it takes a long time to complete the entire upgrade operation.

If your application has a maintenance window for the database to be stopped for maintenance, you can use the offline upgrade method to quickly perform the upgrade operation.

### Online upgrade

```
tiup cluster upgrade <cluster-name> <version>
```

For example, if you want to upgrade the cluster to v5.3.0:

```
tiup cluster upgrade <cluster-name> v5.3.0
```

#### Note:

- Performing a rolling upgrade to the cluster will upgrade all components one by one. During the upgrade of TiKV, all leaders in a TiKV instance are evicted before stopping the instance. The default timeout time is 5 minutes (300 seconds). The instance is directly stopped after this timeout time.
- To perform the upgrade immediately without evicting the leader, specify `--force` in the command above. This method causes performance jitter but not data loss.
- To keep a stable performance, make sure that all leaders in a TiKV instance are evicted before stopping the instance. You can set `--transfer` → `-timeout` to a larger value, for example, `--transfer-timeout 3600` (unit: second).

### Offline upgrade

1. Before the offline upgrade, you first need to stop the entire cluster.

```
tiup cluster stop <cluster-name>
```

2. Use the `upgrade` command with the `--offline` option to perform the offline upgrade.

```
tiup cluster upgrade <cluster-name> <version> --offline
```

3. After the upgrade, the cluster will not be automatically restarted. You need to use the `start` command to restart it.

```
tiup cluster start <cluster-name>
```

### 5.1.1.3.2 Verify the cluster version

Execute the `display` command to view the latest cluster version TiDB Version:

```
tiup cluster display <cluster-name>
```

```
Cluster type: tidb
Cluster name: <cluster-name>
Cluster version: v5.3.0
```

#### Note:

By default, TiUP and TiDB share usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see [Telemetry](#).

### 5.1.1.4 FAQ

This section describes common problems encountered when updating the TiDB cluster using TiUP.

#### 5.1.1.4.1 If an error occurs and the upgrade is interrupted, how to resume the upgrade after fixing this error?

Re-execute the `tiup cluster upgrade` command to resume the upgrade. The upgrade operation restarts the nodes that have been previously upgraded. If you do not want the upgraded nodes to be restarted, use the `replay` sub-command to retry the operation:

1. Execute `tiup cluster audit` to see the operation records:

```
tiup cluster audit
```

Find the failed upgrade operation record and keep the ID of this operation record. The ID is the `<audit-id>` value in the next step.

2. Execute `tiup cluster replay <audit-id>` to retry the corresponding operation:

```
tiup cluster replay <audit-id>
```

#### 5.1.1.4.2 The evict leader has waited too long during the upgrade. How to skip this step for a quick upgrade?

You can specify `--force`. Then the processes of transferring PD leader and evicting TiKV leader are skipped during the upgrade. The cluster is directly restarted to update the version, which has a great impact on the cluster that runs online. Here is the command:

```
tiup cluster upgrade <cluster-name> <version> --force
```

#### 5.1.1.4.3 How to update the version of tools such as pd-ctl after upgrading the TiDB cluster?

You can upgrade the tool version by using TiUP to install the `ctl` component of the corresponding version:

```
tiup install ctl:v5.3.0
```

#### 5.1.1.5 TiDB 5.3 compatibility changes

- See TiDB 5.3 Release Notes for the compatibility changes.
- Try to avoid creating a new clustered index table when you apply rolling updates to the clusters using TiDB Binlog.

### 5.1.2 Use TiDB Operator

## 5.2 Scale

### 5.2.1 Scale the TiDB Cluster Using TiUP

The capacity of a TiDB cluster can be increased or decreased without interrupting the online services.

This document describes how to scale the TiDB, TiKV, PD, TiCDC, or TiFlash cluster using TiUP. If you have not installed TiUP, refer to the steps in [Install TiUP on the control machine](#).

To view the current cluster name list, run `tiup cluster list`.

For example, if the original topology of the cluster is as follows:

| Host IP  | Service        |
|----------|----------------|
| 10.0.1.3 | TiDB + TiFlash |
| 10.0.1.4 | TiDB + PD      |
| 10.0.1.5 | TiKV + Monitor |
| 10.0.1.1 | TiKV           |
| 10.0.1.2 | TiKV           |

#### 5.2.1.1 Scale out a TiDB/PD/TiKV cluster

If you want to add a TiDB node to the 10.0.1.5 host, take the following steps.

**Note:**

You can take similar steps to add the PD node. Before you add the TiKV node, it is recommended that you adjust the PD scheduling parameters in advance according to the cluster load.

## 1. Configure the scale-out topology:

**Note:**

- The port and directory information is not required by default.
- If multiple instances are deployed on a single machine, you need to allocate different ports and directories for them. If the ports or directories have conflicts, you will receive a notification during deployment or scaling.
- Since TiUP v1.0.0, the scale-out configuration will inherit the global configuration of the original cluster.

Add the scale-out topology configuration in the `scale-out.yaml` file:

```
vi scale-out.yaml
```

```
tidb_servers:
- host: 10.0.1.5
 ssh_port: 22
 port: 4000
 status_port: 10080
 deploy_dir: /data/deploy/install/deploy/tidb-4000
 log_dir: /data/deploy/install/log/tidb-4000
```

Here is a TiKV configuration file template:

```
tikv_servers:
- host: 10.0.1.5
 ssh_port: 22
 port: 20160
 status_port: 20180
 deploy_dir: /data/deploy/install/deploy/tikv-20160
 data_dir: /data/deploy/install/data/tikv-20160
 log_dir: /data/deploy/install/log/tikv-20160
```

Here is a PD configuration file template:

```
pd_servers:
- host: 10.0.1.5
 ssh_port: 22
 name: pd-1
 client_port: 2379
 peer_port: 2380
 deploy_dir: /data/deploy/install/deploy/pd-2379
 data_dir: /data/deploy/install/data/pd-2379
 log_dir: /data/deploy/install/log/pd-2379
```

To view the configuration of the current cluster, run `tiup cluster edit-config → <cluster-name>`. Because the parameter configuration of `global` and `server_configs` is inherited by `scale-out.yaml` and thus also takes effect in `scale-out.yaml`.

After the configuration, the current topology of the cluster is as follows:

| Host IP  | Service                      |
|----------|------------------------------|
| 10.0.1.3 | TiDB + TiFlash               |
| 10.0.1.4 | TiDB + PD                    |
| 10.0.1.5 | <b>TiDB</b> + TiKV + Monitor |
| 10.0.1.1 | TiKV                         |
| 10.0.1.2 | TiKV                         |

## 2. Run the scale-out command:

```
tiup cluster scale-out <cluster-name> scale-out.yaml
```

### Note:

The command above is based on the assumption that the mutual trust has been configured for the user to execute the command and the new machine. If the mutual trust cannot be configured, use the `-p` option to enter the password of the new machine, or use the `-i` option to specify the private key file.

If you see the `Scaled cluster <cluster-name> out successfully`, the scale-out operation is successfully completed.

## 3. Check the cluster status:

```
tiup cluster display <cluster-name>
```

Access the monitoring platform at <http://10.0.1.5:3000> using your browser to monitor the status of the cluster and the new node.

After the scale-out, the cluster topology is as follows:

| Host IP  | Service                      |
|----------|------------------------------|
| 10.0.1.3 | TiDB + TiFlash               |
| 10.0.1.4 | TiDB + PD                    |
| 10.0.1.5 | <b>TiDB</b> + TiKV + Monitor |
| 10.0.1.1 | TiKV                         |
| 10.0.1.2 | TiKV                         |

### 5.2.1.2 Scale out a TiFlash cluster

If you want to add a TiFlash node to the 10.0.1.4 host, take the following steps.

#### Note:

When adding a TiFlash node to an existing TiDB cluster, you need to note the following things:

1. Confirm that the current TiDB version supports using TiFlash. Otherwise, upgrade your TiDB cluster to v5.0 or later versions.
2. Execute the `tiup ctl:<cluster-version> pd -u http://<pd_ip>:<→ pd_port> config set enable-placement-rules true` command to enable the Placement Rules feature. Or execute the corresponding command in `pd-ctl`.

1. Add the node information to the `scale-out.yaml` file:

Create the `scale-out.yaml` file to add the TiFlash node information.

```
tiflash_servers:
 - host: 10.0.1.4
```

Currently, you can only add IP but not domain name.

2. Run the scale-out command:

```
tiup cluster scale-out <cluster-name> scale-out.yaml
```

#### Note:

The command above is based on the assumption that the mutual trust has been configured for the user to execute the command and the new machine. If the mutual trust cannot be configured, use the `-p` option to

enter the password of the new machine, or use the `-i` option to specify the private key file.

3. View the cluster status:

```
tiup cluster display <cluster-name>
```

Access the monitoring platform at <http://10.0.1.5:3000> using your browser, and view the status of the cluster and the new node.

After the scale-out, the cluster topology is as follows:

| Host IP  | Service                    |
|----------|----------------------------|
| 10.0.1.3 | TiDB + TiFlash             |
| 10.0.1.4 | TiDB + PD + <b>TiFlash</b> |
| 10.0.1.5 | TiDB+ TiKV + Monitor       |
| 10.0.1.1 | TiKV                       |
| 10.0.1.2 | TiKV                       |

#### 5.2.1.3 Scale out a TiCDC cluster

If you want to add two TiCDC nodes to the 10.0.1.3 and 10.0.1.4 hosts, take the following steps.

1. Add the node information to the `scale-out.yaml` file:

Create the `scale-out.yaml` file to add the TiCDC node information.

```
cdc_servers:
 - host: 10.0.1.3
 gc-ttl: 86400
 data_dir: /data/deploy/install/data/cdc-8300
 - host: 10.0.1.4
 gc-ttl: 86400
 data_dir: /data/deploy/install/data/cdc-8300
```

2. Run the scale-out command:

```
tiup cluster scale-out <cluster-name> scale-out.yaml
```

**Note:**

The command above is based on the assumption that the mutual trust has been configured for the user to execute the command and the new machine. If the mutual trust cannot be configured, use the **-p** option to enter the password of the new machine, or use the **-i** option to specify the private key file.

3. View the cluster status:

```
tiup cluster display <cluster-name>
```

Access the monitoring platform at <http://10.0.1.5:3000> using your browser, and view the status of the cluster and the new nodes.

After the scale-out, the cluster topology is as follows:

| Host IP  | Service                            |
|----------|------------------------------------|
| 10.0.1.3 | TiDB + TiFlash + <b>TiCDC</b>      |
| 10.0.1.4 | TiDB + PD + TiFlash + <b>TiCDC</b> |
| 10.0.1.5 | TiDB+ TiKV + Monitor               |
| 10.0.1.1 | TiKV                               |
| 10.0.1.2 | TiKV                               |

#### 5.2.1.4 Scale in a TiDB/PD/TiKV cluster

If you want to remove a TiKV node from the 10.0.1.5 host, take the following steps.

**Note:**

- You can take similar steps to remove the TiDB and PD node.
- Because the TiKV, TiFlash, and TiDB Binlog components are taken offline asynchronously and the stopping process takes a long time, TiUP takes them offline in different methods. For details, see [Particular handling of components' offline process](#).

**Note:**

The PD Client in TiKV caches the list of PD nodes. The current version of TiKV has a mechanism to automatically and regularly update PD nodes, which can help mitigate the issue of an expired list of PD nodes cached by TiKV. However, after scaling out PD, you should try to avoid directly removing all PD nodes at once that exist before the scaling. If necessary, before making all the previously existing PD nodes offline, make sure to switch the PD leader to a newly added PD node.

1. View the node ID information:

```
tiup cluster display <cluster-name>
```

```
Starting /root/.tiup/components/cluster/v1.7.0/cluster display <cluster
 ↵ -name>
TiDB Cluster: <cluster-name>
TiDB Version: v5.3.0
ID Role Host Ports Status
 ↵ Data Dir Deploy Dir
--- ---- ---- ----- -----
 ↵ -----
10.0.1.3:8300 cdc 10.0.1.3 8300 Up
 ↵ data/cdc-8300 deploy/cdc-8300
10.0.1.4:8300 cdc 10.0.1.4 8300 Up
 ↵ data/cdc-8300 deploy/cdc-8300
10.0.1.4:2379 pd 10.0.1.4 2379/2380
 ↵ Healthy data/pd-2379 deploy/pd-2379
10.0.1.1:20160 tikv 10.0.1.1 20160/20180
 ↵ data/tikv-20160 deploy/tikv-20160
10.0.1.2:20160 tikv 10.0.1.2 20160/20180
 ↵ data/tikv-20160 deploy/tikv-20160
10.0.1.5:20160 tikv 10.0.1.5 20160/20180
 ↵ data/tikv-20160 deploy/tikv-20160
10.0.1.3:4000 tidb 10.0.1.3 4000/10080
 ↵ - deploy/tidb-4000
10.0.1.4:4000 tidb 10.0.1.4 4000/10080
 ↵ - deploy/tidb-4000
10.0.1.5:4000 tidb 10.0.1.5 4000/10080
 ↵ - deploy/tidb-4000
10.0.1.3:9000 tiflash 10.0.1.3 9000/8123/3930/20170/20292/8234 Up
 ↵ data/tiflash-9000 deploy/tiflash-9000
10.0.1.4:9000 tiflash 10.0.1.4 9000/8123/3930/20170/20292/8234 Up
 ↵ data/tiflash-9000 deploy/tiflash-9000
```

|                                                                                                    |    |
|----------------------------------------------------------------------------------------------------|----|
| 10.0.1.5:9090 prometheus 10.0.1.5 9090<br>↳ data/prometheus-9090 deploy/prometheus-9090            | Up |
| 10.0.1.5:3000 grafana 10.0.1.5 3000<br>↳ - deploy/grafana-3000                                     | Up |
| 10.0.1.5:9093 alertmanager 10.0.1.5 9093/9294<br>↳ data/alertmanager-9093 deploy/alertmanager-9093 | Up |

2. Run the scale-in command:

```
tiup cluster scale-in <cluster-name> --node 10.0.1.5:20160
```

The `--node` parameter is the ID of the node to be taken offline.

If you see the `Scaled cluster <cluster-name>` in successfully, the scale-in operation is successfully completed.

3. Check the cluster status:

The scale-in process takes some time. If the status of the node to be scaled in becomes `Tombstone`, that means the scale-in operation is successful.

To check the scale-in status, run the following command:

```
tiup cluster display <cluster-name>
```

Access the monitoring platform at <http://10.0.1.5:3000> using your browser, and view the status of the cluster.

The current topology is as follows:

| Host IP  | Service                                   |
|----------|-------------------------------------------|
| 10.0.1.3 | TiDB + TiFlash + TiCDC                    |
| 10.0.1.4 | TiDB + PD + TiFlash + TiCDC               |
| 10.0.1.5 | TiDB + Monitor ( <b>TiKV is deleted</b> ) |
| 10.0.1.1 | TiKV                                      |
| 10.0.1.2 | TiKV                                      |

### 5.2.1.5 Scale in a TiFlash cluster

If you want to remove a TiFlash node from the 10.0.1.4 host, take the following steps.

#### 5.2.1.5.1 1. Adjust the number of replicas of the tables according to the number of remaining TiFlash nodes

Before the node goes down, make sure that the number of remaining nodes in the TiFlash cluster is no smaller than the maximum number of replicas of all tables. Otherwise, modify the number of TiFlash replicas of the related tables.

- For all tables whose replicas are greater than the number of remaining TiFlash nodes in the cluster, execute the following command in the TiDB client:

```
alter table <db-name>.<table-name> settiflash replica 0;
```

- Wait for the TiFlash replicas of the related tables to be deleted. [Check the table replication progress](#) and the replicas are deleted if the replication information of the related tables is not found.

#### 5.2.1.5.2 2. Perform the scale-in operation

Next, perform the scale-in operation with one of the following solutions.

Solution 1: Use TiUP to remove a TiFlash node

- First, confirm the name of the node to be taken down:

```
tiup cluster display <cluster-name>
```

- Remove the TiFlash node (assume that the node name is 10.0.1.4:9000 from Step 1):

```
tiup cluster scale-in <cluster-name> --node 10.0.1.4:9000
```

Solution 2: Manually remove a TiFlash node

In special cases (such as when a node needs to be forcibly taken down), or if the TiUP scale-in operation fails, you can manually remove a TiFlash node with the following steps.

- Use the store command of pd-ctl to view the store ID corresponding to this TiFlash node.

- Enter the store command in [pd-ctl](#) (the binary file is under `resources/bin` in the `tidb-ansible` directory).
- If you use TiUP deployment, replace `pd-ctl` with `tiup ctl pd`:

```
tiup ctl:<cluster-version> pd -u http://<pd_ip>:<pd_port> store
```

**Note:**

If multiple PD instances exist in the cluster, you only need to specify the IP address:port of an active PD instance in the above command.

- Remove the TiFlash node in pd-ctl:

- Enter `store delete <store_id>` in pd-ctl (`<store_id>` is the store ID of the TiFlash node found in the previous step).
- If you use TiUP deployment, replace pd-ctl with tiup ctl pd:

```
tiup ctl:<cluster-version> pd -u http://<pd_ip>:<pd_port> store
 ↪ delete <store_id>
```

**Note:**

If multiple PD instances exist in the cluster, you only need to specify the IP address:port of an active PD instance in the above command.

3. Wait for the store of the TiFlash node to disappear or for the `state_name` to become `Tombstone` before you stop the TiFlash process.
4. Manually delete TiFlash data files (whose location can be found in the `data_dir` directory under the TiFlash configuration of the cluster topology file).
5. Manually update TiUP's cluster configuration file (delete the information of the TiFlash node that goes down in edit mode).

```
tiup cluster edit-config <cluster-name>
```

**Note:**

Before all TiFlash nodes in the cluster stop running, if not all tables replicated to TiFlash are canceled, you need to manually clean up the replication rules in PD, or the TiFlash node cannot be taken down successfully.

The steps to manually clean up the replication rules in PD are below:

1. View all data replication rules related to TiFlash in the current PD instance:

```
curl http://<pd_ip>:<pd_port>/pd/api/v1/config/rules/group/tiflash
```

```
[
 {
 "group_id": "tiflash",
 "id": "table-45-r",
 "override": true,
 "start_key": "7480000000000000FF2D5F720000000000FA",
 "end_key": "7480000000000000FF2E00000000000000F8",
```

```

"role": "learner",
"count": 1,
"label_constraints": [
{
 "key": "engine",
 "op": "in",
 "values": [
 "tiflash"
]
}
]
]
```

2. Remove all data replication rules related to TiFlash. Take the rule whose `id` is `table-45-r` as an example. Delete it by the following command:

```
curl -v -X DELETE http://<pd_ip>:<pd_port>/pd/api/v1/config/rule/
→ tiflash/table-45-r
```

### 5.2.1.6 Scale in a TiCDC cluster

If you want to remove the TiCDC node from the 10.0.1.4 host, take the following steps:

1. Take the node offline:

```
tiup cluster scale-in <cluster-name> --node 10.0.1.4:8300
```

2. View the cluster status:

```
tiup cluster display <cluster-name>
```

Access the monitoring platform at <http://10.0.1.5:3000> using your browser, and view the status of the cluster.

The current topology is as follows:

| Host IP  | Service                                 |
|----------|-----------------------------------------|
| 10.0.1.3 | TiDB + TiFlash + TiCDC                  |
| 10.0.1.4 | TiDB + PD + ( <b>TiCDC is deleted</b> ) |
| 10.0.1.5 | TiDB + Monitor                          |
| 10.0.1.1 | TiKV                                    |
| 10.0.1.2 | TiKV                                    |

### 5.2.2 Use TiDB Operator

## 5.3 Backup and Restore

### 5.3.1 Use BR Tool (Recommended)

#### 5.3.1.1 BR Tool Overview

BR (Backup & Restore) is a command-line tool for distributed backup and restoration of the TiDB cluster data.

Compared with [dumpling](#), BR is more suitable for scenarios involved huge data volumes.

In addition to regular backup and restoration, you can also use BR for large-scale data migration as long as compatibility is ensured.

This document describes BR's implementation principles, recommended deployment configuration, usage restrictions and several methods to use BR.

#### 5.3.1.1.1 Implementation principles

BR sends the backup or restoration commands to each TiKV node. After receiving these commands, TiKV performs the corresponding backup or restoration operations.

Each TiKV node has a path in which the backup files generated in the backup operation are stored and from which the stored backup files are read during the restoration.

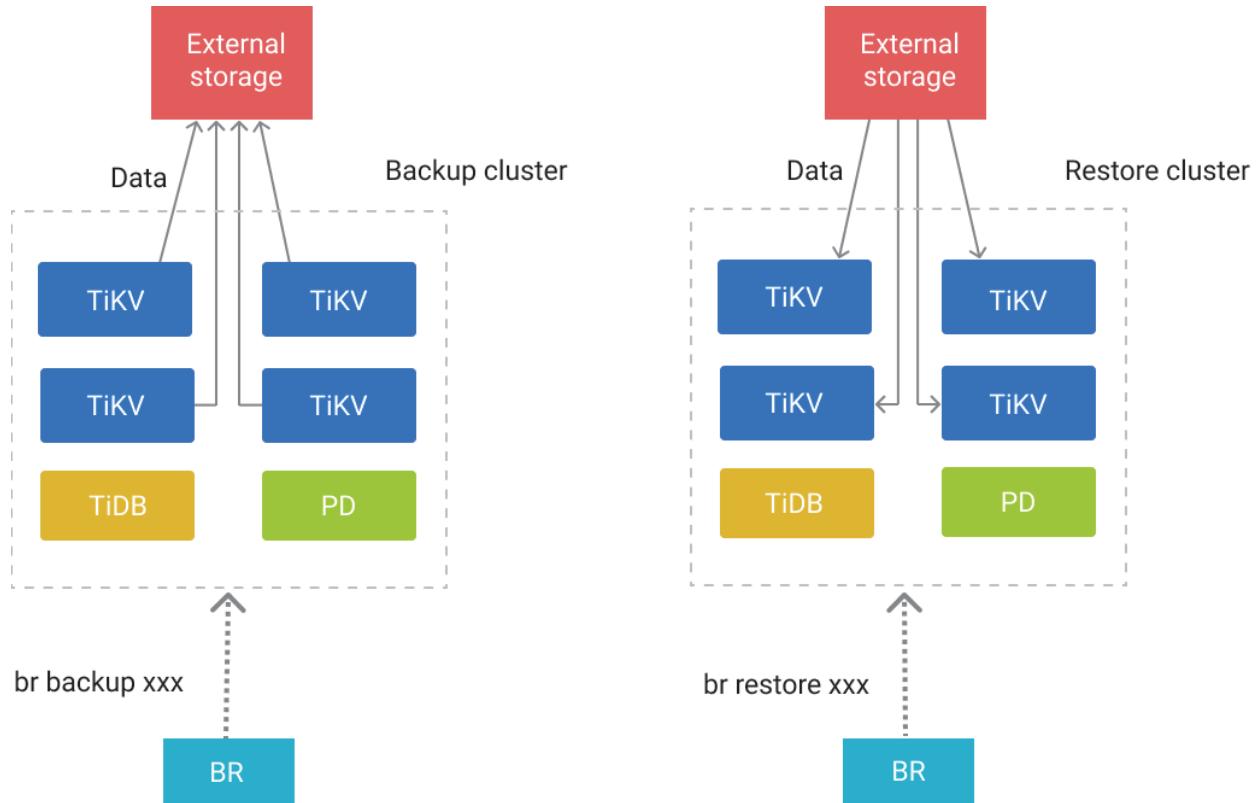


Figure 10: br-arch

### Backup principle

When BR performs a backup operation, it first obtains the following information from PD:

- The current TS (timestamp) as the time of the backup snapshot
- The TiKV node information of the current cluster

According to these information, BR starts a TiDB instance internally to obtain the database or table information corresponding to the TS, and filters out the system databases (`information_schema`, `performance_schema`, `mysql`) at the same time.

According to the backup sub-command, BR adopts the following two types of backup logic:

- Full backup: BR traverses all the tables and constructs the KV range to be backed up according to each table.
- Single table backup: BR constructs the KV range to be backed up according a single table.

Finally, BR collects the KV range to be backed up and sends the complete backup request to the TiKV node of the cluster.

The structure of the request:

```
BackupRequest{
 ClusterId, // The cluster ID.
 StartKey, // The starting key of the backup (backed up).
 EndKey, // The ending key of the backup (not backed up).
 StartVersion, // The version of the last backup snapshot, used for the
 // → incremental backup.
 EndVersion, // The backup snapshot time.
 StorageBackend, // The path where backup files are stored.
 RateLimit, // Backup speed (MB/s).
}
```

After receiving the backup request, the TiKV node traverses all Region leaders on the node to find the Regions that overlap with the KV ranges in this request. The TiKV node backs up some or all of the data within the range, and generates the corresponding SST file.

After finishing backing up the data of the corresponding Region, the TiKV node returns the metadata to BR. BR collects the metadata and stores it in the `backupmeta` file which is used for restoration.

If `StartVersion` is not 0, the backup is seen as an incremental backup. In addition to KVs, BR also collects DDLs between `[StartVersion, EndVersion)`. During data restoration, these DDLs are restored first.

If checksum is enabled when you execute the backup command, BR calculates the checksum of each backed up table for data check.

Types of backup files

Two types of backup files are generated in the path where backup files are stored:

- **The SST file:** stores the data that the TiKV node backed up.
- **The `backupmeta` file:** stores the metadata of this backup operation, including the number, the key range, the size, and the Hash (sha256) value of the backup files.
- **The `backup.lock` file:** prevents multiple backup operations from storing data to the same directory.

The format of the SST file name

The SST file is named in the format of `storeID_regionID_regionEpoch_keyHash_cf`, where

- `storeID` is the TiKV node ID;
- `regionID` is the Region ID;
- `regionEpoch` is the version number of the Region;

- `keyHash` is the Hash (sha256) value of the `startKey` of a range, which ensures the uniqueness of a key;
- `cf` indicates the **Column Family** of RocksDB (`default` or `write` by default).

Restoration principle

During the data restoration process, BR performs the following tasks in order:

1. It parses the `backupmeta` file in the backup path, and then starts a TiDB instance internally to create the corresponding databases and tables based on the parsed information.
2. It aggregates the parsed SST files according to the tables.
3. It pre-splits Regions according to the key range of the SST file so that every Region corresponds to at least one SST file.
4. It traverses each table to be restored and the SST file corresponding to each tables.
5. It finds the Region corresponding to the SST file and sends a request to the corresponding TiKV node for downloading the file. Then it sends a request for loading the file after the file is successfully downloaded.

After TiKV receives the request to load the SST file, TiKV uses the Raft mechanism to ensure the strong consistency of the SST data. After the downloaded SST file is loaded successfully, the file is deleted asynchronously.

After the restoration operation is completed, BR performs a checksum calculation on the restored data to compare the stored data with the backed up data.

### 5.3.1.1.2 Deploy and use BR

Recommended deployment configuration

- It is recommended that you deploy BR on the PD node.
- It is recommended that you mount a high-performance SSD to BR nodes and all TiKV nodes. A 10-gigabit network card is recommended. Otherwise, bandwidth is likely to be the performance bottleneck during the backup and restore process.

#### Note:

- If you do not mount a network disk or use other shared storage, the data backed up by BR will be generated on each TiKV node. Because BR only backs up leader replicas, you should estimate the space reserved for each node based on the leader size.

- Because TiDB uses leader count for load balancing by default, leaders can greatly differ in size. This might result in uneven distribution of backup data on each node.

### Usage restrictions

The following are the limitations of using BR for backup and restoration:

- When BR restores data to the upstream cluster of TiCDC/Drainer, TiCDC/Drainer cannot replicate the restored data to the downstream.
- BR supports operations only between clusters with the same `new_collations_enabled_on_first_backup` value because BR only backs up KV data. If the cluster to be backed up and the cluster to be restored use different collations, the data validation fails. Therefore, before restoring a cluster, make sure that the switch value from the query result of the `select VARIABLE_VALUE from mysql.tidb where VARIABLE_NAME='new_collation_enabled'`; statement is consistent with that during the backup process.

### Compatibility

The compatibility issues of BR and the TiDB cluster are divided into the following categories:

- Some versions of BR are not compatible with the interface of the TiDB cluster.
- The KV format might change when some features are enabled or disabled. If these features are not consistently enabled or disabled during backup and restore, compatibility issues might occur.

These features are as follows:



| Related         | is-    |                                                                                                                                                                                                                                                                   |
|-----------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Features        | Issues | Solutions                                                                                                                                                                                                                                                         |
| Clustered index | #1565  | Make sure that the value of the <code>tidb_enable_clustered_index</code> global variable during re-store is consistent with that during backup. Otherwise, data inconsistency might occur, such as <code>default</code> ↗ <code>not</code> ↗ <code>found</code> ↗ |
| 219             |        | and inconsistency                                                                                                                                                                                                                                                 |

---

| Features      | Issues                                                                                                                                      | Solutions                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| New collation | #352 Make sure that the value of the <code>new_collations_enabled_on_first_bootstrap</code> variable is consistent with that during backup. | Otherwise, inconsistent data index might occur and checksum might fail to pass. |

---

| Feature                    | Issues                                                                                                                                        | Solutions |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| TiCDC <a href="#">#364</a> | Currently, TiKV can- not push down the BR- ingested SST files to TiCDC. Therefore, you need to disable TiCDC when using BR to re- store data. |           |

---

| Features                | Issues                                                                                                                                                                     | Solutions      |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| Global temporary tables | Make sure that you are using BR v5.3.0 or a later version to back up and restore data. Otherwise, an error occurs in the definition of the backed global temporary tables. | Related issues |

---

However, even after you have ensured that the above features are consistently enabled or disabled during backup and restore, compatibility issues might still occur due to the inconsistent internal versions or inconsistent interfaces between BR and TiKV/TiDB/PD. To avoid such cases, BR has the built-in version check.

## Version check

Before performing backup and restore, BR compares and checks the TiDB cluster version and the BR version. If there is a major-version mismatch (for example, BR v4.x and TiDB v5.x), BR prompts a reminder to exit. To forcibly skip the version check, you can set `--check-requirements=false`.

Note that skipping the version check might introduce incompatibility. The version compatibility information between BR and TiDB versions are as follows:

---

Backup

ver-

sion

(ver-  
tical)

| Re-    | Use     | Use    | Use    |
|--------|---------|--------|--------|
| store  | BR      | BR     | BR     |
| ver-   | nightly | v5.0   | v4.0   |
| sion   | to re-  | to re- | to re- |
| (hori- | store   | store  | store  |
| zon-   | TiDB    | TiDB   | TiDB   |
| tal)   | nightly | v5.0   | v4.0   |

---

Use (If  
BR a  
nightly table  
to with  
back the  
up pri-  
TiDB mary  
nightly key  
of  
the  
non-  
integer  
clus-  
tered  
index  
type  
is re-  
stored  
to a  
TiDB  
v4.0  
clus-  
ter,  
BR  
will  
cause  
data  
error  
with-  
out  
warn-  
ing.)

---

Backup

ver-

sion

(ver-

tical)

| Re-    | Use     | Use    | Use    |
|--------|---------|--------|--------|
| store  | BR      | BR     | BR     |
| ver-   | nightly | v5.0   | v4.0   |
| sion   | to re-  | to re- | to re- |
| (hori- | store   | store  | store  |
| zon-   | TiDB    | TiDB   | TiDB   |
| tal)   | nightly | v5.0   | v4.0   |

---

Use (If  
BR a  
v5.0 table  
to with  
back the  
up pri-  
TiDB mary  
v5.0 key  
of  
the  
non-  
integer  
clus-  
tered  
index  
type  
is re-  
stored  
to a  
TiDB  
v4.0  
clus-  
ter,  
BR  
will  
cause  
data  
error  
with-  
out  
warn-  
ing.)

## Backup

ver-

sion

(ver-

tical)

|                       | Use<br>store    | Use<br>nightly  | Use<br>v5.0     |
|-----------------------|-----------------|-----------------|-----------------|
| ver-<br>sion          | to re-<br>store | to re-<br>store | to re-<br>store |
| (hor-<br>zon-<br>tal) | TiDB<br>nightly | TiDB<br>v5.0    | TiDB<br>v4.0    |

Use (If  
BR TiKV  
v4.0 >= v4.0.0-  
to v4.0.0-  
back rc.1,  
up and  
TiDB if BR  
v4.0 contains  
the  
[#233](#)  
bug  
fix  
and  
TiKV  
does  
not  
con-  
tain  
the  
[#7241](#)  
bug  
fix,  
BR  
will  
cause  
the  
TiKV  
node  
to  
restart

---

|         |         |         |         |
|---------|---------|---------|---------|
| Backup  |         |         |         |
| ver-    |         |         |         |
| sion    |         |         |         |
| (ver-   |         |         |         |
| tical)  |         |         |         |
| Re-     | Use     | Use     | Use     |
| store   | BR      | BR      | BR      |
| ver-    | nightly | v5.0    | v4.0    |
| sion    | to re-  | to re-  | to re-  |
| (hori-  | store   | store   | store   |
| zon-    | TiDB    | TiDB    | TiDB    |
| tal)    | nightly | v5.0    | v4.0    |
|         | (If     | (If     | (If     |
| BR      | the     | the     | the     |
| nightly | TiDB    | TiDB    | TiDB    |
| or      | ver-    | ver-    | ver-    |
| v5.0    | sion    | sion    | sion    |
| to      | is      | is      | is      |
| back    | ear-    | ear-    | ear-    |
| up      | lier    | lier    | lier    |
| TiDB    | than    | than    | than    |
| v4.0    | v4.0.9, | v4.0.9, | v4.0.9, |
|         | the     | the     | the     |
|         | #609    | #609    | #609    |
|         | issue   | issue   | issue   |
|         | might   | might   | might   |
|         | oc-     | oc-     | oc-     |
|         | cur.)   | cur.)   | cur.)   |

---

Back up and restore table data in the `mysql` system schema (experimental feature)

**Warning:**

This feature is experimental and not thoroughly tested. It is highly **not recommended** to use this feature in the production environment.

Before v5.1.0, BR filtered out data from the system schema `mysql` during the backup. Since v5.1.0, BR **backs up** all data by default, including the system schemas `mysql.*`. But the technical implementation of restoring the system tables in `mysql.*` is not complete yet, so the tables in the system schema `mysql` are **not** restored by default.

If you want the data of a system table (for example, `mysql.usertable1`) to be restored to the system schema `mysql`, you can set the **filter parameter** to filter the table name (`-r f "mysql.usertable1"`). After the setting, the system table is first restored to the temporary schema, and then to the system schema through renaming.

It should be noted that the following system tables cannot be restored correctly due to technical reasons. Even if `-f "mysql.*"` is specified, these tables will not be restored:

- Tables related to statistics: “`stats_buckets`”, “`stats_extended`”, “`stats_feedback`”, “`stats_fm_sketch`”, “`stats_histograms`”, “`stats_meta`”, “`stats_top_n`”
- Tables related to privileges or the system: “`tidb`”, “`global_variables`”, “`columns_priv`”, “`db`”, “`default_roles`”, “`global_grants`”, “`global_priv`”, “`role_edges`”, “`tables_priv`”, “`user`”, “`gc_delete_range`”, “`Gc_delete_range_done`”, “`schema_index_usage`”

Minimum machine configuration required for running BR

The minimum machine configuration required for running BR is as follows:

| CPU    | Memory | Hard Disk Type | Network              |
|--------|--------|----------------|----------------------|
| 1 core | 4 GB   | HDD            | Gigabit network card |

In general scenarios (less than 1000 tables for backup and restore), the CPU consumption of BR at runtime does not exceed 200%, and the memory consumption does not exceed 4 GB. However, when backing up and restoring a large number of tables, BR might consume more than 4 GB of memory. In a test of backing up 24000 tables, BR consumes about 2.7 GB of memory, and the CPU consumption remains below 100%.

Best practices

The following are some recommended operations for using BR for backup and restoration:

- It is recommended that you perform the backup operation during off-peak hours to minimize the impact on applications.
- BR supports restore on clusters of different topologies. However, the online applications will be greatly impacted during the restore operation. It is recommended that you perform restore during the off-peak hours or use `rate-limit` to limit the rate.
- It is recommended that you execute multiple backup operations serially. Running different backup operations in parallel reduces backup performance and also affects the online application.
- It is recommended that you execute multiple restore operations serially. Running different restore operations in parallel increases Region conflicts and also reduces restore performance.
- It is recommended that you mount a shared storage (for example, NFS) on the backup path specified by `-s`, to make it easier to collect and manage backup files.
- It is recommended that you use a storage hardware with high throughput, because the throughput of a storage hardware limits the backup and restoration speed.

- It is recommended that you disable the checksum feature (`--checksum = false`) during backup operation and only enable it during the restore operation to reduce migration time. This is because BR by default respectively performs checksum calculation after backup and restore operations to compare the stored data with the corresponding cluster data to ensure accuracy.

## How to use BR

Currently, the following methods are supported to run the BR tool:

- Use SQL statements
- Use the command-line tool
- Use BR In the Kubernetes environment

### Use SQL statements

TiDB supports both `BACKUP` and `RESTORE` SQL statements. The progress of these operations can be monitored with the statement `SHOW BACKUPS|RESTORES`.

### Use the command-line tool

The `br` command-line utility is available as a [separate download](#). For details, see [Use BR Command-line for Backup and Restoration](#).

### In the Kubernetes environment

In the Kubernetes environment, you can use the BR tool to back up TiDB cluster data to S3-compatible storage, Google Cloud Storage (GCS) and persistent volumes (PV), and restore them:

#### Note:

For Amazon S3 and Google Cloud Storage parameter descriptions, see the [External Storages](#) document.

- Back up Data to S3-Compatible Storage Using BR
- Restore Data from S3-Compatible Storage Using BR
- Back up Data to GCS Using BR
- Restore Data from GCS Using BR
- Back up Data to PV Using BR
- Restore Data from PV Using BR

### 5.3.1.1.3 Other documents about BR

- Use BR Command-line
- BR Use Cases
- BR FAQ
- External Storages

### 5.3.1.2 Use BR Command-line for Backup and Restoration

This document describes how to back up and restore TiDB cluster data using the BR command line.

Make sure you have read [BR Tool Overview](#), especially [Usage Restrictions](#) and [Best Practices](#).

#### 5.3.1.2.1 BR command-line description

A `br` command consists of sub-commands, options, and parameters.

- Sub-command: the characters without `-` or `--`.
- Option: the characters that start with `-` or `--`.
- Parameter: the characters that immediately follow behind and are passed to the sub-command or the option.

This is a complete `br` command:

```
br backup full --pd "${PDIP}:2379" -s "local:///tmp/backup"
```

Explanations for the above command are as follows:

- `backup`: the sub-command of `br`.
- `full`: the sub-command of `backup`.
- `-s` (or `--storage`): the option that specifies the path where the backup files are stored.
- `"local:///tmp/backup"`: the parameter of `-s`. `/tmp/backup` is the path in the local disk where the backed up files of each TiKV node are stored.
- `--pd`: the option that specifies the Placement Driver (PD) service address.
- `"${PDIP}:2379"`: the parameter of `--pd`.

#### Note:

- When the `local` storage is used, the backup data are scattered in the local file system of each node.

- It is **not recommended** to back up to a local disk in the production environment because you **have to** manually aggregate these data to complete the data restoration. For more information, see [Restore Cluster Data](#).
- Aggregating these backup data might cause redundancy and bring troubles to operation and maintenance. Even worse, if restoring data without aggregating these data, you can receive a rather confusing error message **SST file not found**.
- It is recommended to mount the NFS disk on each node, or back up to the **S3** object storage.

### Sub-commands

A **br** command consists of multiple layers of sub-commands. Currently, BR has the following three sub-commands:

- **br backup**: used to back up the data of the TiDB cluster.
- **br restore**: used to restore the data of the TiDB cluster.

Each of the above three sub-commands might still include the following three sub-commands to specify the scope of an operation:

- **full**: used to back up or restore all the cluster data.
- **db**: used to back up or restore the specified database of the cluster.
- **table**: used to back up or restore a single table in the specified database of the cluster.

### Common options

- **--pd**: used for connection, specifying the PD server address. For example, "`#${PDIP} ↵ }:2379`".
- **-h** (or **--help**): used to get help on all sub-commands. For example, `br backup -- ↵ help`.
- **-V** (or **--version**): used to check the version of BR.
- **--ca**: specifies the path to the trusted CA certificate in the PEM format.
- **--cert**: specifies the path to the SSL certificate in the PEM format.
- **--key**: specifies the path to the SSL certificate key in the PEM format.
- **--status-addr**: specifies the listening address through which BR provides statistics to Prometheus.

### 5.3.1.2.2 Use BR command-line to back up cluster data

To back up the cluster data, use the `br backup` command. You can add the `full` or `table` sub-command to specify the scope of your backup operation: the whole cluster or a single table.

Back up all the cluster data

To back up all the cluster data, execute the `br backup full` command. To get help on this command, execute `br backup full -h` or `br backup full --help`.

#### Usage example:

Back up all the cluster data to the `/tmp/backup` path of each TiKV node and write the `backupmeta` file to this path.

#### Note:

- If the backup disk and the service disk are different, it has been tested that online backup reduces QPS of the read-only online service by about 15%-25% in case of full-speed backup. If you want to reduce the impact on QPS, use `--ratelimit` to limit the rate.
- If the backup disk and the service disk are the same, the backup competes with the service for I/O resources. This might decrease the QPS of the read-only online service by more than half. Therefore, it is **highly not recommended** to back up the online service data to the TiKV data disk.

```
br backup full \
--pd "${PDIP}:2379" \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backupfull.log
```

Explanations for some options in the above command are as follows:

- `--ratelimit`: specifies the maximum speed at which a backup operation is performed (MiB/s) on each TiKV node.
- `--log-file`: specifies writing the BR log to the `backupfull.log` file.

A progress bar is displayed in the terminal during the backup. When the progress bar advances to 100%, the backup is complete. Then the BR also checks the backup data to ensure data safety. The progress bar is displayed as follows:

```
br backup full \
--pd "${PDIP}:2379" \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backupfull.log
Full Backup <-----/.....>
↪ 17.12%.
```

Back up a database

To back up a database in the cluster, execute the `br backup db` command. To get help on this command, execute `br backup db -h` or `br backup db --help`.

#### Usage example:

Back up the data of the `test` database to the `/tmp/backup` path on each TiKV node and write the `backupmeta` file to this path.

```
br backup db \
--pd "${PDIP}:2379" \
--db test \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backuptable.log
```

In the above command, `--db` specifies the name of the database to be backed up. For descriptions of other options, see [Back up all the cluster data](#).

A progress bar is displayed in the terminal during the backup. When the progress bar advances to 100%, the backup is complete. Then the BR also checks the backup data to ensure data safety.

Back up a table

To back up the data of a single table in the cluster, execute the `br backup table` command. To get help on this command, execute `br backup table -h` or `br backup table --help`.

#### Usage example:

Back up the data of the `test.usertable` table to the `/tmp/backup` path on each TiKV node and write the `backupmeta` file to this path.

```
br backup table \
--pd "${PDIP}:2379" \
--db test \
--table usertable \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backuptable.log
```

The `table` sub-command has two options:

- `--db`: specifies the database name
- `--table`: specifies the table name.

For descriptions of other options, see [Back up all cluster data](#).

A progress bar is displayed in the terminal during the backup operation. When the progress bar advances to 100%, the backup is complete. Then the BR also checks the backup data to ensure data safety.

Back up with table filter

To back up multiple tables with more complex criteria, execute the `br backup full` command and specify the `table filters` with `--filter` or `-f`.

#### Usage example:

The following command backs up the data of all tables in the form `db*.tbl*` to the `/tmp/backup` path on each TiKV node and writes the `backupmeta` file to this path.

```
br backup full \
--pd "${PDIP}:2379" \
--filter 'db*.tbl*' \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backupfull.log
```

Back up data to Amazon S3 backend

If you back up the data to the Amazon S3 backend, instead of `local` storage, you need to specify the S3 storage path in the `storage` sub-command, and allow the BR node and the TiKV node to access Amazon S3.

You can refer to the [AWS Official Document](#) to create an `S3 Bucket` in the specified `Region`. You can also refer to another [AWS Official Document](#) to create a `Folder` in the `Bucket`.

#### Note:

To complete one backup, TiKV and BR usually require the minimum privileges of `s3>ListBucket`, `s3.PutObject`, and `s3AbortMultipartUpload`.

Pass `SecretKey` and `AccessKey` of the account that has privilege to access the S3 backend to the BR node. Here `SecretKey` and `AccessKey` are passed as environment variables. Then pass the privilege to the TiKV node through BR.

```
export AWS_ACCESS_KEY_ID=${AccessKey}
export AWS_SECRET_ACCESS_KEY=${SecretKey}
```

When backing up using BR, explicitly specify the parameters `--s3.region` and `--send-credentials-to-tikv`. `--s3.region` indicates the region where S3 is located, and `--send-credentials-to-tikv` means passing the privilege to access S3 to the TiKV node.

```
br backup full \
 --pd "${PDIP}:2379" \
 --storage "s3://${Bucket}/${Folder}" \
 --s3.region "${region}" \
 --send-credentials-to-tikv=true \
 --ratelimit 128 \
 --log-file backuptable.log
```

Back up incremental data

If you want to back up incrementally, you only need to specify the `last backup timestamp` `--lastbackups`.

The incremental backup has two limitations:

- The incremental backup needs to be under a different path from the previous full backup.
- GC (Garbage Collection) safepoint must be before the `lastbackups`.

To back up the incremental data between `(LAST_BACKUP_TS, current PD timestamp]`, execute the following command:

```
br backup full\
 --pd ${PDIP}:2379 \
 --ratelimit 128 \
 -s local:///home/tidb/backupdata/incr \
 --lastbackups ${LAST_BACKUP_TS}
```

To get the timestamp of the last backup, execute the `validate` command. For example:

```
LAST_BACKUP_TS=`br validate decode --field="end-version" -s local:///home/
 ↪ tidb/backupdata | tail -n1`
```

In the above example, for the incremental backup data, BR records the data changes and the DDL operations during `(LAST_BACKUP_TS, current PD timestamp]`. When restoring data, BR first restores DDL operations and then the data.

Encrypt data during backup (experimental feature)

Since TiDB v5.3.0, TiDB supports backup encryption. You can configure the following parameters to encrypt data during backup:

- **--crypter.method**: Encryption algorithm. Supports three algorithms `aes128-ctr`/  
↪ `aes192-ctr/aes256-ctr`. The default value is `plaintext` and indicates no encryption.
- **--crypter.key**: Encryption key in hexadecimal string format. `aes128-ctr` means 128 bit (16 bytes) key length, `aes192-ctr` means 24 bytes and `aes256-ctr` means 32 bytes.
- **--crypter.key-file**: The key file. You can directly pass in the file path where the key is stored as a parameter without passing in “crypter.key”

**Warning:**

- This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.
- If the key is lost, the backup data cannot be restored to the cluster.
- The encryption feature needs to be used on BR tools and TiDB clusters v5.3.0 or later versions, and the encrypted backup data cannot be restored on clusters earlier than v5.3.0.

The configuration example for backup encryption is as follows:

```
br backup full\
 --pd ${PDIP}:2379 \
 -s local:///home/tidb/backupdata/incr \
 --crypter.method aes128-ctr \
 --crypter.key 0123456789abcdef0123456789abcdef
```

Point-in-time recovery (experimental feature)

Point-in-time recovery (PITR) allows you to restore data to a point in time of your choice.

An example scenario would be to take a full backup every day and take incremental backups every 6 hours and then use TiCDC for PITR. Assume that on one day, the full backup was performed at 00:00 and the first incremental backup was performed at 06:00. If you want to restore the database to the state of 07:16, you can first restore the full backup (taken at 00:00) and the incremental backup (taken at 06:00), and then restore TiCDC logs that fill in the gap between 06:00 and 07:16.

To perform the PITR, you can take the following steps:

1. Restore a full backup using `br restore full`.
2. (optional) Restore incremental backup(s).

3. Use `br restore cdclog` to restore the transactions that happened after the last incremental backup. The complete command to execute is as follows:

```
br restore cdclog --storage local:///data/cdclog --start-ts $START_TS
 ↪ --end-ts $END_TS
```

In the command above:

- `local:///data/cdclog` is the location of the TiCDC logs. This might be on the local filesystem or on the external storage like S3.
- `$START_TS` is the end position of the restore from the last restored backup (either a full backup or an incremental backup).
- `$END_TS` is the point to which you want to restore your data.

Back up Raw KV (experimental feature)

#### Warning:

This feature is experimental and not thoroughly tested. It is highly **not recommended** to use this feature in the production environment.

In some scenarios, TiKV might run independently of TiDB. Given that, BR also supports bypassing the TiDB layer and backing up data in TiKV.

For example, you can execute the following command to back up all keys between `[0x31, 0x3130303030303030]` in the default CF to `$BACKUP_DIR`:

```
br backup raw --pd $PD_ADDR \
 -s "local://$BACKUP_DIR" \
 --start 31 \
 --ratelimit 128 \
 --end 3130303030303030 \
 --format hex \
 --cf default
```

Here, the parameters of `--start` and `--end` are decoded using the method specified by `--format` before being sent to TiKV. Currently, the following methods are available:

- “raw”: The input string is directly encoded as a key in binary format.
- “hex”: The default encoding method. The input string is treated as a hexadecimal number.
- “escape”: First escape the input string, and then encode it into binary format.

### 5.3.1.2.3 Use BR command-line to restore cluster data

To restore the cluster data, use the `br restore` command. You can add the `full`, `db` or `table` sub-command to specify the scope of your restoration: the whole cluster, a database or a single table.

#### Note:

If you use the local storage, you **must** copy all back up SST files to every TiKV node in the path specified by `--storage`.

Even if each TiKV node eventually only need to read a part of the all SST files, they all need full access to the complete archive because:

- Data are replicated into multiple peers. When ingesting SSTs, these files have to be present on *all* peers. This is unlike back up where reading from a single node is enough.
- Where each peer is scattered to during restore is random. We don't know in advance which node will read which file.

These can be avoided using shared storage, for example mounting an NFS on the local path, or using S3. With network storage, every node can automatically read every SST file, so these caveats no longer apply.

Also, note that you can only run one restore operation for a single cluster at the same time. Otherwise, unexpected behaviors might occur. For details, see [FAQ](#).

#### Restore all the backup data

To restore all the backup data to the cluster, execute the `br restore full` command. To get help on this command, execute `br restore full -h` or `br restore full --help`.

#### Usage example:

Restore all the backup data in the `/tmp/backup` path to the cluster.

```
br restore full \
--pd "${PDIP}:2379" \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file restorefull.log
```

Explanations for some options in the above command are as follows:

- `--ratelimit`: specifies the maximum speed at which a restoration operation is performed (MiB/s) on each TiKV node.

- `--log-file`: specifies writing the BR log to the `restorefull.log` file.

A progress bar is displayed in the terminal during the restoration. When the progress bar advances to 100%, the restoration is complete. Then the BR also checks the backup data to ensure data safety.

```
br restore full \
 --pd "${PDIP}:2379" \
 --storage "local:///tmp/backup" \
 --ratelimit 128 \
 --log-file restorefull.log
Full Restore <-----/.....>
↪ 17.12%.
```

### Restore a database

To restore a database to the cluster, execute the `br restore db` command. To get help on this command, execute `br restore db -h` or `br restore db --help`.

#### Usage example:

Restore a database backed up in the `/tmp/backup` path to the cluster.

```
br restore db \
 --pd "${PDIP}:2379" \
 --db "test" \
 --ratelimit 128 \
 --storage "local:///tmp/backup" \
 --log-file restorefull.log
```

In the above command, `--db` specifies the name of the database to be restored. For descriptions of other options, see [Restore all backup data](#)).

#### Note:

When you restore the backup data, the name of the database specified by `--db` must be the same as the one specified by `--db` in the backup command. Otherwise, the restore fails. This is because the metafile of the backup data (`backupmeta` file) records the database name, you can only restore data to the database with the same name. The recommended method is to restore the backup data to the database with the same name in another cluster.

### Restore a table

To restore a single table to the cluster, execute the `br restore table` command. To get help on this command, execute `br restore table -h` or `br restore table --help`.

### Usage example:

Restore a table backed up in the `/tmp/backup` path to the cluster.

```
br restore table \
--pd "${PDIP}:2379" \
--db "test" \
--table "usertable" \
--ratelimit 128 \
--storage "local:///tmp/backup" \
--log-file restorefull.log
```

In the above command, `--table` specifies the name of the table to be restored. For descriptions of other options, see [Restore all backup data](#) and [Restore a database](#).

### Restore with table filter

To restore multiple tables with more complex criteria, execute the `br restore full` command and specify the `table filters` with `--filter` or `-f`.

### Usage example:

The following command restores a subset of tables backed up in the `/tmp/backup` path to the cluster.

```
br restore full \
--pd "${PDIP}:2379" \
--filter 'db*.tbl*' \
--storage "local:///tmp/backup" \
--log-file restorefull.log
```

### Restore data from Amazon S3 backend

If you restore data from the Amazon S3 backend, instead of `local` storage, you need to specify the S3 storage path in the `storage` sub-command, and allow the BR node and the TiKV node to access Amazon S3.

#### Note:

To complete one restore, TiKV and BR usually require the minimum privileges of `s3>ListBucket` and `s3GetObject`.

Pass `SecretKey` and `AccessKey` of the account that has privilege to access the S3 backend to the BR node. Here `SecretKey` and `AccessKey` are passed as environment variables. Then pass the privilege to the TiKV node through BR.

```
export AWS_ACCESS_KEY_ID=${AccessKey}
export AWS_SECRET_ACCESS_KEY=${SecretKey}
```

When restoring data using BR, explicitly specify the parameters `--s3.region` and `--send-credentials-to-tikv`. `--s3.region` indicates the region where S3 is located, and `--send-credentials-to-tikv` means passing the privilege to access S3 to the TiKV node.

`Bucket` and `Folder` in the `--storage` parameter represent the S3 bucket and the folder where the data to be restored is located.

```
br restore full \
 --pd "${PDIP}:2379" \
 --storage "s3://${Bucket}/${Folder}" \
 --s3.region "${region}" \
 --ratelimit 128 \
 --send-credentials-to-tikv=true \
 --log-file restorefull.log
```

In the above command, `--table` specifies the name of the table to be restored. For descriptions of other options, see [Restore a database](#).

#### Restore incremental data

Restoring incremental data is similar to [restoring full data using BR](#). Note that when restoring incremental data, make sure that all the data backed up before `last backup ts` has been restored to the target cluster.

#### Restore tables created in the `mysql` schema (experimental feature)

BR backs up tables created in the `mysql` schema by default.

When you restore data using BR, the tables created in the `mysql` schema are not restored by default. If you need to restore these tables, you can explicitly include them using the [table filter](#). The following example restores `mysql.usertable` created in `mysql` schema. The command restores `mysql.usertable` along with other data.

```
br restore full -f '*.*' -f '!mysql.*' -f 'mysql.usertable' -s
 ↪ $external_storage_url --ratelimit 128
```

In the above command, `-f '*.*'` is used to override the default rules and `-f '!mysql.*'` instructs BR not to restore tables in `mysql` unless otherwise stated. `-f 'mysql.usertable'` indicates that `mysql.usertable` is required for restore. For detailed implementation, refer to the [table filter document](#).

If you only need to restore `mysql.usertable`, use the following command:

```
br restore full -f 'mysql.usertable' -s $external_storage_url --ratelimit
 ↪ 128
```

**Warning:**

Although you can back up and restore system tables (such as `mysql.tidb`) using the BR tool, some unexpected situations might occur after the restore, including:

- the statistical information tables (`mysql.stat_*`) cannot be restored.
- the system variable tables (`mysql.tidb`,`mysql.global_variables`) cannot be restored.
- the user information tables (such as `mysql.user` and `mysql.columns_priv`) cannot be restored.
- GC data cannot be restored.

Restoring system tables might cause more compatibility issues. To avoid unexpected issues, **DO NOT** restore system tables in the production environment.

Decrypt data during restore (experimental feature)

**Warning:**

This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

After encrypting the backup data, you need to pass in the corresponding decryption parameters to restore the data. You need to ensure that the decryption parameters and encryption parameters are consistent. If the decryption algorithm or key is incorrect, the data cannot be restored.

The following is an example of decrypting the backup data:

```
br restore full\
 --pd ${PDIP}:2379 \
 -s local:///home/tidb/backupdata/incr \
 --crypter.method aes128-ctr \
 --crypter.key 0123456789abcdef0123456789abcdef
```

Restore Raw KV (experimental feature)

**Warning:**

This feature is in the experiment, without being thoroughly tested. It is highly **not recommended** to use this feature in the production environment.

Similar to [backing up Raw KV](#), you can execute the following command to restore Raw KV:

```
br restore raw --pd $PD_ADDR \
-s "local://$BACKUP_DIR" \
--start 31 \
--end 3130303030303030 \
--ratelimit 128 \
--format hex \
--cf default
```

In the above example, all the backed up keys in the range [0x31, 0x3130303030303030 ↵ ) are restored to the TiKV cluster. The coding methods of these keys are identical to that of [keys during the backup process](#)

Online restore (experimental feature)

#### Warning:

This feature is in the experiment, without being thoroughly tested. It also relies on the unstable [Placement Rules](#) feature of PD. It is highly **not recommended** to use this feature in the production environment.

During data restoration, writing too much data affects the performance of the online cluster. To avoid this effect as much as possible, BR supports [Placement rules](#) to isolate resources. In this case, downloading and importing SST are only performed on a few specified nodes (or “restore nodes” for short). To complete the online restore, take the following steps.

1. Configure PD, and start Placement rules:

```
echo "config set enable-placement-rules true" | pd-ctl
```

2. Edit the configuration file of the “restore node” in TiKV, and specify “restore” to the `server` configuration item:

```
[server]
labels = { exclusive = "restore" }
```

3. Start TiKV of the “restore node” and restore the backed up files using BR. Compared with the offline restore, you only need to add the `--online` flag:

```
br restore full \
-s "local://$BACKUP_DIR" \
--ratelimit 128 \
```

```
--pd $PD_ADDR \
--online
```

### 5.3.1.3 BR Use Cases

[BR](#) is a tool for distributed backup and restoration of the TiDB cluster data.

This document describes how to run BR in the following use cases:

- Back up a single table to a network disk (recommended in production environment)
- Restore data from a network disk (recommended in production environment)
- Back up a single table to a local disk (recommended in testing environment)
- Restore data from a local disk (recommended in testing environment)

This document aims to help you achieve the following goals:

- Back up and restore data using a network disk or local disk correctly.
- Get the status of a backup or restoration operation through monitoring metrics.
- Learn how to tune performance during the operation.
- Troubleshoot the possible anomalies during the backup operation.

#### 5.3.1.3.1 Audience

You are expected to have a basic understanding of TiDB and [TiKV](#).

Before reading on, make sure you have read [BR Tool Overview](#), especially [Usage Restrictions](#) and [Best Practices](#).

#### 5.3.1.3.2 Prerequisites

This section introduces the recommended method of deploying TiDB, cluster versions, the hardware information of the TiKV cluster, and the cluster configuration for the use case demonstrations.

You can estimate the performance of your backup or restoration operation based on your own hardware and configuration.

Deployment method

It is recommended that you deploy the TiDB cluster using [TiUP](#) and get BR by downloading [TiDB Toolkit](#).

Cluster versions

- TiDB: v5.0.0
- TiKV: v5.0.0
- PD: v5.0.0

- BR: v5.0.0

**Note:**

v5.0.0 was the latest version at the time this document was written. It is recommended that you use the latest version of TiDB/TiKV/PD/BR and make sure that the BR version is **consistent with** the TiDB version.

### TiKV hardware information

- Operating system: CentOS Linux release 7.6.1810 (Core)
- CPU: 16-Core Common KVM processor
- RAM: 32GB
- Disk: 500G SSD \* 2
- NIC: 10 Gigabit network card

### Cluster configuration

BR directly sends commands to the TiKV cluster and are not dependent on the TiDB server, so you do not need to configure the TiDB server when using BR.

- TiKV: default configuration
- PD: default configuration

#### 5.3.1.3.3 Use cases

This document describes the following use cases:

- Back up a single table to a network disk (recommended in production environment)
- Restore data from a network disk (recommended in production environment)
- Back up a single table to a local disk (recommended in testing environment)
- Restore data from a local disk (recommended in testing environment)

It is recommended that you use a network disk to back up and restore data. This spares you from collecting backup files and greatly improves the backup efficiency especially when the TiKV cluster is in a large scale.

Before the backup or restoration operations, you need to do some preparations:

- Preparation for backup
- Preparation for restoration

## Preparation for backup

The BR tool already supports self-adapting to GC. It automatically registers `backupTS` (the latest PD timestamp by default) to PD's `safePoint` to ensure that TiDB's GC Safe Point does not move forward during the backup, thus avoiding manually setting GC configurations.

For the detailed usage of the `br backup` command, refer to [Use BR Command-line for Backup and Restoration](#).

1. Before executing the `br backup` command, ensure that no DDL is running on the TiDB cluster.
2. Ensure that the storage device where the backup will be created has sufficient space.

## Preparation for restoration

Before executing the `br restore` command, check the new cluster to make sure that the table in the cluster does not have a duplicate name.

Back up a single table to a network disk (recommended in production environment)

Use the `br backup` command to back up the single table data `--db batchmark --table order_line` to the specified path `local:///br_data` in the network disk.

## Backup prerequisites

- [Preparation for backup](#)
- Configure a high-performance SSD hard disk host as the NFS server to store data, and all BR nodes, TiKV nodes, and TiFlash nodes as NFS clients. Mount the same path (for example, `/br_data`) to the NFS server for NFS clients to access the server.
- The total transfer rate between the NFS server and all NFS clients must reach at least `the number of TiKV instances * 150MB/s`. Otherwise the network I/O might become the performance bottleneck.

### Note:

- During data backup, because only the data of leader replicas are backed up, even if there is a TiFlash replica in the cluster, BR can complete the backup without mounting TiFlash nodes.
- When restoring data, BR will restore the data of all replicas. Also, TiFlash nodes need access to the backup data for BR to complete the restore. Therefore, before the restore, you must mount TiFlash nodes to the NFS server.

## Topology

The following diagram shows the typology of BR:

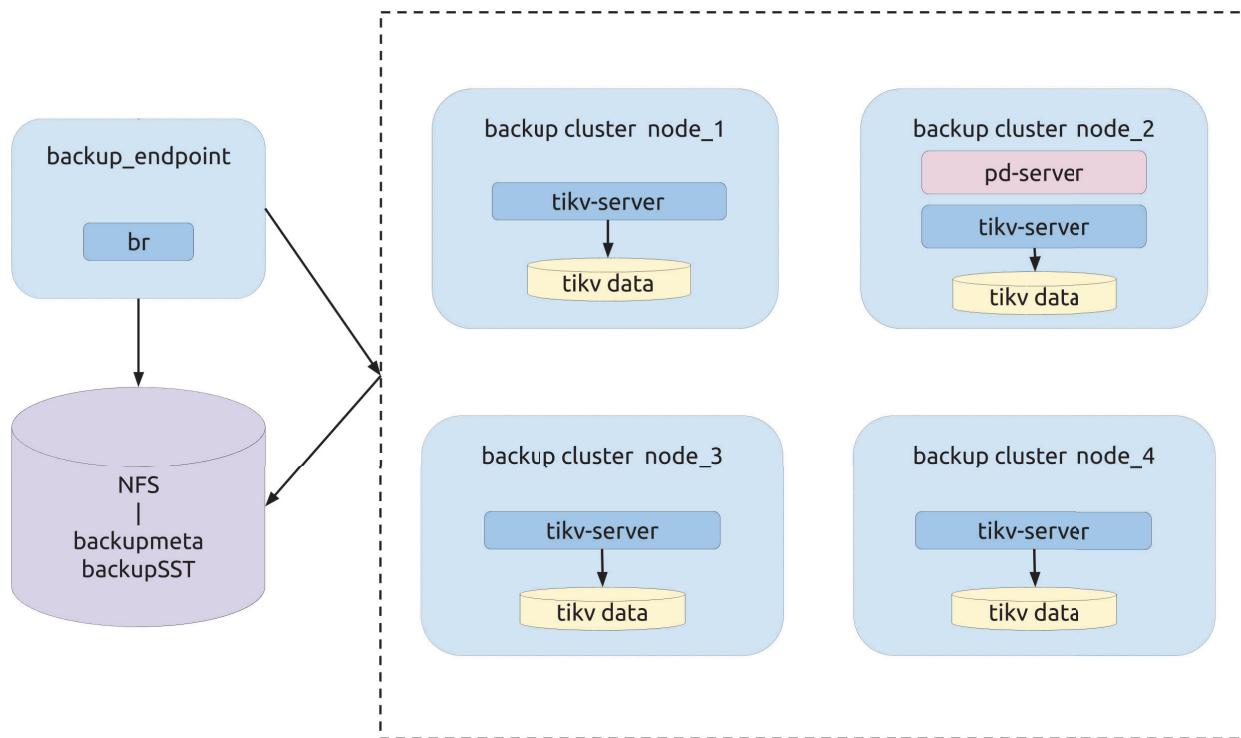


Figure 11: img

## Backup operation

Before the backup operation, execute the `admin checksum table order_line` command to get the statistical information of the table to be backed up (`--db batchmark` ↪ `--table order_line`). The following image shows an example of this information:

| Db_name   | Table_name | Checksum_crc64_xor   | Total_kvs  | Total_bytes  |
|-----------|------------|----------------------|------------|--------------|
| batchmark | order_line | 10912722838344822475 | 5659888624 | 370385538778 |

1 row in set (5 min 47.59 sec)

Figure 12: img

Execute the `br backup` command:

```
bin/br backup table \
--db batchmark \
--table order_line \
-s local:///br_data \
--pd ${PD_ADDR}:2379 \
--log-file backup-nfs.log
```

Monitoring metrics for the backup

During the backup process, pay attention to the following metrics on the monitoring panels to get the status of the backup process.

**Backup CPU Utilization:** the CPU usage rate of each working TiKV node in the backup operation (for example, backup-worker and backup-endpoint).

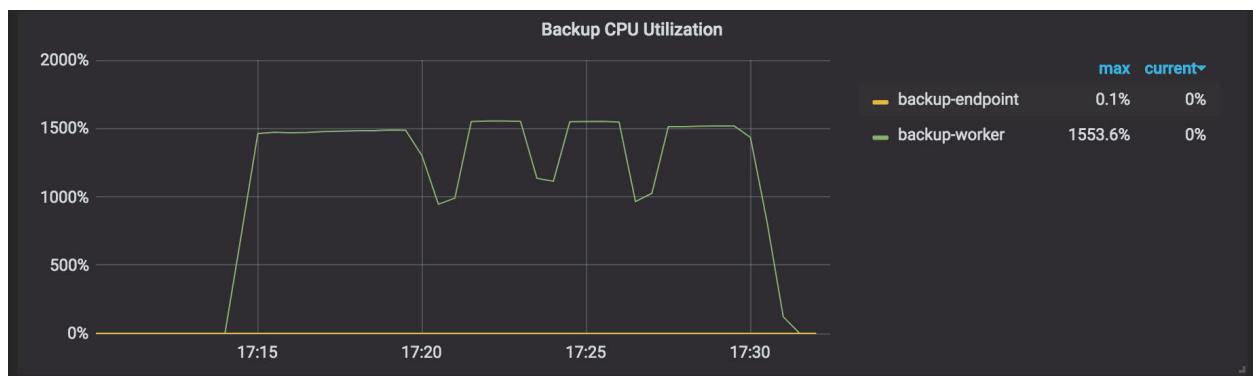


Figure 13: img

**IO Utilization:** the I/O usage rate of each working TiKV node in the backup operation.

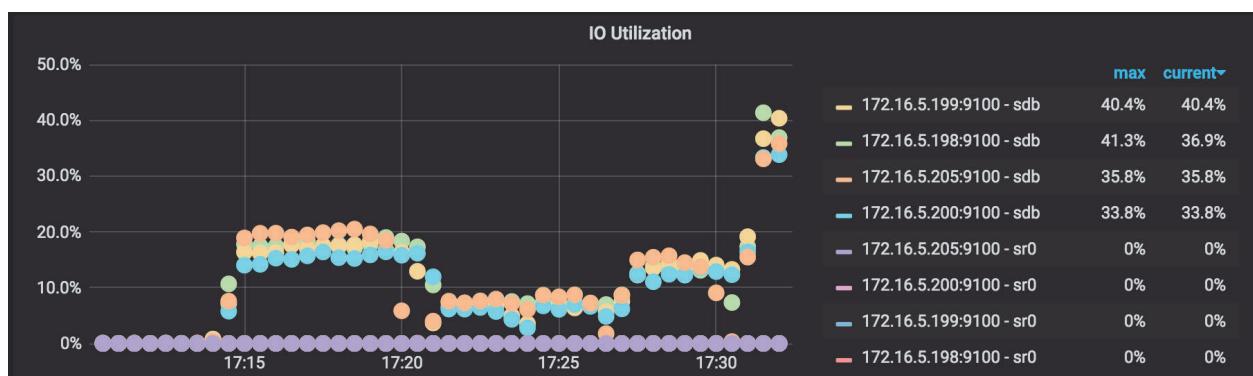


Figure 14: img

**BackupSST Generation Throughput:** the backupSST generation throughput of each working TiKV node in the backup operation, which is normally around 150MB/s.

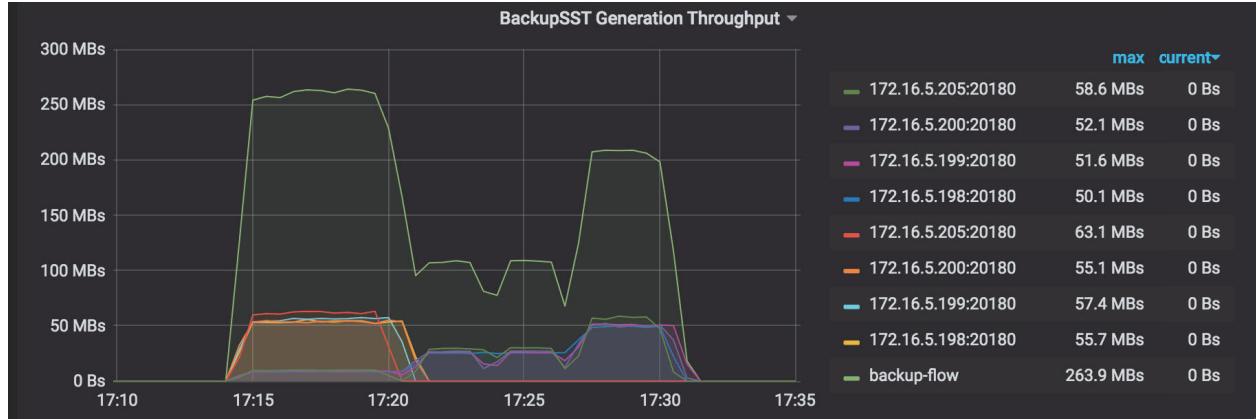


Figure 15: img

**One Backup Range Duration:** the duration of backing up a range, which is the total time cost of scanning KVs and storing the range as the backupSST file.

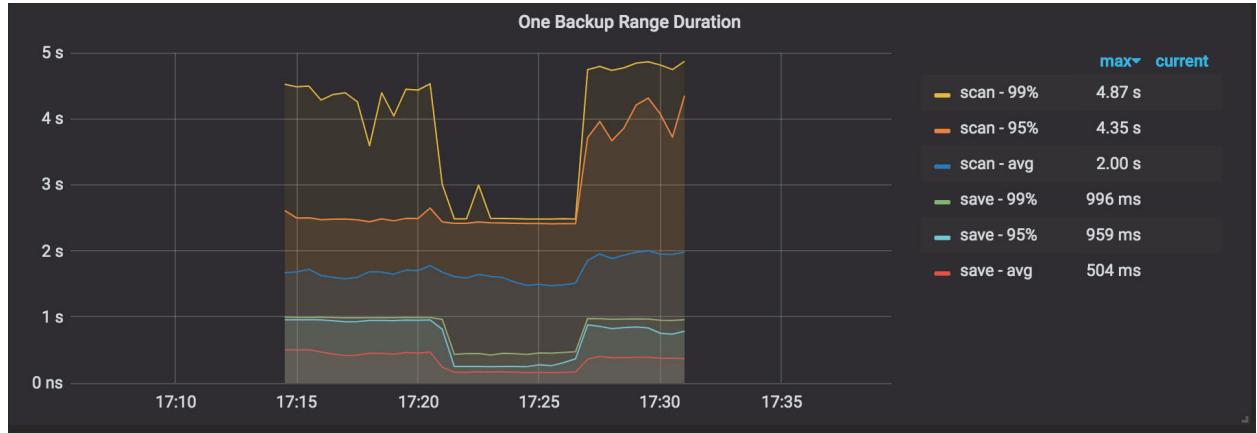


Figure 16: img

**One Backup Subtask Duration:** the duration of each sub-task into which a backup task is divided.

#### Note:

- In this task, the single table to be backed up has three indexes and the task is normally divided into four sub-tasks.

- The panel in the following image has thirteen points on it, which means nine (namely, 13-4) retries. Region scheduling might occur during the backup process, so a few retries is normal.

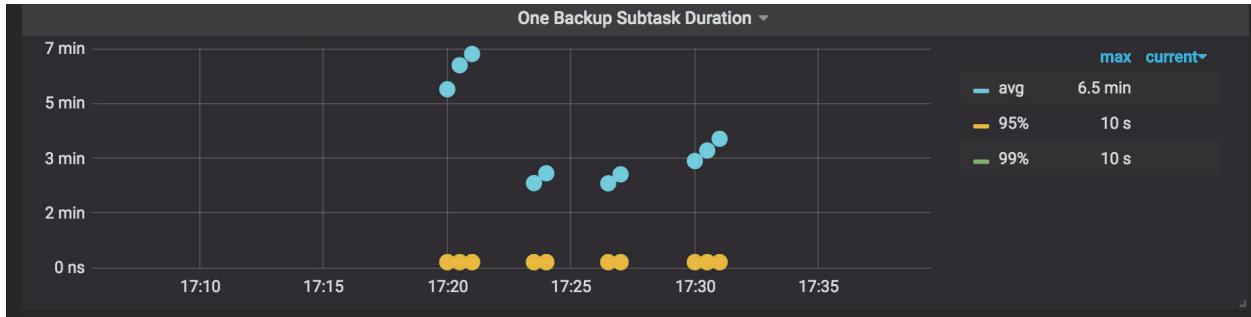


Figure 17: img

**Backup Errors:** the errors occurred during the backup process. No error occurs in normal situations. Even if a few errors occur, the backup operation has the retry mechanism which might increase the backup time but does not affect the operation correctness.

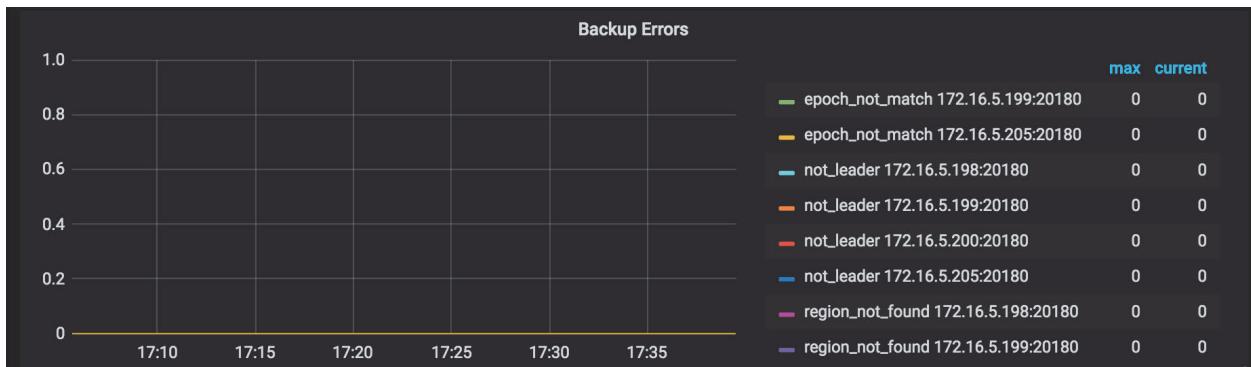


Figure 18: img

**Checksum Request Duration:** the duration of the admin checksum request in the backup cluster.

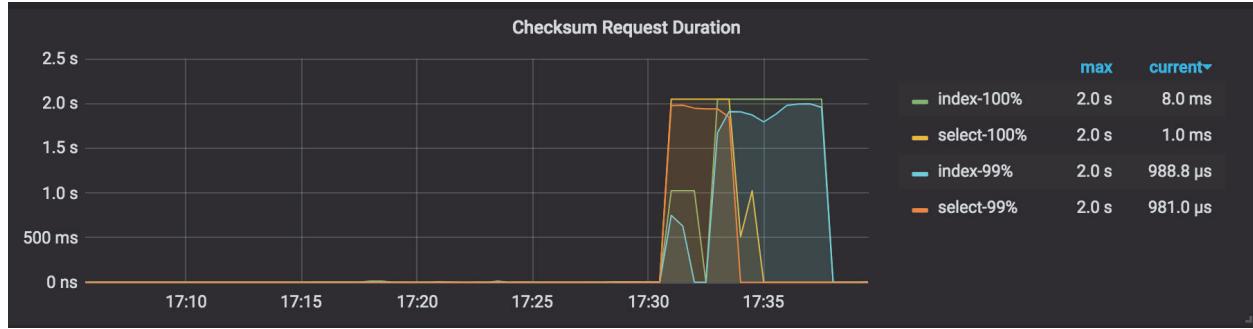


Figure 19: img

### Backup results explanation

When finishing the backup, BR outputs the backup summary to the console.

Before executing the backup command, a path in which the log is stored has been specified. You can get the statistical information of the backup operation from this log. Search “summary” in this log, you can see the following information:

```
["Full backup Success summary:
 total backup ranges: 2,
 total success: 2,
 total failed: 0,
 total take(Full backup time): 31.802912166s,
 total take(real time): 49.799662427s,
 total size(MB): 5997.49,
 avg speed(MB/s): 188.58,
 total kv: 120000000"]
 ["backup checksum"=17.907153678s]
 ["backup fast checksum"=349.333µs]
 ["backup total regions"=43]
 [BackupTS=422618409346269185]
 [Size=826765915]
```

The above log includes the following information:

- Backup duration: `total take(Full backup time): 31.802912166s`
- Total runtime of the application: `total take(real time): 49.799662427s`
- Backup data size: `total size(MB): 5997.49`
- Backup throughput: `avg speed(MB/s): 188.58`
- Number of backed-up KV pairs: `total kv: 120000000`
- Backup checksum duration: `["backup checksum"=17.907153678s]`
- Total duration of calculating the checksum, KV pairs, and bytes of each table: `["→ backup fast checksum"=349.333µs]`

- Total number of backup Regions: `["backup total regions"=43]`
- The actual size of the backup data in the disk after compression: `[Size=826765915]`
- Snapshot timestamp of the backup data: `[BackupTS=422618409346269185]`

From the above information, the throughput of a single TiKV instance can be calculated:  
 $\text{avg speed(MB/s)}/\text{tikv\_count} = 62.86$ .

#### Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the backup process (for example, in the [Monitoring metrics for the backup](#), the highest CPU usage rate of backup-worker is around 1500% and the overall I/O usage rate is below 30%), you can try to increase the value of `--concurrency` (4 by default) to tune the performance. But this performance tuning method is not suitable for the use cases of many small tables. See the following example:

```
bin/br backup table \
 --db batchmark \
 --table order_line \
 -s local:///br_data/ \
 --pd ${PD_ADDR}:2379 \
 --log-file backup-nfs.log \
 --concurrency 16
```

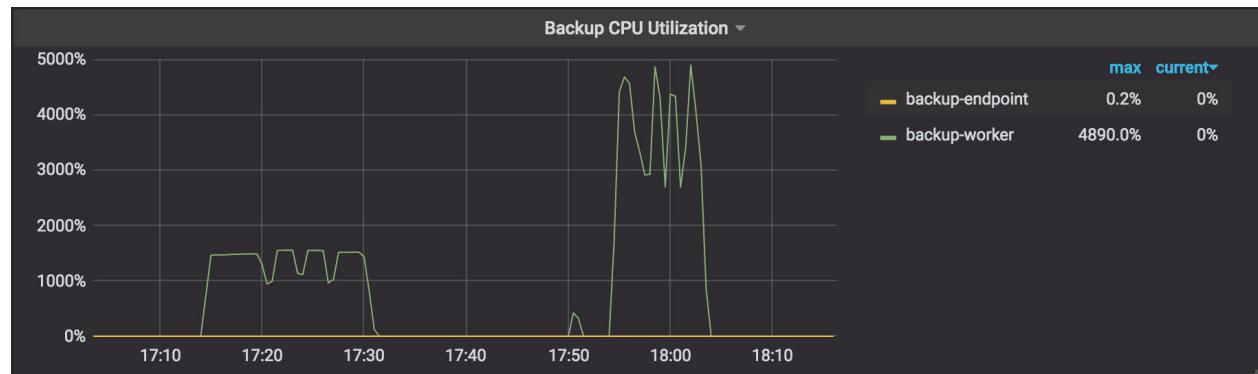


Figure 20: img

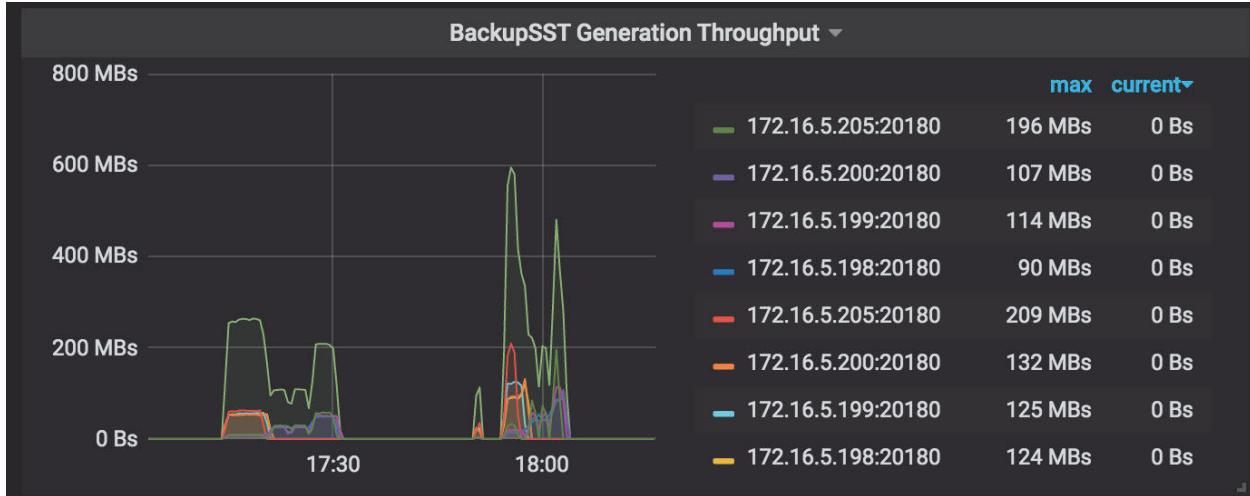


Figure 21: img

The tuned performance results are as follows (with the same data size):

- Backup duration: `total take(s)` reduced from 986.43 to 535.53
- Backup throughput: `avg speed(MB/s)` increased from 358.09 to 659.59
- Throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` increased from 89 to 164.89

Restore data from a network disk (recommended in production environment)

Use the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

Restoration prerequisites

- Preparation for restoration

Topology

The following diagram shows the typology of BR:

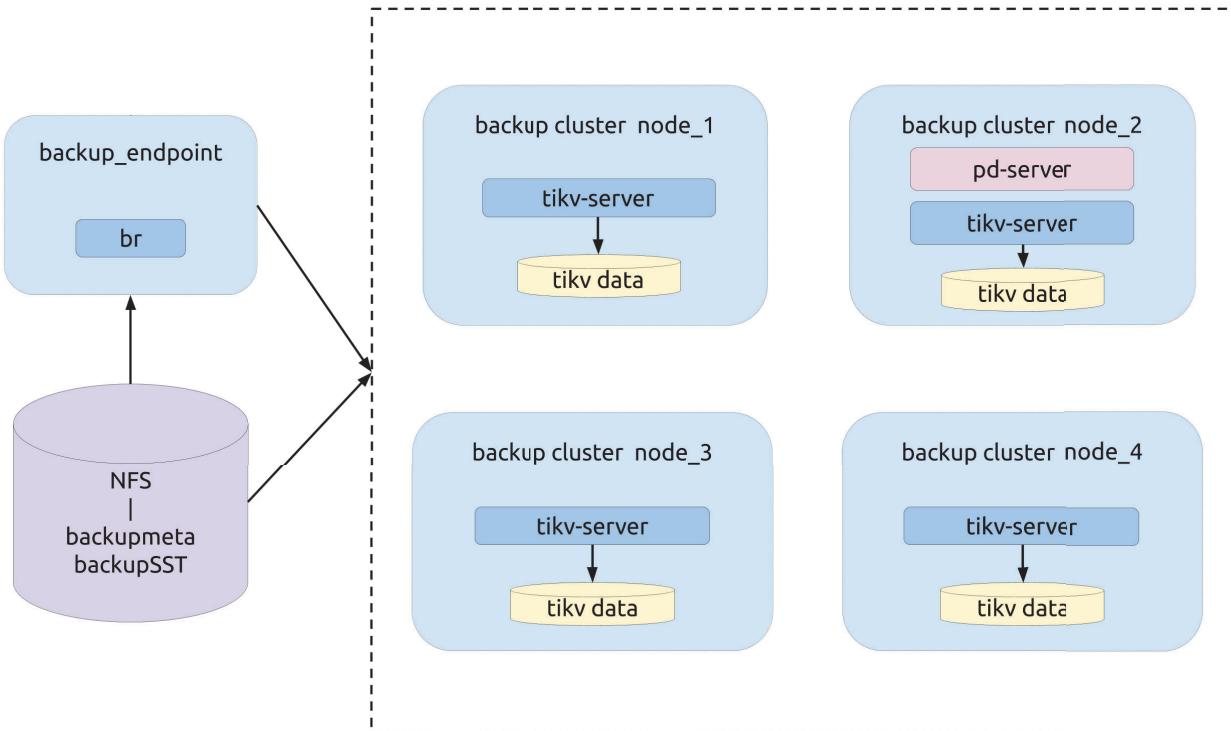


Figure 22: img

### Restoration operation

Before the restoration, refer to [Preparation for restoration](#) for the preparation.

Execute the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data
↪ --pd 172.16.5.198:2379 --log-file restore-nfs.log
```

### Monitoring metrics for the restoration

During the restoration process, pay attention to the following metrics on the monitoring panels to get the status of the restoration process.

**CPU Utilization:** the CPU usage rate of each working TiKV node in the restoration operation.

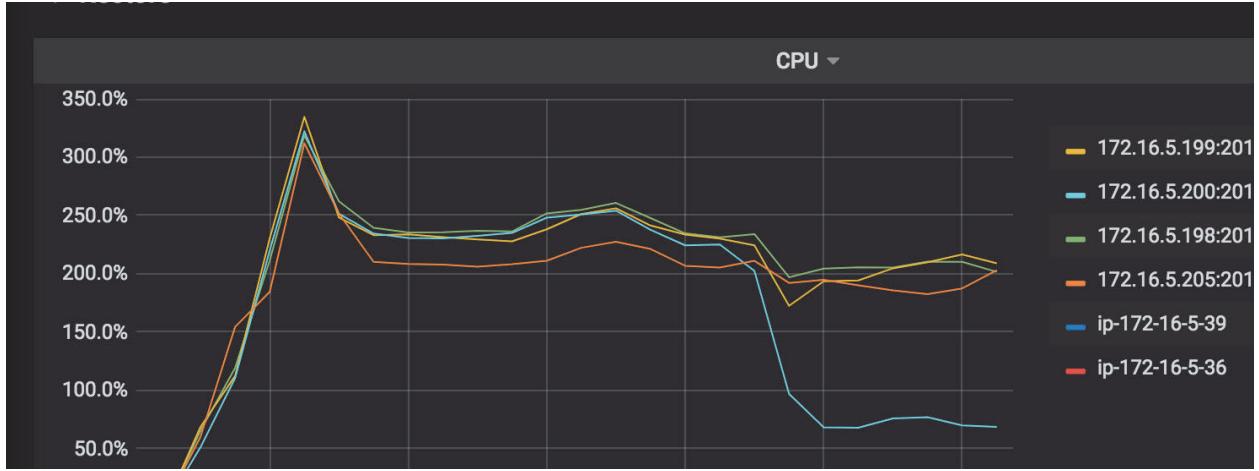


Figure 23: img

**IO Utilization:** the I/O usage rate of each working TiKV node in the restoration operation.



Figure 24: img

**Region:** the Region distribution. The more even Regions are distributed, the better the restoration resources are used.

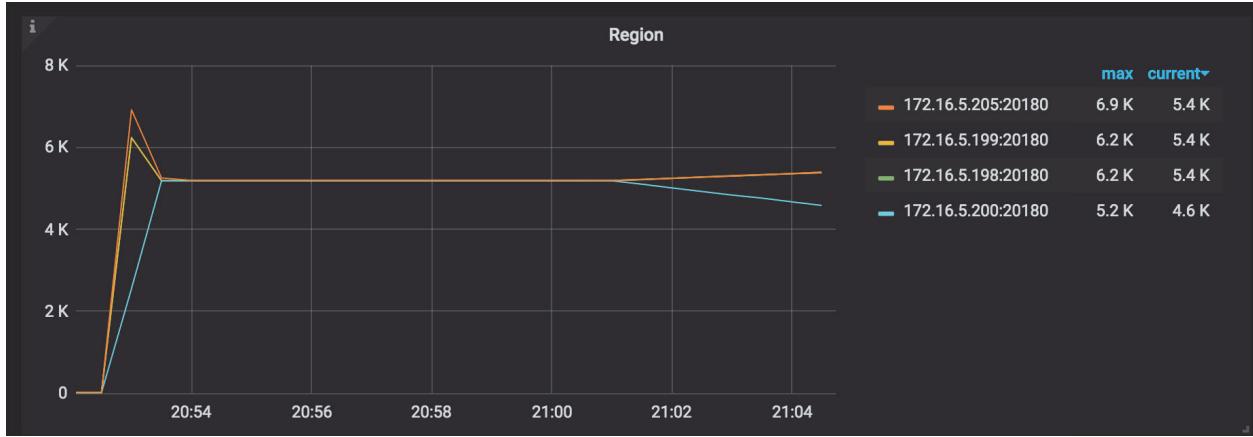


Figure 25: img

**Process SST Duration:** the delay of processing the SST files. When restoring a table, if `tableID` is changed, you need to rewrite `tableID`. Otherwise, `tableID` is renamed. Generally, the delay of rewriting is longer than that of renaming.



Figure 26: img

**DownLoad SST Throughput:** the throughput of downloading SST files from External Storage.

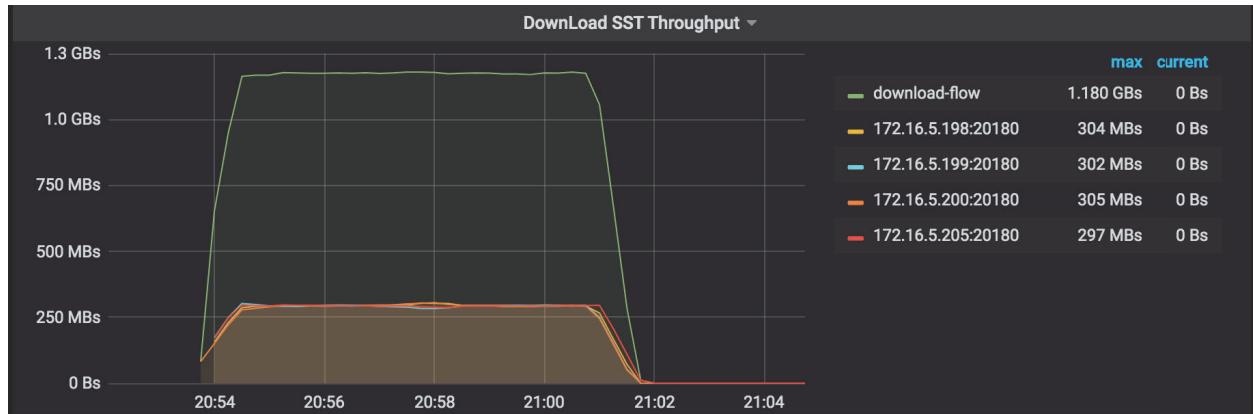


Figure 27: img

**Restore Errors:** the errors occurred during the restoration process.

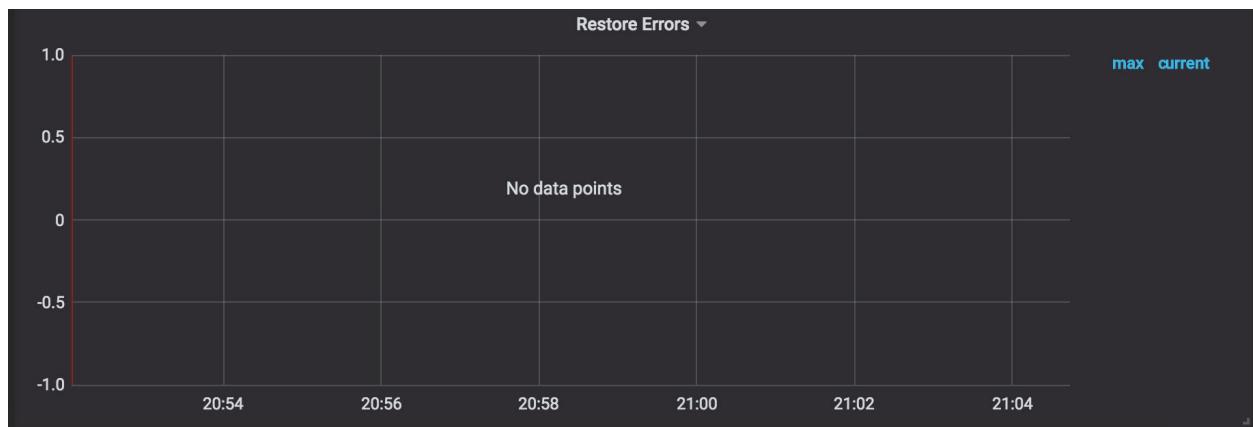


Figure 28: img

**Checksum Request duration:** the duration of the admin checksum request. This duration for the restoration is longer than that for the backup.

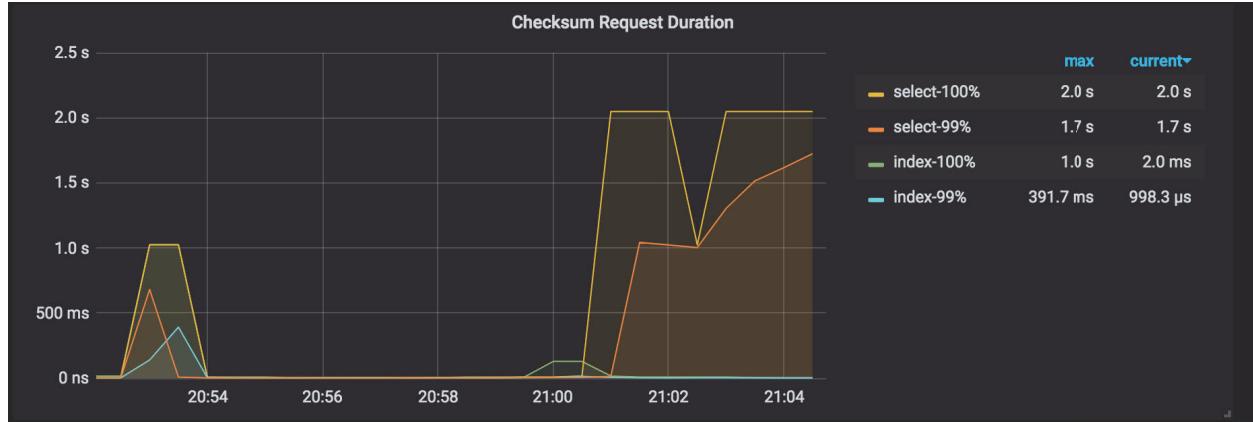


Figure 29: img

### Restoration results explanation

Before executing the restoration command, a path in which the log is stored has been specified. You can get the statistical information of the restoration operation from this log. Search “summary” in this log, you can see the following information:

```
["Table Restore summary:
total restore tables: 1,
total success: 1,
total failed: 0,
total take(Full restore time): 17m1.001611365s,
total take(real time): 16m1.371611365s,
total kv: 5659888624,
total size(MB): 353227.18,
avg speed(MB/s): 367.42"]
[{"restore files":9263]
[{"restore ranges":6888]
[{"split region":49.049182743s]
[{"restore checksum":6m34.879439498s]
[Size=48693068713]
```

The above log includes the following information:

- Restore duration: `total take(Full restore time): 17m1.001611365s`
- Total runtime of the application: `total take(real time): 16m1.371611365s`
- Restore data size: `total size(MB): 353227.18`
- Restore KV pair number: `total kv: 5659888624`
- Restore throughput: `avg speed(MB/s): 367.42`
- Region Split duration: `take=49.049182743s`
- Restore checksum duration: `restore checksum=6m34.879439498s`

- The actual size of the restored data in the disk: [Size=48693068713]

From the above information, the following items can be calculated:

- The throughput of a single TiKV instance:  $\text{avg speed(MB/s)}/\text{tikv\_count} = 91.8$
- The average restore speed of a single TiKV instance:  $\text{total size(MB)}/(\text{split time} + \text{restore time})/\text{tikv\_count} = 87.4$

### Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the restore process, you can try to increase the value of `--concurrency` which is 128 by default. See the following example:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data/
 ↪ --pd 172.16.5.198:2379 --log-file restore-concurrency.log --
 ↪ concurrency 1024
```

The tuned performance results are as follows (with the same data size):

- Restore duration: `total take(s)` reduced from 961.37 to 443.49
- Restore throughput: `avg speed(MB/s)` increased from 367.42 to 796.47
- Throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` increased from 91.8 to 199.1
- Average restore speed of a single TiKV instance: `total size(MB)/(split time + restore time)/tikv_count` increased from 87.4 to 162.3

Back up a single table to a local disk (recommended in testing environment)

Use the `br backup` command to back up the single table `--db batchmark --table order_line` to the specified path `local:///home/tidb/backup_local` in the local disk.

### Backup prerequisites

- **Preparation for backup**
- Each TiKV node has a separate disk to store the `backupSST` file.
- The `backup_endpoint` node has a separate disk to store the `backupmeta` file.
- TiKV and the `backup_endpoint` node must have the same directory for the backup (for example, `/home/tidb/backup_local`).

### Topology

The following diagram shows the typology of BR:

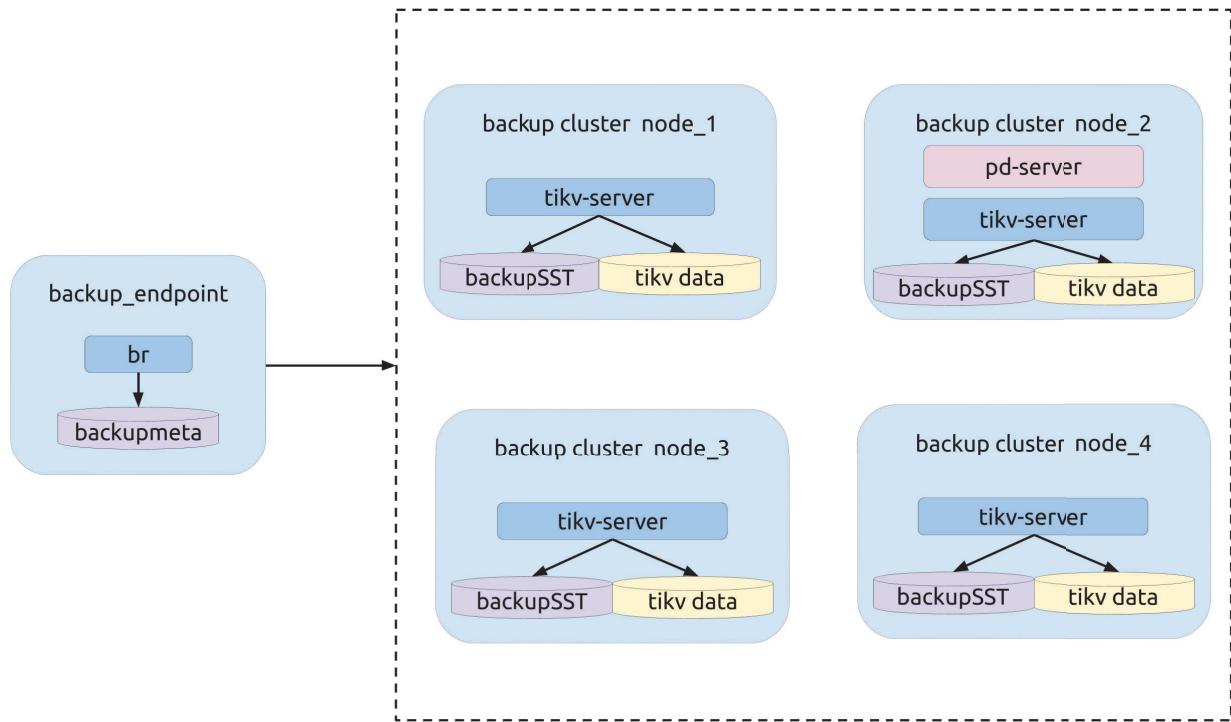


Figure 30: img

### Backup operation

Before the backup operation, execute the `admin checksum table order_line` command to get the statistical information of the table to be backed up (`--db batchmark`  $\rightarrow$  `--table order_line`). The following image shows an example of this information:

| Db_name   | Table_name | Checksum_crc64_xor   | Total_kvs  | Total_bytes  |
|-----------|------------|----------------------|------------|--------------|
| batchmark | order_line | 10912722838344822475 | 5659888624 | 370385538778 |

1 row in set (5 min 47.59 sec)

Figure 31: img

Execute the `br backup` command:

```
bin/br backup table \
--db batchmark \
--table order_line \
-s local:///home/tidb/backup_local/ \
```

```
--pd ${PD_ADDR}:2379 \
--log-file backup_local.log
```

During the backup process, pay attention to the metrics on the monitoring panels to get the status of the backup process. See [Monitoring metrics for the backup](#) for details.

#### Backup results explanation

Before executing the backup command, a path in which the log is stored has been specified. You can get the statistical information of the backup operation from this log. Search “summary” in this log, you can see the following information:

```
["Table backup summary: total backup ranges: 4, total success: 4, total
 ↪ failed: 0, total take(s): 551.31, total kv: 5659888624, total size(MB)
 ↪): 353227.18, avg speed(MB/s): 640.71"] ["backup total regions"=6795]
 ↪ ["backup checksum"=6m33.962719217s] ["backup fast checksum
 ↪ "=22.995552ms]
```

The information from the above log includes:

- Backup duration: `total take(s): 551.31`
- Data size: `total size(MB): 353227.18`
- Backup throughput: `avg speed(MB/s): 640.71`
- Backup checksum duration: `take=6m33.962719217s`

From the above information, the throughput of a single TiKV instance can be calculated:  $\text{avg speed(MB/s)}/\text{tikv\_count} = 160$ .

Restore data from a local disk (recommended in testing environment)

Use the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

#### Restoration prerequisites

- [Preparation for restoration](#)
- The TiKV cluster and the backup data do not have a duplicate database or table. Currently, BR does not support table route.
- Each TiKV node has a separate disk to store the `backupSST` file.
- The `restore_endpoint` node has a separate disk to store the `backupmeta` file.
- TiKV and the `restore_endpoint` node must have the same directory for the restoration (for example, `/home/tidb/backup_local/`).

Before the restoration, follow these steps:

1. Collect all `backupSST` files into the same directory.
2. Copy the collected `backupSST` files to all TiKV nodes of the cluster.

3. Copy the `backupmeta` file to the `restore endpoint` node.

### Topology

The following diagram shows the typology of BR:

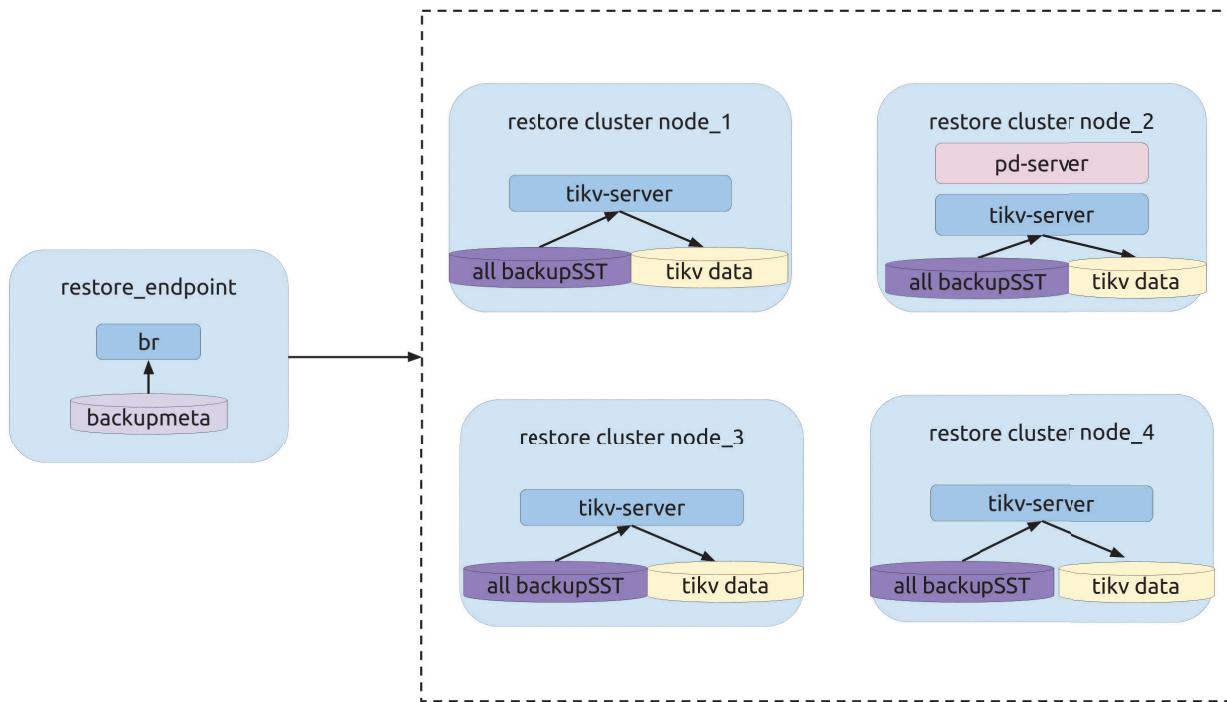


Figure 32: img

### Restoration operation

Execute the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///home/tidb
↪ /backup_local/ --pd 172.16.5.198:2379 --log-file restore_local.log
```

During the restoration process, pay attention to the metrics on the monitoring panels to get the status of the restoration process. See [Monitoring metrics for the restoration](#) for details.

### Restoration results explanation

Before executing the restoration command, a path in which the log is stored has been specified. You can get the statistical information of the restoration operation from this log. Search “summary” in this log, you can see the following information:

```
[{"Table Restore summary: total restore tables: 1, total success: 1, total
 ↪ failed: 0, total take(s): 908.42, total kv: 5659888624, total size(MB)
 ↪): 353227.18, avg speed(MB/s): 388.84"] [{"restore files":9263} [""
 ↪ restore ranges":6888] [{"split region":58.7885518s} [{"restore checksum
 ↪ "=6m19.349067937s}]]
```

The above log includes the following information:

- Restoration duration: `total take(s): 908.42`
- Data size: `total size(MB): 353227.18`
- Restoration throughput: `avg speed(MB/s): 388.84`
- Region Split duration: `take=58.7885518s`
- Restoration checksum duration: `take=6m19.349067937s`

From the above information, the following items can be calculated:

- The throughput of a single TiKV instance:  $\text{avg speed(MB/s)}/\text{tikv\_count} = 97.2$
- The average restoration speed of a single TiKV instance:  $\text{total size(MB)}/(\text{split time} + \text{restore time})/\text{tikv\_count} = 92.4$

#### 5.3.1.3.4 Error handling during backup

This section introduces the common errors occurred during the backup process.

`key locked` Error in the backup log

Error message in the log: `log - [{"backup occur kv error"}]` [error="{"KvError
 ↪ ":"locked"}"]

If a key is locked during the backup process, BR tries to resolve the lock. A small number of these errors do not affect the correctness of the backup.

Backup failure

Error message in the log: `log - Error: msg:"Io(Custom { kind: AlreadyExists,
 ↪ error: \"[5_5359_42_123_default.sst] is already exists in /dir/backup_local
 ↪ /\\" })"`

If the backup operation fails and the above message occurs, perform one of the following operations and then start the backup operation again:

- Change the directory for the backup. For example, change `/dir/backup-2020-01-01/` to `/dir/backup_local/`.
- Delete the backup directory of all TiKV nodes and BR nodes.

### 5.3.1.4 External Storages

Backup & Restore (BR), TiDB Lightning, and Dumpling support reading and writing data on the local filesystem and on Amazon S3. BR also supports reading and writing data on the Google Cloud Storage (GCS). These are distinguished by the URL scheme in the `--storage` parameter passed into BR, in the `-d` parameter passed into TiDB Lightning, and in the `--output (-o)` parameter passed into Dumpling.

#### 5.3.1.4.1 Schemes

The following services are supported:

| Service                                     | Schemes | Example URL                                    |
|---------------------------------------------|---------|------------------------------------------------|
| Local filesystem, distributed on every node | local   | <code>local:///path/to/dest/</code>            |
| Amazon S3 and compatible services           | s3      | <code>s3://bucket-name/prefix/of/dest/</code>  |
| Google Cloud Storage (GCS)                  | gcs, gs | <code>gcs://bucket-name/prefix/of/dest/</code> |
| Write to nowhere (for benchmarking only)    | noop    | <code>noop://</code>                           |

#### 5.3.1.4.2 URL parameters

Cloud storages such as S3 and GCS sometimes require additional configuration for connection. You can specify parameters for such configuration. For example:

- Use Dumpling to export data to S3:

```
./dumpling -u root -h 127.0.0.1 -P 3306 -B mydb -F 256MiB \
-o 's3://my-bucket/sql-backup?region=us-west-2'
```

- Use TiDB Lightning to import data from S3:

```
./tidb-lightning --tidb-port=4000 --pd-urls=127.0.0.1:2379 --backend=
→ local --sorted-kv-dir=/tmp/sorted-kvs \
-d 's3://my-bucket/sql-backup?region=us-west-2'
```

- Use TiDB Lightning to import data from S3 (using the path style in the request mode):

```
./tidb-lightning --tidb-port=4000 --pd-urls=127.0.0.1:2379 --backend=
→ local --sorted-kv-dir=/tmp/sorted-kvs \
-d 's3://my-bucket/sql-backup?force-path-style=true&endpoint=http
→ ://10.154.10.132:8088'
```

- Use BR to back up data to GCS:

```
./br backup full -u 127.0.0.1:2379 \
-s 'gcs://bucket-name/prefix'
```

S3 URL parameters

| URL parameter                        | Description                                                                              |
|--------------------------------------|------------------------------------------------------------------------------------------|
| <code>access-key</code>              | The access key                                                                           |
| <code>secret-access-key</code>       | The secret access key                                                                    |
| <code>region</code>                  | Service Region for Amazon S3 (default to us-east-1)                                      |
| <code>use-accelerate-endpoint</code> | Whether to use the accelerate endpoint                                                   |
| <code>endpoint</code>                | endpoint on Amazon S3 (default to false)                                                 |
| <code>endpoint</code>                | URL of custom endpoint for S3-compatible services (for example, https://s3.example.com/) |

| URL<br>parameter                     | Description                                                                                                                   |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>force-path</b><br>↪ <b>-style</b> | Use path<br>style<br>access<br>rather<br>than<br>virtual<br>hosted<br>style<br>access<br>(default to<br><b>false</b> )        |
| <b>storage-</b><br>↪ <b>class</b>    | Storage<br>class of the<br>uploaded<br>objects<br>(for<br>example,<br><b>STANDARD</b> ,<br><b>STANDARD_IA</b><br>↪ )          |
| <b>sse</b>                           | Server-side<br>encryption<br>algorithm<br>used to<br>encrypt<br>the upload<br>(empty,<br><b>AES256</b> or<br><b>aws:kms</b> ) |
| <b>sse-kms-</b><br>↪ <b>key-id</b>   | If <b>sse</b> is<br>set to<br><b>aws:kms</b> ,<br>specifies<br>the KMS<br>ID                                                  |

| URL parameter | Description                                                                         |
|---------------|-------------------------------------------------------------------------------------|
| acl           | Canned ACL of the uploaded objects (for example, private, authenticated<br>↪ -read) |

**Note:**

It is not recommended to pass in the access key and secret access key directly in the storage URL, because these keys are logged in plain text. The migration tools try to infer these keys from the environment in the following order:

1. \$AWS\_ACCESS\_KEY\_ID and \$AWS\_SECRET\_ACCESS\_KEY environment variables
2. \$AWS\_ACCESS\_KEY and \$AWS\_SECRET\_KEY environment variables
3. Shared credentials file on the tool node at the path specified by the \$AWS\_SHARED\_CREDENTIALS\_FILE  
↪ environment variable
4. Shared credentials file on the tool node at `~/.aws/credentials`
5. Current IAM role of the Amazon EC2 container
6. Current IAM role of the Amazon ECS task

### GCS URL parameters

| URL parameter          | Description                                            |
|------------------------|--------------------------------------------------------|
| credentials<br>↪ -file | The path to the credentials JSON file on the tool node |

| URL parameter               | Description                                                                                |
|-----------------------------|--------------------------------------------------------------------------------------------|
| <code>storage-class</code>  | Storage class of the uploaded objects (for example, STANDARD, COLDLINE)                    |
| <code>predefined-acl</code> | Predefined ACL of the uploaded objects (for example, private, project-<br>→ private<br>→ ) |

When `credentials-file` is not specified, the migration tool will try to infer the credentials from the environment, in the following order:

1. Content of the file on the tool node at the path specified by the `$GOOGLE_APPLICATION_CREDENTIALS` environment variable
2. Content of the file on the tool node at `~/.config/gcloud/application_default_credentials.json`
3. When running in GCE or GAE, the credentials fetched from the metadata server.

#### 5.3.1.4.3 Command-line parameters

In addition to the URL parameters, BR and Dumpling also support specifying these configurations using command-line parameters. For example:

```
./dumpling -u root -h 127.0.0.1 -P 3306 -B mydb -F 256MiB \
-o 's3://my-bucket/sql-backup' \
--s3.region 'us-west-2'
```

If you have specified URL parameters and command-line parameters at the same time, the URL parameters are overwritten by the command-line parameters.

S3 command-line parameters

---

| Command-line parameter     | Description                                                                                  |
|----------------------------|----------------------------------------------------------------------------------------------|
| --s3.<br>↪ <b>region</b>   | Amazon S3's service region, which de-faults to us-east-1.                                    |
| --s3.<br>↪ <b>endpoint</b> | The URL of custom endpoint for S3-compatible services. For example, https://s3.example.com/. |

---

| Command-line parameter                       | Description                                                                                                                   |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| --s3.<br>↳ <b>storage</b><br>↳ <b>-class</b> | The storage class of the upload object. For example, STANDARD<br>↳ and STANDARD_IA<br>↳ .                                     |
| --s3.sse                                     | The server-side encryption algorithm used to encrypt the upload. The value options are empty, AES256 and aws:<br>↳ kms<br>↳ . |

---

| Command-line parameter          | Description                                                                                                                                                   |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| --s3.sse-<br>↳ kms-key<br>↳ -id | If --s3.<br>↳ kms<br>↳ ,<br>this<br>parameter is<br>used to<br>specify<br>the<br>KMS<br>ID.                                                                   |
| --s3.acl                        | The<br>canned<br>ACL of<br>the<br>upload<br>object.<br>For ex-<br>ample,<br><b>private</b><br>↳<br>and<br><b>authenticated</b><br>↳ -<br>↳ <b>read</b><br>↳ . |

| Command-line parameter                                       | Description                                                                                                            |
|--------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| --s3.<br>→ provider<br>→<br>→<br>→ ,<br>→<br>→ and<br>other. | The type of the S3-compatible service. The supported types are aws, alibaba<br>→ , ceph,<br>netease<br>→<br>and other. |

## GCS command-line parameters

| Command-line parameter         | Description                                               |
|--------------------------------|-----------------------------------------------------------|
| --gcs.<br>→ <b>credentials</b> | The path of the credential of the tool node.              |
| --gcs.<br>→ <b>storage</b>     | The storage                                               |
| --gcs.<br>→ <b>-class</b>      | type of the upload object, such as STANDARD and COLDLINE. |

| Command-line parameter                   | Description                                                                                |
|------------------------------------------|--------------------------------------------------------------------------------------------|
| --gcs .<br>→ predefined<br>→ -acl<br>→ . | The pre-defined ACL of the upload object, such as private and project-<br>→ private<br>→ . |

#### 5.3.1.4.4 BR sending credentials to TiKV

By default, when using S3 and GCS destinations, BR will send the credentials to every TiKV nodes to reduce setup complexity.

However, this is unsuitable on cloud environment, where every node has their own role and permission. In such cases, you need to disable credentials sending with --send-credentials-to-tikv=false (or the short form -c=0):

```
./br backup full -c=0 -u pd-service:2379 -s 's3://bucket-name/prefix'
```

When using SQL statements to `back up` and `restore` data, you can add the `SEND_CREDENTIALS_TO_TIKV = FALSE` option:

```
BACKUP DATABASE * TO 's3://bucket-name/prefix' SEND_CREDENTIALS_TO_TIKV =
→ FALSE;
```

This option is not supported in TiDB Lightning and Dumpling, because the two applications are currently standalone.

#### 5.3.1.5 Backup & Restore FAQ

This document lists the frequently asked questions (FAQs) and the solutions about Backup & Restore (BR).

##### 5.3.1.5.1 What should I do if the error message could not read local://...:download sst failed is returned during data restoration?

When you restore data, each node must have access to **all** backup files (SST files). By default, if `local` storage is used, you cannot restore data because the backup files are

scattered among different nodes. Therefore, you have to copy the backup file of each TiKV node to the other TiKV nodes.

It is recommended to mount an NFS disk as a backup disk during backup. For details, see [Back up a single table to a network disk](#).

#### 5.3.1.5.2 How much does it affect the cluster during backup using BR?

When you use the `oltp_read_only` scenario of `sysbench` to back up to a disk (make sure the backup disk and the service disk are different) at full rate, the cluster QPS is decreased by 15%-25%. The impact on the cluster depends on the table schema.

To reduce the impact on the cluster, you can use the `--ratelimit` parameter to limit the backup rate.

#### 5.3.1.5.3 Does BR back up system tables? During data restoration, do they raise conflict?

Before v5.1.0, BR filtered out data from the system schema `mysql` during the backup. Since v5.1.0, BR **backs up** all data by default, including the system schemas `mysql.*`. But the technical implementation of restoring the system tables in `mysql.*` is not complete yet, so the tables in the system schema `mysql` are **not** restored by default. For more details, refer to the [Back up and restore table data in the mysql system schema \(experimental feature\)](#).

#### 5.3.1.5.4 What should I do to handle the Permission denied or No such file or directory error, even if I have tried to run BR using root in vain?

You need to confirm whether TiKV has access to the backup directory. To back up data, confirm whether TiKV has the write permission. To restore data, confirm whether it has the read permission.

During the backup operation, if the storage medium is the local disk or a network file system (NFS), make sure that the user to start BR and the user to start TiKV are consistent (if BR and TiKV are on different machines, the users' UIDs must be consistent). Otherwise, the `Permission denied` issue might occur.

Running BR with the root access might fail due to the disk permission, because the backup files (SST files) are saved by TiKV.

##### Note:

You might encounter the same problem during data restoration. When the SST files are read for the first time, the read permission is verified. The execution duration of DDL suggests that there might be a long interval between checking the permission and running BR. You might receive the error message `Permission denied` after waiting for a long time.

Therefore, it is recommended to check the permission before data restore according to the following steps:

1. Run the Linux-native command for process query:

```
ps aux | grep tikv-server
```

The output of the above command:

```
tidb_ouo 9235 10.9 3.8 2019248 622776 ? Ssl 08:28 1:12 bin/tikv-
 ↳ server --addr 0.0.0.0:20162 --advertise-addr 172.16.6.118:20162
 ↳ --status-addr 0.0.0.0:20188 --advertise-status-addr
 ↳ 172.16.6.118:20188 --pd 172.16.6.118:2379 --data-dir /home/user1/
 ↳ tidb-data/tikv-20162 --config conf/tikv.toml --log-file /home/
 ↳ user1/tidb-deploy/tikv-20162/log/tikv.log
tidb_ouo 9236 9.8 3.8 2048940 631136 ? Ssl 08:28 1:05 bin/tikv-
 ↳ server --addr 0.0.0.0:20161 --advertise-addr 172.16.6.118:20161
 ↳ --status-addr 0.0.0.0:20189 --advertise-status-addr
 ↳ 172.16.6.118:20189 --pd 172.16.6.118:2379 --data-dir /home/user1/
 ↳ tidb-data/tikv-20161 --config conf/tikv.toml --log-file /home/
 ↳ user1/tidb-deploy/tikv-20161/log/tikv.log
```

Or you can run the following command:

```
ps aux | grep tikv-server | awk '{print $1}'
```

The output of the above command:

```
tidb_ouo
tidb_ouo
```

2. Query the startup information of the cluster using the TiUP command:

```
tiup cluster list
```

The output of the above command:

```
[root@Copy-of-VM-EE-CentOS76-v1 br]# tiup cluster list
Starting component `cluster`: /root/.tiup/components/cluster/v1.5.2/
 ↳ tiup-cluster list
Name User Version Path
 ↳
----- ----- ----- -----
 ↳
```

```
tidb_cluster tidb_ouo v5.0.2 /root/.tiup/storage/cluster/clusters/
 ↳ tidb_cluster /root/.tiup/storage/cluster/clusters/tidb_cluster/
 ↳ ssh/id_rsa
```

3. Check the permission for the backup directory. For example, `backup` is for backup data storage:

```
ls -al backup
```

The output of the above command:

```
[root@Copy-of-VM-EE-CentOS76-v1 user1]# ls -al backup
total 0
drwxr-xr-x 2 root root 6 Jun 28 17:48 .
drwxr-xr-x 11 root root 310 Jul 4 10:35 ..
```

From the above output, you can find that the `tikv-server` instance is started by the user `tidb_ouo`. But the user `tidb_ouo` does not have the write permission for `backup`, the backup fails.

#### 5.3.1.5.5 What should I do to handle the `Io(Os...)` error?

Almost all of these problems are system call errors that occur when TiKV writes data to the disk. For example, if you encounter error messages such as `Io(Os {code: 13, kind: : PermissionDenied...})` or `Io(Os {code: 2, kind: NotFound...})`, you can first check the mounting method and the file system of the backup directory, and try to back up data to another folder or another hard disk.

For example, you might encounter the `Code: 22(invalid argument)` error when backing up data to the network disk built by `samba`.

#### 5.3.1.5.6 What should I do to handle the `rpc error: code = Unavailable desc =...` error occurred in BR?

This error might occur when the capacity of the cluster to restore (using BR) is insufficient. You can further confirm the cause by checking the monitoring metrics of this cluster or the TiKV log.

To handle this issue, you can try to scale out the cluster resources, reduce the concurrency during restore, and enable the `RATE_LIMIT` option.

#### 5.3.1.5.7 Where are the backed up files stored when I use local storage?

When you use `local` storage, `backupmeta` is generated on the node where BR is running, and backup files are generated on the Leader nodes of each Region.

### 5.3.1.5.8 How about the size of the backup data? Are there replicas of the backup?

During data backup, backup files are generated on the Leader nodes of each Region. The size of the backup is equal to the data size, with no redundant replicas. Therefore, the total data size is approximately the total number of TiKV data divided by the number of replicas.

However, if you want to restore data from local storage, the number of replicas is equal to that of the TiKV nodes, because each TiKV must have access to all backup files.

### 5.3.1.5.9 What should I do when BR restores data to the upstream cluster of TiCDC/Drainer?

- **The data restored using BR cannot be replicated to the downstream.** This is because BR directly imports SST files but the downstream cluster currently cannot obtain these files from the upstream.
- Before v4.0.3, DDL jobs generated during the BR restore might cause unexpected DDL executions in TiCDC/Drainer. Therefore, if you need to perform restore on the upstream cluster of TiCDC/Drainer, add all tables restored using BR to the TiCDC/Drainer block list.

You can use `filter.rules` to configure the block list for TiCDC and use `syncer.ignore → -table` to configure the block list for Drainer.

### 5.3.1.5.10 Does BR back up the SHARD\_ROW\_ID\_BITS and PRE\_SPLIT\_REGIONS information of a table? Does the restored table have multiple Regions?

Yes. BR backs up the `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` information of a table. The data of the restored table is also split into multiple Regions.

### 5.3.1.5.11 Why is the `region is unavailable` error reported for a SQL query after I use BR to restore the backup data?

If the cluster backed up using BR has TiFlash, `TableInfo` stores the TiFlash information when BR restores the backup data. If the cluster to be restored does not have TiFlash, the `region is unavailable` error is reported.

### 5.3.1.5.12 Does BR support in-place full recovery of some historical backup?

No. BR does not support in-place full recovery of some historical backup.

### 5.3.1.5.13 How can I use BR for incremental backup in the Kubernetes environment?

To get the `commitTs` field of the last BR backup, run the `kubectl -n ${namespace} → } get bk ${name}` command using kubectl. You can use the content of this field as `--lastbackups`.

### 5.3.1.5.14 How can I convert BR backupTS to Unix time?

BR `backupTS` defaults to the latest timestamp obtained from PD before the backup starts. You can use `pd-ctl tso timestamp` to parse the timestamp to obtain an accurate value, or use `backupTS >> 18` to quickly obtain an estimated value.

### 5.3.1.5.15 After BR restores the backup data, do I need to execute the ANALYZE statement on the table to update the statistics of TiDB on the tables and indexes?

BR does not back up statistics (except in v4.0.9). Therefore, after restoring the backup data, you need to manually execute `ANALYZE TABLE` or wait for TiDB to automatically execute `ANALYZE`.

In v4.0.9, BR backs up statistics by default, which consumes too much memory. To ensure that the backup process goes well, the backup for statistics is disabled by default starting from v4.0.10.

If you do not execute `ANALYZE` on the table, TiDB will fail to select the optimized execution plan due to inaccurate statistics. If query performance is not a key concern, you can ignore `ANALYZE`.

### 5.3.1.5.16 Can I use multiple BR processes at the same time to restore the data of a single cluster?

**It is strongly not recommended** to use multiple BR processes at the same time to restore the data of a single cluster for the following reasons:

- When BR restores data, it modifies some global configurations of PD. Therefore, if you use multiple BR processes for data restore at the same time, these configurations might be mistakenly overwritten and cause abnormal cluster status.
- BR consumes a lot of cluster resources to restore data, so in fact, running BR processes in parallel improves the restore speed only to a limited extent.
- There has been no test for running multiple BR processes in parallel for data restore, so it is not guaranteed to succeed.

## 5.4 Time Zone Support

The time zone in TiDB is decided by the global `time_zone` system variable and the session `time_zone` system variable. The default value of `time_zone` is `SYSTEM`. The actual time

zone corresponding to `System` is configured when the TiDB cluster bootstrap is initialized. The detailed logic is as follows:

- Prioritize the use of the `TZ` environment variable.
- If the `TZ` environment variable fails, extract the time zone from the actual soft link address of `/etc/localtime`.
- If both of the above methods fail, use UTC as the system time zone.

You can use the following statement to set the global server `time_zone` value at runtime:

```
SET GLOBAL time_zone = timezone;
```

Each client has its own time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

```
SET time_zone = timezone;
```

You can use the following statement to view the current values of the global, client-specific and system time zones:

```
SELECT @@global.time_zone, @@session.time_zone, @@global.system_time_zone;
```

To set the format of the value of the `time_zone`:

- The value ‘SYSTEM’ indicates that the time zone should be the same as the system time zone.
- The value can be given as a string indicating an offset from UTC, such as ‘+10:00’ or ‘-6:00’.
- The value can be given as a named time zone, such as ‘Europe/Helsinki’, ‘US/Eastern’, or ‘MET’.

The current session time zone setting affects the display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`.

#### Note:

Only the values of the `Timestamp` data type is affected by time zone. This is because the `Timestamp` data type uses the literal value + time zone information. Other data types, such as `Datetime/Date/Time`, do not have time zone information, thus their values are not affected by the changes of time zone.

```
create table t (ts timestamp, dt datetime);
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
set @@time_zone = 'UTC';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
insert into t values ('2017-09-30 11:11:11', '2017-09-30 11:11:11');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
set @@time_zone = '+8:00';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select * from t;
```

| ts                  | dt                  |
|---------------------|---------------------|
| 2017-09-30 19:11:11 | 2017-09-30 11:11:11 |

1 row in set (0.00 sec)

In this example, no matter how you adjust the value of the time zone, the value of the Datetime data type is not affected. But the displayed value of the Timestamp data type changes if the time zone information changes. In fact, the value that is stored in the storage does not change, it's just displayed differently according to different time zone setting.

#### Note:

- Time zone is involved during the conversion of the value of Timestamp and Datetime, which is handled based on the current `time_zone` of the session.
- For data migration, you need to pay special attention to the time zone setting of the primary database and the secondary database.

## 5.5 Daily Check

As a distributed database, TiDB is more complicated than the stand-alone database in terms of the mechanism, and monitoring items. To help operate and maintain TiDB in a more convenient way, this document introduces some key performance indicators.

### 5.5.1 Key indicators of TiDB Dashboard

Starting from v4.0, TiDB provides a new operation and maintenance management tool, [TiDB Dashboard](#). This tool is integrated into the PD component. You can access TiDB Dashboard at the default address `http://{$pd-ip}:{$pd_port}/dashboard`.

TiDB Dashboard simplifies the operation and maintenance of the TiDB database. You can view the running status of the entire TiDB cluster through one interface. The following are descriptions of some performance indicators.

#### 5.5.1.1 Instance panel

| Instances |                 | Hosts  |                  |         |                               |                      |
|-----------|-----------------|--------|------------------|---------|-------------------------------|----------------------|
| ▼         | Address         | Status | Up Time          | Version | Deployment Directory          | Git Hash             |
| ▼         | <b>tidb (1)</b> |        |                  |         |                               |                      |
|           | 127.0.0.1:4000  | ● Up   | Today at 2:19 PM | v4.0.0  | /Users/wangjun/.tiup/compo... | 689a6b6439ae7835...  |
| ▼         | <b>tikv (1)</b> |        |                  |         |                               |                      |
|           | 127.0.0.1:20160 | ● Up   | Today at 2:19 PM | v4.0.0  | /Users/wangjun/.tiup/compo... | 198a2cea01734ce8...  |
| ▼         | <b>pd (1)</b>   |        |                  |         |                               |                      |
|           | 127.0.0.1:2379  | ● Up   | Today at 2:19 PM | v4.0.0  | /Users/wangjun/.tiup/compo... | 56d4c3d2237f5bf6f... |

Figure 33: Instance panel

- **Status:** This indicator is used to check whether the status is normal. For an online node, this can be ignored.
- **Up Time:** The key indicator. If you find that the **Up Time** is changed, you need to locate the reason why the component is restarted.
- **Version, Deployment Directory, Git Hash:** These indicators need to be checked to avoid inconsistent or even incorrect version/deployment directory.

#### 5.5.1.2 Host panel

| Instances | Hosts     |        |                                  |         |                                  |                            |           |                                  |
|-----------|-----------|--------|----------------------------------|---------|----------------------------------|----------------------------|-----------|----------------------------------|
|           | Address   | CPU    | CPU Usage                        | Memory  | Memory Usage                     | Disk                       | Disk Size | Disk Usage                       |
|           | 127.0.0.1 | 4 vCPU | <div style="width: 10%;"> </div> | 8.0 GiB | <div style="width: 10%;"> </div> | 1 TiDB, 1 TiKV, 1 PD; APFS | 233.5 GiB | <div style="width: 10%;"> </div> |

Figure 34: Host panel

You can view the usage of CPU, memory, and disk. When the usage of any resource exceeds 80%, it is recommended to scale out the capacity accordingly.

#### 5.5.1.3 SQL analysis panel

| Statement Template ⓘ                | Total Latency ⓘ ↓ | Mean Latency ⓘ                   | Execution Count ⓘ | Mean Memory ⓘ                    | Database ⓘ |
|-------------------------------------|-------------------|----------------------------------|-------------------|----------------------------------|------------|
| SELECT * FROM information_schema... | 1.7 s             | <div style="width: 10%;"> </div> | 1                 | <div style="width: 10%;"> </div> | 32.4 KiB   |
| SELECT * FROM information_schema... | 396.4 ms          | <div style="width: 10%;"> </div> | 1                 | <div style="width: 10%;"> </div> | 38.0 KiB   |
| SELECT * FROM information_schema... | 139.7 ms          | <div style="width: 10%;"> </div> | 1                 | <div style="width: 10%;"> </div> | 61.7 KiB   |
| SELECT * FROM information_schema... | 98.6 ms           | <div style="width: 10%;"> </div> | 1                 | <div style="width: 10%;"> </div> | 0 B        |
| SELECT DISTINCT stmt_type FROM i... | 64.2 ms           | <div style="width: 10%;"> </div> | 3                 | <div style="width: 10%;"> </div> | 4.1 KiB    |
| SELECT DISTINCT floor (unix_time... | 57.1 ms           | <div style="width: 10%;"> </div> | 3                 | <div style="width: 10%;"> </div> | 70.7 KiB   |
| SELECT *, (unix_timestamp (time)... | 35.2 ms           | <div style="width: 10%;"> </div> | 2                 | <div style="width: 10%;"> </div> | 19.6 KiB   |
| SELECT @ @global.tidb_enable_stm... | 22.9 ms           | <div style="width: 10%;"> </div> | 3                 | <div style="width: 10%;"> </div> | 0 B        |
| SELECT @ @global.tidb_stmt_summa... | 15.5 ms           | <div style="width: 10%;"> </div> | 3                 | <div style="width: 10%;"> </div> | 0 B        |
| SELECT @ @global.tidb_stmt_summa... | 14.1 ms           | <div style="width: 10%;"> </div> | 3                 | <div style="width: 10%;"> </div> | 0 B        |
| SELECT any_value (table_names) A... | 13.8 ms           | <div style="width: 10%;"> </div> | 2                 | <div style="width: 10%;"> </div> | 420.8 KiB  |
| SHOW DATABASES                      | 3.9 ms            | <div style="width: 10%;"> </div> | 3                 | <div style="width: 10%;"> </div> | 0 B        |

Figure 35: SQL analysis panel

You can locate the slow SQL statement executed in the cluster. Then you can optimize the specific SQL statement.

#### 5.5.1.4 Region panel

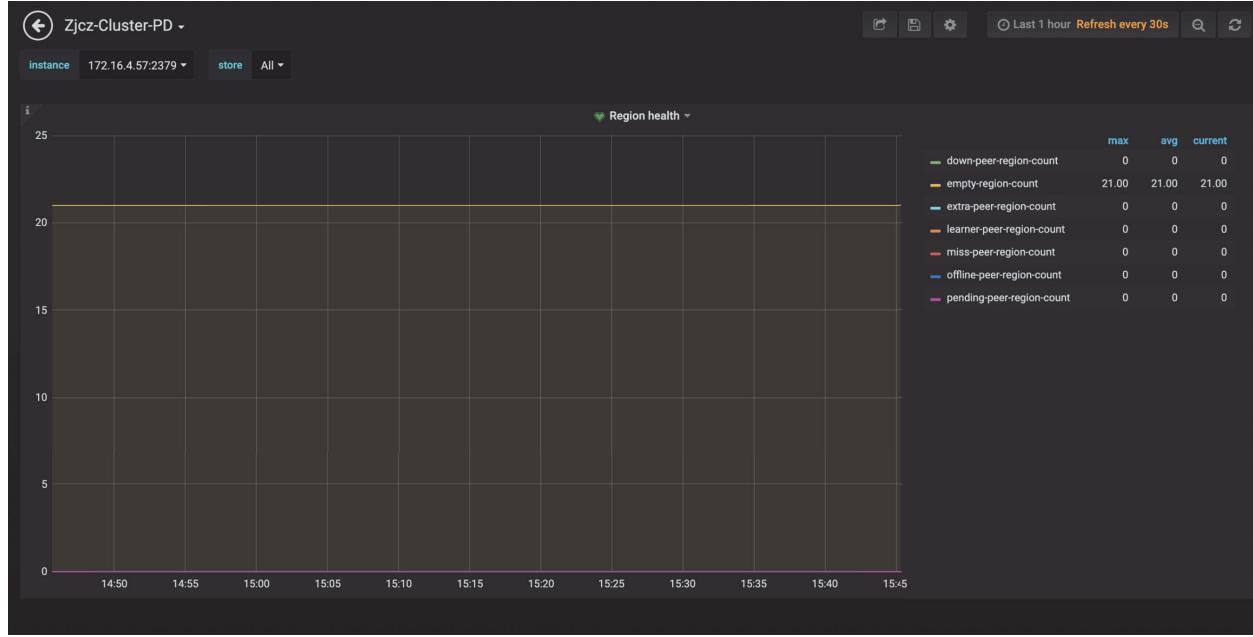


Figure 36: Region panel

- **miss-peer-region-count**: The number of Regions without enough replicas. This value is not always greater than 0.
- **extra-peer-region-count**: The number of Regions with extra replicas. These Regions are generated during the scheduling process.
- **empty-region-count**: The number of empty Regions, generated by executing the TRUNCATE TABLE/DROP TABLE statement. If this number is large, you can consider enabling **Region Merge** to merge Regions across tables.
- **pending-peer-region-count**: The number of Regions with outdated Raft logs. It is normal that a few pending peers are generated in the scheduling process. However, it is not normal if this value is large for a period of time.
- **down-peer-region-count**: The number of Regions with an unresponsive peer reported by the Raft leader.
- **offline-peer-region-count**: The number of Regions during the offline process.

Generally, it is normal that these values are not 0. However, it is not normal that they are not 0 for quite a long time.

#### 5.5.1.5 KV Request Duration



Figure 37: TiKV request duration

The KV request duration 99 in TiKV. If you find nodes with a long duration, check whether there are hot spots, or whether there are nodes with poor performance.

#### 5.5.1.6 PD TSO Wait Duration

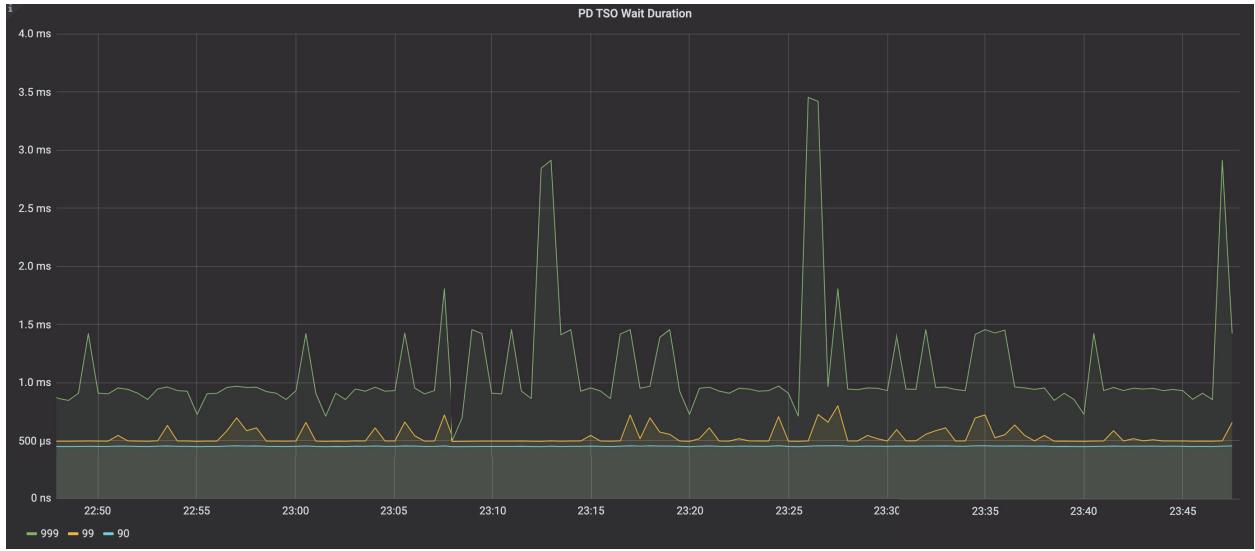


Figure 38: TiDB TSO Wait Duration

The time it takes for TiDB to obtain TSO from PD. The following are reasons for the long wait duration:

- High network latency from TiDB to PD. You can manually execute the ping command to test the network latency.
- High load for the TiDB server.
- High load for the PD server.

#### 5.5.1.7 Overview panel



Figure 39: Overview panel

You can view the load, memory available, network traffic, and I/O utilities. When a bottleneck is found, it is recommended to scale out the capacity, or to optimize the cluster topology, SQL, cluster parameters, etc.

#### 5.5.1.8 Exceptions

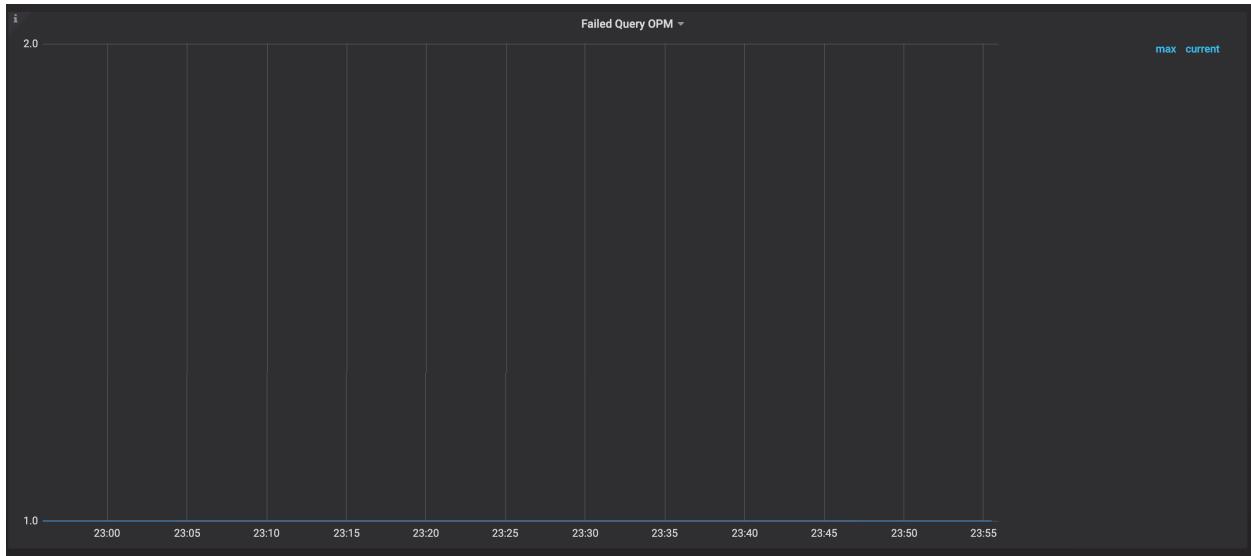


Figure 40: Exceptions

You can view the errors triggered by the execution of SQL statements on each TiDB instance. These include syntax error, primary key conflicts, etc.

#### 5.5.1.9 GC status

| VARIABLE_NAME            | VARIABLE_VALUE                                                                                           | COMMENT                                                                   |
|--------------------------|----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| bootstrapped             | True                                                                                                     | Bootstrap flag. Do not <code>delete</code> .                              |
| tidb_server_version      | 44                                                                                                       | Bootstrap version. Do not <code>delete</code> .                           |
| system_tz                | Asia/Shanghai                                                                                            | TiDB Global System Timezone.                                              |
| new_collation_enabled    | False                                                                                                    | If the new collations are enabled. Do not <code>delete</code> .           |
| tikv_gc_leader_uuid      | 5cb8865ca80006                                                                                           | Current GC worker leader UUID. (DO NOT <code>delete</code> .)             |
| tikv_gc_leader_desc      | host:wangjundeMacBook-Pro.local, pid:81322, start at 2020-05-20 20:49:11.735312 +0800 CST m=>7.904021075 | Host name <code>and</code> pid of current GC leader.                      |
| tikv_gc_leader_lease     | 20200522-00:06:30 +0800                                                                                  | Current GC worker leader lease. (DO NOT <code>delete</code> .)            |
| tikv_gc_enable           | true                                                                                                     | Current GC enable status.                                                 |
| tikv_gc_run_interval     | 10m0s                                                                                                    | GC run interval, at least 10m, <code>in Go</code> mode.                   |
| tikv_gc_life_time        | 10m0s                                                                                                    | All versions within life <code>time</code> will not be GC'd.              |
| tikv_gc_last_run_time    | 20200521-23:55:30 +0800                                                                                  | The <code>time</code> when last GC starts. (DO NOT <code>delete</code> .) |
| tikv_gc_safe_point       | 20200521-23:45:30 +0800                                                                                  | All versions after safe <code>point</code> can be automatically GC'd.     |
| tikv_gc_auto_concurrency | true                                                                                                     | Let TiDB pick the concurrency automatically.                              |
| tikv_gc_mode             | distributed                                                                                              | Mode of GC, "central" or "distributed".                                   |

Figure 41: GC status

You can check whether the GC (Garbage Collection) status is normal by viewing the time when the last GC happens. If the GC is abnormal, it might lead to excessive historical data, thereby decreasing the access efficiency.

## 5.6 Maintain a TiFlash Cluster

This document describes how to perform common operations when you maintain a **TiFlash** cluster, including checking the TiFlash version. This document also introduces critical logs and a system table of TiFlash.

### 5.6.1 Check the TiFlash version

There are two ways to check the TiFlash version:

- If the binary file name of TiFlash is `tiflash`, you can check the version by executing the `./tiflash version` command.

However, to execute the above command, you need to add the directory path which includes the `libtiflash_proxy.so` dynamic library to the `LD_LIBRARY_PATH` environment variable. This is because the running of TiFlash relies on the `libtiflash_proxy ↳ .so` dynamic library.

For example, when `tiflash` and `libtiflash_proxy.so` are in the same directory, you can first switch to this directory, and then use the following command to check the TiFlash version:

```
LD_LIBRARY_PATH=./ ./tiflash version
```

- Check the TiFlash version by referring to the TiFlash log. For the log path, see the `[logger]` part in [the `tiflash.toml` file](#). For example:

```
<information>: TiFlash version: TiFlash 0.2.0 master-375035282451103999
↳ f3863c691e2fc2
```

### 5.6.2 TiFlash critical logs

| Log Information    | Log Description       |
|--------------------|-----------------------|
| [INFO]             | Data starts to be     |
| [<unknown>]        | replicated (the       |
| ["KVStore: Start   | number in the square  |
| to persist [region | brackets at the start |
| 47, applied: term  | of the log refers to  |
| 6 index 10]" ]     | the thread ID         |
| [thread_id=23]     |                       |

| Log Information                                              | Log Description                                                                     |
|--------------------------------------------------------------|-------------------------------------------------------------------------------------|
| [DEBUG] [<unknown>] [“CoprocessorHandler::Status”]           | Handling DAG request, that is, TiFlash starts to handle a Coprocessor request       |
| DB::CoprocessorHandler::execute(): Handling DAG request”]    |                                                                                     |
| [thread_id=30]                                               |                                                                                     |
| [DEBUG] [<unknown>] [“CoprocessorHandler::Status”]           | Handling DAG request done, that is, TiFlash finishes handling a Coprocessor request |
| DB::CoprocessorHandler::execute(): Handle DAG request done”] |                                                                                     |
| [thread_id=30]                                               |                                                                                     |

You can find the beginning or the end of a Coprocessor request, and then locate the related logs of the Coprocessor request through the thread ID printed at the start of the log.

### 5.6.3 TiFlash system table

The column names and their descriptions of the `information_schema.tiflash_replica` system table are as follows:

| Column Name   | Description                    |
|---------------|--------------------------------|
| TABLE_SCHEMA  | database name                  |
| TABLE_NAME    | table name                     |
| TABLE_ID      | table ID                       |
| REPLICA_COUNT | number of TiFlash replicas     |
| AVAILABLE     | available or not (0/1)         |
| PROGRESS      | replication progress [0.0~1.0] |

## 5.7 TiUP Common Operations

This document describes the following common operations when you operate and maintain a TiDB cluster using TiUP.

- View the cluster list

- Start the cluster
- View the cluster status
- Modify the configuration
- Stop the cluster
- Destroy the cluster

### 5.7.1 View the cluster list

You can manage multiple TiDB clusters using the TiUP cluster component. When a TiDB cluster is deployed, the cluster appears in the TiUP cluster list.

To view the list, run the following command:

```
tiup cluster list
```

### 5.7.2 Start the cluster

The components in the TiDB cluster are started in the following order:

**PD > TiKV > Pump > TiDB > TiFlash > Drainer > TiCDC > Prometheus > Grafana > Alertmanager**

To start the cluster, run the following command:

```
tiup cluster start ${cluster-name}
```

#### Note:

Replace  `${cluster-name}` with the name of your cluster. If you forget the cluster name, check it by running `tiup cluster list`.

You can start only some of the components by adding the `-R` or `-N` parameters in the command. For example:

- This command starts only the PD component:

```
tiup cluster start ${cluster-name} -R pd
```

- This command starts only the PD components on the 1.2.3.4 and 1.2.3.5 hosts:

```
tiup cluster start ${cluster-name} -N 1.2.3.4:2379,1.2.3.5:2379
```

**Note:**

If you start the specified component by using the `-R` or `-N` parameters, make sure the starting order is correct. For example, start the PD component before the TiKV component. Otherwise, the start might fail.

### 5.7.3 View the cluster status

After starting the cluster, check the status of each component to ensure that they work normally. TiUP provides the `display` command, so you do not have to log in to every machine to view the component status.

```
tiup cluster display ${cluster-name}
```

### 5.7.4 Modify the configuration

When the cluster is in operation, if you need to modify the parameters of a component, run the `edit-config` command. The detailed steps are as follows:

1. Open the configuration file of the cluster in the editing mode:

```
tiup cluster edit-config ${cluster-name}
```

2. Configure the parameters:

- If the configuration is globally effective for a component, edit `server_configs`:

```
server_configs:
tidb:
log.slow-threshold: 300
```

- If the configuration takes effect on a specific node, edit the configuration in `config` of the node:

```
tidb_servers:
- host: 10.0.1.11
 port: 4000
 config:
 log.slow-threshold: 300
```

For the parameter format, see the [TiUP parameter template](#).

**Use `.` to represent the hierarchy of the configuration items.**

For more information on the configuration parameters of components, refer to [TiDB config.toml.example](#), [TiKV config.toml.example](#), and [PD config.toml.example](#) ↗ .

3. Rolling update the configuration and restart the corresponding components by running the `reload` command:

```
tiup cluster reload ${cluster-name} [-N <nodes>] [-R <roles>]
```

#### 5.7.4.1 Example

If you want to set the transaction size limit parameter (`txn-total-size-limit` in the [performance](#) module) to 1G in tidb-server, edit the configuration as follows:

```
server_configs:
tidb:
 performance.txn-total-size-limit: 1073741824
```

Then, run the `tiup cluster reload ${cluster-name} -R tidb` command to rolling restart the TiDB component.

#### 5.7.5 Replace with a hotfix package

For normal upgrade, see [Upgrade TiDB Using TiUP](#). But in some scenarios, such as debugging, you might need to replace the currently running component with a temporary package. To achieve this, use the `patch` command:

```
tiup cluster patch --help
```

Replace the remote package with a specified package and restart the service

Usage:

```
cluster patch <cluster-name> <package-path> [flags]
```

Flags:

|                                 |                                                                                                                              |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>-h, --help</code>         | help for patch                                                                                                               |
| <code>-N, --node strings</code> | Specify the nodes                                                                                                            |
| <code>--overwrite</code>        | Use this package in the future scale-out<br>→ operations                                                                     |
| <code>-R, --role strings</code> | Specify the role<br>--transfer-timeout int Timeout in seconds when transferring PD and<br>→ TiKV store leaders (default 300) |

#### Global Flags:

```
--native-ssh Use the system's native SSH client
--wait-timeout int Timeout of waiting the operation
--ssh-timeout int Timeout in seconds to connect host via SSH, ignored
 → for operations that don't need an SSH connection. (default 5)
-y, --yes Skip all confirmations and assumes 'yes'
```

If a TiDB hotfix package is in `/tmp/tidb-hotfix.tar.gz` and you want to replace all the TiDB packages in the cluster, run the following command:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -R tidb
```

You can also replace only one TiDB package in the cluster:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -N 172.16.4.5:4000
```

#### 5.7.6 Rename the cluster

After deploying and starting the cluster, you can rename the cluster using the `tiup cluster rename` command:

```
tiup cluster rename ${cluster-name} ${new-name}
```

#### Note:

- The operation of renaming a cluster restarts the monitoring system (Prometheus and Grafana).
- After a cluster is renamed, some panels with the old cluster name might remain on Grafana. You need to delete them manually.

#### 5.7.7 Stop the cluster

The components in the TiDB cluster are stopped in the following order (The monitoring component is also stopped):

**Alertmanager > Grafana > Prometheus > TiCDC > Drainer > TiFlash > TiDB > Pump > TiKV > PD**

To stop the cluster, run the following command:

```
tiup cluster stop ${cluster-name}
```

Similar to the `start` command, the `stop` command supports stopping some of the components by adding the `-R` or `-N` parameters. For example:

- This command stops only the TiDB component:

```
tiup cluster stop ${cluster-name} -R tidb
```

- This command stops only the TiDB components on the 1.2.3.4 and 1.2.3.5 hosts:

```
tiup cluster stop ${cluster-name} -N 1.2.3.4:4000,1.2.3.5:4000
```

### 5.7.8 Clean up cluster data

The operation of cleaning up cluster data stops all the services and cleans up the data directory or/and log directory. The operation cannot be reverted, so proceed **with caution**.

- Clean up the data of all services in the cluster, but keep the logs:

```
tiup cluster clean ${cluster-name} --data
```

- Clean up the logs of all services in the cluster, but keep the data:

```
tiup cluster clean ${cluster-name} --log
```

- Clean up the data and logs of all services in the cluster:

```
tiup cluster clean ${cluster-name} --all
```

- Clean up the logs and data of all services except Prometheus:

```
tiup cluster clean ${cluster-name} --all --ignore-role prometheus
```

- Clean up the logs and data of all services except the 172.16.13.11:9000 instance:

```
tiup cluster clean ${cluster-name} --all --ignore-node
↪ 172.16.13.11:9000
```

- Clean up the logs and data of all services except the 172.16.13.12 node:

```
tiup cluster clean ${cluster-name} --all --ignore-node 172.16.13.12
```

### 5.7.9 Destroy the cluster

The destroy operation stops the services and clears the data directory and deployment directory. The operation cannot be reverted, so proceed **with caution**.

```
tiup cluster destroy ${cluster-name}
```

## 5.8 Modify Configuration Online

This document describes how to modify the cluster configuration online.

### Note:

This feature is experimental. It is **NOT** recommended to use this feature in the production environment.

You can update the configuration of components (including TiDB, TiKV, and PD) online using SQL statements, without restarting the cluster components. Currently, the method of changing TiDB instance configuration is different from that of changing configuration of other components (such TiKV and PD).

### 5.8.1 Common Operations

This section describes the common operations of modifying configuration online.

#### 5.8.1.1 View instance configuration

To view the configuration of all instances in the cluster, use the `show config` statement. The result is as follows:

```
show config;
```

```
+--+
→ -----+-----+
→
| Type | Instance | Name
→ Value
→
→ |
+--+
→ -----+-----+
→
| tidb | 127.0.0.1:4001 | advertise-address
→ 127.0.0.1
→
→ |
| tidb | 127.0.0.1:4001 | binlog.binlog-socket
→
→
→ |
```

```
| tidb | 127.0.0.1:4001 | binlog.enable
 ↵ false
 ↵
 ↵ |
| tidb | 127.0.0.1:4001 | binlog.ignore-error
 ↵ false
 ↵
 ↵ |
| tidb | 127.0.0.1:4001 | binlog.strategy
 ↵ range
 ↵
 ↵ |
| tidb | 127.0.0.1:4001 | binlog.write-timeout
 ↵ 15s
 ↵
 ↵ |
| tidb | 127.0.0.1:4001 | check-mb4-value-in-utf8
 ↵ true
 ↵
 ↵ |

```

...

You can filter the result by fields. For example:

```
show config where type='tidb'
show config where instance in (...)

show config where name like '%log%'
show config where type='tikv' and name='log-level'
```

### 5.8.1.2 Modify TiKV configuration online

#### Note:

- After changing TiKV configuration items online, the TiKV configuration file is automatically updated. However, you also need to modify the corresponding configuration items by executing `tiup edit-config`; otherwise, operations such as `upgrade` and `reload` will overwrite your changes. For details of modifying configuration items, refer to [Modify configuration using TiUP](#).
- After executing `tiup edit-config`, you do not need to execute `tiup ↵ reload`.

When using the `set config` statement, you can modify the configuration of a single instance or of all instances according to the instance address or the component type.

- Modify the configuration of all TiKV instances:

**Note:**

It is recommended to wrap variable names in backticks.

```
set config tikv `split.qps-threshold`=1000
```

- Modify the configuration of a single TiKV instance:

```
set config "127.0.0.1:20180" `split.qps-threshold`=1000
```

If the modification is successful, `Query OK` is returned:

```
Query OK, 0 rows affected (0.01 sec)
```

If an error occurs during the batch modification, a warning is returned:

```
set config `tikv log-level`='warn';
```

```
Query OK, 0 rows affected, 1 warning (0.04 sec)
```

```
show warnings;
```

```
+--+
→ -----+-----+
→
| Level | Code | Message
→
→ |
+--+
→ -----+-----+
→
| Warning | 1105 | bad request to http://127.0.0.1:20180/config: fail to
→ update, error: "config log-level can not be changed" |
+--+
→ -----+-----+
→
1 row in set (0.00 sec)
```

The batch modification does not guarantee atomicity. The modification might succeed on some instances, while failing on others. If you modify the configuration of the entire TiKV cluster using `set tikv key=val`, your modification might fail on some instances. You can use `show warnings` to check the result.

If some modifications fail, you need to re-execute the corresponding statement or modify each failed instance. If some TiKV instances cannot be accessed due to network issues or machine failure, modify these instances after they are recovered.

If a configuration item is successfully modified, the result is persisted in the configuration file, which will prevail in the subsequent operations. The names of some configuration items might conflict with TiDB reserved words, such as `limit` and `key`. For these configuration items, use backtick ` to enclose them. For example, ``raftstore.raft-log-gc-size-limit``.

The following TiKV configuration items can be modified online:

| Configuration item      | Description |
|-------------------------|-------------|
| <code>raftstore</code>  |             |
| <code>. maxi</code>     |             |
| <code>raftnum</code>    |             |
| <code>- size</code>     |             |
| <code>entryf a</code>   |             |
| <code>- sin-</code>     |             |
| <code>max gle</code>    |             |
| <code>- log</code>      |             |
| <code>size</code>       |             |
| <code>raftstore</code>  |             |
| <code>. time</code>     |             |
| <code>raftinter-</code> |             |
| <code>- val</code>      |             |
| <code>log at</code>     |             |
| <code>- which</code>    |             |
| <code>gc the</code>     |             |
| <code>- polling</code>  |             |
| <code>ticktask</code>   |             |
| <code>- of</code>       |             |
| <code>interval</code>   |             |
| <code>ing</code>        |             |
| <code>Raft</code>       |             |
| <code>logs</code>       |             |
| <code>is</code>         |             |
| <code>sched-</code>     |             |
| <code>uled</code>       |             |

| Configuration item | Description |
|--------------------|-------------|
| <b>raftstoThe</b>  |             |
| ↳ . soft           |             |
| ↳ <b>raftlimit</b> |             |
| ↳ - on             |             |
| ↳ <b>log</b> the   |             |
| ↳ - maxi-          |             |
| ↳ <b>gc</b> mum    |             |
| ↳ - al-            |             |
| ↳ <b>threshold</b> |             |
| ↳      able        |             |
| ↳      num-        |             |
| ↳      ber         |             |
| ↳      of          |             |
| ↳      resid-      |             |
| ↳      ual         |             |
| Raft logs          |             |
| <b>raftstoThe</b>  |             |
| ↳ . hard           |             |
| ↳ <b>raftlimit</b> |             |
| ↳ - on             |             |
| ↳ <b>log</b> the   |             |
| ↳ - al-            |             |
| ↳ <b>gc</b> low-   |             |
| ↳ - able           |             |
| ↳ <b>countum-</b>  |             |
| ↳ - ber            |             |
| ↳ <b>limif</b>     |             |
| ↳      resid-      |             |
| ↳      ual         |             |
| Raft logs          |             |

| Configuration item | Description |
|--------------------|-------------|
| <b>raftstore</b>   |             |
| ↳ .                | hard        |
| ↳ <b>raftlimit</b> |             |
| ↳ -                | on          |
| ↳ log              | the         |
| ↳ -                | al-         |
| ↳ gc               | low-        |
| ↳ -                | able        |
| ↳ size             | size        |
| ↳ -                | of          |
| ↳ limits           | sid-        |
| ↳ -                | ual         |
| ↳ Raft             |             |
| ↳ logs             |             |
| <b>raftstore</b>   |             |
| ↳ .                | maxi-       |
| ↳ <b>raftnum</b>   |             |
| ↳ -                | re-         |
| ↳ entry            | main-       |
| ↳ -                | ing         |
| ↳ cache            | me          |
| ↳ -                | al-         |
| ↳ lifedow          | ed          |
| ↳ -                | for         |
| ↳ time             | the         |
| ↳ log              |             |
| ↳ cache            |             |
| ↳ in               |             |
| ↳ mem-             |             |
| ↳ ory              |             |

| Configuration item       | Description |
|--------------------------|-------------|
| <code>raftstore</code>   |             |
| <code>→ . time</code>    |             |
| <code>→ splinter-</code> |             |
| <code>→ - val</code>     |             |
| <code>→ region</code>    |             |
| <code>→ - which</code>   |             |
| <code>→ check</code>     |             |
| <code>→ - check</code>   |             |
| <code>→ tick</code>      | whether     |
| <code>→ - the</code>     |             |
| <code>→ interval</code>  |             |
| <code>→ gion</code>      |             |
| <code>→ split</code>     |             |
| <code>→ is</code>        |             |
| <code>→ needed</code>    |             |
| <code>raftstore</code>   |             |
| <code>→ . maxi-</code>   |             |
| <code>→ regionnm</code>  |             |
| <code>→ - value</code>   |             |
| <code>→ split</code>     |             |
| <code>→ - which</code>   |             |
| <code>→ check</code>     |             |
| <code>→ - Re-</code>     |             |
| <code>→ diffgion</code>  |             |
| <code>→ data</code>      |             |
| <code>→ is al-</code>    |             |
| <code>→ lowed</code>     |             |
| <code>→ to ex-</code>    |             |
| <code>→ ceed</code>      |             |
| <code>→ be-</code>       |             |
| <code>→ fore</code>      |             |
| <code>→ Re-</code>       |             |
| <code>→ gion</code>      |             |
| <code>→ split</code>     |             |

| Configuration item                | Description |
|-----------------------------------|-------------|
| <b>raftstore</b>                  | The         |
| $\hookrightarrow$ .               | time        |
| $\hookrightarrow$ <b>register</b> |             |
| $\hookrightarrow$ -               | val         |
| $\hookrightarrow$ <b>compact</b>  |             |
| $\hookrightarrow$ -               | which       |
| $\hookrightarrow$ <b>check</b>    |             |
| $\hookrightarrow$ -               | check       |
| $\hookrightarrow$ <b>interval</b> | whether     |
| $\hookrightarrow$                 | it is       |
|                                   | nec-        |
|                                   | es-         |
|                                   | sary        |
|                                   | to          |
|                                   | man-        |
|                                   | ually       |
|                                   | trig-       |
|                                   | ger         |
|                                   | RocksDB     |
|                                   | com-        |
|                                   | paction     |
| <b>raftstore</b>                  | The         |
| $\hookrightarrow$ .               | num-        |
| $\hookrightarrow$ <b>region</b>   |             |
| $\hookrightarrow$ -               | of          |
| $\hookrightarrow$ <b>compact</b>  |             |
| $\hookrightarrow$ -               | gions       |
| $\hookrightarrow$ <b>checked</b>  |             |
| $\hookrightarrow$ -               | at          |
| $\hookrightarrow$ <b>stepone</b>  |             |
| $\hookrightarrow$                 | time        |
|                                   | for         |
|                                   | each        |
|                                   | round       |
|                                   | of          |
|                                   | man-        |
|                                   | ual         |
|                                   | com-        |
|                                   | paction     |

| item                                | Description |
|-------------------------------------|-------------|
| <b>raftstore</b>                    |             |
| $\hookrightarrow$ . num-            |             |
| $\hookrightarrow$ <b>region</b>     |             |
| $\hookrightarrow$ - of              |             |
| $\hookrightarrow$ <b>compact</b> -  |             |
| $\hookrightarrow$ - stones          |             |
| $\hookrightarrow$ min re-           |             |
| $\hookrightarrow$ - quired          |             |
| $\hookrightarrow$ <b>tombstones</b> |             |
| $\hookrightarrow$ trig-             |             |
| $\hookrightarrow$ ger               |             |
| RocksDB                             |             |
| com-                                |             |
| paction                             |             |
| <b>raftstore</b>                    |             |
| $\hookrightarrow$ . pro-            |             |
| $\hookrightarrow$ <b>region-</b>    |             |
| $\hookrightarrow$ - tion            |             |
| $\hookrightarrow$ <b>compact</b>    |             |
| $\hookrightarrow$ - tomb-           |             |
| $\hookrightarrow$ <b>tombstones</b> |             |
| $\hookrightarrow$ - re-             |             |
| $\hookrightarrow$ <b>percentage</b> |             |
| $\hookrightarrow$ to                |             |
| $\hookrightarrow$ trig-             |             |
| $\hookrightarrow$ ger               |             |
| RocksDB                             |             |
| com-                                |             |
| paction                             |             |

| Configuration item           | Description |
|------------------------------|-------------|
| <b>raftstore</b>             |             |
| $\hookrightarrow$ . time     |             |
| $\hookrightarrow$ pd inter-  |             |
| $\hookrightarrow$ - val      |             |
| $\hookrightarrow$ heartbeat  |             |
| $\hookrightarrow$ - which    |             |
| $\hookrightarrow$ tick Re-   |             |
| $\hookrightarrow$ - gion's   |             |
| $\hookrightarrow$ interval-  |             |
| $\hookrightarrow$ beat       |             |
| $\hookrightarrow$ to         |             |
| $\hookrightarrow$ PD         |             |
| $\hookrightarrow$ is         |             |
| $\hookrightarrow$ trig-      |             |
| $\hookrightarrow$ gered      |             |
| <b>raftstore</b>             |             |
| $\hookrightarrow$ . time     |             |
| $\hookrightarrow$ pd inter-  |             |
| $\hookrightarrow$ - val      |             |
| $\hookrightarrow$ store      |             |
| $\hookrightarrow$ - which    |             |
| $\hookrightarrow$ heartbeat  |             |
| $\hookrightarrow$ - store's  |             |
| $\hookrightarrow$ tickheart- |             |
| $\hookrightarrow$ - beat     |             |
| $\hookrightarrow$ interval   |             |
| $\hookrightarrow$ PD         |             |
| $\hookrightarrow$ is         |             |
| $\hookrightarrow$ trig-      |             |
| $\hookrightarrow$ gered      |             |

| Configuration item                   | Description |
|--------------------------------------|-------------|
| <b>raftstore</b>                     |             |
| $\hookrightarrow$ . time             |             |
| $\hookrightarrow$ <b>snap</b> inter- |             |
| $\hookrightarrow$ - val              |             |
| $\hookrightarrow$ <b>mgr</b> at      |             |
| $\hookrightarrow$ - which            |             |
| $\hookrightarrow$ <b>gc</b> the      |             |
| $\hookrightarrow$ - recy-            |             |
| $\hookrightarrow$ <b>tickle</b> of   |             |
| $\hookrightarrow$ - ex-              |             |
| $\hookrightarrow$ <b>interval</b>    |             |
| $\hookrightarrow$ snap-              |             |
| shot                                 |             |
| files                                |             |
| is                                   |             |
| trig-                                |             |
| gered                                |             |
| <b>raftstore</b>                     |             |
| $\hookrightarrow$ . longest          |             |
| $\hookrightarrow$ <b>snap</b> time   |             |
| $\hookrightarrow$ - for              |             |
| $\hookrightarrow$ <b>gc</b> which    |             |
| $\hookrightarrow$ - a                |             |
| $\hookrightarrow$ <b>timeout</b> -   |             |
| $\hookrightarrow$ shot               |             |
| file is                              |             |
| saved                                |             |

| Configuration item                           | Description |
|----------------------------------------------|-------------|
| <b>raftstore</b>                             |             |
| <code>→ . time</code>                        |             |
| <code>→ lockinter-</code>                    |             |
| <code>→ - val</code>                         |             |
| <code>→ cf at</code>                         |             |
| <code>→ - which</code>                       |             |
| <code>→ compactTiKV</code>                   |             |
| <code>→ - trig-</code>                       |             |
| <code>→ interval</code>                      |             |
| <code>→ a</code>                             |             |
| manual compaction for the Lock Column Family |             |
| <b>raftstore</b>                             |             |
| <code>→ . size</code>                        |             |
| <code>→ lockat</code>                        |             |
| <code>→ - which</code>                       |             |
| <code>→ cf TiKV</code>                       |             |
| <code>→ - trig-</code>                       |             |
| <code>→ compact</code>                       |             |
| <code>→ - a</code>                           |             |
| <code>→ bytesan-</code>                      |             |
| <code>→ - ual</code>                         |             |
| <code>→ threshold</code>                     |             |
| paction for the Lock Column Family           |             |

| item                              | Description |
|-----------------------------------|-------------|
| <b>raftstoThe</b>                 |             |
| $\hookrightarrow$ .               | maxi-       |
| $\hookrightarrow$ <b>messages</b> |             |
| $\hookrightarrow$ -               | num-        |
| $\hookrightarrow$ <b>per</b>      | ber         |
| $\hookrightarrow$ -               | of          |
| $\hookrightarrow$ <b>ticknes-</b> |             |
| $\hookrightarrow$                 | sages       |
|                                   | pro-        |
|                                   | cessed      |
|                                   | per         |
|                                   | batch       |
| <b>raftstoThe</b>                 |             |
| $\hookrightarrow$ .               | longest     |
| $\hookrightarrow$ <b>max</b>      | inac-       |
| $\hookrightarrow$ -               | tive        |
| $\hookrightarrow$ <b>peer</b>     | dura-       |
| $\hookrightarrow$ -               | tion        |
| $\hookrightarrow$ <b>downal-</b>  |             |
| $\hookrightarrow$ -               | lowed       |
| $\hookrightarrow$ <b>duration</b> |             |
| $\hookrightarrow$                 | peer        |

| Configuration item                                                                                                                                                                                                                                                                                                          | Description |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| <b>raftstore</b><br>↳ . longest<br>↳ <b>max_dura-</b><br>↳ - tion<br>↳ <b>leader</b><br>↳ - lowed<br>↳ <b>missing</b><br>↳ - peer<br>↳ <b>duration</b><br>↳ with-<br>out a<br>leader.<br>If<br>this<br>value<br>is ex-<br>ceeded,<br>the<br>peer<br>veri-<br>fies<br>with<br>PD<br>whether<br>it<br>has<br>been<br>deleted. |             |

| Configuration item     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>raftstore</code> | <pre>         ↳ . nor-         ↳ abnormal         ↳ - dura-         ↳ leader         ↳ - al-         ↳ missingd         ↳ - for a         ↳ duration         ↳      to be             with-             out a             leader.          If         this         value         is ex-         ceeded,         the         peer         is         seen         as         ab-         nor-         mal         and         marked         in         met-         rics         and         logs.     </pre> |

| Configuration item             | Description |
|--------------------------------|-------------|
| <b>raftstore</b>               |             |
| $\hookrightarrow$ . time       |             |
| $\hookrightarrow$ peerinter-   |             |
| $\hookrightarrow$ - val        |             |
| $\hookrightarrow$ stale        |             |
| $\hookrightarrow$ - check      |             |
| $\hookrightarrow$ statewhether |             |
| $\hookrightarrow$ - a          |             |
| $\hookrightarrow$ checker      |             |
| $\hookrightarrow$ - is         |             |
| $\hookrightarrow$ interval     |             |
| $\hookrightarrow$ out a        |             |
| $\hookrightarrow$ leader       |             |
| <b>raftstore</b>               |             |
| $\hookrightarrow$ . time       |             |
| $\hookrightarrow$ consistency  |             |
| $\hookrightarrow$ - val        |             |
| $\hookrightarrow$ check        |             |
| $\hookrightarrow$ - check      |             |
| $\hookrightarrow$ interval     |             |
| $\hookrightarrow$ sis-         |             |
| $\hookrightarrow$ tency        |             |
| <b>raftstore</b>               |             |
| $\hookrightarrow$ . longest    |             |
| $\hookrightarrow$ rafttrusted  |             |
| $\hookrightarrow$ - pe-        |             |
| $\hookrightarrow$ stored       |             |
| $\hookrightarrow$ - of a       |             |
| $\hookrightarrow$ max Raft     |             |
| $\hookrightarrow$ - leader     |             |
| $\hookrightarrow$ leader       |             |
| $\hookrightarrow$ -            |             |
| $\hookrightarrow$ lease        |             |
| $\hookrightarrow$              |             |

| Configuration item | Description                |
|--------------------|----------------------------|
| <b>raftstd</b>     | Determines whether . allow |
| <b>allow</b>       | - allow                    |
| <b>remote</b>      | -det-                      |
| <b>leader</b>      | - ing                      |
| <b>main</b>        | leader                     |
| <b>switch</b>      | main                       |
| <b>raftstofhe</b>  | The time                   |
| <b>merge</b>       | -ter-                      |
| <b>val</b>         | - val                      |
| <b>check</b>       | - check                    |
| <b>interval</b>    | -                          |
| <b>raftstofhe</b>  | The time                   |
| <b>cleaner</b>     | -ter-                      |
| <b>import</b>      | - val                      |
| <b>check</b>       | - check                    |
| <b>sst ex-</b>     | - check                    |
| <b>pired</b>       | - pired                    |
| <b>interval</b>    | - interval                 |
|                    | files                      |

| Configuration item  | Description |
|---------------------|-------------|
| <b>raftstore</b>    |             |
| <b>localnum</b>     | maxi-       |
| <b>- num-</b>       | num-        |
| <b>reader</b>       | reader      |
| <b>- of</b>         | of          |
| <b>batch</b>        | batch       |
| <b>- re-</b>        | re-         |
| <b>size</b>         | size        |
| <b>quests</b>       | quests      |
| <b>      pro-</b>   | processed   |
| <b>      cessed</b> | in          |
| <b>      in</b>     | one         |
| <b>      one</b>    | batch       |

| Configuration item | Description |
|--------------------|-------------|
| <b>raftstop</b>    | The         |
| ↳ .                | short-      |
| ↳ <b>hibernate</b> |             |
| ↳ -                | wait        |
| ↳ <b>timebut-</b>  |             |
| ↳      tion        |             |
| ↳      be-         |             |
| ↳      fore        |             |
| ↳      enter-      |             |
| ↳      ing         |             |
| ↳      hi-         |             |
| ↳      ber-        |             |
| ↳      na-         |             |
| ↳      tion        |             |
| ↳      upon        |             |
| ↳      start.      |             |
| ↳      Within      |             |
| ↳      this        |             |
| ↳      dura-       |             |
| ↳      tion,       |             |
| ↳      TiKV        |             |
| ↳      does        |             |
| ↳      not         |             |
| ↳      hi-         |             |
| ↳      ber-        |             |
| ↳      nate        |             |
| ↳      (not        |             |
| ↳      re-         |             |
| ↳      leased).    |             |
| <b>coprocessor</b> | Enables     |
| ↳ .                | to          |
| ↳ <b>split</b>     | split       |
| ↳ -                | Re-         |
| ↳ <b>region</b>    | region      |
| ↳ -                | by          |
| ↳ <b>on</b>        | on table    |
| ↳ -                |             |
| ↳ <b>table</b>     |             |
| ↳                  |             |

| item                         | Description |
|------------------------------|-------------|
| <b>coprocessor</b>           |             |
| $\hookrightarrow$ . thresh-  |             |
| $\hookrightarrow$ batchId    |             |
| $\hookrightarrow$ - of       |             |
| $\hookrightarrow$ splitRe-   |             |
| $\hookrightarrow$ - gion     |             |
| $\hookrightarrow$ limitSplit |             |
| $\hookrightarrow$ in         |             |
| batches                      |             |
| <b>coprocessor</b>           |             |
| $\hookrightarrow$ . maxi-    |             |
| $\hookrightarrow$ regionNm   |             |
| $\hookrightarrow$ - size     |             |
| $\hookrightarrow$ max of a   |             |
| $\hookrightarrow$ - Re-      |             |
| $\hookrightarrow$ sizeRegion |             |
| $\hookrightarrow$            |             |
| <b>coprocessor</b>           |             |
| $\hookrightarrow$ . size     |             |
| $\hookrightarrow$ regionN    |             |
| $\hookrightarrow$ - the      |             |
| $\hookrightarrow$ splitNewly |             |
| $\hookrightarrow$ - split    |             |
| $\hookrightarrow$ sizeRe-    |             |
| $\hookrightarrow$ gion       |             |
| <b>coprocessor</b>           |             |
| $\hookrightarrow$ . maxi-    |             |
| $\hookrightarrow$ regionNm   |             |
| $\hookrightarrow$ - num-     |             |
| $\hookrightarrow$ max ber    |             |
| $\hookrightarrow$ - of       |             |
| $\hookrightarrow$ keys       |             |
| $\hookrightarrow$ al-        |             |
| lowed                        |             |
| in a                         |             |
| Re-                          |             |
| gion                         |             |

| item                  | Description |
|-----------------------|-------------|
| <b>coprocessor</b>    |             |
| ⇨ . num-              |             |
| ⇨ <b>region</b>       |             |
| ⇨ - of                |             |
| ⇨ <b>splitkeys</b>    |             |
| ⇨ - in                |             |
| ⇨ <b>keysthe</b>      |             |
| ⇨       newly         |             |
| ⇨       split         |             |
| ⇨ Re-                 |             |
| ⇨ gion                |             |
| <b>pessimistic</b>    |             |
| ⇨ - longest           |             |
| ⇨ <b>txn dura-</b>    |             |
| ⇨ . tion              |             |
| ⇨ <b>waitthat</b>     |             |
| ⇨ - a pes-            |             |
| ⇨ <b>for simistic</b> |             |
| ⇨ - trans-            |             |
| ⇨ <b>lockac-</b>      |             |
| ⇨ - tion              |             |
| ⇨ <b>timeouts</b>     |             |
| ⇨       for           |             |
| ⇨       the           |             |
| ⇨       lock          |             |
| <b>pessimistic</b>    |             |
| ⇨ - dura-             |             |
| ⇨ <b>txn tion</b>     |             |
| ⇨ . after             |             |
| ⇨ <b>wakewhich</b>    |             |
| ⇨ - a pes-            |             |
| ⇨ <b>up simistic</b>  |             |
| ⇨ - trans-            |             |
| ⇨ <b>delay-</b>       |             |
| ⇨ - tion              |             |
| ⇨ <b>duration</b>     |             |
| ⇨       ken           |             |
| ⇨       up            |             |

| Configuration item | Description |
|--------------------|-------------|
| <b>pessimistic</b> | Whether     |
| ↳ -                | to en-      |
| ↳ <b>txn</b>       | able        |
| ↳ .                | the         |
| ↳ <b>pipelined</b> |             |
| ↳                  | pes-        |
|                    | simistic    |
|                    | lock-       |
|                    | ing         |
|                    | pro-        |
|                    | cess        |
| <b>gc.</b>         | The         |
| ↳ <b>ratio</b>     | thresh-     |
| ↳ -                | old         |
| ↳ <b>threshold</b> |             |
| ↳                  | which       |
|                    | Re-         |
|                    | gion        |
|                    | GC          |
|                    | is          |
|                    | skipped     |
|                    | (the        |
|                    | num-        |
|                    | ber         |
|                    | of          |
|                    | GC          |
|                    | ver-        |
|                    | sion-       |
|                    | s/the       |
|                    | num-        |
|                    | ber         |
|                    | of          |
|                    | keys)       |

| Configuration item | Description |
|--------------------|-------------|
| gc.                | The         |
| ↳ batchum-         |             |
| ↳ - ber            |             |
| ↳ keysf            |             |
| ↳    keys          |             |
| pro-               |             |
| cessed             |             |
| in                 |             |
| one                |             |
| batch              |             |
| gc.                | The         |
| ↳ max maxi-        |             |
| ↳ - mum            |             |
| ↳ writbytes        |             |
| ↳ - that           |             |
| ↳ bytesn           |             |
| ↳ - be             |             |
| ↳ per writ-        |             |
| ↳ - ten            |             |
| ↳ sec into         |             |
| ↳    RocksDB       |             |
| per                |             |
| sec-               |             |
| ond                |             |
| gc.                | Whether     |
| ↳ enableen-        |             |
| ↳ - able           |             |
| ↳ compactiion      |             |
| ↳ - paction        |             |
| ↳ filtfiter        |             |
| ↳                  |             |

| item                | Description |
|---------------------|-------------|
| gc.                 | Whether     |
| ↳ <b>compaction</b> |             |
| ↳ - skip            |             |
| ↳ <b>filter</b>     |             |
| ↳ - clus-           |             |
| ↳ <b>skipper</b>    |             |
| ↳ - ver-            |             |
| ↳ <b>version</b>    |             |
| ↳ - check           |             |
| ↳ <b>checkf</b>     |             |
| ↳ com-              |             |
| paction             |             |
| filter              |             |
| (not                |             |
| re-                 |             |
| leased)             |             |
| {db-                | The         |
| ↳ <b>name</b>       | maximum-    |
| ↳ }.                | number      |
| ↳ <b>max size</b>   |             |
| ↳ - of              |             |
| ↳ <b>total</b>      |             |
| ↳ - WAL             |             |
| ↳ <b>wal</b>        |             |
| ↳ -                 |             |
| ↳ <b>size</b>       |             |
| ↳                   |             |
| {db-                | The         |
| ↳ <b>name</b>       | number-     |
| ↳ }.                | background  |
| ↳ <b>max of</b>     |             |
| ↳ - back-           |             |
| ↳ <b>background</b> |             |
| ↳ - threads         |             |
| ↳ <b>jobs</b>       |             |
| ↳     RocksDB       |             |

---

| item                          | Description |
|-------------------------------|-------------|
| {db-                          | The         |
| ↳ name <del>total</del>       |             |
| ↳ }. num-                     |             |
| ↳ max ber                     |             |
| ↳ - of                        |             |
| ↳ openfiles                   |             |
| ↳ - that                      |             |
| ↳ files <del>RocksDB</del>    |             |
| ↳ can                         |             |
| ↳ open                        |             |
| {db-                          | The         |
| ↳ name <del>size</del>        |             |
| ↳ }. of                       |             |
| ↳ compact <del>deadhead</del> |             |
| ↳ - ↳                         |             |
| ↳ read <del>ahead</del>       |             |
| ↳ - ing                       |             |
| ↳ size <del>com-</del>        |             |
| ↳ paction                     |             |
| {db-                          | The         |
| ↳ name <del>rate</del>        |             |
| ↳ }. at                       |             |
| ↳ bytes <del>which</del>      |             |
| ↳ - OS                        |             |
| ↳ per incre-                  |             |
| ↳ - men-                      |             |
| ↳ synd <del>ally</del>        |             |
| ↳ syn-                        |             |
| ↳ chro-                       |             |
| ↳ nizes                       |             |
| ↳ files                       |             |
| ↳ to                          |             |
| ↳ disk                        |             |
| ↳ while                       |             |
| ↳ these                       |             |
| ↳ files                       |             |
| ↳ are                         |             |
| ↳ being                       |             |
| ↳ writ-                       |             |
| ↳ ten                         |             |
| ↳ asyn-                       |             |
| ↳ chronously                  |             |

| item       | Description |
|------------|-------------|
| {db-       | The         |
| ↳ name     | rate        |
| ↳ }.       | at          |
| ↳ wal      | which       |
| ↳ -        | OS          |
| ↳ bytes    | acre-       |
| ↳ -        | men-        |
| ↳ per      | tally       |
| ↳ -        | syn-        |
| ↳ syncro-  |             |
| ↳          | nizes       |
| ↳ WAL      |             |
| ↳ files    |             |
| ↳ to       |             |
| ↳ disk     |             |
| ↳ while    |             |
| ↳ the      |             |
| ↳ WAL      |             |
| ↳ files    |             |
| ↳ are      |             |
| ↳ being    |             |
| ↳ writ-    |             |
| ↳ ten      |             |
| {db-       | The         |
| ↳ name     | axis-       |
| ↳ }.       | mum         |
| ↳ writable |             |
| ↳ -        | size        |
| ↳ file     | ased        |
| ↳ -        | in          |
| ↳ max      | Writable-   |
| ↳ -        | FileWrite   |
| ↳ buffer   |             |
| ↳ -        |             |
| ↳ size     |             |
| ↳          |             |

---

| item     | Description |
|----------|-------------|
| {db-     | The         |
| ↳ name   | cache       |
| ↳ }.     | { size      |
| ↳ cf     | of a        |
| ↳ -      | block       |
| ↳ name   |             |
| ↳ }.     |             |
| ↳ block  |             |
| ↳ -      |             |
| ↳ cache  |             |
| ↳ -      |             |
| ↳ size   |             |
| ↳        |             |
| {db-     | The         |
| ↳ name   | size        |
| ↳ }.     | { of a      |
| ↳ cf     | memtable    |
| ↳ -      |             |
| ↳ name   |             |
| ↳ }.     |             |
| ↳ write  |             |
| ↳ -      |             |
| ↳ buffer |             |
| ↳ -      |             |
| ↳ size   |             |
| ↳        |             |
| {db-     | The         |
| ↳ name   | maxi-       |
| ↳ }.     | { num       |
| ↳ cf     | num-        |
| ↳ -      | ber         |
| ↳ name   | of          |
| ↳ }.     | memta-      |
| ↳ max    | bles        |
| ↳ -      |             |
| ↳ write  |             |
| ↳ -      |             |
| ↳ buffer |             |
| ↳ -      |             |
| ↳ number |             |
| ↳        |             |

---

| item     | Description |
|----------|-------------|
| {db-     | The         |
| ↳ name   | maximum-    |
| ↳ }.     | { number of |
| ↳ cf     | number of   |
| ↳ -      | bytes per   |
| ↳ name   | name of     |
| ↳ }.     | bytes at    |
| ↳ max    | base        |
| ↳ -      | bytes level |
| ↳ -      | (L1)        |
| ↳ for    |             |
| ↳ -      |             |
| ↳ level  |             |
| ↳ -      |             |
| ↳ base   |             |
| ↳        |             |
| {db-     | The         |
| ↳ name   | size        |
| ↳ }.     | { of        |
| ↳ cf     | the         |
| ↳ -      | target      |
| ↳ name   | target      |
| ↳ }.     | file        |
| ↳ target |             |
| ↳ -      | base        |
| ↳ file   | level       |
| ↳ -      |             |
| ↳ size   |             |
| ↳ -      |             |
| ↳ base   |             |
| ↳        |             |

---

| item         | Description |
|--------------|-------------|
| {db-         | The         |
| ↳ name       | maximum     |
| ↳ }.         | number of   |
| ↳ cf         | number of   |
| ↳ -          | compaction  |
| ↳ nameof     | files       |
| ↳ }.         | trigger     |
| ↳ level      | L0          |
| ↳ -          | that        |
| ↳ file       | trigger     |
| ↳ -          | ger         |
| ↳ num        | compaction  |
| ↳ -          | action      |
| ↳ compaction |             |
| ↳ -          |             |
| ↳ trigger    |             |
| ↳            |             |
| {db-         | The         |
| ↳ name       | maximum     |
| ↳ }.         | number of   |
| ↳ cf         | number of   |
| ↳ -          | compaction  |
| ↳ nameof     | files       |
| ↳ }.         | trigger     |
| ↳ level      | L0          |
| ↳ -          | that        |
| ↳ slowdown   |             |
| ↳ -          | ger         |
| ↳ writewait  |             |
| ↳ -          | stall       |
| ↳ trigger    |             |
| ↳            |             |

---

| item         | Description |
|--------------|-------------|
| {db-         | The         |
| ↳ name       | maximum-    |
| ↳ }.{        | memory      |
| ↳ cf         | number      |
| ↳ -          | of bytes    |
| ↳ name@f     | files       |
| ↳ }.         | level       |
| ↳ level10L0  | that        |
| ↳ -          | stop        |
| ↳ -          | completely  |
| ↳ write@ck   | write       |
| ↳ -          | trigger     |
| ↳            |             |
| {db-         | The         |
| ↳ name       | maximum-    |
| ↳ }.{        | memory      |
| ↳ cf         | number      |
| ↳ -          | of bytes    |
| ↳ name@f     | bytes       |
| ↳ }.         | max write-  |
| ↳ -          | ten         |
| ↳ compaction | compaction  |
| ↳ -          | disk        |
| ↳ bytes@r    | bytes per   |
| ↳            | compaction  |

| Configuration item | Description |
|--------------------|-------------|
| {db-               | The         |
| ↳ namele-          |             |
| ↳ }.{ fault        |             |
| ↳ cf am-           |             |
| ↳ - plifi-         |             |
| ↳ nameca-          |             |
| ↳ }. tion          |             |
| ↳ max mul-         |             |
| ↳ - tiple          |             |
| ↳ bytes            |             |
| ↳ - each           |             |
| ↳ for layer        |             |
| ↳ -                |             |
| ↳ level            |             |
| ↳ -                |             |
| ↳ multiplier       |             |
| ↳                  |             |
| {db-               | Enables     |
| ↳ nameor           |             |
| ↳ }.{ dis-         |             |
| ↳ cf ables         |             |
| ↳ - auto-          |             |
| ↳ namenatic        |             |
| ↳ }. com-          |             |
| ↳ disable          |             |
| ↳ -                |             |
| ↳ auto             |             |
| ↳ -                |             |
| ↳ compactions      |             |
| ↳                  |             |

---

| item         | Description |
|--------------|-------------|
| {db-         | The         |
| ↳ name       | soft        |
| ↳ }.         | { limit     |
| ↳ cf         | on          |
| ↳ -          | the         |
| ↳ name       | pend-       |
| ↳ }.         | ing         |
| ↳ soft       | com-        |
| ↳ -          | paction     |
| ↳ pending    | s           |
| ↳ -          |             |
| ↳ compaction |             |
| ↳ -          |             |
| ↳ bytes      |             |
| ↳ -          |             |
| ↳ limit      |             |
| ↳            |             |
| {db-         | The         |
| ↳ name       | hard        |
| ↳ }.         | { limit     |
| ↳ cf         | on          |
| ↳ -          | the         |
| ↳ name       | pend-       |
| ↳ }.         | ing         |
| ↳ hard       | com-        |
| ↳ -          | paction     |
| ↳ pending    | s           |
| ↳ -          |             |
| ↳ compaction |             |
| ↳ -          |             |
| ↳ bytes      |             |
| ↳ -          |             |
| ↳ limit      |             |
| ↳            |             |

---

| item            | Description |
|-----------------|-------------|
| {db-            | The         |
| ↳ namenode      |             |
| ↳ }.{ of        |             |
| ↳ cf pro-       |             |
| ↳ - cess-       |             |
| ↳ naméng        |             |
| ↳ }. blob       |             |
| ↳ titanles      |             |
| ↳ .             |             |
| ↳ blob          |             |
| ↳ -             |             |
| ↳ run           |             |
| ↳ -             |             |
| ↳ mode          |             |
| ↳               |             |
| storage         | The         |
| ↳ . size        |             |
| ↳ blockf        |             |
| ↳ - shared      |             |
| ↳ cache         |             |
| ↳ . cache       |             |
| ↳ capacity      |             |
| ↳       ported  |             |
| ↳       since   |             |
| ↳       v4.0.3) |             |
| backup          | The         |
| ↳ . num-        |             |
| ↳ num ber       |             |
| ↳ - of          |             |
| ↳ threads       |             |
| ↳       threads |             |
| ↳       (sup-   |             |
| ↳       ported  |             |
| ↳       since   |             |
| ↳       v4.0.3) |             |

| item         | Description                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>split</b> | The<br>→ . thresh-<br>→ <b>qps old</b><br>→ - to ex-<br>→ <b>threshold</b><br>→ load<br>→ -<br>→ <b>base</b><br>→ -<br>→ <b>split</b><br>→<br>on a<br>Re-<br>gion.<br>If the<br>QPS<br>of<br>read<br>re-<br>quests<br>for a<br>Re-<br>gion<br>ex-<br>ceeds<br><b>qps-</b><br>→ <b>threshold</b><br>→<br>for a<br>con-<br>secu-<br>tive<br>pe-<br>riod<br>of<br>time,<br>this<br>Re-<br>gion<br>should<br>be<br>split. |

| item         | Description                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>split</b> | The<br>→ . thresh-<br>→ <b>bytold</b><br>→ - to ex-<br>→ <b>threshold</b><br>→ load<br>→ -<br>→ <b>base</b><br>→ -<br>→ <b>split</b><br>→<br>on a<br>Re-<br>gion.<br>If the<br>traf-<br>fic of<br>read<br>re-<br>quests<br>for a<br>Re-<br>gion<br>ex-<br>ceeds<br>the<br><b>byte</b><br>→ -<br>→ <b>threshold</b><br>→<br>for a<br>con-<br>secu-<br>tive<br>pe-<br>riod<br>of<br>time,<br>this<br>Re-<br>gion<br>should<br>be<br>split. |

| Configuration item | Description |
|--------------------|-------------|
|--------------------|-------------|

**split** The  
 ↳ . pa-  
 ↳ **split**ame-  
 ↳ - ter of  
 ↳ **balanced**  
 ↳ - ↳ -  
 ↳ **score** base  
 ↳ ↳ -  
 ↳ **split**  
 ↳ ,  
 which  
 en-  
 sures  
 the  
 load  
 of  
 the  
 two  
 split  
 Re-  
 gions  
 is as  
 bal-  
 anced  
 as  
 possi-  
 ble.  
 The  
 smaller  
 the  
 value  
 is,  
 the  
 more  
 bal-  
 anced  
 the  
 load  
 is.  
 But  
 set-  
 ting  
 it too  
 small  
 329 might  
 cause  
 split  
 file

| Configuration item | Description                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>split</code> | The<br>→ . pa-<br>→ <b>splitame-</b><br>→ - ter of<br>→ <b>contained</b><br>→ - → -<br>→ <b>score</b> base<br>→ → -<br>→ <b>split</b><br>→ .<br>The<br>smaller<br>the<br>value,<br>the<br>fewer<br>cross-<br>Region<br>visits<br>after<br>Re-<br>gion<br>split. |

In the table above, parameters with the `{db-name}` or `{db-name}.{cf-name}` prefix are configurations related to RocksDB. The optional values of `db-name` are `rocksdb` and `raftdb`.

- When `db-name` is `rocksdb`, the optional values of `cf-name` are `defaultcf`, `writecf`, `lockcf`, and `raftcf`.
- When `db-name` is `raftdb`, the value of `cf-name` can be `defaultcf`.

For detailed parameter description, refer to [TiKV Configuration File](#).

#### 5.8.1.3 Modify PD configuration online

Currently, PD does not support the separate configuration for each instance. All PD instances share the same configuration.

You can modify the PD configurations using the following statement:

```
set config pd `log.level`='info'
```

If the modification is successful, `Query OK` is returned:

```
Query OK, 0 rows affected (0.01 sec)
```

If a configuration item is successfully modified, the result is persisted in etcd instead of in the configuration file; the configuration in etcd will prevail in the subsequent operations. The names of some configuration items might conflict with TiDB reserved words. For these configuration items, use backtick ` to enclose them. For example, `schedule.leader-  
→ schedule-limit`.

The following PD configuration items can be modified online:

| Configuration item       | Description  |
|--------------------------|--------------|
| <code>log.</code>        | The          |
| → <code>level</code>     | → level      |
| <code>cluster</code>     | The          |
| → <code>-</code>         | clus-        |
| → <code>version</code>   | → version    |
| → <code>size</code>      | ver-<br>sion |
| <code>schedule</code>    | Controls     |
| → <code>.</code>         | the          |
| → <code>max size</code>  | → max size   |
| → <code>- limit</code>   | → - limit    |
| → <code>merge</code>     | → merge      |
| → <code>- Region</code>  | → - Region   |
| → <code>region</code>    | → region     |
| → <code>- → Merge</code> | → - → Merge  |
| → <code>size →</code>    | → size →     |
| → <code>(in MB)</code>   | → (in<br>MB) |
| <code>schedule</code>    | Specifies    |
| → <code>.</code>         | the          |
| → <code>max max-</code>  | → max maxi-  |
| → <code>- mum</code>     | → - mum      |
| → <code>merge am-</code> | → merge am-  |
| → <code>- bers</code>    | → - bers     |
| → <code>region</code>    | → region     |
| → <code>- the</code>     | → - the      |
| → <code>keys</code>      | → keys       |
| <code>Region</code>      | Region       |
| → <code>→</code>         | → →          |
| → <code>Merge</code>     | → Merge      |
| → <code>→</code>         | → →          |
| → <code>keys</code>      | → keys       |

| Configuration item | Description |
|--------------------|-------------|
| <b>schedule</b>    | Determines  |
| ↳ .                | the         |
| ↳ <b>patfd</b>     |             |
| ↳ -                | quency      |
| ↳ <b>region</b>    |             |
| ↳ -                | which       |
| ↳ <b>interval</b>  |             |
| ↳      ↳           |             |
|                    | checks      |
|                    | the         |
|                    | health      |
|                    | state       |
|                    | of a        |
|                    | Re-         |
|                    | gion        |
| <b>schedule</b>    | Determines  |
| ↳ .                | the         |
| ↳ <b>split</b>     |             |
| ↳ -                | inter-      |
| ↳ <b>merge</b>     |             |
| ↳ -                | per-        |
| ↳ <b>interval</b>  |             |
| ↳      ↳           |             |
|                    | ing         |
|                    | split       |
|                    | and         |
|                    | merge       |
|                    | oper-       |
|                    | a-          |
|                    | tions       |
|                    | on          |
|                    | the         |
|                    | same        |
|                    | Re-         |
|                    | gion        |

| Configuration item | Description |
|--------------------|-------------|
| <b>schedule</b>    | Determines  |
| → .                | the         |
| → <b>max</b>       | maxi-       |
| → -                | mum         |
| → <b>snapshot</b>  |             |
| → -                | ber         |
| → <b>count</b>     |             |
| →                  | snap-       |
|                    | shots       |
|                    | that        |
|                    | a sin-      |
|                    | gle         |
|                    | store       |
|                    | can         |
|                    | send        |
|                    | or re-      |
|                    | ceive       |
|                    | at          |
|                    | the         |
|                    | same        |
|                    | time        |
| <b>schedule</b>    | Determines  |
| → .                | the         |
| → <b>max</b>       | maxi-       |
| → -                | mum         |
| → <b>pending</b> - |             |
| → -                | ber         |
| → <b>peer</b>      | f           |
| → -                | pend-       |
| → <b>count</b>     |             |
| →                  | peers       |
|                    | in a        |
|                    | sin-        |
|                    | gle         |
|                    | store       |

| Configuration item   | Description |
|----------------------|-------------|
| <b>schedule</b>      | the         |
| ↳ .                  | down-       |
| ↳ <b>max_time</b>    | time        |
| ↳ -                  | after       |
| ↳ <b>store</b>       | which       |
| ↳ -                  | PD          |
| ↳ <b>down_judges</b> | judges      |
| ↳ -                  | that        |
| ↳ <b>time</b>        | the         |
| ↳                    | dis-        |
| ↳                    | con-        |
| ↳                    | nected      |
| ↳                    | store       |
| ↳                    | can         |
| ↳                    | not         |
| ↳                    | be re-      |
| ↳                    | cov-        |
| ↳                    | ered        |
| <b>schedule</b>      | determines  |
| ↳ .                  | the         |
| ↳ <b>leader</b>      | leader      |
| ↳ -                  | icy of      |
| ↳ <b>scheduler</b>   | scheduler   |
| ↳ -                  | schedul-    |
| ↳ <b>poling</b>      | poling      |
| ↳                    |             |
| <b>schedule</b>      | the         |
| ↳ .                  | num-        |
| ↳ <b>leader</b>      | leader      |
| ↳ -                  | of          |
| ↳ <b>scheduler</b>   | scheduler   |
| ↳ -                  | schedul-    |
| ↳ <b>limiting</b>    | limiting    |
| ↳                    | tasks       |
| ↳                    | per-        |
| ↳                    | formed      |
| ↳                    | at          |
| ↳                    | the         |
| ↳                    | same        |
| ↳                    | time        |

| Configuration item | Description |
|--------------------|-------------|
| <b>schedule</b>    | the         |
| ↳ .                | num-        |
| ↳ <b>region</b>    | region      |
| ↳ -                | of          |
| ↳ <b>schedule</b>  | schedule    |
| ↳ -                | gion        |
| ↳ <b>limit</b>     | limit       |
| ↳ <b>task</b>      | ing         |
| ↳ <b>per</b>       | tasks       |
| ↳ <b>form</b>      | per-        |
| ↳ <b>at</b>        | formed      |
| ↳ <b>the</b>       | at          |
| ↳ <b>same</b>      | the         |
| ↳ <b>time</b>      | same        |
| <b>schedule</b>    | the         |
| ↳ .                | num-        |
| ↳ <b>replica</b>   | replica     |
| ↳ -                | of          |
| ↳ <b>schedule</b>  | schedule    |
| ↳ -                | schedul-    |
| ↳ <b>limi</b>      | ing         |
| ↳ <b>task</b>      | tasks       |
| ↳ <b>per</b>       | per-        |
| ↳ <b>form</b>      | formed      |
| ↳ <b>at</b>        | at          |
| ↳ <b>the</b>       | the         |
| ↳ <b>same</b>      | same        |
| ↳ <b>time</b>      | time        |

| Configuration item   | Description |
|----------------------|-------------|
| <b>schedule</b>      | the         |
| ↳ .                  | num-        |
| ↳ <b>merge</b>       | merger      |
| ↳ -                  | of          |
| ↳ <b>schedule</b>    | schedule    |
| ↳ -                  | Region      |
| ↳ <b>limit</b>       | limit       |
| ↳     ↳ <b>Merge</b> | Merge       |
| ↳                    |             |
|                      | schedul-    |
|                      | ing         |
|                      | tasks       |
|                      | per-        |
|                      | formed      |
|                      | at          |
|                      | the         |
|                      | same        |
|                      | time        |
| <b>schedule</b>      | the         |
| ↳ .                  | num-        |
| ↳ <b>hot</b>         | hot ber     |
| ↳ -                  | of          |
| ↳ <b>region</b>      | region      |
| ↳ -                  | Re-         |
| ↳ <b>schedule</b>    | schedule    |
| ↳ -                  | schedul-    |
| ↳ <b>limit</b>       | limit       |
| ↳     ↳ tasks        | tasks       |
|                      | per-        |
|                      | formed      |
|                      | at          |
|                      | the         |
|                      | same        |
|                      | time        |

| Configuration item     | Description |
|------------------------|-------------|
| <b>schedule</b>        | Determines  |
| → .                    | the         |
| → <b>hot threshold</b> | hot thresh- |
| → - old                | - old       |
| → <b>region</b>        | region      |
| → - which              | which       |
| → <b>cacheRegion</b>   | cacheRe-    |
| → - gion               | - gion      |
| → <b>hits</b>          | hits        |
| → - con-               | - con-      |
| → <b>threshold</b>     | threshold   |
| →        ered          | ered        |
| →        a hot         | a hot       |
| →        spot          | spot        |
| <b>schedule</b>        | The         |
| → .                    | thresh-     |
| → <b>hold</b>          | hold        |
| → - ratio              | - ratio     |
| → <b>spade</b>         | spade-      |
| → - low                | - low       |
| → <b>ratio</b>         | ratio       |
| →        which         | which       |
| →        the           | the         |
| →        ca-           | ca-         |
| →        pac-          | pac-        |
| →        ity of        | ity of      |
| →        the           | the         |
| →        store         | store       |
| →        is            | is          |
| →        suffi-        | suffi-      |
| →        cient         | cient       |

| Configuration item        | Description |
|---------------------------|-------------|
| <b>schedule</b>           |             |
| <b>.</b>                  | threshold   |
| <b>low</b>                | old         |
| <b>-ratio</b>             |             |
| <b>spaceabove</b>         |             |
| <b>-which</b>             |             |
| <b>ratio</b>              |             |
| <b>ca-</b>                |             |
| <b>pac-</b>               |             |
| <b>ity</b>                | of          |
| <b>the</b>                |             |
| <b>store</b>              |             |
| <b>is in-</b>             |             |
| <b>suffi-</b>             |             |
| <b>cient</b>              |             |
| <b>scheduleControls</b>   |             |
| <b>.</b>                  | the         |
| <b>tolerance</b>          |             |
| <b>-</b>                  |             |
| <b>sizebuffer</b>         |             |
| <b>-size</b>              |             |
| <b>ratio</b>              |             |
| <b></b>                   |             |
| <b>scheduleDetermines</b> |             |
| <b>.</b>                  | whether     |
| <b>enable</b>             | been-       |
| <b>-able</b>              |             |
| <b>remove</b>             |             |
| <b>-fea-</b>              |             |
| <b>downtime</b>           |             |
| <b>-that</b>              |             |
| <b>replica-</b>           |             |
| <b>-mati-</b>             |             |
| <b>cally</b>              |             |
| <b>re-</b>                |             |
| <b>moves</b>              |             |
| <b>DownReplica</b>        |             |
| <b></b>                   |             |

| Configuration item | Description                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------------------|
| <b>schedule</b>    | Determines whether . been- able fea- offline that repli- grates OfflineReplica                           |
| <b>schedule</b>    | Determines whether . been- able make the fea- up ture that repli- mati- cally sup- ple- ments repli- cas |
| <b>schedule</b>    | Determines whether . been- able remove fea- extra that repli- moves extra repli- cas                     |

| Configuration item | Description                    |
|--------------------|--------------------------------|
| <b>schedule</b>    | Determines whether . been-able |
| <b>enable</b>      | location                       |
| <b>location</b>    | - tion                         |
| <b>replacement</b> | check                          |
| <b>schedule</b>    | Determines whether . been-able |
| <b>enable</b>      | cross-table                    |
| <b>cross-table</b> | merge                          |
| <b>table</b>       | -                              |
| <b>merge</b>       |                                |
| <b>schedule</b>    | Enables                        |
| <b>enable</b>      | . one-                         |
| <b>enable</b>      | only                           |
| <b>one</b>         | merge, which                   |
| <b>only</b>        | only                           |
| <b>way</b>         | al-                            |
| <b>ways</b>        | lows                           |
| <b>merge</b>       | erg-                           |
| <b>merge</b>       | ing                            |
| <b>merge</b>       | with                           |
| <b>merge</b>       | the                            |
| <b>next</b>        | next                           |
| <b>adjacent</b>    | adja-                          |
| <b>region</b>      | cent                           |
| <b>region</b>      | Re-                            |
| <b>region</b>      | gion                           |

| item                               | Description |
|------------------------------------|-------------|
| <b>replication</b>                 |             |
| $\hookrightarrow$ .                | the         |
| $\hookrightarrow$ <b>max</b>       | maxi-       |
| $\hookrightarrow$ -                | num         |
| $\hookrightarrow$ <b>replicas</b>  |             |
| $\hookrightarrow$ ber              |             |
| $\hookrightarrow$ of               |             |
| $\hookrightarrow$ repli-           |             |
| $\hookrightarrow$ cas              |             |
| <b>replication</b>                 |             |
| $\hookrightarrow$ .                | topol-      |
| $\hookrightarrow$ <b>location</b>  |             |
| $\hookrightarrow$ -                | infor-      |
| $\hookrightarrow$ <b>labels</b> -  |             |
| $\hookrightarrow$ tion             |             |
| $\hookrightarrow$ of a             |             |
| $\hookrightarrow$ TiKV             |             |
| $\hookrightarrow$ clus-            |             |
| $\hookrightarrow$ ter              |             |
| <b>replications</b>                |             |
| $\hookrightarrow$ .                | Place-      |
| $\hookrightarrow$ <b>enable</b>    |             |
| $\hookrightarrow$ -                | Rules       |
| $\hookrightarrow$ <b>placement</b> |             |
| $\hookrightarrow$ -                |             |
| $\hookrightarrow$ <b>rules</b>     |             |
| $\hookrightarrow$                  |             |
| <b>replications</b>                |             |
| $\hookrightarrow$ .                | the         |
| $\hookrightarrow$ <b>striably</b>  |             |
| $\hookrightarrow$ -                | check       |
| $\hookrightarrow$ <b>match</b>     |             |
| $\hookrightarrow$ -                |             |
| $\hookrightarrow$ <b>label</b>     |             |
| $\hookrightarrow$                  |             |

| item                | Description |
|---------------------|-------------|
| pd-                 | Enables     |
| ↳ <b>server</b> -   |             |
| ↳ . pen-            |             |
| ↳ <b>use</b> dent   |             |
| ↳ - Re-             |             |
| ↳ <b>region</b>     |             |
| ↳ - stor-           |             |
| ↳ <b>storage</b>    |             |
| ↳                   |             |
| pd-                 | Sets        |
| ↳ <b>server</b>     |             |
| ↳ . maxi-           |             |
| ↳ <b>max</b> mum    |             |
| ↳ - inter-          |             |
| ↳ <b>gap</b> val of |             |
| ↳ - reset-          |             |
| ↳ <b>reset</b> ing  |             |
| ↳ - times-          |             |
| ↳ <b>ts</b> tamp    |             |
| ↳      (BR)         |             |
| pd-                 | Sets        |
| ↳ <b>server</b>     |             |
| ↳ . clus-           |             |
| ↳ <b>key</b> ter    |             |
| ↳ - key             |             |
| ↳ <b>type</b> type  |             |
| ↳                   |             |
| pd-                 | Sets        |
| ↳ <b>server</b>     |             |
| ↳ . stor-           |             |
| ↳ <b>metric</b>     |             |
| ↳ - ad-             |             |
| ↳ <b>storage</b> s  |             |
| ↳      of           |             |
| ↳      the          |             |
| ↳      clus-        |             |
| ↳      ter          |             |
| ↳      met-         |             |
| ↳      rics         |             |

| Configuration item | Description |
|--------------------|-------------|
| pd-                | Sets        |
| ↳ server           |             |
| ↳ . dash-          |             |
| ↳ dashboard        |             |
| ↳ - ad-            |             |
| ↳ address          |             |
| ↳                  |             |
| ↳ replication      |             |
| ↳ - the            |             |
| ↳ mode             |             |
| ↳ backup           |             |
| ↳ . mode           |             |
| ↳ replication      |             |
| ↳ -                |             |
| ↳ mode             |             |
| ↳                  |             |

For detailed parameter description, refer to [PD Configuration File](#).

#### 5.8.1.4 Modify TiDB configuration online

Currently, the method of changing TiDB configuration is different from that of changing TiKV and PD configurations. You can modify TiDB configuration by using [system variables](#).

The following example shows how to modify `slow-threshold` online by using the `tidb_slow_log_threshold` variable.

The default value of `slow-threshold` is 300 ms. You can set it to 200 ms by using `tidb_slow_log_threshold`.

```
set tidb_slow_log_threshold = 200;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select @@tidb_slow_log_threshold;
```

```
+-----+
| @@tidb_slow_log_threshold |
+-----+
| 200 |
+-----+
1 row in set (0.00 sec)
```

The following TiDB configuration items can be modified online:

| Configuration item      | SQL variable                        |
|-------------------------|-------------------------------------|
| mem-quota-query         | tidb_mem_quota_query                |
| log.enable-slow-log     | tidb_enable_slow_log                |
| log.slow-threshold      | tidb_slow_log_threshold             |
| log.expensive-threshold | tidb_expensive_query_time_threshold |

## 5.9 Online Unsafe Recovery

**Warning:**

- Online Unsafe Recovery is a type of lossy recovery. If you use this feature, the integrity of data and data indexes cannot be guaranteed.
- Online Unsafe Recovery is an experimental feature, and it is **NOT** recommended to use it in the production environment. The interface, strategy, and internal implementation of this feature might change when it becomes generally available (GA). Although this feature has been tested in some scenarios, it is not thoroughly validated and might cause system unavailability.
- It is recommended to perform the feature-related operations with the support from the TiDB team. If any misoperation is performed, it might be hard to recover the cluster.

When permanently damaged replicas cause part of data on TiKV to be unreadable and unwritable, you can use the Online Unsafe Recovery feature to perform a lossy recovery operation.

### 5.9.1 Feature description

In TiDB, the same data might be stored in multiple stores at the same time according to the replica rules defined by users. This guarantees that data is still readable and writable even if a single or a few stores are temporarily offline or damaged. However, when most or all replicas of a Region go offline during a short period of time, the Region becomes temporarily unavailable, by design, to ensure data integrity.

Suppose that multiple replicas of a data range encounter issues like permanent damage (such as disk damage), and these issues cause the stores to stay offline. In this case, this data range is temporarily unavailable. If you want the cluster back in use and also accept data rewind or data loss, in theory, you can re-form the majority of replicas by manually removing the failed replicas from the group. This allows application-layer services to read and write this data range (might be stale or empty) again.

In this case, if some stores with loss-tolerant data are permanently damaged, you can perform a lossy recovery operation by using Online Unsafe Recovery. Using this feature, PD, under its global perspective, collects the metadata of data shards from all stores and generates a real-time and complete recovery plan. Then, PD distributes the plan to all surviving stores to make them perform data recovery tasks. In addition, once the data recovery plan is distributed, PD periodically monitors the recovery progress and re-send the plan when necessary.

### 5.9.2 User scenarios

The Online Unsafe Recovery feature is suitable for the following scenarios:

- The data for application services is unreadable and unwritable, because permanently damaged stores cause the stores to fail to restart.
- You can accept data loss and want the affected data to be readable and writable.
- You want to perform a one-stop online data recovery operation.

### 5.9.3 Usage

#### 5.9.3.1 Prerequisites

Before using Online Unsafe Recovery, make sure that the following requirements are met:

- The offline stores indeed cause some pieces of data to be unavailable.
- The offline stores cannot be automatically recovered or restarted.

#### 5.9.3.2 Step 1. Disable all types of scheduling

You need to temporarily disable all types of internal scheduling, such as load balancing. After disabling them, it is recommended to wait for about 10 minutes so that the triggered scheduling can have sufficient time to complete the scheduled tasks.

##### Note:

After the scheduling is disabled, the system cannot resolve data hotspot issues. Therefore, you need to enable the scheduling as soon as possible after the recovery is completed.

1. Use PD Control to get the current configuration by running the `config show` command.
2. Use PD Control to disable all types of scheduling. For example:

- `config set region-schedule-limit 0`
- `config set replica-schedule-limit 0`
- `config set merge-schedule-limit 0`

### 5.9.3.3 Step 2. Remove the stores that cannot be automatically recovered

Use PD Control to remove the stores that cannot be automatically recovered by running the `unsafe remove-failed-stores <store_id>[,<store_id>,...]` command.

**Note:**

The returned result of this command only indicates that the request is accepted, not that the recovery is completed successfully. The stores are actually recovered in the background.

### 5.9.3.4 Step 3. Check the progress

When the above store removal command runs successfully, you can use PD Control to check the removal progress by running the `unsafe remove-failed-stores show` command. When the command result shows “Last recovery has finished”, the system recovery is completed.

### 5.9.3.5 Step 4. Test read and write tasks

After the progress command shows that the recovery task is completed, you can try to execute some simple SQL queries like the following example or perform write tasks to ensure that the data is readable and writable.

```
select count(*) from table_that_suffered_from_group_majority_failure;
```

**Note:**

The situation that data can be read and written does not indicate there is no data loss.

### 5.9.3.6 Step 5. Restart the scheduling

To restart the scheduling, you need to adjust the 0 value of `config set region-schedule-limit 0`, `config set replica-schedule-limit 0`, and `config set merge-schedule-limit 0` modified in step 1 to the initial values.

Then, the whole process is finished.

## 6 Monitor and Alert

### 6.1 TiDB Monitoring Framework Overview

The TiDB monitoring framework adopts two open source projects: Prometheus and Grafana. TiDB uses [Prometheus](#) to store the monitoring and performance metrics and [Grafana](#) to visualize these metrics.

#### 6.1.1 About Prometheus in TiDB

As a time series database, Prometheus has a multi-dimensional data model and flexible query language. As one of the most popular open source projects, Prometheus has been adopted by many companies and organizations and has a very active community. PingCAP is one of the active developers and adopters of Prometheus for monitoring and alerting in TiDB, TiKV and PD.

Prometheus consists of multiple components. Currently, TiDB uses the following of them:

- The Prometheus Server to scrape and store time series data
- The client libraries to customize necessary metrics in the application
- An Alertmanager for the alerting mechanism

The diagram is as follows:

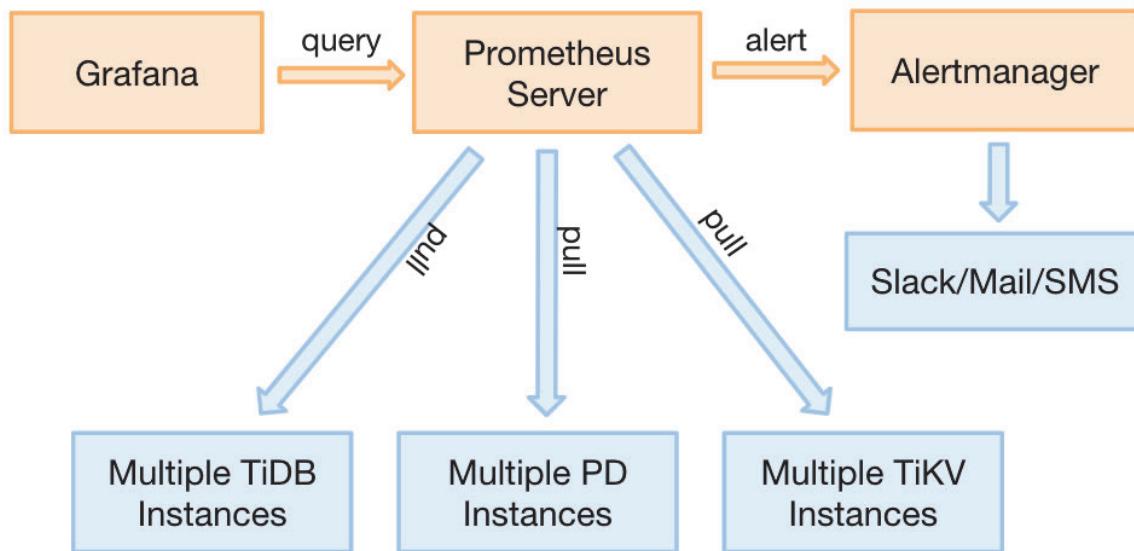


Figure 42: diagram

### 6.1.2 About Grafana in TiDB

Grafana is an open source project for analyzing and visualizing metrics. TiDB uses Grafana to display the performance metrics as follows:

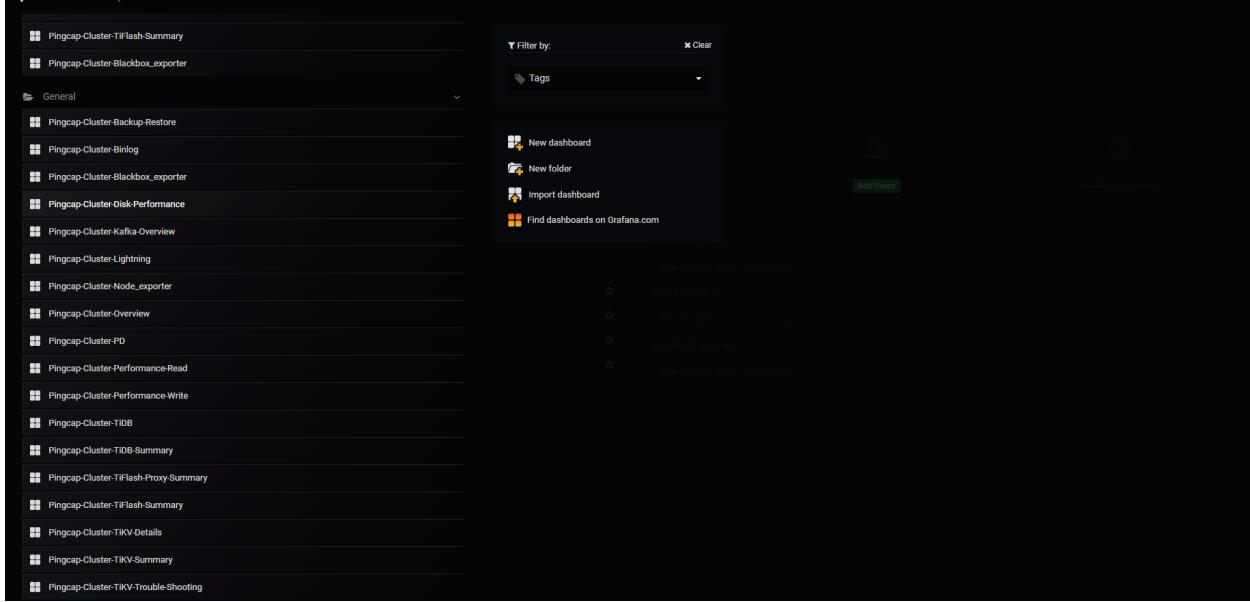


Figure 43: Grafana monitored\_groups

- {TiDB\_Cluster\_name}-Backup-Restore: Monitoring metrics related to backup and restore.
- {TiDB\_Cluster\_name}-Binlog: Monitoring metrics related to TiDB Binlog.
- {TiDB\_Cluster\_name}-Blackbox\_exporter: Monitoring metrics related to network probe.
- {TiDB\_Cluster\_name}-Disk-Performance: Monitoring metrics related to disk performance.
- {TiDB\_Cluster\_name}-Kafka-Overview: Monitoring metrics related to Kafka.
- {TiDB\_Cluster\_name}-Lightning: Monitoring metrics related to TiDB Lightning.
- {TiDB\_Cluster\_name}-Node\_exporter: Monitoring metrics related to the operating system.
- {TiDB\_Cluster\_name}-Overview: Monitoring overview related to important components.
- {TiDB\_Cluster\_name}-PD: Monitoring metrics related to the PD server.
- {TiDB\_Cluster\_name}-Performance-Read: Monitoring metrics related to read performance.
- {TiDB\_Cluster\_name}-Performance-Write: Monitoring metrics related to write performance.
- {TiDB\_Cluster\_name}-TiDB: Detailed monitoring metrics related to the TiDB server.

- {TiDB\_Cluster\_name}-TiDB-Summary: Monitoring overview related to TiDB.
- {TiDB\_Cluster\_name}-TiFlash-Proxy-Summary: Monitoring overview of the proxy server that is used to replicate data to TiFlash.
- {TiDB\_Cluster\_name}-TiFlash-Summary: Monitoring overview related to TiFlash.
- {TiDB\_Cluster\_name}-TiKV-Details: Detailed monitoring metrics related to the TiKV server.
- {TiDB\_Cluster\_name}-TiKV-Summary: Monitoring overview related to the TiKV server.
- {TiDB\_Cluster\_name}-TiKV-Trouble-Shooting: Monitoring metrics related to the TiKV error diagnostics.
- {TiDB\_Cluster\_name}-TiCDC: Detailed monitoring metrics related to TiCDC.

Each group has multiple panel labels of monitoring metrics, and each panel contains detailed information of multiple monitoring metrics. For example, the **Overview** monitoring group has five panel labels, and each labels corresponds to a monitoring panel. See the following UI:

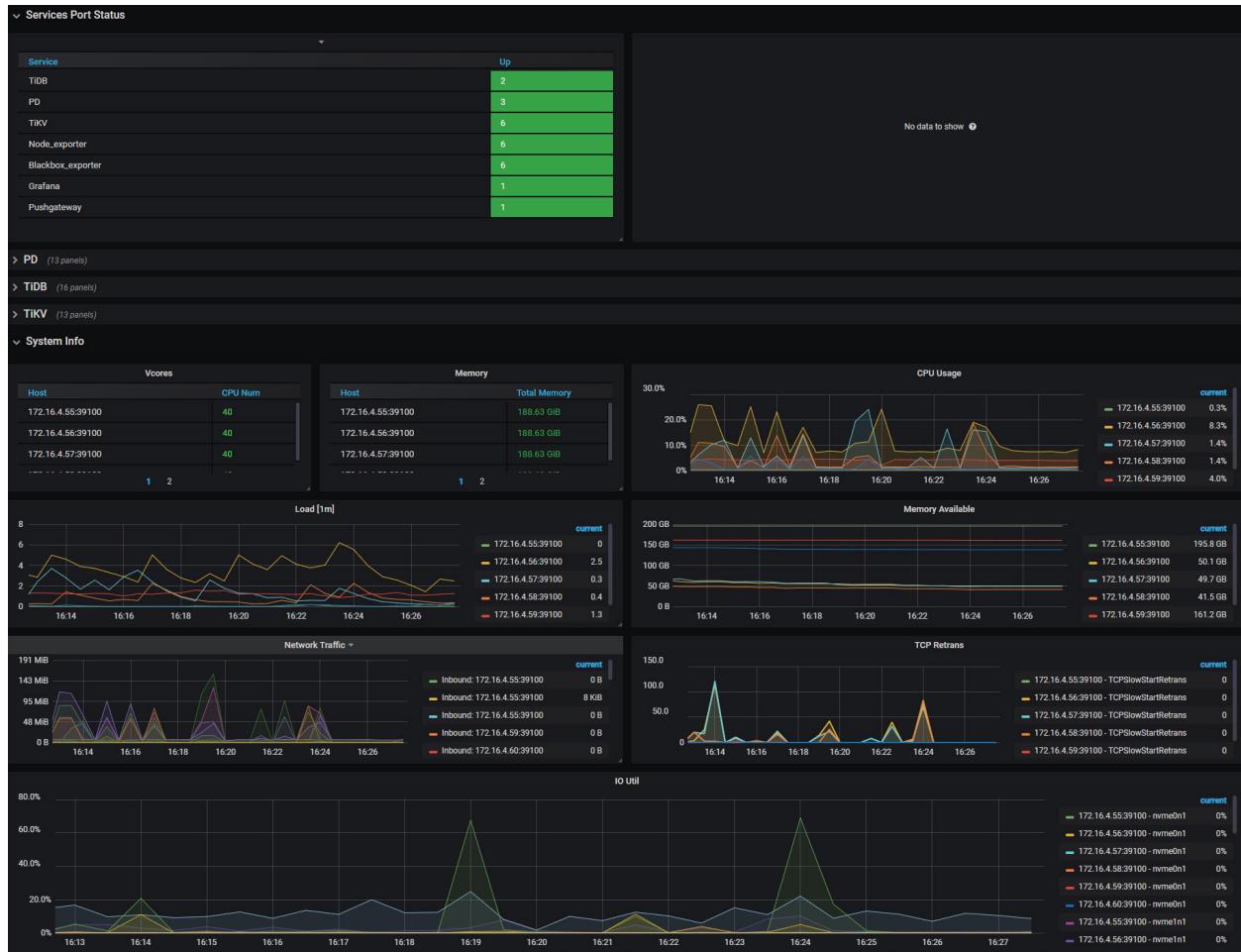


Figure 44: Grafana Overview

## 6.2 TiDB Monitoring API

You can use the following types of interfaces to monitor the TiDB cluster status:

- **The status interface**: this interface uses the HTTP interface to get the component information. Using this interface, you can get the **running status** of the current TiDB server and the **storage information** of a table.
- **The metrics interface**: this interface uses Prometheus to record the detailed information of the various operations in components and views these metrics using Grafana.

### 6.2.1 Use the status interface

The status interface monitors the basic information of a specific component in the TiDB cluster. It can also act as the monitor interface for Keepalive messages. In addition, the status interface for the Placement Driver (PD) can get the details of the entire TiKV cluster.

#### 6.2.1.1 TiDB server

- TiDB API address: `http://${host}:${port}`
- Default port: 10080

#### 6.2.1.2 Running status

The following example uses `http://${host}:${port}/status` to get the current status of the TiDB server and to determine whether the server is alive. The result is returned in **JSON** format.

```
curl http://127.0.0.1:10080/status
{
 connections: 0, # The current number of clients connected to the TiDB
 ↪ server.
 version: "5.7.25-TiDB-v3.0.0-beta-250-g778c3f4a5", # The TiDB version
 ↪ number.
 git_hash: "778c3f4a5a716880bcd1d71b257c8165685f0d70" # The Git Hash of
 ↪ the current TiDB code.
}
```

#### 6.2.1.2.1 Storage information

The following example uses `http://${host}:${port}/schema_storage/${db}/${table}` to get the storage information of the specific data table. The result is returned in **JSON** format.

```
curl http://127.0.0.1:10080/schema_storage/mysql/stats_histograms
```

```
{
 "table_schema": "mysql",
 "table_name": "stats_histograms",
 "table_rows": 0,
 "avg_row_length": 0,
 "data_length": 0,
 "max_data_length": 0,
 "index_length": 0,
 "data_free": 0
}
```

```
curl http://127.0.0.1:10080/schema_storage/test
```

```
[
 {
 "table_schema": "test",
 "table_name": "test",
 "table_rows": 0,
 "avg_row_length": 0,
 "data_length": 0,
 "max_data_length": 0,
 "index_length": 0,
 "data_free": 0
 }
]
```

### 6.2.1.3 PD server

- PD API address: `http://${host}:${port}/pd/api/v1/${api_name}`
- Default port: 2379
- Details about API names: see [PD API doc](#)

The PD interface provides the status of all the TiKV servers and the information about load balancing. See the following example for the information about a single-node TiKV cluster:

```
curl http://127.0.0.1:2379/pd/api/v1/stores
{
 "count": 1, # The number of TiKV nodes.
 "stores": [# The list of TiKV nodes.
 # The details about the single TiKV node.
 {
 "store": {
 "id": 1,
 "region_count": 1,
 "store_id": 1
 }
 }
]
}
```

```

 "id": 1,
 "address": "127.0.0.1:20160",
 "version": "3.0.0-beta",
 "state_name": "Up"
},
"status": {
 "capacity": "20 GiB", # The total capacity.
 "available": "16 GiB", # The available capacity.
 "leader_count": 17,
 "leader_weight": 1,
 "leader_score": 17,
 "leader_size": 17,
 "region_count": 17,
 "region_weight": 1,
 "region_score": 17,
 "region_size": 17,
 "start_ts": "2019-03-21T14:09:32+08:00", # The starting timestamp.
 "last_heartbeat_ts": "2019-03-21T14:14:22.961171958+08:00", # The
 ↪ timestamp of the last heartbeat.
 "uptime": "4m50.961171958s"
}
]

```

### 6.2.2 Use the metrics interface

The metrics interface monitors the status and performance of the entire TiDB cluster.

- If you use other deployment ways, [deploy Prometheus and Grafana](#) before using this interface.

After Prometheus and Grafana are successfully deployed, [configure Grafana](#).

## 6.3 Deploy Monitoring Services for the TiDB Cluster

This document is intended for users who want to manually deploy TiDB monitoring and alert services.

If you deploy the TiDB cluster using TiUP, the monitoring and alert services are automatically deployed, and no manual deployment is needed.

### 6.3.1 Deploy Prometheus and Grafana

Assume that the TiDB cluster topology is as follows:

| Name  | Host IP         | Services                                    |
|-------|-----------------|---------------------------------------------|
| Node1 | 192.168.199.113 | PD1, TiDB, node_export, Prometheus, Grafana |
| Node2 | 192.168.199.114 | PD2, node_export                            |
| Node3 | 192.168.199.115 | PD3, node_export                            |
| Node4 | 192.168.199.116 | TiKV1, node_export                          |
| Node5 | 192.168.199.117 | TiKV2, node_export                          |
| Node6 | 192.168.199.118 | TiKV3, node_export                          |

### 6.3.1.1 Step 1: Download the binary package

```
Downloads the package.
wget https://download.pingcap.org/prometheus-2.27.1.linux-amd64.tar.gz
wget https://download.pingcap.org/node_exporter-0.17.0.linux-amd64.tar.gz
wget https://download.pingcap.org/grafana-6.1.6.linux-amd64.tar.gz
```

```
Extracts the package.
tar -xzf prometheus-2.27.1.linux-amd64.tar.gz
tar -xzf node_exporter-0.17.0.linux-amd64.tar.gz
tar -xzf grafana-6.1.6.linux-amd64.tar.gz
```

### 6.3.1.2 Step 2: Start node\_exporter on Node1, Node2, Node3, and Node4

```
cd node_exporter-0.17.0.linux-amd64

Starts the node_exporter service.
$./node_exporter --web.listen-address=:9100 \
--log.level="info" &
```

### 6.3.1.3 Step 3: Start Prometheus on Node1

Edit the Prometheus configuration file:

```
cd prometheus-2.27.1.linux-amd64 &&
vi prometheus.yml
```

```
...

global:
 scrape_interval: 15s # By default, scrape targets every 15 seconds.
 evaluation_interval: 15s # By default, scrape targets every 15 seconds.
 # scrape_timeout is set to the global default value (10s).
 external_labels:
 cluster: 'test-cluster'
 monitor: "prometheus"
```

```

scrape_configs:
 - job_name: 'overwritten-nodes'
 honor_labels: true # Do not overwrite job & instance labels.
 static_configs:
 - targets:
 - '192.168.199.113:9100'
 - '192.168.199.114:9100'
 - '192.168.199.115:9100'
 - '192.168.199.116:9100'
 - '192.168.199.117:9100'
 - '192.168.199.118:9100'

 - job_name: 'tidb'
 honor_labels: true # Do not overwrite job & instance labels.
 static_configs:
 - targets:
 - '192.168.199.113:10080'

 - job_name: 'pd'
 honor_labels: true # Do not overwrite job & instance labels.
 static_configs:
 - targets:
 - '192.168.199.113:2379'
 - '192.168.199.114:2379'
 - '192.168.199.115:2379'

 - job_name: 'tikv'
 honor_labels: true # Do not overwrite job & instance labels.
 static_configs:
 - targets:
 - '192.168.199.116:20180'
 - '192.168.199.117:20180'
 - '192.168.199.118:20180'

...

```

Start the Prometheus service:

```
$./prometheus \
--config.file="../prometheus.yml" \
--web.listen-address=:9090" \
--web.external-url="http://192.168.199.113:9090/" \
--web.enable-admin-api \
--log.level="info" \
```

```
--storage.tsdb.path=".~/data.metrics" \
--storage.tsdb.retention="15d" &
```

#### 6.3.1.4 Step 4: Start Grafana on Node1

Edit the Grafana configuration file:

```
cd grafana-6.1.6 &&
vi conf/grafana.ini

...
[paths]
data = ./data
logs = ./data/log
plugins = ./data/plugins
[server]
http_port = 3000
domain = 192.168.199.113
[database]
[session]
[analytics]
check_for_updates = true
[security]
admin_user = admin
admin_password = admin
[snapshots]
[users]
[auth.anonymous]
[auth.basic]
[auth.ldap]
[smtp]
[emails]
[log]
mode = file
[log.console]
[log.file]
level = info
format = text
[log.syslog]
[event_publisher]
[dashboards.json]
enabled = false
path = ./data/dashboards
[metrics]
```

```
[grafana_net]
url = https://grafana.net

...
```

Start the Grafana service:

```
$./bin/grafana-server \
--config="../conf/grafana.ini" &
```

### 6.3.2 Configure Grafana

This section describes how to configure Grafana.

#### 6.3.2.1 Step 1: Add a Prometheus data source

1. Log in to the Grafana Web interface.

- Default address: <http://localhost:3000>
- Default account: admin
- Default password: admin

**Note:**

For the **Change Password** step, you can choose **Skip**.

2. In the Grafana sidebar menu, click **Data Source** within the **Configuration**.

3. Click **Add data source**.

4. Specify the data source information.

- Specify a **Name** for the data source.
- For **Type**, select **Prometheus**.
- For **URL**, specify the Prometheus address.
- Specify other fields as needed.

5. Click **Add** to save the new data source.

### 6.3.2.2 Step 2: Import a Grafana dashboard

To import a Grafana dashboard for the PD server, the TiKV server, and the TiDB server, take the following steps respectively:

1. Click the Grafana logo to open the sidebar menu.
2. In the sidebar menu, click **Dashboards -> Import** to open the **Import Dashboard** window.
3. Click **Upload .json File** to upload a JSON file (Download [TiDB Grafana configuration file](#)).

**Note:**

For the TiKV, PD, and TiDB dashboards, the corresponding JSON files are `tikv_summary.json`, `tikv_details.json`, `tikv_trouble_shooting.json`, `pd.json`, `tidb.json`, and `tidb_summary.json`.

4. Click **Load**.
5. Select a Prometheus data source.
6. Click **Import**. A Prometheus dashboard is imported.

### 6.3.3 View component metrics

Click **New dashboard** in the top menu and choose the dashboard you want to view.

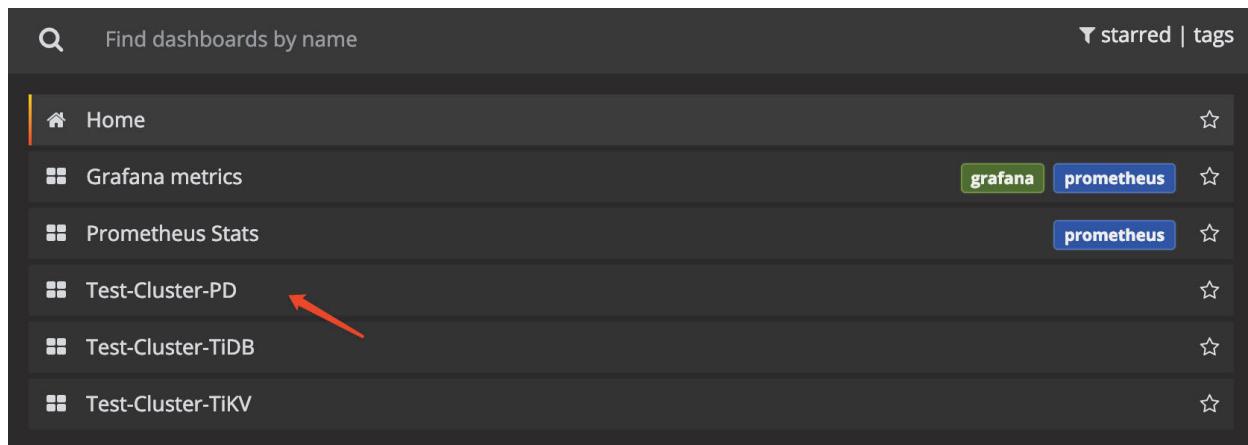


Figure 45: view dashboard

You can get the following metrics for cluster components:

- **TiDB server:**

- Query processing time to monitor the latency and throughput
- The DDL process monitoring
- TiKV client related monitoring
- PD client related monitoring

- **PD server:**

- The total number of times that the command executes
- The total number of times that a certain command fails
- The duration that a command succeeds
- The duration that a command fails
- The duration that a command finishes and returns result

- **TiKV server:**

- Garbage Collection (GC) monitoring
- The total number of times that the TiKV command executes
- The duration that Scheduler executes commands
- The total number of times of the Raft propose command
- The duration that Raft executes commands
- The total number of times that Raft commands fail
- The total number of times that Raft processes the ready state

## 6.4 Export Grafana Snapshots

Metrics data is important in troubleshooting. When you request remote assistance, sometimes the support staff need to view the Grafana dashboards to diagnose problems. [MetricsTool](#) can help export snapshots of Grafana dashboards as local files and visualize these snapshots. You can share these snapshots with outsiders and allow them to accurately read out the graphs, without giving out access to other sensitive information on the Grafana server.

### 6.4.1 Usage

MetricsTool can be accessed from <https://metricstool.pingcap.com/>. It consists of three sets of tools:

- **Export:** A user script running on the browser's Developer Tool, allowing you to download a snapshot of all visible panels in the current dashboard on any Grafana v6.x.x server.

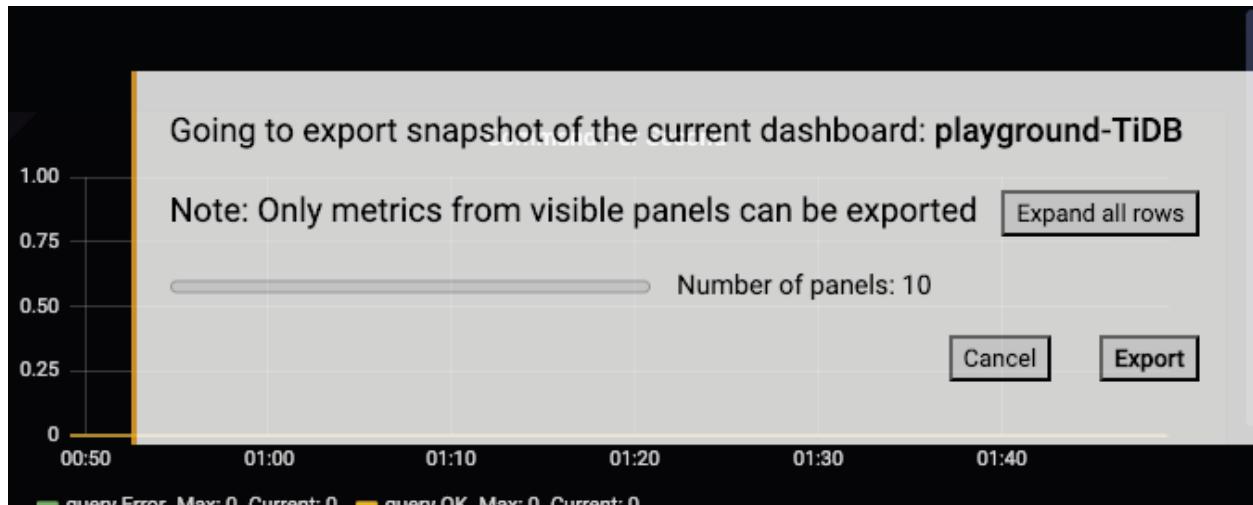


Figure 46: Screenshot of MetricsTool Exporter after running the user script

- **Visualize:** A web page visualizing the exported snapshot files. The visualized snapshots can be operated in the same way as live Grafana dashboards.

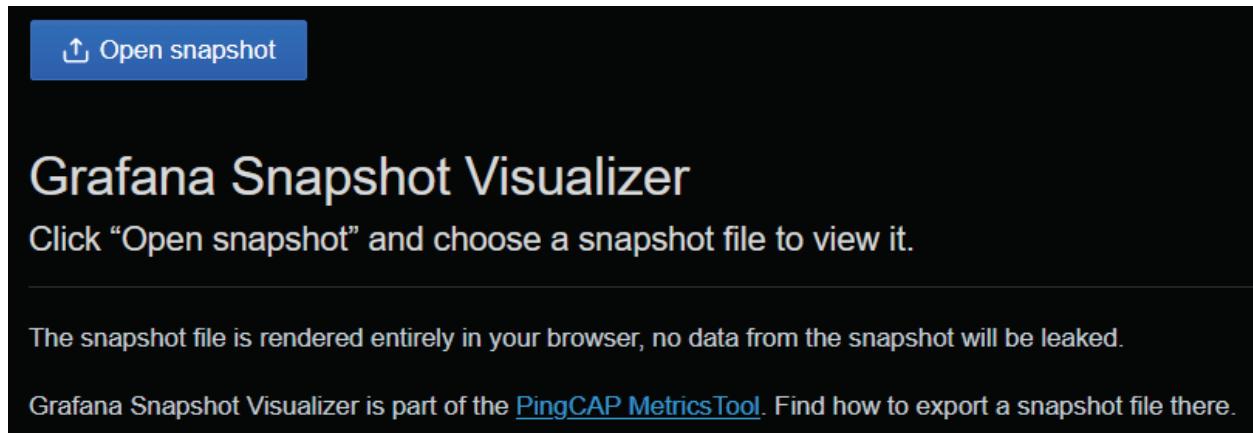


Figure 47: Screenshot of MetricsTool Visualizer

- **Import:** Instructions to import the exported snapshot back into an actual Grafana instance.

#### 6.4.2 FAQs

##### 6.4.2.1 What is the advantage of this tool compared with screenshot or PDF printing?

The snapshot files exported by MetricsTool contain the actual values when they are taken. And the Visualizer allows you to interact with the rendered graphs as if it is a live

Grafana dashboard, supporting operations like toggling series, zooming into a smaller time range, and checking the precise value at a given time. This makes MetricsTool much more powerful than images or PDFs.

#### 6.4.2.2 What are included in the snapshot file?

The snapshot file contains the values of all graphs and panels in the selected time range. It does not save the original metrics from the data sources (and thus you cannot edit the query expression in the Visualizer).

#### 6.4.2.3 Will the Visualizer save the uploaded snapshot files in PingCAP's servers?

No, the Visualizer parses the snapshot files entirely inside your browser. Nothing will be sent to PingCAP. You are free to view snapshot files received from sensitive sources, and no need to worry about these leaking to third parties through the Visualizer.

#### 6.4.2.4 Can it export metrics besides Grafana?

No, we only support Grafana v6.x.x at the moment.

#### 6.4.2.5 Will there be problems to execute the script before all metrics are loaded?

No, the script UI will notify you to wait for all metrics to be loaded. However, you can manually skip waiting and export the snapshot in case of some metrics loading for too long.

#### 6.4.2.6 Can we share a link to a visualized snapshot?

No, but you can share the snapshot file, with instruction on how to use the Visualizer to view it. If you truly need a world-readable URL, you may also try the public [snapshot](#) → [raintank.io](#) service built into Grafana, but make sure all privacy concerns are cleared before doing so.

### 6.5 TiDB Cluster Alert Rules

This document describes the alert rules for different components in a TiDB cluster, including the rule descriptions and solutions of the alert items in TiDB, TiKV, PD, TiFlash, TiDB Binlog, TiCDC, Node\_exporter and Blackbox\_exporter.

According to the severity level, alert rules are divided into three categories (from high to low): emergency-level, critical-level, and warning-level. This division of severity levels applies to all alert items of each component below.

| Severity level                               | Description                                                     |
|----------------------------------------------|-----------------------------------------------------------------|
| Emergency-level                              | The highest severity level at which the service is unavailable. |
| Emergency-level alerts                       | are often caused by a service or node failure.                  |
| Manual intervention is required immediately. |                                                                 |

| Severity level | Description                                                                                                       |
|----------------|-------------------------------------------------------------------------------------------------------------------|
| Critical-level | Decreased service availability. For the critical-level alerts, a close watch on the abnormal metrics is required. |
| Warning-level  | Warning-level alerts are a reminder for an issue or error.                                                        |

## 6.5.1 TiDB alert rules

This section gives the alert rules for the TiDB component.

### 6.5.1.1 Emergency-level alerts

#### 6.5.1.1.1 TiDB\_schema\_error

- Alert rule:

```
increase(tidb_session_schema_lease_error_total{type="outdated"}[15m])>
↪ 0
```

- Description:

The latest schema information is not reloaded in TiDB within one lease. When TiDB fails to continue providing services, an alert is triggered.

- Solution:

It is often caused by an unavailable Region or a TiKV timeout. You need to locate the issue by checking the TiKV monitoring items.

#### 6.5.1.1.2 TiDB\_tikvclient\_region\_err\_total

- Alert rule:

```
increase(tidb_tikvclient_region_err_total[10m])> 6000
```

- Description:

When TiDB accesses TiKV, a Region error occurs. When the error is reported over 6000 times in 10 minutes, an alert is triggered.

- Solution:

View the monitoring status of TiKV.

#### 6.5.1.1.3 TiDB\_domain\_load\_schema\_total

- Alert rule:

```
increase(tidb_domain_load_schema_total{type="failed"}[10m])> 10
```

- Description:

The total number of failures to reload the latest schema information in TiDB. If the reloading failure occurs over 10 times in 10 minutes, an alert is triggered.

- Solution:

Same as [TiDB\\_schema\\_error](#).

#### 6.5.1.1.4 TiDB\_monitor\_keep\_alive

- Alert rule:

```
increase(tidb_monitor_keep_alive_total[10m])< 100
```

- Description:

Indicates whether the TiDB process still exists. If the number of times for `tidb_monitor_keep_alive_total` increases less than 100 in 10 minutes, the TiDB process might already exit and an alert is triggered.

- Solution:

- Check whether the TiDB process is out of memory.
- Check whether the machine has restarted.

### 6.5.1.2 Critical-level alerts

#### 6.5.1.2.1 TiDB\_server\_panic\_total

- Alert rule:

```
increase(tiDB_server_panic_total[10m]) > 0
```

- Description:

The number of panicked TiDB threads. When a panic occurs, an alert is triggered. The thread is often recovered, otherwise, TiDB will frequently restart.

- Solution:

Collect the panic logs to locate the issue.

### 6.5.1.3 Warning-level alerts

#### 6.5.1.3.1 TiDB\_memory\_abnormal

- Alert rule:

```
go_memstats_heap_inuse_bytes{job="tiDB"} > 1e+10
```

- Description:

The monitoring on the TiDB memory usage. If the usage exceeds 10 G, an alert is triggered.

- Solution:

Use the HTTP API to troubleshoot the goroutine leak issue.

#### 6.5.1.3.2 TiDB\_query\_duration

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tiDB_server_handle_query_duration_seconds_bucket
 [1m])) BY (le, instance)) > 1
```

- Description:

The latency of handling a request in TiDB. If the ninety-ninth percentile latency exceeds 1 second, an alert is triggered.

- Solution:

View TiDB logs and search for the SLOW\_QUERY and TIME\_COP\_PROCESS keywords to locate the slow SQL queries.

#### 6.5.1.3.3 TiDB\_server\_event\_error

- Alert rule:

```
increase(tidb_server_event_total{type=~"server_start|server_hang"}[15m
↪]) > 0
```

- Description:

The number of events that happen in the TiDB service. An alert is triggered when the following events happen:

1. start: The TiDB service starts.
2. hang: When a critical-level event (currently there is only one scenario: TiDB cannot write binlog) happens, TiDB enters the `hang` mode and waits to be killed manually.

- Solution:

- Restart TiDB to recover the service.
- Check whether the TiDB Binlog service is normal.

#### 6.5.1.3.4 TiDB\_tikvclient\_backoff\_seconds\_count

- Alert rule:

```
increase(tidb_tikvclient_backoff_seconds_count[10m]) > 10
```

- Description:

The number of retries when TiDB fails to access TiKV. When the retry times is over 10 in 10 minutes, an alert is triggered.

- Solution:

View the monitoring status of TiKV.

#### 6.5.1.3.5 TiDB\_monitor\_time\_jump\_back\_error

- Alert rule:

```
increase(tidb_monitor_time_jump_back_total[10m]) > 0
```

- Description:

When the time of the machine that holds TiDB rewinds, an alert is triggered.

- Solution:

Troubleshoot the NTP configurations.

#### 6.5.1.3.6 TiDB\_ddl\_waiting\_jobs

- Alert rule:

```
sum(tidb_ddl_waiting_jobs)> 5
```

- Description:

When the number of DDL tasks pending for execution in TiDB exceeds 5, an alert is triggered.

- Solution:

Check whether there is any time-consuming `add index` operation that is being executed by running `admin show ddl`.

### 6.5.2 PD alert rules

This section gives the alert rules for the PD component.

#### 6.5.2.1 Emergency-level alerts

##### 6.5.2.1.1 PD\_cluster\_offline\_tikv\_nums

- Alert rule:

```
sum(pd_cluster_status{type="store_down_count"})> 0
```

- Description:

PD has not received a TiKV heartbeat for a long time (the default configuration is 30 minutes).

- Solution:

- Check whether the TiKV process is normal, the network is isolated or the load is too high, and recover the service as much as possible.
- If the TiKV instance cannot be recovered, you can make it offline.

#### 6.5.2.2 Critical-level alerts

#### 6.5.2.2.1 PD\_etcd\_write\_disk\_latency

- Alert rule:

```
histogram_quantile(0.99, sum(rate(etcd_disk_wal_fsync_duration_seconds_bucket
↪ [1m]))by (instance,job,le))> 1
```

- Description:

If the latency of the fsync operation exceeds 1 second, it indicates that etcd writes data to disk at a lower speed than normal. It might lead to PD leader timeout or failure to store TSO on disk in time, which will shut down the service of the entire cluster.

- Solution:

- Find the cause of slow writes. It might be other services that overload the system. You can check whether PD itself occupies a large amount of CPU or I/O resources.
- Try to restart PD or manually transfer leader to another PD to recover the service.
- If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

#### 6.5.2.2.2 PD\_miss\_peer\_region\_count

- Alert rule:

```
sum(pd_regions_status{type="miss_peer_region_count"})> 100
```

- Description:

The number of Region replicas is smaller than the value of `max-replicas`. When a TiKV machine is down and its downtime exceeds `max-down-time`, it usually leads to missing replicas for some Regions during a period of time.

- Solution:

- Find the cause of the issue by checking whether there is any TiKV machine that is down or being made offline.
- Watch the Region health panel and see whether `miss_peer_region_count` is continuously decreasing.

#### 6.5.2.3 Warning-level alerts

#### 6.5.2.3.1 PD\_cluster\_lost\_connect\_tikv\_nums

- Alert rule:

```
sum(pd_cluster_status{type="store_disconnected_count"}) > 0
```

- Description:

PD does not receive a TiKV heartbeat within 20 seconds. Normally a TiKV heartbeat comes in every 10 seconds.

- Solution:

- Check whether the TiKV instance is being restarted.
- Check whether the TiKV process is normal, the network is isolated, and the load is too high, and recover the service as much as possible.
- If you confirm that the TiKV instance cannot be recovered, you can make it offline.
- If you confirm that the TiKV instance can be recovered, but not in the short term, you can consider increasing the value of `max-down-time`. It will prevent the TiKV instance from being considered as irrecoverable and the data from being removed from the TiKV.

#### 6.5.2.3.2 PD\_cluster\_low\_space

- Alert rule:

```
sum(pd_cluster_status{type="store_low_space_count"}) > 0
```

- Description:

Indicates that there is no sufficient space on the TiKV node.

- Solution:

- Check whether the space in the cluster is generally insufficient. If so, increase its capacity.
- Check whether there is any issue with Region balance scheduling. If so, it will lead to uneven data distribution.
- Check whether there is any file that occupies a large amount of disk space, such as the log, snapshot, core dump, etc.
- Lower the Region weight of the node to reduce the data volume.
- When it is not possible to release the space, consider proactively making the node offline. This prevents insufficient disk space that leads to downtime.

#### 6.5.2.3.3 PD\_etcd\_network\_peer\_latency

- Alert rule:

```
histogram_quantile(0.99, sum(rate(etcd_network_peer_round_trip_time_seconds_bucket
↪ [1m]))by (To,instance,job,le))> 1
```

- Description:

The network latency between PD nodes is high. It might lead to the leader timeout and TSO disk storage timeout, which impacts the service of the cluster.

- Solution:

- Check the network and system load status.
- If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

#### 6.5.2.3.4 PD\_tidb\_handle\_requests\_duration

- Alert rule:

```
histogram_quantile(0.99, sum(rate(pd_client_request_handle_requests_duration_seconds
↪ {type="tso"}[1m]))by (instance,job,le))> 0.1
```

- Description:

It takes a longer time for PD to handle the TSO request. It is often caused by a high load.

- Solution:

- Check the load status of the server.
- Use pprof to analyze the CPU profile of PD.
- Manually switch the PD leader.
- If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

#### 6.5.2.3.5 PD\_down\_peer\_region\_nums

- Alert rule:

```
sum(pd_regions_status{type="down_peer_region_count"})> 0
```

- Description:

The number of Regions with an unresponsive peer reported by the Raft leader.

- Solution:

- Check whether there is any TiKV that is down, or that was just restarted, or that is busy.
- Watch the Region health panel and see whether `down_peer_region_count` is continuously decreasing.
- Check the network between TiKV servers.

#### 6.5.2.3.6 PD\_pending\_peer\_region\_count

- Alert rule:

```
sum(pd_regions_status{type="pending_peer_region_count"}) > 100
```

- Description:

There are too many Regions that have lagged Raft logs. It is normal that scheduling leads to a small number of pending peers, but if the number remains high, there might be an issue.

- Solution:

- Watch the Region health panel and see whether `pending_peer_region_count` is continuously decreasing.
- Check the network between TiKV servers, especially whether there is enough bandwidth.

#### 6.5.2.3.7 PD\_leader\_change

- Alert rule:

```
count(changes(pd_tso_events{type="save"}[10m]) > 0) >= 2
```

- Description:

The PD leader is recently switched.

- Solution:

- Exclude the human factors, such as restarting PD, manually transferring leader, adjusting leader priority, etc.
- Check the network and system load status.
- If the problematic PD instance cannot be recovered due to environmental factors, make it offline and replace it.

#### 6.5.2.3.8 TiKV\_space\_used\_more\_than\_80%

- Alert rule:

```
sum(pd_cluster_status{type="storage_size"}) / sum(pd_cluster_status{type
↪ ="storage_capacity"})* 100 > 80
```

- Description:

Over 80% of the cluster space is occupied.

- Solution:

- Check whether it is needed to increase capacity.
- Check whether there is any file that occupies a large amount of disk space, such as the log, snapshot, core dump, etc.

#### 6.5.2.3.9 PD\_system\_time\_slow

- Alert rule:

```
changes(pd_tso_events{type="system_time_slow"}[10m])>= 1
```

- Description:

The system time rewind might happen.

- Solution:

Check whether the system time is configured correctly.

#### 6.5.2.3.10 PD\_no\_store\_for\_making\_replica

- Alert rule:

```
increase(pd_checker_event_count{type="replica_checker", name="no_target_store
↪ "}[1m])> 0
```

- Description:

There is no appropriate store for additional replicas.

- Solution:

- Check whether there is enough space in the store.
- Check whether there is any store for additional replicas according to the label configuration if it is configured.

### 6.5.3 TiKV alert rules

This section gives the alert rules for the TiKV component.

### 6.5.3.1 Emergency-level alerts

#### 6.5.3.1.1 TiKV\_memory\_used\_too\_fast

- Alert rule:

```
process_resident_memory_bytes{job=~"tikv",instance=~".*"} - (process_resident_memory_bytes{job=~"tikv",instance=~".*"} offset 5m) > 5*1024*1024*1024
```

- Description:

Currently, there are no TiKV monitoring items about memory. You can monitor the memory usage of the machines in the cluster by Node\_exporter. The above rule indicates that when the memory usage exceeds 5 GB within 5 minutes (the memory is occupied too fast in TiKV), an alert is triggered.

- Solution:

Adjust the `block-cache-size` value of both `rockdb.defaultcf` and `rocksdb.writecf` .

#### 6.5.3.1.2 TiKV\_GC\_can\_not\_work

- Alert rule:

```
sum(increase(tikv_gcworker_gc_tasks_vec{task="gc"}[1d])) < 1 and sum(increase(tikv_gc_compaction_filter_perform[1d])) < 1
```

- Description:

GC is not performed successfully on a TiKV instance within 24 hours, which indicates that GC is not working properly. If GC does not run in a short term, it will not cause much trouble; but if GC keeps down, more and more versions are retained, which slows down the query.

- Solution:

1. Perform `SELECT VARIABLE_VALUE FROM mysql.tidb WHERE VARIABLE_NAME = "tikv_gc_leader_desc"` to locate the tidb-server corresponding to the GC leader;
2. View the log of the tidb-server, and grep `gc_worker` `tidb.log`;
3. If you find that the GC worker has been resolving locks (the last log is “start resolve locks”) or deleting ranges (the last log is “start delete {number} ranges”) during this time, it means the GC process is running normally. Otherwise, contact [support@pingcap.com](mailto:support@pingcap.com) to resolve this issue.

### 6.5.3.2 Critical-level alerts

### 6.5.3.2.1 TiKV\_server\_report\_failure\_msg\_total

- Alert rule:

```
sum(rate(tikv_server_report_failure_msg_total{type="unreachable"}[10m])
 ↪)BY (store_id)> 10
```

- Description:

Indicates that the remote TiKV cannot be connected.

- Solution:

1. Check whether the network is clear.
2. Check whether the remote TiKV is down.
3. If the remote TiKV is not down, check whether the pressure is too high. Refer to the solution in [TiKV\\_channel\\_full\\_total](#).

### 6.5.3.2.2 TiKV\_channel\_full\_total

- Alert rule:

```
sum(rate(tikv_channel_full_total[10m]))BY (type, instance)> 0
```

- Description:

This issue is often caused by the stuck Raftstore thread and high pressure on TiKV.

- Solution:

1. Watch the Raft Propose monitor, and see whether the alerted TiKV node has a much higher Raft propose than other TiKV nodes. If so, it means that there are one or more hot spots on this TiKV. You need to check whether the hot spot scheduling can work properly.
2. Watch the Raft I/O monitor, and see whether the latency increases. If the latency is high, it means a bottleneck might exist in the disk. One feasible but unsafe solution is setting `sync-log` to `false`.
3. Watch the Raft Process monitor, and see whether the tick duration is high. If so, you need to add `raft-base-tick-interval = "2s"` under the `[raftstore]` configuration.

### 6.5.3.2.3 TiKV\_write\_stall

- Alert rule:

```
delta(tikv_engine_write_stall[10m])> 0
```

- Description:

The write pressure on RocksDB is too high, and a stall occurs.

- Solution:

1. View the disk monitor, and troubleshoot the disk issues;
2. Check whether there is any write hot spot on the TiKV;
3. Set `max-sub-compactions` to a larger value under the `[rocksdb]` and `[raftdb]` configurations.

#### 6.5.3.2.4 TiKV\_raft\_log\_lag

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_raftstore_log_lag_bucket[1m]))by
↪ (le, instance))> 5000
```

- Description:

If this value is relatively large, it means Follower has lagged far behind Leader, and Raft cannot be replicated normally. It is possibly because the TiKV machine where Follower is located is stuck or down.

#### 6.5.3.2.5 TiKV\_async\_request\_snapshot\_duration\_seconds

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_storage_engine_async_request_duration_second
↪ {type="snapshot"}[1m]))by (le, instance, type))> 1
```

- Description:

If this value is relatively large, it means the load pressure on Raftstore is too high, and it might be stuck already.

- Solution:

Refer to the solution in [TiKV\\_channel\\_full\\_total](#).

#### 6.5.3.2.6 TiKV\_async\_request\_write\_duration\_seconds

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_storage_engine_async_request_duration_second
↪ {type="write"}[1m]))by (le, instance, type))> 1
```

- Description:

If this value is relatively large, it means the Raft write takes a long time.

- Solution:

1. Check the pressure on Raftstore. See the solution in [TiKV\\_channel\\_full\\_total](#).
2. Check the pressure on the apply worker thread.

#### 6.5.3.2.7 TiKV\_coprocessor\_request\_wait\_seconds

- Alert rule:

```
histogram_quantile(0.9999, sum(rate(tikv_coprocessor_request_wait_seconds_bucket
↪ [1m]))by (le, instance, req))> 10
```

- Description:

If this value is relatively large, it means the pressure on the Coprocessor worker is high. There might be a slow task that makes the Coprocessor thread stuck.

- Solution:

1. View the slow query log from the TiDB log to see whether the index or full table scan is used in a query, or see whether it is needed to analyze;
2. Check whether there is a hot spot;
3. View the Coprocessor monitor and see whether `total` and `process` in `coprocessor table/index scan` match. If they differ a lot, it indicates too many invalid queries are performed. You can see whether there is `over seek bound`. If so, there are too many versions that GC does not handle in time. Then you need to increase the number of parallel GC threads.

#### 6.5.3.2.8 TiKV\_raftstore\_thread\_cpu\_seconds\_total

- Alert rule:

```
sum(rate(tikv_thread_cpu_seconds_total{name=~"raftstore_.*"}[1m]))by (
↪ instance, name)> 1.6
```

- Description:

The pressure on the Raftstore thread is too high.

- Solution:

Refer to the solution in [TiKV\\_channel\\_full\\_total](#).

#### 6.5.3.2.9 TiKV\_raft\_append\_log\_duration\_secs

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_raftstore_append_log_duration_seconds_bucket
↪ [1m]))by (le, instance))> 1
```

- Description:

Indicates the time cost of appending Raft log. If it is high, it usually means I/O is too busy.

#### 6.5.3.2.10 TiKV\_raft\_apply\_log\_duration\_secs

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_raftstore_apply_log_duration_seconds_bucket
↪ [1m]))by (le, instance))> 1
```

- Description:

Indicates the time cost of applying Raft log. If it is high, it usually means I/O is too busy.

#### 6.5.3.2.11 TiKV\_scheduler\_latch\_wait\_duration\_seconds

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_scheduler_latch_wait_duration_seconds_bucket
↪ [1m]))by (le, instance, type))> 1
```

- Description:

The waiting time for the write operations to obtain the memory lock in Scheduler. If it is high, there might be many write conflicts, or that some operations that lead to conflicts take a long time to finish and block other operations that wait for the same lock.

- Solution:

1. View the scheduler command duration in the Scheduler-All monitor and see which command is most time-consuming;
2. View the scheduler scan details in the Scheduler-All monitor and see whether `total` and `process` match. If they differ a lot, there are many invalid scans. You can also see whether there is `over seek bound`. If there is too much, it indicates GC does not work in time;
3. View the storage async snapshot/write duration in the Storage monitor and see whether the Raft operation is performed in time.

#### 6.5.3.2.12 TiKV\_thread\_apply\_worker\_cpu\_seconds

- Alert rule:

```
sum(rate(tikv_thread_cpu_seconds_total{name="apply_worker"}[1m]))by (
↪ instance)> 1.8
```

- Description:

The pressure on the apply Raft log thread is too high. It is often caused by a burst of writes.

### 6.5.3.3 Warning-level alerts

#### 6.5.3.3.1 TiKV\_leader\_drops

- Alert rule:

```
delta(tikv_pd_heartbeat_tick_total{type="leader"})[30s]) < -10
```

- Description:

It is often caused by a stuck Raftstore thread.

- Solution:

1. Refer to [TiKV\\_channel\\_full\\_total](#).

2. If there is low pressure on TiKV, consider whether the PD scheduling is too frequent. You can view the Operator Create panel on the PD page, and check the types and number of the PD scheduling.

#### 6.5.3.3.2 TiKV\_raft\_process\_ready\_duration\_secs

- Alert rule:

```
histogram_quantile(0.999, sum(rate(tikv_raftstore_raft_process_duration_secs_bucket
↪ {type='ready'})[1m]))by (le, instance, type)) > 2
```

- Description:

Indicates the time cost of handling Raft ready. If this value is large, it is often caused by the stuck appending log task.

#### 6.5.3.3.3 TiKV\_raft\_process\_tick\_duration\_secs

- Alert rule:

```
histogram_quantile(0.999, sum(rate(tikv_raftstore_raft_process_duration_secs_bucket
↪ {type=' tick' })[1m]))by (le, instance, type)) > 2
```

- Description:

Indicates the time cost of handling Raft tick. If this value is large, it is often caused by too many Regions.

- Solution:

1. Consider using a higher-level log such as `warn` or `error`.

2. Add `raft-base-tick-interval = "2s"` under the `[raftstore]` configuration.

#### 6.5.3.3.4 TiKV\_scheduler\_context\_total

- Alert rule:

```
abs(delta(tikv_scheduler_context_total[5m])) > 1000
```

- Description:

The number of write commands that are being executed by Scheduler. If this value is large, it means the task is not finished timely.

- Solution:

Refer to [TiKV\\_scheduler\\_latch\\_wait\\_duration\\_seconds](#).

#### 6.5.3.3.5 TiKV\_scheduler\_command\_duration\_seconds

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_scheduler_command_duration_seconds_bucket
 [1m])) by (le, instance, type) / 1000) > 1
```

- Description:

Indicates the time cost of executing the Scheduler command.

- Solution:

Refer to [TiKV\\_scheduler\\_latch\\_wait\\_duration\\_seconds](#).

#### 6.5.3.3.6 TiKV\_coprocessor\_outdated\_request\_wait\_seconds

- Alert rule:

```
delta(tikv_coprocessor_outdated_request_wait_seconds_count[10m]) > 0
```

- Description:

The waiting time of the expired requests by Coprocessor. If this value is large, it means there is high pressure on Coprocessor.

- Solution:

Refer to [TiKV\\_coprocessor\\_request\\_wait\\_seconds](#).

#### 6.5.3.3.7 TiKV\_coprocessor\_request\_error

- Alert rule:

```
increase(tikv_coprocessor_request_error{reason!="meet_lock"}[10m])> 100
```

- Description:

The request error of Coprocessor.

- Solution:

The reasons for the Coprocessor error can be divided into three types: “lock”, “outdated” and “full”. “outdated” indicates that the request has a timeout. It might be caused by a long queue time or a long time to handle a single request. “full” indicates that the request queue is full. It is possibly because the running request is time-consuming, which sends all new requests in the queue. You need to check whether the time-consuming query’s execution plan is correct.

#### 6.5.3.3.8 TiKV\_coprocessor\_request\_lock\_error

- Alert rule:

```
increase(tikv_coprocessor_request_error{reason="meet_lock"}[10m])>
↪ 10000
```

- Description:

The lock requesting error of Coprocessor.

- Solution:

The reasons for the Coprocessor error can be divided into three types: “lock”, “outdated” and “full”. “lock” indicates that the read data is being written and you need to wait a while and read again (the automatic retry happens inside TiDB). If just a few errors of this kind occur, you can ignore them; but if there are a lot of them, you need to check whether there is a conflict between the write and the query.

#### 6.5.3.3.9 TiKV\_coprocessor\_pending\_request

- Alert rule:

```
delta(tikv_coprocessor_pending_request[10m])> 5000
```

- Description:

The queuing requests of Coprocessor.

- Solution:

Refer to [TiKV\\_coprocessor\\_request\\_wait\\_seconds](#).

#### 6.5.3.3.10 TiKV\_batch\_request\_snapshot\_nums

- Alert rule:

```
sum(rate(tikv_thread_cpu_seconds_total{name=~"cop_.*"}[1m]))by (instance
 ↵)/ (count(tikv_thread_cpu_seconds_total{name=~"cop_.*"})* 0.9)/
 ↵ count(count(tikv_thread_cpu_seconds_total)by (instance))> 0
```

- Description:

The Coprocessor CPU usage of a TiKV machine exceeds 90%.

#### 6.5.3.3.11 TiKV\_pending\_task

- Alert rule:

```
sum(tikv_worker_pending_task_total)BY (instance,name)> 1000
```

- Description:

The number of pending tasks of TiKV.

- Solution:

Check which kind of tasks has a higher value. You can normally find a solution to the Coprocessor and apply worker tasks from other metrics.

#### 6.5.3.3.12 TiKV\_low\_space

- Alert rule:

```
sum(tikv_store_size_bytes{type="available"})by (instance)/ sum(tikv_store_size_byte
 ↵ {type="capacity"})by (instance)< 0.2
```

- Description:

The data volume of TiKV exceeds 80% of the configured node capacity or the disk capacity of the machine.

- Solution:

- Check the balance condition of node space.
- Make a plan to increase the disk capacity or delete some data or increase cluster node depending on different situations.

#### 6.5.3.3.13 TiKV\_approximate\_region\_size

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tikv_raftstore_region_size_bucket[1m
↪]))by (le))> 1073741824
```

- Description:

The maximum Region approximate size that is scanned by the TiKV split checker is continually larger than 1 GB within one minute.

- Solution:

The speed of splitting Regions is slower than the write speed. To alleviate this issue, you'd better update TiDB to a version that supports batch-split ( $\geq 2.1.0\text{-rc1}$ ). If it is not possible to update temporarily, you can use `pd-ctl operator add split-  
↪ region <region_id> --policy=approximate` to manually split Regions.

#### 6.5.4 TiFlash alert rules

For the detailed descriptions of TiFlash alert rules, see [TiFlash Alert Rules](#).

#### 6.5.5 TiDB Binlog alert rules

For the detailed descriptions of TiDB Binlog alert rules, see [TiDB Binlog monitoring document](#).

#### 6.5.6 TiCDC Alert rules

For the detailed descriptions of TiCDC alert rules, see [TiCDC Alert Rules](#).

#### 6.5.7 Node\_exporter host alert rules

This section gives the alert rules for the Node\_exporter host.

##### 6.5.7.1 Emergency-level alerts

###### 6.5.7.1.1 NODE\_disk\_used\_more\_than\_80%

- Alert rule:

```
node_filesystem_avail_bytes{fstype=~"(ext.|xfs)", mountpoint!~"/boot"}
↪ / node_filesystem_size_bytes{fstype=~"(ext.|xfs)", mountpoint!~"/
↪ boot"} * 100 <= 20
```

- Description:

The disk space usage of the machine exceeds 80%.

- Solution:

- Log in to the machine, run the `df -h` command to check the disk space usage.
- Make a plan to increase the disk capacity or delete some data or increase cluster node depending on different situations.

#### 6.5.7.1.2 NODE\_disk\_inode\_more\_than\_80%

- Alert rule:

```
node_filesystem_files_free{fstype=~"(ext.|xfs)"} / node_filesystem_files
↪ {fstype=~"(ext.|xfs)"} * 100 < 20
```

- Description:

The inode usage of the filesystem on the machine exceeds 80%.

- Solution:

- Log in to the machine and run the `df -i` command to view the node usage of the filesystem.
- Make a plan to increase the disk capacity or delete some data or increase cluster node depending on different situations.

#### 6.5.7.1.3 NODE\_disk\_READONLY

- Alert rule:

```
node_filesystem_READONLY{fstype=~"(ext.|xfs)"} == 1
```

- Description:

The filesystem is read-only and data cannot be written in it. It is often caused by disk failure or filesystem corruption.

- Solution:

- Log in to the machine and create a file to test whether it is normal.
- Check whether the disk LED is normal. If not, replace the disk and repair the filesystem of the machine.

#### 6.5.7.2 Critical-level alerts

#### 6.5.7.2.1 NODE\_memory\_used\_more\_than\_80%

- Alert rule:

```
((node_memory_MemTotal_bytes-node_memory_MemFree_bytes-node_memory_Cached_bytes
→)/(node_memory_MemTotal_bytes)*100))>= 80
```

- Description:

The memory usage of the machine exceeds 80%.

- Solution:

- View the Memory panel of the host in the Grafana Node Exporter dashboard, and see whether Used memory is too high and Available memory is too low.
- Log in to the machine and run the `free -m` command to view the memory usage. You can run `top` to check whether there is any abnormal process that has an overly high memory usage.

#### 6.5.7.3 Warning-level alerts

##### 6.5.7.3.1 NODE\_node\_overload

- Alert rule:

```
(node_load5 / count without (cpu, mode)(node_cpu_seconds_total{mode="system"})> 1
```

- Description:

The CPU load on the machine is relatively high.

- Solution:

- View the CPU Usage and Load Average of the host in the Grafana Node Exporter dashboard to check whether they are too high.
- Log in to the machine and run `top` to check the load average and the CPU usage, and see whether there is any abnormal process that has an overly high CPU usage.

##### 6.5.7.3.2 NODE\_cpu\_used\_more\_than\_80%

- Alert rule:

```
avg(irate(node_cpu_seconds_total{mode="idle"}[5m]))by(instance)* 100 <=
→ 20
```

- Description:

The CPU usage of the machine exceeds 80%.

- Solution:

- View the CPU Usage and Load Average of the host on the Grafana Node Exporter dashboard to check whether they are too high.
- Log in to the machine and run `top` to check the Load Average and the CPU Usage, and see whether there is any abnormal process that has an overly high CPU usage.

#### 6.5.7.3.3 NODE\_tcp\_estab\_num\_more\_than\_50000

- Alert rule:

```
node_netstat_Tcp_CurrEstab > 50000
```

- Description:

There are more than 50,000 TCP links in the “establish” status on the machine.

- Solution:

- Log in to the machine and run `ss -s` to check the number of TCP links in the “estab” status in the current system.
- Run `netstat` to check whether there is any abnormal link.

#### 6.5.7.3.4 NODE\_disk\_read\_latency\_more\_than\_32ms

- Alert rule:

```
((rate(node_disk_read_time_seconds_total{device=~".+"}[5m])/ rate(
 node_disk_reads_completed_total{device=~".+"}[5m])) or (irate(node_disk_read_time_
 {device=~".+"}[5m])/ irate(node_disk_reads_completed_total{device
 =~".+"}[5m])))* 1000 > 32
```

- Description:

The read latency of the disk exceeds 32 ms.

- Solution:

- Check the disk status by viewing the Grafana Disk Performance dashboard.
- Check the read latency of the disk by viewing the Disk Latency panel.
- Check the I/O usage by viewing the Disk I/O Utilization panel.

### 6.5.7.3.5 NODE\_disk\_write\_latency\_more\_than\_16ms

- Alert rule:

```
((rate(node_disk_write_time_seconds_total{device=~".+"}[5m])/ rate
↪ (node_disk_writes_completed_total{device=~".+"}[5m]))or (irate(
↪ node_disk_write_time_seconds_total{device=~".+"}[5m])/ irate(node_disk_writes_co
↪ {device=~".+"}[5m])))> 16
```

- Description:

The write latency of the disk exceeds 16ms.

- Solution:

- Check the disk status by viewing the Grafana Disk Performance dashboard.
- Check the write latency of the disk by viewing the Disk Latency panel.
- Check the I/O usage by viewing the Disk I/O Utilization panel.

## 6.5.8 Blackbox\_exporter TCP, ICMP, and HTTP alert rules

This section gives the alert rules for the Blackbox\_exporter TCP, ICMP, and HTTP.

### 6.5.8.1 Emergency-level alerts

#### 6.5.8.1.1 TiDB\_server\_is\_down

- Alert rule:

```
probe_success{group="tidb"} == 0
```

- Description:

Failure to probe the TiDB service port.

- Solution:

- Check whether the machine that provides the TiDB service is down.
- Check whether the TiDB process exists.
- Check whether the network between the monitoring machine and the TiDB machine is normal.

#### 6.5.8.1.2 TiFlash\_server\_is\_down

- Alert rule:

```
probe_success{group="tiflash"} == 0
```

- Description:

Failure to probe the TiFlash service port.

- Solution:

- Check whether the machine that provides the TiFlash service is down.
- Check whether the TiFlash process exists.
- Check whether the network between the monitoring machine and the TiFlash machine is normal.

#### 6.5.8.1.3 Pump\_server\_is\_down

- Alert rule:

```
probe_success{group="pump"} == 0
```

- Description:

Failure to probe the pump service port.

- Solution:

- Check whether the machine that provides the pump service is down.
- Check whether the pump process exists.
- Check whether the network between the monitoring machine and the pump machine is normal.

#### 6.5.8.1.4 Drainer\_server\_is\_down

- Alert rule:

```
probe_success{group="drainer"} == 0
```

- Description:

Failure to probe the Drainer service port.

- Solution:

- Check whether the machine that provides the Drainer service is down.
- Check whether the Drainer process exists.
- Check whether the network between the monitoring machine and the Drainer machine is normal.

#### 6.5.8.1.5 TiKV\_server\_is\_down

- Alert rule:

```
probe_success{group="tikv"} == 0
```

- Description:

Failure to probe the TiKV service port.

- Solution:

- Check whether the machine that provides the TiKV service is down.
- Check whether the TiKV process exists.
- Check whether the network between the monitoring machine and the TiKV machine is normal.

#### 6.5.8.1.6 PD\_server\_is\_down

- Alert rule:

```
probe_success{group="pd"} == 0
```

- Description:

Failure to probe the PD service port.

- Solution:

- Check whether the machine that provides the PD service is down.
- Check whether the PD process exists.
- Check whether the network between the monitoring machine and the PD machine is normal.

#### 6.5.8.1.7 Node\_exporter\_server\_is\_down

- Alert rule:

```
probe_success{group="node_exporter"} == 0
```

- Description:

Failure to probe the Node\_exporter service port.

- Solution:

- Check whether the machine that provides the Node\_exporter service is down.
- Check whether the Node\_exporter process exists.
- Check whether the network between the monitoring machine and the Node\_exporter machine is normal.

#### 6.5.8.1.8 Blackbox\_exporter\_server\_is\_down

- Alert rule:

```
probe_success{group="blackbox_exporter"} == 0
```

- Description:

Failure to probe the Blackbox\_Exporter service port.

- Solution:

- Check whether the machine that provides the Blackbox\_Exporter service is down.
- Check whether the Blackbox\_Exporter process exists.
- Check whether the network between the monitoring machine and the Blackbox\_Exporter machine is normal.

#### 6.5.8.1.9 Grafana\_server\_is\_down

- Alert rule:

```
probe_success{group="grafana"} == 0
```

- Description:

Failure to probe the Grafana service port.

- Solution:

- Check whether the machine that provides the Grafana service is down.
- Check whether the Grafana process exists.
- Check whether the network between the monitoring machine and the Grafana machine is normal.

#### 6.5.8.1.10 Pushgateway\_server\_is\_down

- Alert rule:

```
probe_success{group="pushgateway"} == 0
```

- Description:

Failure to probe the Pushgateway service port.

- Solution:

- Check whether the machine that provides the Pushgateway service is down.
- Check whether the Pushgateway process exists.
- Check whether the network between the monitoring machine and the Pushgateway machine is normal.

#### 6.5.8.1.11 Kafka\_exporter\_is\_down

- Alert rule:

```
probe_success{group="kafka_exporter"} == 0
```

- Description:

Failure to probe the Kafka\_Exporter service port.

- Solution:

- Check whether the machine that provides the Kafka\_Exporter service is down.
- Check whether the Kafka\_Exporter process exists.
- Check whether the network between the monitoring machine and the Kafka\_Exporter machine is normal.

#### 6.5.8.1.12 Pushgateway\_metrics\_interface

- Alert rule:

```
probe_success{job="blackbox_exporter_http"} == 0
```

- Description:

Failure to probe the Pushgateway service http interface.

- Solution:

- Check whether the machine that provides the Pushgateway service is down.
- Check whether the Pushgateway process exists.
- Check whether the network between the monitoring machine and the Pushgateway machine is normal.

### 6.5.8.2 Warning-level alerts

#### 6.5.8.2.1 BLACKER\_ping\_latency\_more\_than\_1s

- Alert rule:

```
max_over_time(probe_duration_seconds{job=~"blackbox_exporter.*_icmp"}[1
↪ m]) > 1
```

- Description:

The ping latency exceeds 1 second.

- Solution:

- View the ping latency between the two nodes on the Grafana Blackbox Exporter dashboard to check whether it is too high.
- Check the tcp panel on the Grafana Node Exporter dashboard to check whether there is any packet loss.

## 6.6 TiFlash Alert Rules

This document introduces the alert rules of the TiFlash cluster.

### 6.6.1 TiFlash\_schema\_error

- Alert rule:

```
increase(tiflash_schema_apply_count{type="failed"}[15m]) > 0
```

- Description:

When the schema apply error occurs, an alert is triggered.

- Solution:

The error might be caused by some wrong logic. Contact [TiFlash R&D](#) for support.

### 6.6.2 TiFlash\_schema\_apply\_duration

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tiflash_schema_apply_duration_seconds_bucket
↪ [1m]))BY (le, instance)) > 20
```

- Description:

When the probability that the apply duration exceeds 20 seconds is over 99%, an alert is triggered.

- Solution:

It might be caused by the internal problems of the TiFlash storage engine. Contact [TiFlash R&D](#) for support.

### 6.6.3 TiFlash\_raft\_read\_index\_duration

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tiflash_raft_read_index_duration_seconds_bucket
↪ [1m]))BY (le, instance)) > 3
```

- Description:

When the probability that the read index duration exceeds 3 seconds is over 99%, an alert is triggered.

**Note:**

`read index` is the kvproto request sent to the TiKV leader. TiKV region retries, busy store, or network problems might lead to long request time of `read index`.

- Solution:

The frequent retries might be caused by frequent splitting or migration of the TiKV cluster. You can check the TiKV cluster status to identify the retry reason.

#### 6.6.4 `TiFlash_raft_wait_index_duration`

- Alert rule:

```
histogram_quantile(0.99, sum(rate(tiflash_raft_wait_index_duration_seconds_bucket
→ [1m]))BY (le, instance))> 2
```

- Description:

When the probability that the waiting time for Region Raft Index in TiFlash exceeds 2 seconds is over 99%, an alert is triggered.

- Solution:

It might be caused by a communication error between TiKV and the proxy. Contact [TiFlash R&D](#) for support.

## 7 Troubleshoot

### 7.1 TiDB Troubleshooting Map

This document summarizes common issues in TiDB and other components. You can use this map to diagnose and solve issues when you encounter related problems.

#### 7.1.1 1. Service Unavailable

##### 7.1.1.1 1.1 The client reports `Region is Unavailable` error

- 1.1.1 The `Region is Unavailable` error is usually because a Region is not available for a period of time. You might encounter `TiKV server is busy`, or the request to TiKV fails due to `not leader` or `epoch not match`, or the request to TiKV time out. In such cases, TiDB performs a `backoff` retry mechanism. When the `backoff` exceeds a threshold (20s by default), the error will be sent to the client. Within the `backoff` threshold, this error is not visible to the client.
- 1.1.2 Multiple TiKV instances are OOM at the same time, which causes no Leader in a Region for a period of time. See [case-991](#) in Chinese.
- 1.1.3 TiKV reports `TiKV server is busy`, and exceeds the `backoff` time. For more details, refer to [4.3. TiKV server is busy](#) is a result of the internal flow control mechanism and should not be counted in the `backoff` time. This issue will be fixed.

- 1.1.4 Multiple TiKV instances failed to start, which causes no Leader in a Region. When multiple TiKV instances are deployed in a physical machine, the failure of the physical machine can cause no Leader in a Region if the label is not properly configured. See [case-228](#) in Chinese.
- 1.1.5 When a Follower apply is lagged in a previous epoch, after the Follower becomes a Leader, it rejects the request with `epoch not match`. See [case-958](#) in Chinese (TiKV needs to optimize its mechanism).

### 7.1.1.2 1.2 PD errors cause service unavailable

Refer to [5 PD issues](#).

## 7.1.2 2. Latency increases significantly

### 7.1.2.1 2.1 Transient increase

- 2.1.1 Wrong TiDB execution plan causes latency increase. Refer to [3.3](#).
- 2.1.2 PD Leader election issue or OOM. Refer to [5.2](#) and [5.3](#).
- 2.1.3 A significant number of Leader drops in some TiKV instances. Refer to [4.4](#).

### 7.1.2.2 2.2 Persistent and significant increase

- 2.2.1 TiKV single thread bottleneck
  - Too many Regions in a TiKV instance causes a single gRPC thread to be the bottleneck (Check the **Grafana -> TiKV-details -> Thread CPU/gRPC CPU Per Thread** metric). In v3.x or later versions, you can enable **Hibernate → Region** to resolve the issue. See [case-612](#) in Chinese.
  - For versions earlier than v3.0, when the raftstore thread or the apply thread becomes the bottleneck (**Grafana -> TiKV-details -> Thread CPU/raft store CPU** and **Async apply CPU** metrics exceed 80%), you can scale out TiKV (v2.x) instances or upgrade to v3.x with multi-threading.
- 2.2.2 CPU load increases.
- 2.2.3 TiKV slow write. Refer to [4.5](#).
- 2.2.4 TiDB wrong execution plan. Refer to [3.3](#).

### 7.1.3 3. TiDB issues

#### 7.1.3.1 3.1 DDL

- 3.1.1 An error `ERROR 1105 (HY000): unsupported modify decimal column ↪ precision` is reported when you modify the length of the `decimal` field. TiDB does not support changing the length of the `decimal` field.
- 3.1.2 TiDB DDL job hangs or executes slowly (use `admin show ddl jobs` to check DDL progress)
  - Cause 1: Network issue with other components (PD/TiKV).
  - Cause 2: Early versions of TiDB (earlier than v3.0.8) have heavy internal load because of a lot of goroutine at high concurrency.
  - Cause 3: In early versions (v2.1.15 & versions < v3.0.0-rc1), PD instances fail to delete TiDB keys, which causes every DDL change to wait for two leases.
  - For other unknown causes, [report a bug](#).
  - Solution:
    - \* For cause 1, check the network connection between TiDB and TiKV/PD.
    - \* For cause 2 and 3, the issues are already fixed in later versions. You can upgrade TiDB to a later version.
    - \* For other causes, you can use the following solution of migrating the DDL owner.
      - \* If you can connect to the TiDB server, execute the owner election command again: `curl -X POST http://[TiDBIP]:10080/ddl/owner/resign`
      - \* If you cannot connect to the TiDB server, use `tidb-ctl` to delete the DDL owner from the etcd of the PD cluster to trigger re-election: `tidb-ctl etcd ↪ delowner [LeaseID] [flags] + ownerKey`
  - DDL owner migration:
    - \* If you can connect to the TiDB server, execute the owner election command again: `curl -X POST http://[TiDBIP]:10080/ddl/owner/resign`
    - \* If you cannot connect to the TiDB server, use `tidb-ctl` to delete the DDL owner from the etcd of the PD cluster to trigger re-election: `tidb-ctl etcd ↪ delowner [LeaseID] [flags] + ownerKey`
- 3.1.3 TiDB reports `information schema is changed` error in log
  - Cause 1: The DML operation touches a table that is under DDL. You can use `admin show ddl job` to check the DDLs that are currently in progress.
  - Cause 2: The current DML operation is executed too long. During the time, many DDL operations are executed, which causes `schema version` changes to be more than 1024. The new version `lock table` might also cause schema version changes.
  - Cause 3: The TiDB instance that is currently executing DML statements cannot load the new `schema information` (maybe caused by network issues with PD or TiKV). During this time, many DDL statements are executed (including `lock ↪ table`), which causes `schema version` changes to be more than 1024.

- Solution: The first two causes do not impact the application, as the related DML operations retry after failure. For cause 3, you need to check the network between TiDB and TiKV/PD.
- Background: The increased number of `schema version` is consistent with the number of `schema state` of each DDL change operation. For example, the `create ↪ table` operation has 1 version change, and the `add column` operation has 4 version changes. Therefore, too many column change operations might cause `schema version` to increase fast. For details, refer to [online schema change](#).
- 3.1.4 TiDB reports `information schema is out of date` in log
  - Cause 1: The TiDB server that is executing the DML statement is stopped by `graceful kill` and prepares to exit. The execution time of the transaction that contains the DML statement exceeds one DDL lease. An error is reported when the transaction is committed.
  - Cause 2: The TiDB server cannot connect to PD or TiKV when it is executing the DML statement, which causes the following problems:
    - \* The TiDB server did not load the new schema within one DDL lease (45s by default); or
    - \* The TiDB server disconnects from PD with the `keep alive` setting.
  - Cause 3: TiKV has high load or network timed out. Check the node loads in **Grafana -> TiDB and TiKV**.
  - Solution:
    - \* For cause 1, retry the DML operation when TiDB is started.
    - \* For cause 2, check the network between the TiDB server and PD/TiKV.
    - \* For cause 3, investigate why TiKV is busy. Refer to [4 TiKV issues](#).

### 7.1.3.2 3.2 OOM issues

- 3.2.1 Symptom
  - Client: The client reports the error `ERROR 2013 (HY000): Lost connection ↪ to MySQL server during query`.
  - Check the log
    - \* Execute `dmesg -T | grep tidb-server`. The result shows the OOM-killer log around the time point when the error occurs.
    - \* Grep the “Welcome to TiDB” log in `tidb.log` around the time point after the error occurs (namely, the time when tidb-server restarts).
    - \* Grep `fatal error: runtime: out of memory or cannot allocate ↪ memory in tidb_stderr.log`.
    - \* In v2.1.8 or earlier versions, you can grep `fatal error: stack overflow` in the `tidb_stderr.log`.

- Monitor: The memory usage of tidb-server instances increases sharply in a short period of time.
- 3.2.2 Locate the SQL statement that causes OOM. (Currently all versions of TiDB cannot locate SQL accurately. You still need to analyze whether OOM is caused by the SQL statement after you locate one.)
  - For versions  $\geq$  v3.0.0, grep “expensive\_query” in `tidb.log`. That log message records SQL queries that timed out or exceed memory quota.
  - For versions  $<$  v3.0.0, grep “memory exceeds quota” in `tidb.log` to locate SQL queries that exceed memory quota.

**Note:**

The default threshold for a single SQL memory usage is 1GB (in bytes, scope:SESSION). You can set this parameter by configuring `tidb_mem_quota_query`. You can also modify the `mem-quota-query` item (in bytes) in the configuration file by hot loading the configuration items.

- 3.2.3 Mitigate OOM issues
  - By enabling SWAP, you can mitigate the OOM issue caused by overuse of memory by large queries. When the memory is insufficient, this method can have impact on the performance of large queries due to the I/O overhead. The degree to which the performance is affected depends on the remaining memory space and the disk I/O speed.
- 3.2.4 Typical reasons for OOM
  - The SQL query has `join`. If you view the SQL statement by using `explain`, you can find that the `join` operation selects the `HashJoin` algorithm and the inner table is large.
  - The data volume of a single `UPDATE/DELETE` query is too large. See [case-882](#) in Chinese.
  - The SQL contains multiple sub-queries connected by `Union`. See [case-1828](#) in Chinese.

### 7.1.3.3 3.3 Wrong execution plan

- 3.3.1 Symptom

- SQL query execution time is much longer compared with that of previous executions, or the execution plan suddenly changes. If the execution plan is logged in the slow log, you can directly compare the execution plans.
- SQL query execution time is much longer compared with that of other databases such as MySQL. Compare the execution plan with other databases to see the differences, such as `Join Order`.
- In slow log, the number of SQL execution time `Scan Keys` is large.
- 3.3.2 Investigate the execution plan
  - `explain analyze {SQL}`. When the execution time is acceptable, compare `count` in the result of `explain analyze` and the number of `row` in `execution info`. If a large difference is found in the `TableScan/IndexScan` row, it is likely that the statistics is incorrect. If a large difference is found in other rows, the problem might not be in the statistics.
  - `select count(*)`. When the execution plan contains a `join` operation, `explain ↗ analyze` might take a long time. You can check whether the problem is in the statistics by executing `select count(*)` for the conditions on `TableScan/ ↗ IndexScan` and comparing the `row count` information in the `explain` result.
- 3.3.3 Mitigation
  - For v3.0 and later versions, use the `SQL Bind` feature to bind the execution plan.
  - Update the statistics. If you are roughly sure that the problem is caused by the statistics, `dump the statistics`. If the cause is outdated statistics, such as the `modify count/row count` in `show stats_meta` is greater than a certain value (for example, 0.3), or the table has an index of time column, you can try recovering by using `analyze table`. If `auto analyze` is configured, check whether the `tidb_auto_analyze_ratio` system variable is too large (for example, greater than 0.3), and whether the current time is between `tidb_auto_analyze_start_time` and `tidb_auto_analyze_end_time`.
  - For other situations, [report a bug](#).

#### 7.1.3.4 3.4 SQL execution error

- 3.4.1 The client reports the `ERROR 1265(01000)Data Truncated` error. This is because the way TiDB internally calculates the precision of `Decimal` type is incompatible with that of MySQL. This issue has been fixed in v3.0.10 ([#14438](#)).
  - Cause:  
In MySQL, if two large-precision `Decimal` are divided and the result exceeds the maximum decimal precision (30), only 30 digits are reserved and no error is reported;

In TiDB, the calculation result is the same as in MySQL, but inside the data structure that represents `Decimal`, a field for decimal precision still retains the actual precision.

Take  $(0.1^{30}) / 10$  as an example. The results in TiDB and MySQL are both 0, because the precision is 30 at most. However, in TiDB, the field for decimal precision is still 31.

After multiple `Decimal` divisions, even though the result is correct, this precision field could grow larger and larger, and eventually exceeds the threshold in TiDB (72), and the `Data Truncated` error is reported.

The multiplication of `Decimal` does not have this issue, because the out-of-bounds is bypassed, and the precision is set to the maximum precision limit.

- Solution: You can bypass this issue by manually adding `Cast(xx as decimal(a → , b))`, in which `a` and `b` are the target precisions.

#### 7.1.4 4. TiKV issues

##### 7.1.4.1 4.1 TiKV panics and fails to start

- 4.1.1 `sync-log = false`. The `unexpected raft log index: last_index X < → applied_index Y` error is returned after the machine is powered off.

This issue is expected. You can restore the Region using `tikv-ctl`.

- 4.1.2 If TiKV is deployed on a virtual machine, when the virtual machine is killed or the physical machine is powered off, the `entries[X, Y] is unavailable from storage` error is reported.

This issue is expected. The `fsync` of virtual machines is not reliable, so you need to restore the Region using `tikv-ctl`.

- 4.1.3 For other unexpected causes, [report a bug](#).

##### 7.1.4.2 4.2 TiKV OOM

- 4.2.1 If the `block-cache` configuration is too large, it might cause OOM.

To verify the cause of the problem, check the `block cache size` of RocksDB by selecting the corresponding instance in the monitor **Grafana -> TiKV-details**.

Meanwhile, check whether the `[storage.block-cache] capacity = # "1GB" →` parameter is set properly. By default, TiKV's `block-cache` is set to 45% of the total memory of the machine. You need to explicitly specify this parameter when you deploy TiKV in the container, because TiKV obtains the memory of the physical machine, which might exceed the memory limit of the container.

- 4.2.2 Coprocessor receives many large queries and returns a large volume of data. gRPC fails to send data as quickly as the coprocessor returns data, which results in OOM.

To verify the cause, you can check whether `response size` exceeds the `network → outbound` traffic by viewing the monitor **Grafana -> TiKV-details -> coprocessor overview**.

- 4.2.3 Other components occupy too much memory.

This issue is unexpected. You can [report a bug](#).

#### 7.1.4.3 4.3 The client reports the `server is busy` error

Check the specific cause for busy by viewing the monitor **Grafana -> TiKV -> errors**. `server is busy` is caused by the flow control mechanism of TiKV, which informs `tidb/ti → -client` that TiKV is currently under too much pressure and will retry later.

- 4.3.1 TiKV RocksDB encounters `write stall`.

A TiKV instance has two RocksDB instances, one in `data/raft` to save the Raft log, another in `data/db` to save the real data. You can check the specific cause for stall by running `grep "Stalling" RocksDB` in the log. The RocksDB log is a file starting with `LOG`, and `LOG` is the current log.

- Too many `level0 sst` causes stall. You can add the `[rocksdb] max-sub- → compactions = 2` (or 3) parameter to speed up `level0 sst` compaction. The compaction task from `level0` to `level1` is divided into several subtasks (the max number of subtasks is the value of `max-sub-compactions`) to be executed concurrently. See [case-815](#) in Chinese.
  - Too many `pending compaction bytes` causes stall. The disk I/O fails to keep up with the write operations in business peaks. You can mitigate this problem by increasing the `soft-pending-compaction-bytes-limit` and `hard-pending- → compaction-bytes-limit` of the corresponding CF.
    - \* The default value of `[rocksdb.defaultcf] soft-pending-compaction → -bytes-limit` is 64GB. If the pending compaction bytes reaches the threshold, RocksDB slows down the write speed. You can set `[rocksdb. → defaultcf] soft-pending-compaction-bytes-limit` to 128GB.
    - \* The default value of `hard-pending-compaction-bytes-limit` is 256GB. If the pending compaction bytes reaches the threshold (this is not likely to happen, because RocksDB slows down the write after the pending compaction bytes reaches `soft-pending-compaction-bytes-limit`), RocksDB stops the write operation. You can set `hard-pending-compaction-bytes-limit` to 512GB.
    - \* If the disk I/O capacity fails to keep up with the write for a long time, it is recommended to scale up your disk. If the disk throughput reaches the

upper limit and causes write stall (for example, the SATA SSD is much lower than NVME SSD), while the CPU resources is sufficient, you may apply a compression algorithm of higher compression ratio. This way, the CPU resources is traded for disk resources, and the pressure on the disk is eased.

- \* If the default CF compaction sees a high pressure, change the [rocksdb.defaultcf] compression-per-level parameter from ["no", "no", "lz4", "lz4", "lz4", "zstd", "zstd"] to ["no", "no", "zstd", "zstd", "zstd", "zstd", "zstd", "zstd"].
- Too many memtables causes stall. This usually occurs when the amount of instant writes is large and the memtables flush to the disk slowly. If the disk write speed cannot be improved, and this issue only occurs during business peaks, you can mitigate it by increasing the `max-write-buffer-number` of the corresponding CF.
  - \* For example, set [rocksdb.defaultcf] `max-write-buffer-number` to 8 (5 by default). Note that this might cause more memory usage in the peak, because more memtables might be in the memory.

- 4.3.2 scheduler too busy

- Serious write conflict. `latch wait duration` is high. You can view `latch → wait duration` in the monitor **Grafana -> TiKV-details -> scheduler prewrite/scheduler commit**. When the write tasks pile up in the scheduler, the pending write tasks exceed the threshold set in [storage] `scheduler-pending → -write-threshold` (100MB). You can verify the cause by viewing the metric corresponding to `MVCC_CONFLICT_COUNTER`.
- Slow write causes write tasks to pile up. The data being written to TiKV exceeds the threshold set by [storage] `scheduler-pending-write-threshold` (100MB). Refer to [4.5](#).
- 4.3.3 raftstore is busy. The processing of messages is slower than the receiving of messages. The short-term `channel full` status does not affect the service, but if the error persists for a long time, it might cause Leader switch.
  - `append log` encounters stall. Refer to [4.3.1](#).
  - `append log duration` is high, which causes slow processing of messages. You can refer to [4.5](#) to analyze why `append log duration` is high.
  - raftstore receives a large batch of messages in an instant (check in the TiKV Raft messages dashboard), and fails to process them. Usually the short-term `channel full` status does not affect the service.
- 4.3.4 TiKV coprocessor is in a queue. The number of piled up tasks exceeds `coprocessor threads * readpool.coprocessor.max-tasks-per-worker-[normal → |low|high]`. Too many large queries leads to the tasks piling up in coprocessor. You need to check whether a execution plan change causes a large number of table scan operations. Refer to [3.3](#).

#### 7.1.4.4 4.4 Some TiKV nodes drop Leader frequently

- 4.4.1 Re-election because TiKV is restarted
  - After TiKV panics, it is pulled up by systemd and runs normally. You can check whether panic has occurred by viewing the TiKV log. Because this issue is unexpected, [report a bug](#) if it happens.
  - TiKV is stopped or killed by a third party and then pulled up by systemd. Check the cause by viewing `dmesg` and the TiKV log.
  - TiKV is OOM, which causes restart. Refer to [4.2](#).
  - TiKV is hung because of dynamically adjusting THP (Transparent Hugepage). See case [case-500](#) in Chinese.
- 4.4.2 TiKV RocksDB encounters write stall and thus results in re-election. You can check if the monitor **Grafana -> TiKV-details -> errors** shows **server is busy**. Refer to [4.3.1](#).
- 4.4.3 Re-election because of network isolation.

#### 7.1.4.5 4.5 TiKV write is slow

- 4.5.1 Check whether the TiKV write is low by viewing the `prewrite/commit/raw-put` duration of TiKV gRPC (only for raw KV clusters). Generally, you can locate the slow phase according to the [performance-map](#). Some common situations are listed as follows.
- 4.5.2 The scheduler CPU is busy (only for transaction kv).

The `scheduler command duration` of prewrite/commit is longer than the sum of `scheduler latch wait duration` and `storage async write duration`. The scheduler worker has a high CPU demand, such as over 80% of `scheduler-worker-pool-size * 100%`, or the CPU resources of the entire machine are relatively limited. If the write workload is large, check if `[storage] scheduler-worker-pool-size` is set too small.

For other situations, [report a bug](#).

- 4.5.3 Append log is slow.

The `Raft IO/append log duration` in TiKV Grafana is high, usually because the disk write operation is slow. You can verify the cause by checking the `WAL Sync Duration max` value of RocksDB - raft.

For other situations, [report a bug](#).

- 4.5.4 The raftstore thread is busy.

The `Raft Propose/propose wait duration` is significantly larger than the append log duration in TiKV Grafana. Take the following methods:

- Check whether the `[raftstore] store-pool-size` configuration value is too small. It is recommended to set the value between 1 and 5 and not too large.
- Check whether the CPU resources on the machine are insufficient.

- 4.5.5 Apply is slow.

The **Raft IO/apply log duration** in TiKV Grafana is high, which usually comes with a high **Raft Propose/apply wait duration**. The possible causes are as follows:

- `[raftstore] apply-pool-size` is too small (it is recommended to set the value between 1 and 5 and not too large), and the **Thread CPU/apply CPU** is large.
- The CPU resources on the machine are insufficient.
- Region write hot spot. A single apply thread has high CPU usage. Currently, we cannot properly address the hot spot problem on a single Region, which is being improved. To view the CPU usage of each thread, modify the Grafana expression and add `by (instance, name)`.
- RocksDB write is slow. **RocksDB kv/max write duration** is high. A single Raft log might contain multiple KVs. When writing into RocksDB, 128 KVs are written into RocksDB in a write batch. Therefore, an apply log might be associated with multiple writes in RocksDB.
- For other situations, [report a bug](#).

- 4.5.6 Raft commit log is slow.

The **Raft IO/commit log duration** in TiKV Grafana is high (this metric is only supported in Grafana after v4.x). Every Region corresponds to an independent Raft group. Raft has a flow control mechanism, similar to the sliding window mechanism of TCP. You can control the size of the sliding window by configuring the `[raftstore] → raft-max-inflight-msgs = 256` parameter. If there is a write hot spot and the `commit log duration` is high, you can adjust the parameter, such as increasing it to 1024.

- 4.5.7 For other situations, refer to the write path on [performance-map](#) and analyze the cause.

## 7.1.5 5. PD issues

### 7.1.5.1 5.1 PD scheduling

- 5.1.1 Merge

- Empty Regions across tables cannot be merged. You need to modify the `[coprocessor] split-region-on-table` parameter in TiKV, which is set to `false` in v4.x by default. See [case-896](#) in Chinese.

- Region merge is slow. You can check whether the merged operator is generated by accessing the monitor dashboard in **Grafana -> PD -> operator**. To accelerate the merge, increase the value of `merge-schedule-limit`.
- 5.1.2 Add replicas or take replicas online/offline
  - The TiKV disk uses 80% of the capacity, and PD does not add replicas. In this situation, the number of miss peers increases, so TiKV needs to be scaled out. See [case-801](#) in Chinese.
  - When a TiKV node is taken offline, some Region cannot be migrated to other nodes. This issue has been fixed in v3.0.4 ([#5526](#)). See [case-870](#) in Chinese.
- 5.1.3 Balance
  - The Leader/Region count is not evenly distributed. See [case-394](#) and [case-759](#) in Chinese. The major cause is that the balance performs scheduling based on the size of Region/Leader, so this might result in the uneven distribution of the count. In TiDB 4.0, the `[leader-schedule-policy]` parameter is introduced, which enables you to set the scheduling policy of Leader to be `count-based` or `size-based`.

### 7.1.5.2 5.2 PD election

- 5.2.1 PD switches Leader.
  - Cause 1: Disk. The disk where the PD node is located has full I/O load. Investigate whether PD is deployed with other components with high I/O demand and the health of the disk. You can verify the cause by viewing the monitor metrics in **Grafana -> disk performance -> latency/load**. You can also use the FIO tool to run a check on the disk if necessary. See [case-292](#) in Chinese.
  - Cause 2: Network. The PD log shows `lost the TCP streaming connection`. You need to check whether there is a problem with the network between PD nodes and verify the cause by viewing `round trip` in the monitor **Grafana -> PD -> etcd**. See [case-177](#) in Chinese.
  - Cause 3: High system load. The log shows `server is likely overloaded`. See [case-214](#) in Chinese.
- 5.2.2 PD cannot elect a Leader or the election is slow.
  - PD cannot elect a Leader: The PD log shows `lease is not expired`. **This issue** has been fixed in v3.0.x and v2.1.19. See [case-875](#) in Chinese.

- The election is slow: The Region loading duration is long. You can check this issue by running `grep "regions cost"` in the PD log. If the result is in seconds, such as `load 460927 regions cost 11.77099s`, it means the Region loading is slow. You can enable the `region storage` feature in v3.0 by setting `use-region ↪ -storage` to `true`, which significantly reduce the Region loading duration. See [case-429](#) in Chinese.
- 5.2.3 PD timed out when TiDB executes SQL statements.
  - PD doesn't have a Leader or switches Leader. Refer to [5.2.1](#) and [5.2.2](#).
  - Network issue. Check whether the network from TiDB to PD Leader is running normally by accessing the monitor **Grafana** -> **blackbox\_exporter** -> **ping latency**.
  - PD panics. [Report a bug](#).
  - PD is OOM. Refer to [5.3](#).
  - If the issue has other causes, get goroutine by running `curl http://127.0.0.1:2379/ ↪ debug/pprof/goroutine?debug=2` and [report a bug](#).
- 5.2.4 Other issues
  - PD reports the FATAL error, and the log shows `range failed to find revision ↪ pair`. This issue has been fixed in v3.0.8 ([#2040](#)). For details, see [case-947](#) in Chinese.
  - For other situations, [report a bug](#).

#### 7.1.5.3 5.3 PD OOM

- 5.3.1 When the `/api/v1/regions` interface is used, too many Regions might cause PD OOM. This issue has been fixed in v3.0.8 ([#1986](#)).
- 5.3.2 PD OOM during the rolling upgrade. The size of gRPC messages is not limited, and the monitor shows that TCP InSegs is relatively large. This issue has been fixed in v3.0.6 ([#1952](#)).

#### 7.1.5.4 5.4 Grafana display

- 5.4.1 The monitor in **Grafana** -> **PD** -> **cluster** -> **role** displays follower. The Grafana expression issue has been fixed in v3.0.8 ([#1065](#)). For details, see [case-1022](#).

## 7.1.6 6. Ecosystem tools

### 7.1.6.1 6.1 TiDB Binlog

- 6.1.1 TiDB Binlog is a tool that collects changes from TiDB and provides backup and replication to downstream TiDB or MySQL platforms. For details, see [TiDB Binlog on GitHub](#).
- 6.1.2 The `UpdateTime` in Pump/Drainer Status is updated normally, and no anomaly shows in the log, but no data is written to the downstream.
  - Binlog is not enabled in the TiDB configuration. Modify the `[binlog]` configuration in TiDB.
- 6.1.3 `sarama` in Drainer reports the `EOF` error.
  - The Kafka client version in Drainer is inconsistent with the version of Kafka. You need to modify the `[syncer.to]` `kafka-version` configuration.
- 6.1.4 Drainer fails to write to Kafka and panics, and Kafka reports the `Message was ↞ too large` error.
  - The binlog data is too large, so the single message written to Kafka is too large. You need to modify the following configuration of Kafka:

`message.max.bytes=1073741824  
replica.fetch.max.bytes=1073741824  
fetch.message.max.bytes=1073741824`

For details, see [case-789](#) in Chinese.

- 6.1.5 Inconsistent data in upstream and downstream
  - Some TiDB nodes do not enable binlog. For v3.0.6 or later versions, you can check the binlog status of all the nodes by accessing the <http://127.0.0.1:10080/info/all> interface. For versions earlier than v3.0.6, you can check the binlog status by viewing the configuration file.
  - Some TiDB nodes go into the `ignore binlog` status. For v3.0.6 or later versions, you can check the binlog status of all the nodes by accessing the <http://127.0.0.1:10080/info/all> interface. For versions earlier than v3.0.6, check the TiDB log to see whether it contains the `ignore binlog` keyword.
  - The value of the timestamp column is inconsistent in upstream and downstream.
    - \* This is caused by different time zones. You need to ensure that Drainer is in the same time zone as the upstream and downstream databases. Drainer obtains its time zone from `/etc/localtime` and does not support the `TZ` environment variable. See [case-826](#) in Chinese.

- \* In TiDB, the default value of timestamp is `null`, but the same default value in MySQL 5.7 (not including MySQL 8) is the current time. Therefore, when the timestamp in upstream TiDB is `null` and the downstream is MySQL 5.7, the data in the timestamp column is inconsistent. You need to run `SET @@global.explicit_defaults_for_timestamp=on;` in the upstream before enabling binlog.
  - For other situations, [report a bug](#).
- 6.1.6 Slow replication
  - The downstream is TiDB/MySQL, and the upstream performs frequent DDL operations. See [case-1023](#) in Chinese.
  - The downstream is TiDB/MySQL, and the table to be replicated has no primary key and no unique index, which causes reduced performance in binlog. It is recommended to add the primary key or unique index.
  - If the downstream outputs to files, check whether the output disk or network disk is slow.
  - For other situations, [report a bug](#).
- 6.1.7 Pump cannot write binlog and reports the `no space left on device` error.
  - The local disk space is insufficient for Pump to write binlog data normally. You need to clean up the disk space and then restart Pump.
- 6.1.8 Pump reports the `fail to notify all living drainer` error when it is started.
  - Cause: When Pump is started, it notifies all Drainer nodes that are in the `online` state. If it fails to notify Drainer, this error log is printed.
  - Solution: Use the `binlogctl` tool to check whether each Drainer node is normal or not. This is to ensure that all Drainer nodes in the `online` state are working normally. If the state of a Drainer node is not consistent with its actual working status, use the `binlogctl` tool to change its state and then restart Pump. See the case [fail-to-notify-all-living-drainer](#).
- 6.1.9 Drainer reports the `gen update sqls failed: table xxx: row data is ↪ corruption []` error.
  - Trigger: The upstream performs DML operations on this table while performing `DROP COLUMN` DDL. This issue has been fixed in v3.0.6. See [case-820](#) in Chinese.
- 6.1.10 Drainer replication is hung. The process remains active but the checkpoint is not updated.
  - This issue has been fixed in v3.0.4. See [case-741](#) in Chinese.

- 6.1.11 Any component panics.
  - Report a bug.

#### 7.1.6.2 6.2 Data Migration

- 6.2.1 TiDB Data Migration (DM) is a migration tool that supports data migration from MySQL/MariaDB into TiDB. For details, see [DM on GitHub](#).
- 6.2.2 `Access denied for user 'root'@'172.31.43.27'` (using password: YES) shows when you run `query status` or check the log.
  - The database related passwords in all the DM configuration files should be encrypted by `dmctl`. If a database password is empty, it is unnecessary to encrypt the password. Cleartext passwords can be used since v1.0.6.
  - During DM operation, the user of the upstream and downstream databases must have the corresponding read and write privileges. Data Migration also [prechecks the corresponding privileges](#) automatically while starting the data replication task.
  - To deploy different versions of DM-worker/DM-master/dmctl in a DM cluster, see the [case study on AskTUG](#) in Chinese.
- 6.2.3 A replication task is interrupted with the `driver: bad connection` error returned.
  - The `driver: bad connection` error indicates that an anomaly has occurred in the connection between DM and the downstream TiDB database (such as network failure, TiDB restart and so on), and that the data of the current request has not yet been sent to TiDB.
    - \* For versions earlier than DM 1.0.0 GA, stop the task by running `stop-task` and then restart the task by running `start-task`.
    - \* For DM 1.0.0 GA or later versions, an automatic retry mechanism for this type of error is added. See [#265](#).
- 6.2.4 A replication task is interrupted with the `invalid connection` error.
  - The `invalid connection` error indicates that an anomaly has occurred in the connection between DM and the downstream TiDB database (such as network failure, TiDB restart, TiKV busy and so on), and that a part of the data for the current request has been sent to TiDB. Because DM has the feature of concurrently replicating data to the downstream in replication tasks, several errors might occur when a task is interrupted. You can check these errors by running `query-status` or `query-error`.
    - \* If only the `invalid connection` error occurs during the incremental replication process, DM retries the task automatically.

- \* If DM does not retry or fails to retry automatically because of version problems (automatic retry is introduced in v1.0.0-rc.1), use `stop-task` to stop the task and then use `start-task` to restart the task.
- 6.2.5 The relay unit reports the error `event from * in * diff from passed-in ↳ event *`, or a replication task is interrupted with an error that fails to get or parse binlog, such as `get binlog error ERROR 1236 (HY000) and binlog checksum ↳ mismatch, data may be corrupted returned`
  - During the process that DM pulls relay log or the incremental replication, this two errors might occur if the size of the upstream binlog file exceeds 4 GB.
  - Cause: When writing relay logs, DM needs to perform event verification based on binlog positions and the binlog file size, and store the replicated binlog positions as checkpoints. However, the official MySQL uses uint32 to store binlog positions, which means the binlog position for a binlog file over 4 GB overflows, and then the errors above occur.
  - Solution:
    - \* For relay processing units, [manually recover replication](#).
    - \* For binlog replication processing units, [manually recover replication](#).
- 6.2.6 The DM replication is interrupted, and the log returns `ERROR 1236 (HY000)`
  - ↳ The slave is connecting using `CHANGE MASTER TO MASTER_AUTO_POSITION`
  - ↳ `= 1`, but the master has purged binary logs containing GTIDs that the slave requires.
  - Check whether the master binlog is purged.
  - Check the position information recorded in `relay.meta`.
    - \* `relay.meta` has recorded the empty GTID information. DM-worker saves the GTID information in memory to `relay.meta` when it exits or in every 30s. When DM-worker does not obtain the upstream GTID information, it saves the empty GTID information to `relay.meta`. See [case-772](#) in Chinese.
    - \* The binlog event recorded in `relay.meta` triggers the incomplete recover process and records the wrong GTID information. This issue is fixed in v1.0.2, and might occur in earlier versions.
- 6.2.7 The DM replication process returns an error `Error 1366: incorrect utf8 ↳ value eda0bdedb29d(\ufffd\ufffd\ufffd\ufffd\ufffd\ufffd\ufffd)`.
  - This value cannot be successfully written into MySQL 8.0 or TiDB, but can be written into MySQL 5.7. You can skip the data format check by enabling the `tidb_skip_utf8_check` parameter.

### 7.1.6.3 6.3 TiDB Lightning

- 6.3.1 TiDB Lightning is a tool for fast full import of large amounts of data into a TiDB cluster. See [TiDB Lightning on GitHub](#).
- 6.3.2 Import speed is too slow.
  - `region-concurrency` is set too high, which causes thread contention and reduces performance. Three ways to troubleshoot:
    - \* The setting can be found from the start of the log by searching `region->concurrency`.
    - \* If TiDB Lightning shares a server with other services (for example, Importer), you must manually set `region-concurrency` to 75% of the total number of CPU cores on that server.
    - \* If there is a quota on CPU (for example, limited by Kubernetes settings), TiDB Lightning might not be able to read this out. In this case, `region->concurrency` must also be manually reduced.
  - Every additional index introduces a new KV pair for each row. If there are N indices, the actual size to be imported would be approximately (N+1) times the size of the [Mydumper](#) output. If the indices are negligible, you may first remove them from the schema, and add them back via `CREATE INDEX` after the import is complete.
  - The version of TiDB Lightning is old. Try the latest version, which might improve the import speed.
- 6.3.3 `checksum failed: checksum mismatched remote vs local`.
  - Cause 1: The table might already have data. These old data can affect the final checksum.
  - Cause 2: If the checksum of the target database is 0, which means nothing is imported, it is possible that the cluster is too hot and fails to take in any data.
  - Cause 3: If the data source is generated by the machine and not backed up by [Mydumper](#), ensure it respects the constraints of the table. For example:
    - \* `AUTO_INCREMENT` columns need to be positive, and do not contain the value “0”.
    - \* `UNIQUE` and `PRIMARY KEY`s must not have duplicate entries.
  - Solution: See [Troubleshooting Solution](#).
- 6.3.4 `Checkpoint for ... has invalid status:(error code)`
  - Cause: Checkpoint is enabled, and Lightning/Importer has previously abnormally exited. To prevent accidental data corruption, TiDB Lightning will not start until the error is addressed. The error code is an integer less than 25, with possible

values as 0, 3, 6, 9, 12, 14, 15, 17, 18, 20 and 21. The integer indicates the step where the unexpected exit occurs in the import process. The larger the integer is, the later the exit occurs.

- Solution: See [Troubleshooting Solution](#).
- 6.3.5 ResourceTemporarilyUnavailable("Too many open engines ...: 8")
  - Cause: The number of concurrent engine files exceeds the limit specified by tikv-importer. This could be caused by misconfiguration. In addition, even when the configuration is correct, if tidb-lightning has exited abnormally before, an engine file might be left at a dangling open state, which could cause this error as well.
  - Solution: See [Troubleshooting Solution](#).
- 6.3.6 cannot guess encoding for input file, please convert to UTF-8
  - ↪ manually
  - Cause: TiDB Lightning only supports the UTF-8 and GB-18030 encodings. This error means the file is not in any of these encodings. It is also possible that the file has mixed encoding, such as containing a string in UTF-8 and another string in GB-18030, due to historical ALTER TABLE executions.
  - Solution: See [Troubleshooting Solution](#).
- 6.3.7 [sql2kv] sql encode error = [types:1292]invalid time format:
  - ↪ '{1970 1 1 0 45 0 0}'
  - Cause: A timestamp type entry has a time value that does not exist. This is either because of DST changes or because the time value has exceeded the supported range (from Jan 1, 1970 to Jan 19, 2038).
  - Solution: See [Troubleshooting Solution](#).

## 7.1.7 7. Common log analysis

### 7.1.7.1 7.1 TiDB

- 7.1.1 GC life time is shorter than transaction duration.

The transaction duration exceeds the GC lifetime (10 minutes by default).

You can increase the GC lifetime by modifying the `tidb_gc_life_time` system variable. Generally, it is not recommended to modify this parameter, because changing it might cause many old versions to pile up if this transaction has a large number of UPDATE and DELETE statements.

- 7.1.2 `txn` takes too much time.

This error is returned when you commit a transaction that has not been committed for a long time (over 590 seconds).

If your application needs to execute a transaction of such a long time, you can increase the `[tikv-client] max-txn-time-use = 590` parameter and the GC lifetime to avoid this issue. It is recommended to check whether your application needs such a long transaction time.

- 7.1.3 `coprocessor.go` reports `request outdated`.

This error is returned when the coprocessor request sent to TiKV waits in a queue at TiKV for over 60 seconds.

You need to investigate why the TiKV coprocessor is in a long queue.

- 7.1.4 `region_cache.go` reports a large number of `switch region peer to next` due to send request fail, and the error message is context deadline exceeded.

The request for TiKV timed out and triggers the region cache to switch the request to other nodes. You can continue to run the `grep "<addr> cancelled` command on the `addr` field in the log and take the following steps according to the `grep` results:

- `send request is cancelled`: The request timed out during the sending phase. You can investigate the monitoring **Grafana -> TiDB -> Batch Client/Pending Request Count** by TiKV and see whether the Pending Request Count is greater than 128:

- \* If the value is greater than 128, the sending goes beyond the processing capacity of KV, so the sending piles up.
- \* If the value is not greater than 128, check the log to see if the report is caused by the operation and maintenance changes of the corresponding KV; otherwise, this error is unexpected, and you need to [report a bug](#).

- `wait response is cancelled`: The request timed out after it is sent to TiKV. You need to check the response time of the corresponding TiKV address and the Region logs in PD and KV at that time.

- 7.1.5 `distsql.go` reports `inconsistent index`.

The data index seems to be inconsistent. Run the `admin check table <TableName>` command on the table where the reported index is. If the check fails, disable garbage collection by running the following command, and [report a bug](#):

```
SET GLOBAL tidb_gc_enable = 0;
```

## 7.1.7.2 7.2 TiKV

- **7.2.1 key is locked.**

The read and write have conflict. The read request encounters data that has not been committed and needs to wait until the data is committed.

A small number of this error has no impact on the business, but a large number of this error indicates that the read-write conflict is severe in your business.

- **7.2.2 write conflict.**

This is the write-write conflict in optimistic transactions. If multiple transactions modify the same key, only one transaction succeed and other transactions automatically obtain the timestamp again and retry the operation, with no impact on the business.

If the conflict is severe, it might cause transaction failure after multiple retries. In this case, it is recommended to use the pessimistic lock.

- **7.2.3 TxnLockNotFound.**

This transaction commit is too slow, which is rolled back by other transactions after TTL (3 seconds for a small transaction by default). This transaction will automatically retry, so the business is usually not affected.

- **7.2.4 PessimisticLockNotFound.**

Similar to `TxnLockNotFound`. The pessimistic transaction commit is too slow and thus rolled back by other transactions.

- **7.2.5 stale\_epoch.**

The request epoch is outdated, so TiDB re-sends the request after updating the routing. The business is not affected. Epoch changes when Region has a split/merge operation or a replica is migrated.

- **7.2.6 peer is not leader.**

The request is sent to a replica that is not Leader. If the error response indicates which replica is the latest Leader, TiDB updates the local routing according the error and sends a new request to the latest Leader. Usually, the business is not affected.

In v3.0 and later versions, TiDB tries other peers if the request to the previous Leader fails, which might lead to frequent `peer is not leader` in TiKV log. You can check the `switch region peer to next due to send request fail` log of the corresponding Region in TiDB to determine the root cause of the sending failure. For details, refer to [7.1.4](#).

This error might also be returned if a Region has no Leader due to other reasons. For details, see [4.4](#).

## 7.2 Identify Slow Queries

To help users identify slow queries, analyze and improve the performance of SQL execution, TiDB outputs the statements whose execution time exceeds **slow-threshold** (The default value is 300 milliseconds) to **slow-query-file** (The default value is “tidb-slow.log”).

TiDB enables the slow query log by default. You can enable or disable the feature by modifying the configuration **enable-slow-log**.

### 7.2.1 Usage example

```
Time: 2019-08-14T09:26:59.487776265+08:00
Txn_start_ts: 410450924122144769
User@Host: root[root] @ localhost [127.0.0.1]
Conn_ID: 3086
Exec_retry_time: 5.1 Exec_retry_count: 3
Query_time: 1.527627037
Parse_time: 0.000054933
Compile_time: 0.000129729
Rewrite_time: 0.000000003 Preproc_subqueries: 2 Preproc_subqueries_time:
 ↳ 0.000000002
Process_time: 0.07 Request_count: 1 Total_keys: 131073 Process_keys:
 ↳ 131072 Prewrite_time: 0.335415029 Commit_time: 0.032175429
 ↳ Get_commit_ts_time: 0.000177098 Local_latch_wait_time: 0.106869448
 ↳ Write_keys: 131072 Write_size: 3538944 Prewrite_region: 1
DB: test
Is_internal: false
Digest: 50a2e32d2abbd6c1764b1b7f2058d428ef2712b029282b776beb9506a365c0f1
Stats: t:pseudo
Num_cop_tasks: 1
Cop_proc_avg: 0.07 Cop_proc_p90: 0.07 Cop_proc_max: 0.07 Cop_proc_addr:
 ↳ 172.16.5.87:20171
Cop_wait_avg: 0 Cop_wait_p90: 0 Cop_wait_max: 0 Cop_wait_addr:
 ↳ 172.16.5.87:20171
Cop_backoff_regionMiss_total_times: 200 Cop_backoff_regionMiss_total_time
 ↳ : 0.2 Cop_backoff_regionMiss_max_time: 0.2
 ↳ Cop_backoff_regionMiss_max_addr: 127.0.0.1
 ↳ Cop_backoff_regionMiss_avg_time: 0.2 Cop_backoff_regionMiss_p90_time:
 ↳ 0.2
Cop_backoff_rpcPD_total_times: 200 Cop_backoff_rpcPD_total_time: 0.2
 ↳ Cop_backoff_rpcPD_max_time: 0.2 Cop_backoff_rpcPD_max_addr: 127.0.0.1
 ↳ Cop_backoff_rpcPD_avg_time: 0.2 Cop_backoff_rpcPD_p90_time: 0.2
Cop_backoff_rpcTiKV_total_times: 200 Cop_backoff_rpcTiKV_total_time: 0.2
 ↳ Cop_backoff_rpcTiKV_max_time: 0.2 Cop_backoff_rpcTiKV_max_addr:
 ↳ 127.0.0.1 Cop_backoff_rpcTiKV_avg_time: 0.2
```

```

 ↵ Cop_backoff_rpcTiKV_p90_time: 0.2
Mem_max: 525211
Disk_max: 65536
Prepared: false
Plan_from_cache: false
Succ: true
Plan: tidb_decode_plan(
 ↵ ZJAwCTMyXzcJMAkyMALkYXRhOlRhYmxlU2Nhb182CjEJMTBfNgkxAROAdAEY1Dp0LCByYW5nZTpbLWluZi
 ↵ ==')
use test;
insert into t select * from t;

```

## 7.2.2 Fields description

**Note:**

The unit of all the following time fields in the slow query log is “second”.

Slow query basics:

- **Time**: The print time of log.
- **Query\_time**: The execution time of a statement.
- **Parse\_time**: The parsing time for the statement.
- **Compile\_time**: The duration of the query optimization.
- **Query**: A SQL statement. **Query** is not printed in the slow log, but the corresponding field is called **Query** after the slow log is mapped to the memory table.
- **Digest**: The fingerprint of the SQL statement.
- **Txn\_start\_ts**: The start timestamp and the unique ID of a transaction. You can use this value to search for the transaction-related logs.
- **Is\_internal**: Whether a SQL statement is TiDB internal. **true** indicates that a SQL statement is executed internally in TiDB and **false** indicates that a SQL statement is executed by the user.
- **Index\_ids**: The IDs of the indexes involved in a statement.
- **Succ**: Whether a statement is executed successfully.
- **Backoff\_time**: The waiting time before retry when a statement encounters errors that require a retry. The common errors as such include: **lock occurs**, **Region split**, and **tikv server is busy**.
- **Plan**: The execution plan of the statement. Use the `select tidb_decode_plan('xxx ↵ ...')` statement to parse the specific execution plan.
- **Prepared**: Whether this statement is a **Prepare** or **Execute** request or not.

- **Plan\_from\_cache**: Whether this statement hits the execution plan cache.
- **Rewrite\_time**: The time consumed for rewriting the query of this statement.
- **Preproc\_subqueries**: The number of subqueries (in the statement) that are executed in advance. For example, the `where id in (select if from t)` subquery might be executed in advance.
- **Preproc\_subqueries\_time**: The time consumed for executing the subquery of this statement in advance.
- **Exec\_retry\_count**: The retry times of this statement. This field is usually for pessimistic transactions in which the statement is retried when the lock is failed.
- **Exec\_retry\_time**: The execution retry duration of this statement. For example, if a statement has been executed three times in total (failed for the first two times), `Exec_retry_time` means the total duration of the first two executions. The duration of the last execution is `Query_time` minus `Exec_retry_time`.

The following fields are related to transaction execution:

- **Prewrite\_time**: The duration of the first phase (prewrite) of the two-phase transaction commit.
- **Commit\_time**: The duration of the second phase (commit) of the two-phase transaction commit.
- **Get\_commit\_ts\_time**: The time spent on getting `commit_ts` during the second phase (commit) of the two-phase transaction commit.
- **Local\_latch\_wait\_time**: The time that TiDB spends on waiting for the lock before the second phase (commit) of the two-phase transaction commit.
- **Write\_keys**: The count of keys that the transaction writes to the Write CF in TiKV.
- **Write\_size**: The total size of the keys or values to be written when the transaction commits.
- **Prewrite\_region**: The number of TiKV Regions involved in the first phase (prewrite) of the two-phase transaction commit. Each Region triggers a remote procedure call.

Memory usage fields:

- **Mem\_max**: The maximum memory space used during the execution period of a SQL statement (the unit is byte).

Hard disk fields:

- **Disk\_max**: The maximum disk space used during the execution period of a SQL statement (the unit is byte).

User fields:

- **User**: The name of the user who executes this statement.

- `Conn_ID`: The Connection ID (session ID). For example, you can use the keyword `con:3` to search for the log whose session ID is 3.
- `DB`: The current database.

TiKV Coprocessor Task fields:

- `Request_count`: The number of Coprocessor requests that a statement sends.
- `Total_keys`: The number of keys that Coprocessor has scanned.
- `Process_time`: The total processing time of a SQL statement in TiKV. Because data is sent to TiKV concurrently, this value might exceed `Query_time`.
- `Wait_time`: The total waiting time of a statement in TiKV. Because the Coprocessor of TiKV runs a limited number of threads, requests might queue up when all threads of Coprocessor are working. When a request in the queue takes a long time to process, the waiting time of the subsequent requests increases.
- `Process_keys`: The number of keys that Coprocessor has processed. Compared with `total_keys`, `processed_keys` does not include the old versions of MVCC. A great difference between `processed_keys` and `total_keys` indicates that many old versions exist.
- `Cop_proc_avg`: The average execution time of cop-tasks.
- `Cop_proc_p90`: The P90 execution time of cop-tasks.
- `Cop_proc_max`: The maximum execution time of cop-tasks.
- `Cop_proc_addr`: The address of the cop-task with the longest execution time.
- `Cop_wait_avg`: The average waiting time of cop-tasks.
- `Cop_wait_p90`: The P90 waiting time of cop-tasks.
- `Cop_wait_max`: The maximum waiting time of cop-tasks.
- `Cop_wait_addr`: The address of the cop-task whose waiting time is the longest.
- `Cop_backoff_{backoff-type}_total_times`: The total times of backoff caused by an error.
- `Cop_backoff_{backoff-type}_total_time`: The total time of backoff caused by an error.
- `Cop_backoff_{backoff-type}_max_time`: The longest time of backoff caused by an error.
- `Cop_backoff_{backoff-type}_max_addr`: The address of the cop-task that has the longest backoff time caused by an error.
- `Cop_backoff_{backoff-type}_avg_time`: The average time of backoff caused by an error.
- `Cop_backoff_{backoff-type}_p90_time`: The P90 percentile backoff time caused by an error.

### 7.2.3 Related system variables

- `tidb_slow_log_threshold`: Sets the threshold for the slow log. The SQL statement whose execution time exceeds this threshold is recorded in the slow log. The default value is 300 (ms).

- `tidb_query_log_max_len`: Sets the maximum length of the SQL statement recorded in the slow log. The default value is 4096 (byte).
- `tidb_redact_log`: Determines whether to desensitize user data using ? in the SQL statement recorded in the slow log. The default value is 0, which means to disable the feature.
- `tidb_enable_collect_execution_info`: Determines whether to record the physical execution information of each operator in the execution plan. The default value is 1. This feature impacts the performance by approximately 3%. After enabling this feature, you can view the Plan information as follows:

```
> select tidb_decode_plan(
 ↪ jAOIMAk1XzE3CTAJMQLmdW5jczpj3VudChDb2x1bW4jNyktPkMJC/
 ↪ BMNQkxCXRpbWU6MTAuOTMxNTA1bXMsIGxvb3Bz0jIJMzcyIEJ5dGVzCU4vQQoxCTMyXzE4CTAJMQL
 ↪ ');
+-
|-----+
| |
| | tidb_decode_plan(
| | ↪ jAOIMAk1XzE3CTAJMQLmdW5jczpj3VudChDb2x1bW4jNyktPkMJC/
| | ↪ BMNQkxCXRpbWU6MTAuOTMxNTA1bXMsIGxvb3Bz0jIJMzcyIEJ5dGVzCU4vQQoxCTMyXzE4CTAJMQL
| | |
| +-----+
| | |
| | | id task estRows operator info
| | | actRows execution info
| | | memory
| | | disk
| | | StreamAgg_17 root 1 funcs:count(Column
| | | #7)->Column#5 1 time:10.931505ms,
| | | loops:2 372 Bytes N
| | | /A
| | | - IndexReader_18 root 1 index:StreamAgg_9
| | | loops:2, rpc num: 1, rpc time:10.884355ms, proc keys:25007 206
| | | Bytes N/A
| | | - StreamAgg_9 cop 1 funcs:count(1)->
| | | Column#7 1 time:11ms, loops
| | | :25
| | | N/A
| | | - IndexScan_16 cop 31281.857819905217 table:t, index:idx(
| | | a), range:[-inf,50000), keep order:false 25007 time:11ms, loops
| | | :25
| | | /A
| +-----+
```



If you are conducting a performance test, you can disable the feature of automatically collecting the execution information of operators:

```
set @@tidb_enable_collect_execution_info=0;
```

The returned result of the `Plan` field has roughly the same format with that of `EXPLAIN` or `EXPLAIN ANALYZE`. For more details of the execution plan, see [EXPLAIN](#) or [EXPLAIN ANALYZE](#).

For more information, see [TiDB specific variables and syntax](#).

#### 7.2.4 Memory mapping in slow log

You can query the content of the slow query log by querying the `INFORMATION_SCHEMA.SLOW_QUERY` table. Each column name in the table corresponds to one field name in the slow log. For table structure, see the introduction to the `SLOW_QUERY` table in [Information Schema](#).

**Note:**

Every time you query the `SLOW_QUERY` table, TiDB reads and parses the current slow query log.

For TiDB 4.0, `SLOW_QUERY` supports querying the slow log of any period of time, including the rotated slow log file. You need to specify the `TIME` range to locate the slow log files that need to be parsed. If you don't specify the `TIME` range, TiDB only parses the current slow log file. For example:

- If you don't specify the time range, TiDB only parses the slow query data that TiDB is writing to the slow log file:

```
select count(*),
 min(time),
 max(time)
from slow_query;
```

| count(*) | min(time)                  | max(time)                  |
|----------|----------------------------|----------------------------|
| 122492   | 2020-03-11 23:35:20.908574 | 2020-03-25 19:16:38.229035 |

- If you specify the time range, for example, from 2020-03-10 00:00:00 to 2020-03-11 00:00:00, TiDB first locates the slow log files of the specified time range, and then parses the slow query information:

```
select count(*),
 min(time),
 max(time)
 from slow_query
 where time > '2020-03-10 00:00:00'
 and time < '2020-03-11 00:00:00';
```

| count(*) | min(time)                  | max(time)                  |
|----------|----------------------------|----------------------------|
| 2618049  | 2020-03-10 00:00:00.427138 | 2020-03-10 23:00:22.716728 |

#### Note:

If the slow log files of the specified time range are removed, or there is no slow query, the query returns NULL.

TiDB 4.0 adds the `CLUSTER_SLOW_QUERY` system table to query the slow query information of all TiDB nodes. The table schema of the `CLUSTER_SLOW_QUERY` table differs from that of the `SLOW_QUERY` table in that an `INSTANCE` column is added to `CLUSTER_SLOW_QUERY`. The `INSTANCE` column represents the TiDB node address of the row information on the slow query. You can use `CLUSTER_SLOW_QUERY` the way you do with `SLOW_QUERY`.

When you query the `CLUSTER_SLOW_QUERY` table, TiDB pushes the computation and the judgment down to other nodes, instead of retrieving all slow query information from other nodes and executing the operations on one TiDB node.

### 7.2.5 `SLOW_QUERY / CLUSTER_SLOW_QUERY` usage examples

#### 7.2.5.1 Top-N slow queries

Query the Top 2 slow queries of users. `Is_internal=false` means excluding slow queries inside TiDB and only querying slow queries of users.

```
select query_time, query
 from information_schema.slow_query
 where is_internal = false
 order by query_time desc
 limit 2;
```

Output example:

| query_time  | query                                                             |
|-------------|-------------------------------------------------------------------|
| 12.77583857 | select * from t_slim, t_wide where t_slim.c0=t_wide.c0;           |
| 0.734982725 | select t0.c0, t1.c1 from t_slim t0, t_wide t1 where t0.c0= t1.c0; |

#### 7.2.5.2 Query the Top-N slow queries of the `test` user

In the following example, the slow queries executed by the `test` user are queried, and the first two results are displayed in reverse order of execution time.

```
select query_time, query, user
from information_schema.slow_query
where is_internal = false
 and user = "test"
order by query_time desc
limit 2;
```

Output example:

| Query_time  | query                                                                   |
|-------------|-------------------------------------------------------------------------|
| user        |                                                                         |
| 0.676408014 | select t0.c0, t1.c1 from t_slim t0, t_wide t1 where t0.c0=t1.c1;   test |

#### 7.2.5.3 Query similar slow queries with the same SQL fingerprints

After querying the Top-N SQL statements, continue to query similar slow queries using the same fingerprints.

1. Acquire Top-N slow queries and the corresponding SQL fingerprints.

```
select query_time, query, digest
from information_schema.slow_query
where is_internal = false
order by query_time desc
limit 1;
```

Output example:

| query_time  | query                       | digest                                                               |
|-------------|-----------------------------|----------------------------------------------------------------------|
| 0.302558006 | select * from t1 where a=1; | 4751<br>cb6008fda383e22dacb601fde85425dc8f8cf669338d55d944bafb46a6fa |

2. Query similar slow queries with the fingerprints.

```
select query, query_time
from information_schema.slow_query
where digest = "4751"
→ cb6008fda383e22dacb601fde85425dc8f8cf669338d55d944bafb46a6fa";
```

Output example:

| query                       | query_time  |
|-----------------------------|-------------|
| select * from t1 where a=1; | 0.302558006 |
| select * from t1 where a=2; | 0.401313532 |

### 7.2.6 Query slow queries with pseudo stats

```
select query, query_time, stats
from information_schema.slow_query
where is_internal = false
and stats like '%pseudo%';
```

Output example:

| query                       | query_time  | stats                           |
|-----------------------------|-------------|---------------------------------|
| select * from t1 where a=1; | 0.302558006 | t1:pseudo                       |
| select * from t1 where a=2; | 0.401313532 | t1:pseudo                       |
| select * from t1 where a>2; | 0.602011247 | t1:pseudo                       |
| select * from t1 where a>3; | 0.50077719  | t1:pseudo                       |
| select * from t1 join t2;   | 0.931260518 | t1:407872303825682445,t2:pseudo |
| →                           |             |                                 |
|                             |             |                                 |
|                             |             |                                 |

#### 7.2.6.1 Query slow queries whose execution plan is changed

When the execution plan of SQL statements of the same category is changed, the execution slows down, because the statistics is outdated, or the statistics is not accurate enough to reflect the real data distribution. You can use the following SQL statement to query SQL statements with different execution plans.

```
select count(distinct plan_digest) as count,
 digest,
 min(query)
 from cluster_slow_query
 group by digest
 having count > 1
 limit 3\G
```

Output example:

```
*****[1. row]*****
count | 2
digest | 17b4518fde82e32021877878bec2bb309619d384fca944106fcfa9c93b536e94
min(query) | SELECT DISTINCT c FROM sbtest25 WHERE id BETWEEN ? AND ? ORDER
 → BY c [arguments: (291638, 291737)];
*****[2. row]*****
count | 2
digest | 9337865f3e2ee71c1c2e740e773b6dd85f23ad00f8fa1f11a795e62e15fc9b23
min(query) | SELECT DISTINCT c FROM sbtest22 WHERE id BETWEEN ? AND ? ORDER
 → BY c [arguments: (215420, 215519)];
*****[3. row]*****
count | 2
digest | db705c89ca2dfc1d39d10e0f30f285cbbadec7e24da4f15af461b148d8ffb020
```

```
min(query) | SELECT DISTINCT c FROM sbtest11 WHERE id BETWEEN ? AND ? ORDER
 ↪ BY c [arguments: (303359, 303458)];
```

Then you can query the different plans using the SQL fingerprint in the query result above:

```
select min(plan),
 plan_digest
 from cluster_slow_query
 where digest='17
 ↪ b4518fde82e32021877878bec2bb309619d384fca944106fcfa9c93b536e94'
group by plan_digest\G
```

Output example:

```
***** 1. row *****
min(plan): Sort_6 root 100.00131380758702 sbtest.
 ↪ sbtest25.c:asc
 - HashAgg_10 root 100.00131380758702 group by:sbtest.
 ↪ sbtest25.c, funcs:firstrow(sbtest.sbtest25.c)->sbtest.sbtest25
 ↪ .c
 - TableReader_15 root 100.00131380758702 data:
 ↪ TableRangeScan_14
 - TableScan_14 cop 100.00131380758702 table:sbtest25,
 ↪ range:[502791,502890], keep order:false
plan_digest: 6
 ↪ afbbd21f60ca6c6fdf3d3cd94f7c7a49dd93c00fcf8774646da492e50e204ee
***** 2. row *****
min(plan): Sort_6 root 1 sbtest.
 ↪ sbtest25.c:asc
 - HashAgg_12 root 1 group by:sbtest.
 ↪ sbtest25.c, funcs:firstrow(sbtest.sbtest25.c)->sbtest.sbtest25
 ↪ .c
 - TableReader_13 root 1 data:HashAgg_8
 - HashAgg_8 cop 1 group by:sbtest.
 ↪ sbtest25.c,
 - TableScan_11 cop 1.2440069558121831 table:sbtest25,
 ↪ range:[472745,472844], keep order:false
```

#### 7.2.6.2 Query the number of slow queries for each TiDB node in a cluster

```
select instance, count(*) from information_schema.cluster_slow_query where
 ↪ time >= "2020-03-06 00:00:00" and time < now() group by instance;
```

Output example:

| instance      | count(*) |
|---------------|----------|
| 0.0.0.0:10081 | 124      |
| 0.0.0.0:10080 | 119771   |

### 7.2.6.3 Query slow logs occurring only in abnormal time period

If you find problems such as decreased QPS or increased latency for the time period from 2020-03-10 13:24:00 to 2020-03-10 13:27:00, the reason might be that a large query crops up. Run the following SQL statement to query slow logs that occur only in abnormal time period. The time range from 2020-03-10 13:20:00 to 2020-03-10 13:23:00 refers to the normal time period.

```

SELECT * FROM
 (SELECT /*+ AGG_TO_COP(), HASH_AGG() */ count(*),
 min(time),
 sum(query_time) AS sum_query_time,
 sum(Process_time) AS sum_process_time,
 sum(Wait_time) AS sum_wait_time,
 sum(Commit_time),
 sum(Request_count),
 sum(process_keys),
 sum(Write_keys),
 max(Cop_proc_max),
 min(query),min(prev_stmt),
 digest
 FROM information_schema.CLUSTER_SLOW_QUERY
 WHERE time >= '2020-03-10 13:24:00'
 AND time < '2020-03-10 13:27:00'
 AND Is_internal = false
 GROUP BY digest) AS t1
WHERE t1.digest NOT IN
 (SELECT /*+ AGG_TO_COP(), HASH_AGG() */ digest
 FROM information_schema.CLUSTER_SLOW_QUERY
 WHERE time >= '2020-03-10 13:20:00'
 AND time < '2020-03-10 13:23:00'
 GROUP BY digest)
ORDER BY t1.sum_query_time DESC limit 10\G

```

Output example:

| *****[ 1. row ]***** |     |
|----------------------|-----|
| count(*)             | 200 |

```

min(time) | 2020-03-10 13:24:27.216186
sum_query_time | 50.114126194
sum_process_time | 268.351
sum_wait_time | 8.476
sum(Commit_time) | 1.044304306
sum(Request_count) | 6077
sum(process_keys) | 202871950
sum(Write_keys) | 319500
max(Cop_proc_max) | 0.263
min(query) | delete from test.tcs2 limit 5000;
min(prev_stmt) |
digest | 24
↪ bd6d8a9b238086c9b8c3d240ad4ef32f79ce94cf5a468c0b8fe1eb5f8d03df

```

#### 7.2.6.4 Parse other TiDB slow log files

TiDB uses the session variable `tidb_slow_query_file` to control the files to be read and parsed when querying `INFORMATION_SCHEMA.SLOW_QUERY`. You can query the content of other slow query log files by modifying the value of the session variable.

```
set tidb_slow_query_file = "/path-to-log/tidb-slow.log"
```

#### 7.2.6.5 Parse TiDB slow logs with pt-query-digest

Use `pt-query-digest` to parse TiDB slow logs.

##### Note:

It is recommended to use `pt-query-digest` 3.0.13 or later versions.

For example:

```
pt-query-digest --report tidb-slow.log
```

Output example:

```

320ms user time, 20ms system time, 27.00M rss, 221.32M vsz
Current date: Mon Mar 18 13:18:51 2019
Hostname: localhost.localdomain
Files: tidb-slow.log
Overall: 1.02k total, 21 unique, 0 QPS, 0x concurrency -----
Time range: 2019-03-18-12:22:16 to 2019-03-18-13:08:52
Attribute total min max avg 95% stddev median

```

```
=====
Exec time 218s 10ms 13s 213ms 30ms 1s 19ms
Query size 175.37k 9 2.01k 175.89 158.58 122.36 158.58
Commit time 46ms 2ms 7ms 3ms 7ms 1ms 3ms
Conn ID 71 1 16 8.88 15.25 4.06 9.83
Process keys 581.87k 2 103.15k 596.43 400.73 3.91k 400.73
Process time 31s 1ms 10s 32ms 19ms 334ms 16ms
Request coun 1.97k 1 10 2.02 1.96 0.33 1.96
Total keys 636.43k 2 103.16k 652.35 793.42 3.97k 400.73
Txn start ts 374.38E 0 16.00E 375.48P 1.25P 89.05T 1.25P
Wait time 943ms 1ms 19ms 1ms 2ms 1ms 972us
.
.
.
```

### 7.2.7 Identify problematic SQL statements

Not all of the `SLOW_QUERY` statements are problematic. Only those whose `process_time` is very large increase the pressure on the entire cluster.

The statements whose `wait_time` is very large and `process_time` is very small are usually not problematic. This is because the statement is blocked by real problematic statements and it has to wait in the execution queue, which leads to a much longer response time.

#### 7.2.7.1 admin show slow command

In addition to the TiDB log file, you can identify slow queries by running the `admin ↵ show slow` command:

```
admin show slow recent N
admin show slow top [internal | all] N
```

`recent N` shows the recent N slow query records, for example:

```
admin show slow recent 10
```

`top N` shows the slowest N query records recently (within a few days). If the `internal` option is provided, the returned results would be the inner SQL executed by the system; If the `all` option is provided, the returned results would be the user's SQL combined with inner SQL; Otherwise, this command would only return the slow query records from the user's SQL.

```
admin show slow top 3
admin show slow top internal 3
admin show slow top all 5
```

TiDB stores only a limited number of slow query records because of the limited memory. If the value of N in the query command is greater than the records count, the number of returned records is smaller than N.

The following table shows output details:

| Column   |                                        |
|----------|----------------------------------------|
| name     | Description                            |
| start    | The starting time of the SQL execution |
| duration | The duration of the SQL execution      |
| details  | The details of the SQL execution       |

| Column        |                                                                                             |
|---------------|---------------------------------------------------------------------------------------------|
| name          | Description                                                                                 |
| succ          | Whether the SQL statement is executed successfully.<br>1 means success and 0 means failure. |
| conn_id       | The connection ID for the session                                                           |
| transcationTs | The commit timestamp for a transaction commit                                               |

| Column    |                                                                 |
|-----------|-----------------------------------------------------------------|
| name      | Description                                                     |
| user      | The user name for the execution of the statement                |
| db        | The database involved when the statement is executed            |
| table_ids | The ID of the table involved when the SQL statement is executed |

---

| Column    |                                                                 |
|-----------|-----------------------------------------------------------------|
| name      | Description                                                     |
| index_ids | The ID of the index involved when the SQL statement is executed |
| internal  | This is a TiDB internal SQL statement                           |
| digest    | The finger-print of the SQL statement                           |
| sql       | The SQL statement that is being executed or has been executed   |

---

## 7.3 Analyze Slow Queries

To address the issue of slow queries, you need to take the following two steps:

1. Among many queries, identify which type of queries are slow.
2. Analyze why this type of queries are slow.

You can easily perform step 1 using the [slow query log](#) and the [statement summary table](#) features. It is recommended to use [TiDB Dashboard](#), which integrates the two features and directly displays the slow queries in your browser.

This document focuses on how to perform step 2 - analyze why this type of queries are slow.

Generally, slow queries have the following major causes:

- Optimizer issues, such as wrong index selected, wrong join type or sequence selected.
- System issues. All issues not caused by the optimizer are system issues. For example, a busy TiKV instance processes requests slowly; outdated Region information causes slow queries.

In actual situations, optimizer issues might cause system issues. For example, for a certain type of queries, the optimizer uses a full table scan instead of the index. As a result, the SQL queries consume many resources, which causes the CPU usage of some TiKV instances to soar. This seems like a system issue, but in essence, it is an optimizer issue.

To identify system issues is relatively simple. To analyze optimizer issues, you need to determine whether the execution plan is reasonable or not. Therefore, it is recommended to analyze slow queries by following these procedures:

1. Identify the performance bottleneck of the query, that is, the time-consuming part of the query process.
2. Analyze the system issues: analyze the possible causes according to the query bottleneck and the monitoring/log information of that time.
3. Analyze the optimizer issues: analyze whether there is a better execution plan.

The procedures above are explained in the following sections.

### 7.3.1 Identify the performance bottleneck of the query

First, you need to have a general understanding of the query process. The key stages of the query execution process in TiDB are illustrated in [TiDB performance map](#).

You can get the duration information using the following methods:

- **Slow log.** It is recommended to view the slow log in [TiDB Dashboard](#).
- **EXPLAIN ANALYZE statement.**

The methods above are different in the following aspects:

- The slow log records the duration of almost all stages of a SQL execution, from parsing to returning results, and is relatively comprehensive (you can query and analyze the slow log in TiDB Dashboard in an intuitive way).
- By executing `EXPLAIN ANALYZE`, you can learn the time consumption of each operator in an actual SQL execution. The results have more detailed statistics of the execution duration.

In summary, the slow log and `EXPLAIN ANALYZE` statements help you determine the SQL query is slow in which component (TiDB or TiKV) at which stage of the execution. Therefore, you can accurately identify the performance bottleneck of the query.

In addition, since v4.0.3, the `Plan` field in the slow log also includes the SQL execution information, which is the result of `EXPLAIN ANALYZE`. So you can find all information of SQL duration in the slow log.

### 7.3.2 Analyze system issues

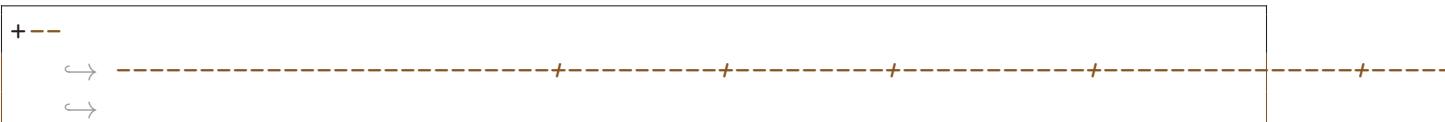
System issues can be divided into the following types according to different execution stages of a SQL statement:

1. TiKV is slow in data processing. For example, the TiKV coprocessor processes data slowly.
2. TiDB is slow in execution. For example, a `Join` operator processes data slowly.
3. Other key stages are slow. For example, getting the timestamp takes a long time.

For each slow query, first determine to which type the query belongs, and then analyze it in detail.

#### 7.3.2.1 TiKV is slow in data processing

If TiKV is slow in data processing, you can easily identify it in the result of `EXPLAIN ANALYZE`. In the following example, `StreamAgg_8` and `TableFullScan_15`, two `tikv-tasks` (as indicated by `cop[tikv]` in the `task` column), take `170ms` to execute. After subtracting `170ms`, the execution time of TiDB operators account for a very small proportion of the total execution time. This indicates that the bottleneck is in TiKV.



|                                                                       |                  | estRows   | actRows | task      | access object |      |
|-----------------------------------------------------------------------|------------------|-----------|---------|-----------|---------------|------|
|                                                                       | ↳ execution info |           |         |           |               |      |
|                                                                       | ↳ operator info  |           |         | memory    | disk          |      |
| +--                                                                   |                  |           |         |           |               |      |
|                                                                       | ↳                |           |         |           |               |      |
| StreamAgg_16                                                          |                  | 1.00      | 1       | root      |               | time |
| ↳ :170.08572ms, loops:2                                               |                  |           |         |           |               |      |
| ↳ funcs:count(Column#5)->Column#3                                     |                  | 372 Bytes | N/A     |           |               |      |
| -TableReader_17                                                       |                  | 1.00      | 1       | root      |               | time |
| ↳ :170.080369ms, loops:2, rpc num: 1, rpc time:17.023347ms, proc keys |                  |           |         |           |               |      |
| ↳ :28672   data:StreamAgg_8                                           |                  | 202 Bytes | N/A     |           |               |      |
| -StreamAgg_8                                                          |                  | 1.00      | 1       | cop[tikv] |               | time |
| ↳ :170ms, loops:29                                                    |                  |           |         |           |               |      |
| ↳ funcs:count(1)->Column#5                                            |                  | N/A       | N/A     |           |               |      |
| -TableFullScan_15                                                     |                  | 7.00      | 28672   | cop[tikv] | table:t       | time |
| ↳ :170ms, loops:29                                                    |                  |           |         |           |               |      |
| ↳ keep order:false, stats:pseudo                                      |                  | N/A       | N/A     |           |               |      |
| +--                                                                   |                  |           |         |           |               |      |
|                                                                       | ↳                |           |         |           |               |      |

In addition, the `Cop_process` and `Cop_wait` fields in the slow log can also help your analysis. In the following example, the total duration of the query is around 180.85ms, and the largest coptask takes 171ms. This indicates that the bottleneck of this query is on the TiKV side.

For the description of each field in the slow log, see [fields description](#).

```
Query_time: 0.18085
...
Num_cop_tasks: 1
Cop_process: Avg_time: 170ms P90_time: 170ms Max_time: 170ms Max_addr:
 ↳ 10.6.131.78
Cop_wait: Avg_time: 1ms P90_time: 1ms Max_time: 1ms Max_addr: 10.6.131.78
```

After identifying that TiKV is the bottleneck, you can find out the cause as described in the following sections.

### 7.3.2.1.1 TiKV instance is busy

During the execution of a SQL statement, TiDB might fetch data from multiple TiKV instances. If one TiKV instance responds slowly, the overall SQL execution speed is slowed down.

The `Cop_wait` field in the slow log can help you determine this cause.

```
Cop_wait: Avg_time: 1ms P90_time: 2ms Max_time: 110ms Max_Addr:
↪ 10.6.131.78
```

The log above shows that a `Cop-task` sent to the 10.6.131.78 instance waits 110ms before being executed. It indicates that this instance is busy. You can check the CPU monitoring of that time to confirm the cause.

### 7.3.2.1.2 Too many outdated keys

A TiKV instance has much outdated data, which needs to be cleaned up for data scan. This impacts the processing speed.

Check `Total_keys` and `Processed_keys`. If they are greatly different, the TiKV instance has too many keys of the older versions.

```
...
Total_keys: 2215187529 Processed_keys: 1108056368
...
```

### 7.3.2.2 Other key stages are slow

#### 7.3.2.2.1 Slow in getting timestamps

You can compare `Wait_TS` and `Query_time` in the slow log. The timestamps are prefetched, so generally `Wait_TS` should be low.

```
Query_time: 0.0300000
...
Wait_TS: 0.02500000
```

#### 7.3.2.2.2 Outdated Region information

Region information on the TiDB side might be outdated. In this situation, TiKV might return the `regionMiss` error. Then TiDB gets the Region information from PD again, which is reflected in the `Cop_backoff` information. Both the failed times and the total duration are recorded.

```
Cop_backoff_regionMiss_total_times: 200 Cop_backoff_regionMiss_total_time
↪ : 0.2 Cop_backoff_regionMiss_max_time: 0.2
↪ Cop_backoff_regionMiss_max_addr: 127.0.0.1
↪ Cop_backoff_regionMiss_avg_time: 0.2 Cop_backoff_regionMiss_p90_time:
↪ 0.2
Cop_backoff_rpcPD_total_times: 200 Cop_backoff_rpcPD_total_time: 0.2
↪ Cop_backoff_rpcPD_max_time: 0.2 Cop_backoff_rpcPD_max_addr: 127.0.0.1
↪ Cop_backoff_rpcPD_avg_time: 0.2 Cop_backoff_rpcPD_p90_time: 0.2
```

### 7.3.2.2.3 Subqueries are executed in advance

For statements with non-correlated subqueries, the subquery part might be executed in advance. For example, in `select * from t1 where a = (select max(a)from t2)`, the `select max(a)from t2` part might be executed in advance in the optimization stage. The result of EXPLAIN ANALYZE does not show the duration of this type of subqueries.

```
mysql> explain analyze select count(*) from t where a=(select max(t1.a)
 ↵ from t t1, t t2 where t1.a=t2.a);
+---+
| id | estRows | actRows | task | access object |
| execution info | | operator info | | memory | disk |
+---+
| StreamAgg_59 | 1.00 | 1 | root | | time
| ↵ :4.69267ms, loops:2 | funcs:count(Column#10)->Column#8 | 372 Bytes |
| ↵ N/A |
| -TableReader_60 | 1.00 | 1 | root | |
| ↵ time:4.690428ms, loops:2 | data:StreamAgg_48 | 141 Bytes | N/A
| ↵ |
| -StreamAgg_48 | 1.00 | | cop[tikv] | |
| ↵ time:0ns, loops:0 | funcs:count(1)->Column#10 | N/A | N/A
| ↵ |
| -Selection_58 | 16384.00 | | cop[tikv] | |
| ↵ time:0ns, loops:0 | eq(test.t.a, 1) | N/A | N/A
| ↵ |
| -TableFullScan_57 | 16384.00 | -1 | cop[tikv] | table:t |
| ↵ time:0s, loops:0 | keep order:false | N/A | N/A
| ↵ |
+---+
| 5 rows in set (7.77 sec)
```

But you can identify this type of subquery execution in the slow log:

```
Query_time: 7.770634843
...
Rewrite_time: 7.765673663 Preproc_subqueries: 1 Preproc_subqueries_time:
 ↵ 7.765231874
```

From log record above, you can see that a subquery is executed in advance and takes 7.76s.

### 7.3.2.3 TiDB is slow in execution

Assume that the execution plan in TiDB is correct but the execution is slow. To solve this type of issue, you can adjust parameters or use the hint according to the result of EXPLAIN ANALYZE for the SQL statement.

If the execution plan is incorrect, see the [Analyze optimizer issues](#) section.

#### 7.3.2.3.1 Low concurrency

If the bottleneck is in the operator with concurrency, speed up the execution by adjusting the concurrency. For example:

```
mysql> explain analyze select sum(t1.a) from t t1, t t2 where t1.a=t2.a;
+---+
| id | estRows | actRows | task | access
| object | execution info
| | | | | operator
| info | | | | disk |
+---+
| HashAgg_11 | 1.00 | 1 | root |
| | time:9.666832189s, loops:2, PartialConcurrency:4,
| | FinalConcurrency:4 | funcs:sum(Column#6)->Column#5
| | 322.125 KB | N/A |
| -Projection_24 | 268435456.00 | 268435456 | root |
| | time:9.098644711s, loops:262145, Concurrency:4
| | | cast(test.t.a, decimal(65,0) BINARY)->
| | Column#6 | 199 KB | N/A |
| -HashJoin_14 | 268435456.00 | 268435456 | root |
| | time:6.616773501s, loops:262145, Concurrency:5, probe
| | collision:0, build:881.404us | inner join, equal:[eq(test.t.a, test.t
| .a)] | 131.75 KB | 0 Bytes |
| -TableReader_21(Build) | 16384.00 | 16384 | root |
| | time:6.553717ms, loops:17
| | | | data:Selection_20
| | | | 33.6318359375 KB | N/A |
| -Selection_20 | 16384.00 | | cop[tikv] |
| | time:0ns, loops:0
| | | | not(isnull(
| | test.t.a)) | N/A | N/A |
| -TableFullScan_19 | 16384.00 | -1 | cop[tikv] | table:
| t2 | time:0s, loops:0
| | | | keep order:
```

```

 ↗ false | N/A | N/A |
| - TableReader_18(Probe) | 16384.00 | 16384 | root |
 ↗ | time:6.880923ms, loops:17
 ↗ | data:Selection_17
 ↗ | 33.6318359375 KB | N/A |
| - Selection_17 | 16384.00 | | cop[tikv] |
 ↗ | time:0ns, loops:0
 ↗ | not(isnull(
 ↗ test.t.a)) | N/A | N/A |
| - TableFullScan_16 | 16384.00 | -1 | cop[tikv] | table:
 ↗ t1 | time:0s, loops:0
 ↗ | keep order:
 ↗ false | N/A | N/A |
+--+
 ↗ -----+-----+-----+-----+
 ↗
9 rows in set (9.67 sec)

```

As shown above, HashJoin\_14 and Projection\_24 consume much of the execution time. Consider increasing their concurrency using SQL variables to speed up execution.

All system variables are documented in [system-variables](#). To increase the concurrency of HashJoin\_14, you can modify the `tidb_hash_join_concurrency` system variable.

### 7.3.2.3.2 Data is spilled to disk

Another cause of slow execution is disk spill that occurs during execution if the memory limit is reached. You can find out this cause in the execution plan and the slow log:

```

+--+
 ↗ -----+-----+-----+-----+
 ↗
| id | estRows | actRows | task | access object |
 ↗ execution info | operator info | memory |
 ↗ disk | |
+--+
 ↗ -----+-----+-----+-----+
 ↗
| Sort_4 | 462144.00 | 462144 | root | | time
 ↗ :2.02848898s, loops:453 | test.t.a | 149.68795776367188 MB |
 ↗ 219.3203125 MB |
| -TableReader_8 | 462144.00 | 462144 | root | | time
 ↗ :616.211272ms, loops:453 | data:TableFullScan_7 | 197.49601364135742
 ↗ MB | N/A |
| -TableFullScan_7 | 462144.00 | -1 | cop[tikv] | table:t | time:0
 ↗ s, loops:0 | keep order:false | N/A | N/A |

```

```

 ↗ |
+---+
 ↗-----+-----+-----+-----+
 ↗

```

```

...
Disk_max: 229974016
...

```

### 7.3.2.3.3 Join operations with Cartesian product

Join operations with Cartesian product generate data volume as large as `left child row count * right child row count`. This is inefficient and should be avoided.

This type of join operations is marked `CARTESIAN` in the execution plan. For example:

```

mysql> explain select * from t t1, t t2 where t1.a>t2.a;
+---+
 ↗-----+-----+-----+-----+
 ↗
| id | estRows | task | access object | operator
| info | | | |
+---+
 ↗-----+-----+-----+-----+
 ↗
| HashJoin_8 | 99800100.00 | root | | CARTESIAN
 ↗ inner join, other cond:gt(test.t.a, test.t.a) |
| -TableReader_15(Build) | 9990.00 | root | | data:
 ↗ Selection_14
| -Selection_14 | 9990.00 | cop[tikv] | | not(
 ↗ isnull(test.t.a))
| -TableFullScan_13 | 10000.00 | cop[tikv] | table:t2 | keep
 ↗ order:false, stats:pseudo
| -TableReader_12(Probe) | 9990.00 | root | | data:
 ↗ Selection_11
| -Selection_11 | 9990.00 | cop[tikv] | | not(
 ↗ isnull(test.t.a))
| -TableFullScan_10 | 10000.00 | cop[tikv] | table:t1 | keep
 ↗ order:false, stats:pseudo
+---+
 ↗-----+-----+-----+-----+
 ↗

```

### 7.3.3 Analyze optimizer issues

To analyze optimizer issues, you need to determine whether the execution plan is reasonable or not. You need to have some understanding of the optimization process and each operator.

For the following examples, assume that the table schema is `create table t (id int → , a int, b int, c int, primary key(id), key(a), key(b, c))`.

1. `select * from t`: There is no filter condition and a full table scan is performed. So the `TableFullScan` operator is used to read data.
2. `select a from t where a=2`: There is a filter condition and only the index columns are read, so the `IndexReader` operator is used to read data.
3. `select * from t where a=2`: There is a filter condition for `a` but the `a` index cannot fully cover the data to be read, so the `IndexLookup` operator is used.
4. `select b from t where c=3`: Without the prefix condition, the multi-column index cannot be used. So the `IndexFullScan` is used.
5. ...

The examples above are operators used for data reads. For more operators, see [Understand TiDB Execution Plan](#).

In addition, reading [SQL Tuning Overview](#) helps you better understand the TiDB optimizer and determine whether the execution plan is reasonable or not.

Most optimizer issues are explained in [SQL Tuning Overview](#). For the solutions, see the following documents:

1. [Wrong Index Solution](#)
2. [Wrong join order](#)
3. [Expressions are not pushed down](#)

## 7.4 SQL Diagnostics

### Warning:

SQL diagnostics is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

SQL diagnostics is a feature introduced in TiDB v4.0. You can use this feature to locate problems in TiDB with higher efficiency. Before TiDB v4.0, you need to use different tools to obtain different information.

The SQL diagnostic system has the following advantages:

- It integrates information from all components of the system as a whole.
- It provides a consistent interface to the upper layer through system tables.
- It provides monitoring summaries and automatic diagnostics.
- You will find it easier to query cluster information.

#### 7.4.1 Overview

The SQL diagnostic system consists of three major parts:

- **Cluster information table:** The SQL diagnostics system introduces cluster information tables that provide a unified way to get the discrete information of each instance. This system fully integrates the cluster topology, hardware information, software information, kernel parameters, monitoring, system information, slow queries, statements, and logs of the entire cluster into the table. So you can query these information using SQL statements.
- **Cluster monitoring table:** The SQL diagnostic system introduces cluster monitoring tables. All of these tables are in `metrics_schema`, and you can query monitoring information using SQL statements. Compared to the visualized monitoring before v4.0, you can use this SQL-based method to perform correlated queries on all the monitoring information of the entire cluster, and compare the results of different time periods to quickly identify performance bottlenecks. Because the TiDB cluster has many monitoring metrics, the SQL diagnostic system also provides monitoring summary tables, so you can find abnormal monitoring items more easily.

**Automatic diagnostics:** Although you can manually execute SQL statements to query cluster information tables, cluster monitoring tables, and summary tables to locate issues, the automatic diagnostics allows you to quickly locate common issues. The SQL diagnostic system performs automatic diagnostics based on the existing cluster information tables and monitoring tables, and provides relevant diagnostic result tables and diagnostic summary tables.

#### 7.4.2 Cluster information tables

The cluster information tables bring together the information of all instances and instances in a cluster. With these tables, you can query all cluster information using only one SQL statement. The following is a list of cluster information tables:

- From the cluster topology table `information_schema.cluster_info`, you can get the current topology information of the cluster, the version of each instance, the Git Hash corresponding to the version, the starting time of each instance, and the running time of each instance.

- From the cluster configuration table `information_schema.cluster_config`, you can get the configuration of all instances in the cluster. For versions earlier than 4.0, you need to access the HTTP API of each instance one by one to get these configuration information.
- On the cluster hardware table `information_schema.cluster_hardware`, you can quickly query the cluster hardware information.
- On the cluster load table `information_schema.cluster_load`, you can query the load information of different instances and hardware types of the cluster.
- On the kernel parameter table `information_schema.cluster_systeminfo`, you can query the kernel configuration information of different instances in the cluster. Currently, TiDB supports querying the sysctl information.
- On the cluster log table `information_schema.cluster_log`, you can query cluster logs. By pushing down query conditions to each instance, the impact of the query on cluster performance is less than that of the `grep` command.

On the system tables earlier than TiDB v4.0, you can only view the current instance. TiDB v4.0 introduces the corresponding cluster tables and you can have a global view of the entire cluster on a single TiDB instance. These tables are currently in `information_schema`, and the query method is the same as other `information_schema` system tables.

#### 7.4.3 Cluster monitoring tables

To dynamically observe and compare cluster conditions in different time periods, the SQL diagnostic system introduces cluster monitoring system tables. All monitoring tables are in `metrics_schema`, and you can query the monitoring information using SQL statements. Using this method, you can perform correlated queries on all monitoring information of the entire cluster and compare the results of different time periods to quickly identify performance bottlenecks.

- `information_schema.metrics_tables`: Because many system tables exist now, you can query meta-information of these monitoring tables on the `information_schema.→ metrics_tables` table.

Because the TiDB cluster has many monitoring metrics, TiDB provides the following monitoring summary tables in v4.0:

- The monitoring summary table `information_schema.metrics_summary` summarizes all monitoring data to for you to check each monitoring metric with higher efficiency.
- `information_schema.metrics_summary_by_label` also summarizes all monitoring data. Particularly, this table aggregates statistics using different labels of each monitoring metric.

#### 7.4.4 Automatic diagnostics

On the cluster information tables and cluster monitoring tables above, you need to manually execute SQL statements to troubleshoot the cluster. TiDB v4.0 supports the automatic diagnostics. You can use diagnostic-related system tables based on the existing basic information tables, so that the diagnostics is automatically executed. The following are the system tables related to the automatic diagnostics:

- The diagnostic result table `information_schema.inspection_result` displays the diagnostic result of the system. The diagnostics is passively triggered. Executing `select → * from inspection_result` triggers all diagnostic rules to diagnose the system, and the faults or risks in the system are displayed in the results.
- The diagnostic summary table `information_schema.inspection_summary` summarizes the monitoring information of a specific link or module. You can troubleshoot and locate problems based on the context of the entire module or link.

## 7.5 Identify Expensive Queries

TiDB allows you to identify expensive queries during SQL execution, so you can diagnose and improve the performance of SQL execution. Specifically, TiDB prints the information about statements whose execution time exceeds `tidb_expensive_query_time_threshold` (60 seconds by default) or memory usage exceeds `mem-quota-query` (1 GB by default) to the `tidb-server log file` (“`tidb.log`” by default).

### Note:

The expensive query log differs from the `slow query log` in this way: TiDB prints statement information to the expensive query log **as soon as** the statement exceeds the threshold of resource usage (execution time or memory usage); while TiDB prints statement information to the slow query log **after** the statement execution.

### 7.5.1 Expensive query log example

```
[2020/02/05 15:32:25.096 +08:00] [WARN] [expensivequery.go:167] [
 ← expensive_query] [cost_time=60.008338935s] [wait_time=0s] [
 ← request_count=1] [total_keys=70] [process_keys=65] [num_cop_tasks=1]
 ← [process_avg_time=0s] [process_p90_time=0s] [process_max_time=0s] [
 ← process_max_addr=10.0.1.9:20160] [wait_avg_time=0.002s] [
 ← wait_p90_time=0.002s] [wait_max_time=0.002s] [wait_max_addr
 ← =10.0.1.9:20160] [stats=t:pseudo] [conn_id=60026] [user=root] [
```

```

↪ database=test] [table_ids="[122]" [txn_start_ts=414420273735139329]
↪ [mem_max="1035 Bytes (1.0107421875 KB)"] [sql="insert into t select
↪ sleep(1) from t"]

```

### 7.5.2 Fields description

Basic fields:

- **cost\_time**: The execution time of a statement when the log is printed.
- **stats**: The version of statistics used by the tables or indexes involved in a statement. If the value is `pseudo`, it means that there are no available statistics. In this case, you need to analyze the tables or indexes.
- **table\_ids**: The IDs of the tables involved in a statement.
- **txn\_start\_ts**: The start timestamp and the unique ID of a transaction. You can use this value to search for the transaction-related logs.
- **sql**: The sql statement.

Memory usage related fields:

- **mem\_max**: Memory usage of a statement when the log is printed. This field has two kinds of units to measure memory usage: byte and other readable and adaptable units (such as MB and GB).

User related fields:

- **user**: The name of the user who executes the statement.
- **conn\_id**: The connection ID (session ID). For example, you can use the keyword `con:60026` to search for the log whose session ID is 60026.
- **database**: The database where the statement is executed.

TiKV Coprocessor task related fields:

- **wait\_time**: The total waiting time of all Coprocessor requests of a statement in TiKV. Because the Coprocessor of TiKV runs a limited number of threads, requests might queue up when all threads of Coprocessor are working. When a request in the queue takes a long time to process, the waiting time of the subsequent requests increases.
- **request\_count**: The number of Coprocessor requests that a statement sends.
- **total\_keys**: The number of keys that Coprocessor has scanned.
- **processed\_keys**: The number of keys that Coprocessor has processed. Compared with `total_keys`, `processed_keys` does not include the old versions of MVCC. A great difference between `processed_keys` and `total_keys` indicates that many old versions exist.

- `num_cop_tasks`: The number of Coprocessor requests that a statement sends.
- `process_avg_time`: The average execution time of Coprocessor tasks.
- `process_p90_time`: The P90 execution time of Coprocessor tasks.
- `process_max_time`: The maximum execution time of Coprocessor tasks.
- `process_max_addr`: The address of the Coprocessor task with the longest execution time.
- `wait_avg_time`: The average waiting time of Coprocessor tasks.
- `wait_p90_time`: The P90 waiting time of Coprocessor tasks.
- `wait_max_time`: The maximum waiting time of Coprocessor tasks.
- `wait_max_addr`: The address of the Coprocessor task with the longest waiting time.

## 7.6 Statement Summary Tables

To better handle SQL performance issues, MySQL has provided [statement summary tables](#) in `performance_schema` to monitor SQL with statistics. Among these tables, `events_statements_summary_by_digest` is very useful in locating SQL problems with its abundant fields such as latency, execution times, rows scanned, and full table scans.

Therefore, starting from v4.0.0-rc.1, TiDB provides system tables in `information_schema` (not `performance_schema`) that are similar to `events_statements_summary_by_digest` in terms of features.

- `statements_summary`
- `statements_summary_history`
- `cluster_statements_summary`
- `cluster_statements_summary_history`
- `statements_summary_evicted`

This document details these tables and introduces how to use them to troubleshoot SQL performance issues.

### 7.6.1 `statements_summary`

`statements_summary` is a system table in `information_schema`. `statements_summary` groups the SQL statements by the SQL digest and the plan digest, and provides statistics for each SQL category.

The “SQL digest” here means the same as used in slow logs, which is a unique identifier calculated through normalized SQL statements. The normalization process ignores constant, blank characters, and is case insensitive. Therefore, statements with consistent syntaxes have the same digest. For example:

```
SELECT * FROM employee WHERE id IN (1, 2, 3) AND salary BETWEEN 1000 AND
 ↪ 2000;
select * from EMPLOYEE where ID in (4, 5) and SALARY between 3000 and 4000;
```

After normalization, they are both of the following category:

```
select * from employee where id in (...) and salary between ? and ?;
```

The “plan digest” here refers to the unique identifier calculated through normalized execution plan. The normalization process ignores constants. The same SQL statements might be grouped into different categories because the same statements might have different execution plans. SQL statements of the same category have the same execution plan.

`statements_summary` stores the aggregated results of SQL monitoring metrics. In general, each of the monitoring metrics includes the maximum value and average value. For example, the execution latency metric corresponds to two fields: `AVG_LATENCY` (average latency) and `MAX_LATENCY` (maximum latency).

To make sure that the monitoring metrics are up to date, data in the `statements_summary` table is periodically cleared, and only recent aggregated results are retained and displayed. The periodical data clearing is controlled by the `tidb_stmt_summary_refresh_interval` system variable. If you happen to make a query right after the clearing, the data displayed might be very little.

The following is a sample output of querying `statements_summary`:

```
SUMMARY_BEGIN_TIME: 2020-01-02 11:00:00
SUMMARY_END_TIME: 2020-01-02 11:30:00
 STMT_TYPE: Select
 SCHEMA_NAME: test
 DIGEST: 0611
 ↪ cc2fe792f8c146cc97d39b31d9562014cf15f8d41f23a4938ca341f54182
 ↪
 DIGEST_TEXT: select * from employee where id = ?
 TABLE_NAMES: test.employee
 INDEX_NAMES: NULL
 SAMPLE_USER: root
 EXEC_COUNT: 3
 SUM_LATENCY: 1035161
 MAX_LATENCY: 399594
 MIN_LATENCY: 301353
 AVG_LATENCY: 345053
 AVG_PARSE_LATENCY: 57000
 MAX_PARSE_LATENCY: 57000
 AVG_COMPILE_LATENCY: 175458
 MAX_COMPILE_LATENCY: 175458

 AVG_MEM: 103
 MAX_MEM: 103
 AVG_DISK: 65535
 MAX_DISK: 65535
 AVG_AFFECTED_ROWS: 0
```

```

FIRST_SEEN: 2020-01-02 11:12:54
LAST_SEEN: 2020-01-02 11:25:24
QUERY_SAMPLE_TEXT: select * from employee where id=3100
PREV_SAMPLE_TEXT:
PLAN_DIGEST:
 ↳ f415b8d52640b535b9b12a9c148a8630d2c6d59e419aad29397842e32e8e5de3
 ↳
PLAN: Point_Get_1 root 1 table:employee, handle:3100

```

#### Note:

In TiDB, the time unit of fields in statement summary tables is nanosecond (ns), whereas in MySQL the time unit is picosecond (ps).

#### 7.6.2 statements\_summary\_history

The table schema of `statements_summary_history` is identical to that of `statements_summary`. `statements_summary_history` saves the historical data of a time range. By checking historical data, you can troubleshoot anomalies and compare monitoring metrics of different time ranges.

The fields `SUMMARY_BEGIN_TIME` and `SUMMARY_END_TIME` represent the start time and the end time of the historical time range.

#### 7.6.3 statements\_summary\_evicted

The `tidb_stmt_summary_max_stmt_count` variable controls the maximum number of statements that the `statement_summary` table stores in memory. The `statement_summary` table uses the LRU algorithm. Once the number of SQL statements exceeds the `tidb_stmt_summary_max_stmt_count` value, the longest unused record is evicted from the table. The number of evicted SQL statements during each period is recorded in the `statements_summary_evicted` table.

The `statements_summary_evicted` table is updated only when a SQL record is evicted from the `statement_summary` table. The `statements_summary_evicted` only records the period during which the eviction occurs and the number of evicted SQL statements.

#### 7.6.4 The cluster tables for statement summary

The `statements_summary`, `statements_summary_history`, and `statements_summary_evicted` tables only show the statement summary of a single TiDB server. To query

the data of the entire cluster, you need to query the `cluster_statements_summary` → , `cluster_statements_summary_history`, or `cluster_statements_summary_evicted` tables.

`cluster_statements_summary` displays the `statements_summary` data of each TiDB server. `cluster_statements_summary_history` displays the `statements_summary_history` → data of each TiDB server. `cluster_statements_summary_evicted` displays the `statements_summary_evicted` data of each TiDB server. These tables use the `INSTANCE` field to represent the address of the TiDB server. The other fields are the same as those in `statements_summary`.

### 7.6.5 Parameter configuration

The following system variables are used to control the statement summary:

- `tidb_enable_stmt_summary`: Determines whether to enable the statement summary feature. 1 represents `enable`, and 0 means `disable`. The feature is enabled by default. The statistics in the system table are cleared if this feature is disabled. The statistics are re-calculated next time this feature is enabled. Tests have shown that enabling this feature has little impact on performance.
- `tidb_stmt_summary_refresh_interval`: The interval at which the `statements_summary` → table is refreshed. The time unit is second (s). The default value is 1800.
- `tidb_stmt_summary_history_size`: The size of each SQL statement category stored in the `statements_summary_history` table, which is also the maximum number of records in the `statement_summary_evicted` table. The default value is 24.
- `tidb_stmt_summary_max_stmt_count`: Limits the number of SQL statements that can be stored in statement summary tables. The default value is 3000. If the limit is exceeded, those SQL statements that recently remain unused are cleared. These cleared SQL statements are recorded in the `statement_summary_evicted` table.
- `tidb_stmt_summary_max_sql_length`: Specifies the longest display length of `DIGEST_TEXT` and `QUERY_SAMPLE_TEXT`. The default value is 4096.
- `tidb_stmt_summary_internal_query`: Determines whether to count the TiDB SQL statements. 1 means to count, and 0 means not to count. The default value is 0.

#### Note:

When a category of SQL statement needs to be removed because the `tidb_stmt_summary_max_stmt_count` limit is exceeded, TiDB removes the data of that SQL statement category of all time ranges from the `statement_summary_history` table. Therefore, even if the number of SQL statement categories in a certain time range does not reach the limit, the number of SQL statements stored in the `statement_summary_history` table is less than the actual number of SQL statements. If this situation occurs and affects performance, you are recommended to increase the value of

```
tidb_stmt_summary_max_stmt_count.
```

An example of the statement summary configuration is shown as follows:

```
set global tidb_enable_stmt_summary = true;
set global tidb_stmt_summary_refresh_interval = 1800;
set global tidb_stmt_summary_history_size = 24;
```

After the configuration above takes effect, every 30 minutes the `statements_summary` table is cleared. The `statements_summary_history` table stores data generated over the recent 12 hours.

The `statements_summary_evicted` table records the recent 24 periods during which SQL statements are evicted from the statement summary. The `statements_summary_evicted` table is updated every 30 minutes.

The system variables above have two scopes: global and session. These scopes work differently from other system variables:

- After setting the global variable, your setting applies to the whole cluster immediately.
- After setting the session variable, your setting applies to the current TiDB server immediately. This is useful when you debug on a single TiDB server instance.
- The session variable has a higher read priority. The global variable is read only when no session variable is set.
- If you set the session variable to a blank string, the global variable is re-read.

#### Note:

The `tidb_stmt_summary_history_size`, `tidb_stmt_summary_max_stmt_count`, and `tidb_stmt_summary_max_sql_length` configuration items affect memory usage. It is recommended that you adjust these configurations based on your needs. It is not recommended to set them too large values.

#### 7.6.5.1 Set a proper size for statement summary

After the system has run for a period of time, you can check the `statement_summary` table to see whether SQL eviction has occurred. For example:

```
select @@global.tidb_stmt_summary_max_stmt_count;
select count(*) from information_schema.statements_summary;
```

```
+-----+
| @@global.tidb_stmt_summary_max_stmt_count |
+-----+
| 3000 |
+-----+
1 row in set (0.001 sec)

+-----+
| count(*) |
+-----+
| 3001 |
+-----+
1 row in set (0.001 sec)
```

You can see that the `statements_summary` table is full of records. Then check the evicted data from the `statements_summary_evicted` table:

```
select * from information_schema.statements_summary_evicted;
```

```
+-----+-----+-----+
| BEGIN_TIME | END_TIME | EVICTED_COUNT |
+-----+-----+-----+
| 2020-01-02 16:30:00 | 2020-01-02 17:00:00 | 59 |
+-----+-----+-----+
| 2020-01-02 16:00:00 | 2020-01-02 16:30:00 | 45 |
+-----+-----+-----+
2 row in set (0.001 sec)
```

From the result above, you can see that a maximum of 59 SQL categories are evicted, which indicates that the proper size of the statement summary is 59 records.

### 7.6.6 Limitation

The statement summary tables have the following limitation:

All data of the statement summary tables above will be lost when the TiDB server is restarted. This is because statement summary tables are all memory tables, and the data is cached in memory instead of being persisted on storage.

### 7.6.7 Troubleshooting examples

This section provides two examples to show how to use the statement summary feature to troubleshoot SQL performance issues.

### 7.6.7.1 Could high SQL latency be caused by the server end?

In this example, the client shows slow performance with point queries on the `employee` table. You can perform a fuzzy search on SQL texts:

```
SELECT avg_latency, exec_count, query_sample_text
 FROM information_schema.statements_summary
 WHERE digest_text LIKE 'select * from employee%';
```

1ms and 0.3ms are considered within the normal range of `avg_latency`. Therefore, it can be concluded that the server end is not the cause. You can troubleshoot with the client or the network.

| avg_latency | exec_count | query_sample_text                                     |
|-------------|------------|-------------------------------------------------------|
| 1042040     | 2          | <code>select * from employee where name='eric'</code> |
| 345053      | 3          | <code>select * from employee where id=3100</code>     |

2 rows in set (0.00 sec)

### 7.6.7.2 Which categories of SQL statements consume the longest total time?

If the QPS decrease significantly from 10:00 to 10:30, you can find out the three categories of SQL statements with the longest time consumption from the history table:

```
SELECT sum_latency, avg_latency, exec_count, query_sample_text
 FROM information_schema.statements_summary_history
 WHERE summary_begin_time='2020-01-02 10:00:00'
 ORDER BY sum_latency DESC LIMIT 3;
```

The result shows that the following three categories of SQL statements consume the longest time in total, which need to be optimized with high priority.

| sum_latency | avg_latency | exec_count | query_sample_text                                     |
|-------------|-------------|------------|-------------------------------------------------------|
| 7855660     | 1122237     | 7          | <code>select avg(salary) from employee</code>         |
| 7241960     | 1448392     | 5          | <code>select * from employee join company</code>      |
| 2084081     | 1042040     | 2          | <code>select * from employee where name='eric'</code> |

```
+--+
→ -----+-----+-----+
→
3 rows in set (0.00 sec)
```

## 7.6.8 Fields description

### 7.6.8.1 statements\_summary fields description

The following are descriptions of fields in the `statements_summary` table.

Basic fields:

- `STMT_TYPE`: SQL statement type.
- `SCHEMA_NAME`: The current schema in which SQL statements of this category are executed.
- `DIGEST`: The digest of SQL statements of this category.
- `DIGEST_TEXT`: The normalized SQL statement.
- `QUERY_SAMPLE_TEXT`: The original SQL statements of the SQL category. Only one original statement is taken.
- `TABLE_NAMES`: All tables involved in SQL statements. If there is more than one table, each is separated by a comma.
- `INDEX_NAMES`: All SQL indexes used in SQL statements. If there is more than one index, each is separated by a comma.
- `SAMPLE_USER`: The users who execute SQL statements of this category. Only one user is taken.
- `PLAN_DIGEST`: The digest of the execution plan.
- `PLAN`: The original execution plan. If there are multiple statements, the plan of only one statement is taken.
- `PLAN_CACHE_HITS`: The total number of times that SQL statements of this category hit the plan cache.
- `PLAN_IN_CACHE`: Indicates whether the previous execution of SQL statements of this category hit the plan cache.

Fields related to execution time:

- `SUMMARY_BEGIN_TIME`: The beginning time of the current summary period.
- `SUMMARY_END_TIME`: The ending time of the current summary period.
- `FIRST_SEEN`: The time when SQL statements of this category are seen for the first time.
- `LAST_SEEN`: The time when SQL statements of this category are seen for the last time.

Fields related to TiDB server:

- **EXEC\_COUNT**: Total execution times of SQL statements of this category.
- **SUM\_ERRORS**: The sum of errors occurred during execution.
- **SUM\_WARNINGS**: The sum of warnings occurred during execution.
- **SUM\_LATENCY**: The total execution latency of SQL statements of this category.
- **MAX\_LATENCY**: The maximum execution latency of SQL statements of this category.
- **MIN\_LATENCY**: The minimum execution latency of SQL statements of this category.
- **AVG\_LATENCY**: The average execution latency of SQL statements of this category.
- **AVG\_PARSE\_LATENCY**: The average latency of the parser.
- **MAX\_PARSE\_LATENCY**: The maximum latency of the parser.
- **AVG\_COMPILE\_LATENCY**: The average latency of the compiler.
- **MAX\_COMPILE\_LATENCY**: The maximum latency of the compiler.
- **AVG\_MEM**: The average memory (byte) used.
- **MAX\_MEM**: The maximum memory (byte) used.
- **AVG\_DISK**: The average disk space (byte) used.
- **MAX\_DISK**: The maximum disk space (byte) used.

Fields related to TiKV Coprocessor task:

- **SUM\_COP\_TASK\_NUM**: The total number of Coprocessor requests sent.
- **MAX\_COP\_PROCESS\_TIME**: The maximum execution time of Coprocessor tasks.
- **MAX\_COP\_PROCESS\_ADDRESS**: The address of the Coprocessor task with the maximum execution time.
- **MAX\_COP\_WAIT\_TIME**: The maximum waiting time of Coprocessor tasks.
- **MAX\_COP\_WAIT\_ADDRESS**: The address of the Coprocessor task with the maximum waiting time.
- **AVG\_PROCESS\_TIME**: The average processing time of SQL statements in TiKV.
- **MAX\_PROCESS\_TIME**: The maximum processing time of SQL statements in TiKV.
- **AVG\_WAIT\_TIME**: The average waiting time of SQL statements in TiKV.
- **MAX\_WAIT\_TIME**: The maximum waiting time of SQL statements in TiKV.
- **AVG\_BACKOFF\_TIME**: The average waiting time before retry when a SQL statement encounters an error that requires a retry.
- **MAX\_BACKOFF\_TIME**: The maximum waiting time before retry when a SQL statement encounters an error that requires a retry.
- **AVG\_TOTAL\_KEYS**: The average number of keys that Coprocessor has scanned.
- **MAX\_TOTAL\_KEYS**: The maximum number of keys that Coprocessor has scanned.
- **AVG\_PROCESSED\_KEYS**: The average number of keys that Coprocessor has processed. Compared with `avg_total_keys`, `avg_processed_keys` does not include the old versions of MVCC. A great difference between `avg_total_keys` and `avg_processed_keys` indicates that many old versions exist.
- **MAX\_PROCESSED\_KEYS**: The maximum number of keys that Coprocessor has processed.

Transaction-related fields:

- **AVG\_Prewrite\_Time**: The average time of the prewrite phase.

- `MAX_PREWRITE_TIME`: The longest time of the prewrite phase.
- `AVG_COMMIT_TIME`: The average time of the commit phase.
- `MAX_COMMIT_TIME`: The longest time of the commit phase.
- `AVG_GET_COMMIT_TS_TIME`: The average time of getting `commit_ts`.
- `MAX_GET_COMMIT_TS_TIME`: The longest time of getting `commit_ts`.
- `AVG_COMMIT_BACKOFF_TIME`: The average waiting time before retry when a SQL statement encounters an error that requires a retry during the commit phase.
- `MAX_COMMIT_BACKOFF_TIME`: The maximum waiting time before retry when a SQL statement encounters an error that requires a retry during the commit phase.
- `AVG_RESOLVE_LOCK_TIME`: The average time for resolving lock conflicts occurred between transactions.
- `MAX_RESOLVE_LOCK_TIME`: The longest time for resolving lock conflicts occurred between transactions.
- `AVG_LOCAL_LATCH_WAIT_TIME`: The average waiting time of the local transaction.
- `MAX_LOCAL_LATCH_WAIT_TIME`: The maximum waiting time of the local transaction.
- `AVG_WRITE_KEYS`: The average count of written keys.
- `MAX_WRITE_KEYS`: The maximum count of written keys.
- `AVG_WRITE_SIZE`: The average amount of written data (in byte).
- `MAX_WRITE_SIZE`: The maximum amount of written data (in byte).
- `AVG_PREWRITE_REGIONS`: The average number of Regions involved in the prewrite phase.
- `MAX_PREWRITE_REGIONS`: The maximum number of Regions during the prewrite phase.
- `AVG_TXN_RETRY`: The average number of transaction retries.
- `MAX_TXN_RETRY`: The maximum number of transaction retries.
- `SUM_BACKOFF_TIMES`: The sum of retries when SQL statements of this category encounter errors that require a retry.
- `BACKOFF_TYPES`: All types of errors that require retries and the number of retries for each type. The format of the field is `type:number`. If there is more than one error type, each is separated by a comma, like `txnLock:2,pdRPC:1`.
- `AVG_AFFECTED_ROWS`: The average number of rows affected.
- `PREV_SAMPLE_TEXT`: When the current SQL statement is `COMMIT`, `PREV_SAMPLE_TEXT` is the previous statement to `COMMIT`. In this case, SQL statements are grouped by the digest and `prev_sample_text`. This means that `COMMIT` statements with different `prev_sample_text` are grouped to different rows. When the current SQL statement is not `COMMIT`, the `PREV_SAMPLE_TEXT` field is an empty string.

#### 7.6.8.2 `statements_summary_evicted` fields description

- `BEGIN_TIME`: Records the starting time.
- `END_TIME`: Records the ending time.
- `EVICTED_COUNT`: The number of SQL categories that are evicted during the record period.

## 7.7 Troubleshoot Hotspot Issues

This document describes how to locate and resolve the problem of read and write hotspots.

As a distributed database, TiDB has a load balancing mechanism to distribute the application loads as evenly as possible to different computing or storage nodes, to make better use of server resources. However, in certain scenarios, some application loads cannot be well distributed, which can affect the performance and form a single point of high load, also known as a hotspot.

TiDB provides a complete solution to troubleshooting, resolving or avoiding hotspots. By balancing load hotspots, overall performance can be improved, including improving QPS and reducing latency.

### 7.7.1 Common hotspots

This section describes TiDB encoding rules, table hotspots, and index hotspots.

#### 7.7.1.1 TiDB encoding rules

TiDB assigns a TableID to each table, an IndexID to each index, and a RowID to each row. By default, if the table uses an integer primary key, the value of the primary key is treated as the RowID. Among these IDs, TableID is unique in the entire cluster, while IndexID and RowID are unique in the table. The type of all these IDs is int64.

Each row of data is encoded as a key-value pair according to the following rule:

```
Key: tablePrefix{tableID}_recordPrefixSep{rowID}
Value: [col1, col2, col3, col4]
```

The `tablePrefix` and `recordPrefixSep` of the key are specific string constants, used to distinguish from other data in the KV space.

For Index data, the key-value pair is encoded according to the following rule:

```
Key: tablePrefix{tableID}_indexPrefixSep{indexID}_indexedColumnsValue
Value: rowID
```

Index data has two types: the unique index and the non-unique index.

- For unique indexes, you can follow the coding rules above.
- For non-unique indexes, a unique key cannot be constructed through this encoding, because the `tablePrefix{tableID}_indexPrefixSep{indexID}` of the same index is the same and the `ColumnsValue` of multiple rows might be the same. The encoding rule for non-unique indexes is as follows:

```
Key: tablePrefix{tableID}_indexPrefixSep{indexID}
 ↳ _indexedColumnsValue_rowID
Value: null
```

### 7.7.1.2 Table hotspots

According to TiDB coding rules, the data of the same table is in a range prefixed by the beginning of the TableID, and the data is arranged in the order of RowID values. When RowID values are incremented during table inserting, the inserted line can only be appended to the end. The Region will split after it reaches a certain size, and then it still can only be appended to the end of the range. The `INSERT` operation can only be executed on one Region, forming a hotspot.

The common auto-increment primary key is sequentially increasing. When the primary key is of the integer type, the value of the primary key is used as the RowID by default. At this time, the RowID is sequentially increasing, and a write hotspot of the table forms when a large number of `INSERT` operations exist.

Meanwhile, the RowID in TiDB is also sequentially auto-incremental by default. When the primary key is not an integer type, you might also encounter the problem of write hotspots.

### 7.7.1.3 Index hotspots

Index hotspots are similar to table hotspots. Common index hotspots appear in fields that are monotonously increasing in time order, or `INSERT` scenarios with a large number of repeated values.

## 7.7.2 Identify hotspot issues

Performance problems are not necessarily caused by hotspots and might be caused by multiple factors. Before troubleshooting issues, confirm whether it is related to hotspots.

- To judge write hotspots, open **Hot Write** in the **TiKV-Trouble-Shooting** monitoring panel to check whether the Raftstore CPU metric value of any TiKV node is significantly higher than that of other nodes.
- To judge read hotspots, open **Thread\_CPU** in the **TiKV-Details** monitoring panel to check whether the coprocessor CPU metric value of any TiKV node is particularly high.

### 7.7.2.1 Use TiDB Dashboard to locate hotspot tables

The **Key Visualizer** feature in **TiDB Dashboard** helps users narrow down hotspot troubleshooting scope to the table level. The following is an example of the thermal diagram

shown by **Key Visualizer**. The horizontal axis of the graph is time, and the vertical axis are various tables and indexes. The brighter the color, the greater the load. You can switch the read or write flow in the toolbar.

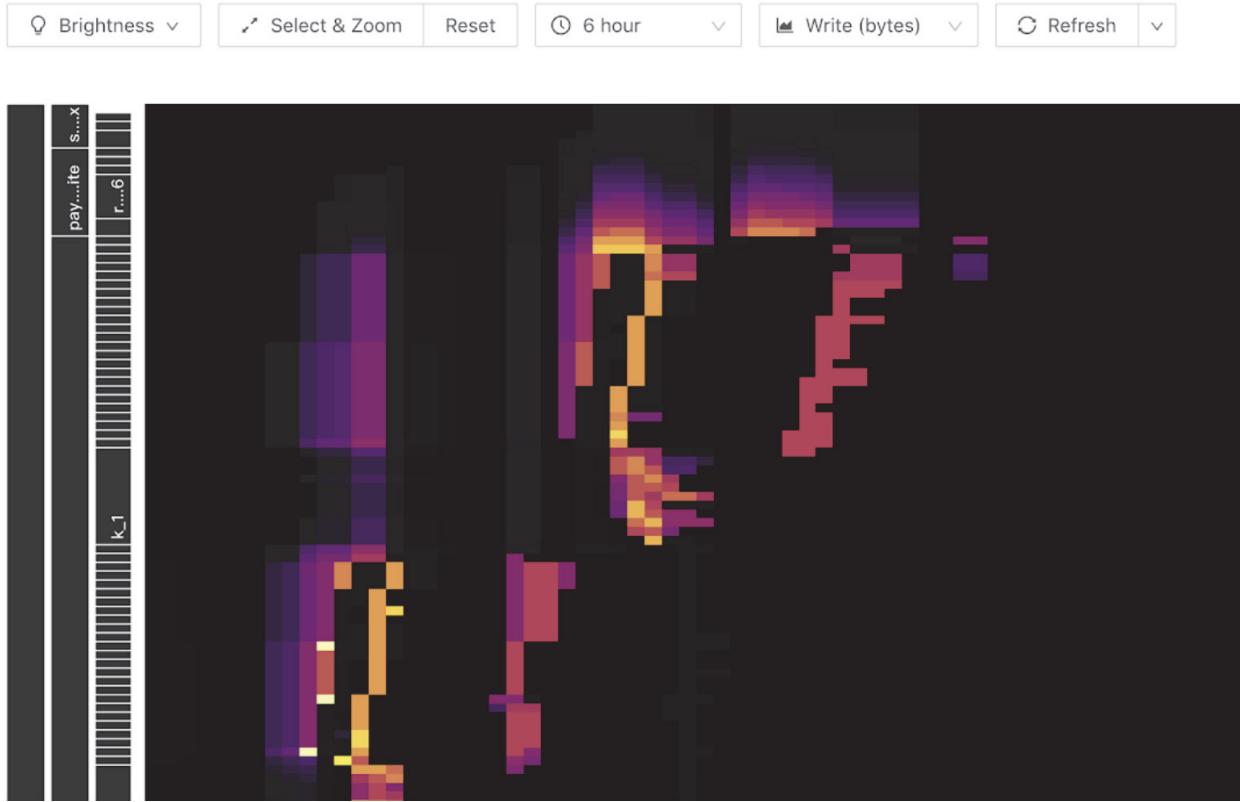


Figure 48: Dashboard Example 1

The following bright diagonal lines (oblique upward or downward) can appear in the write flow graph. Because the write only appears at the end, as the number of table Regions becomes larger, it appears as a ladder. This indicates that a write hotspot shows in this table:

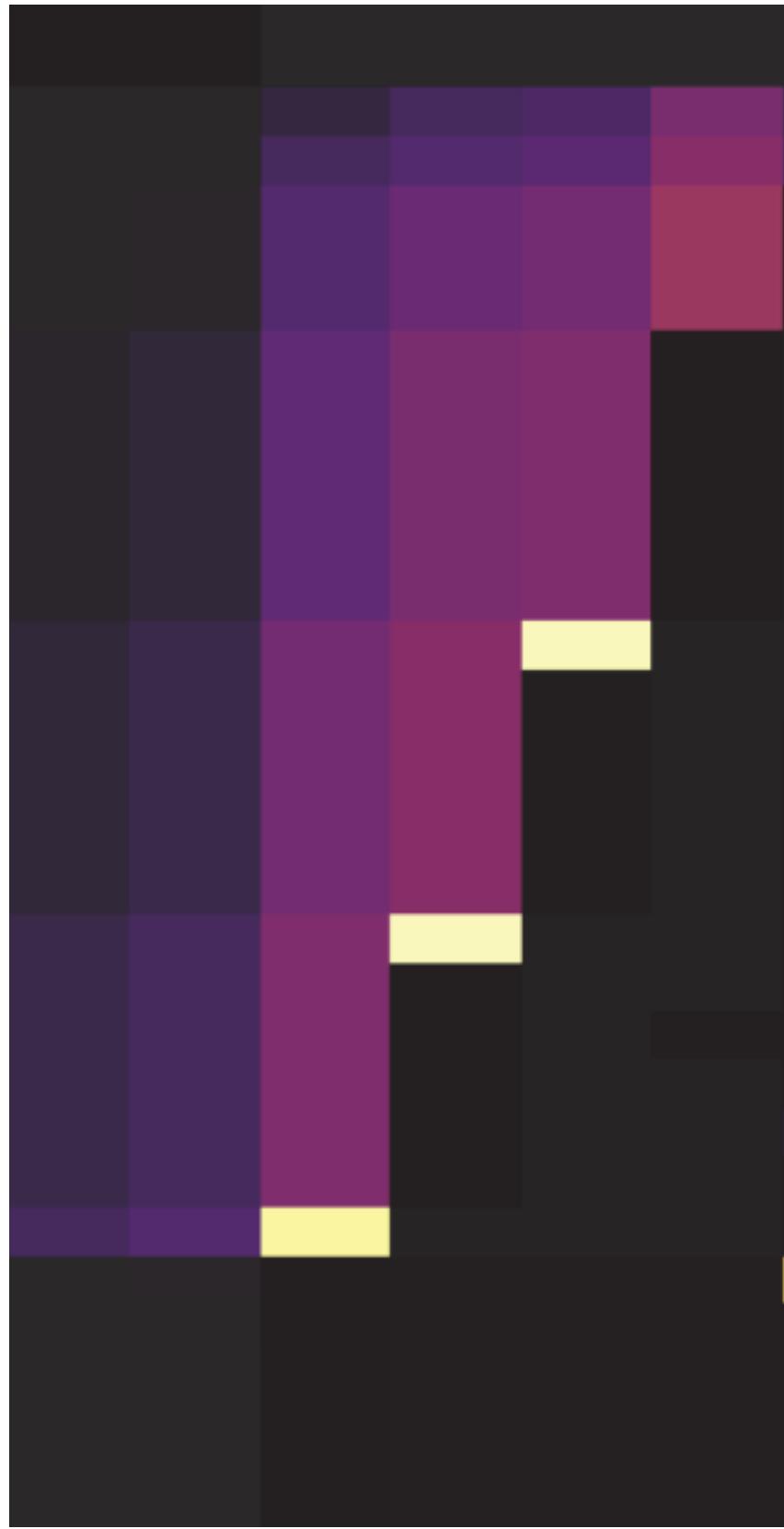


Figure 49: Dashboard Example 2

For read hotspots, a bright horizontal line is generally shown in the thermal diagram. Usually these are caused by small tables with a large number of accesses, shown as follows:

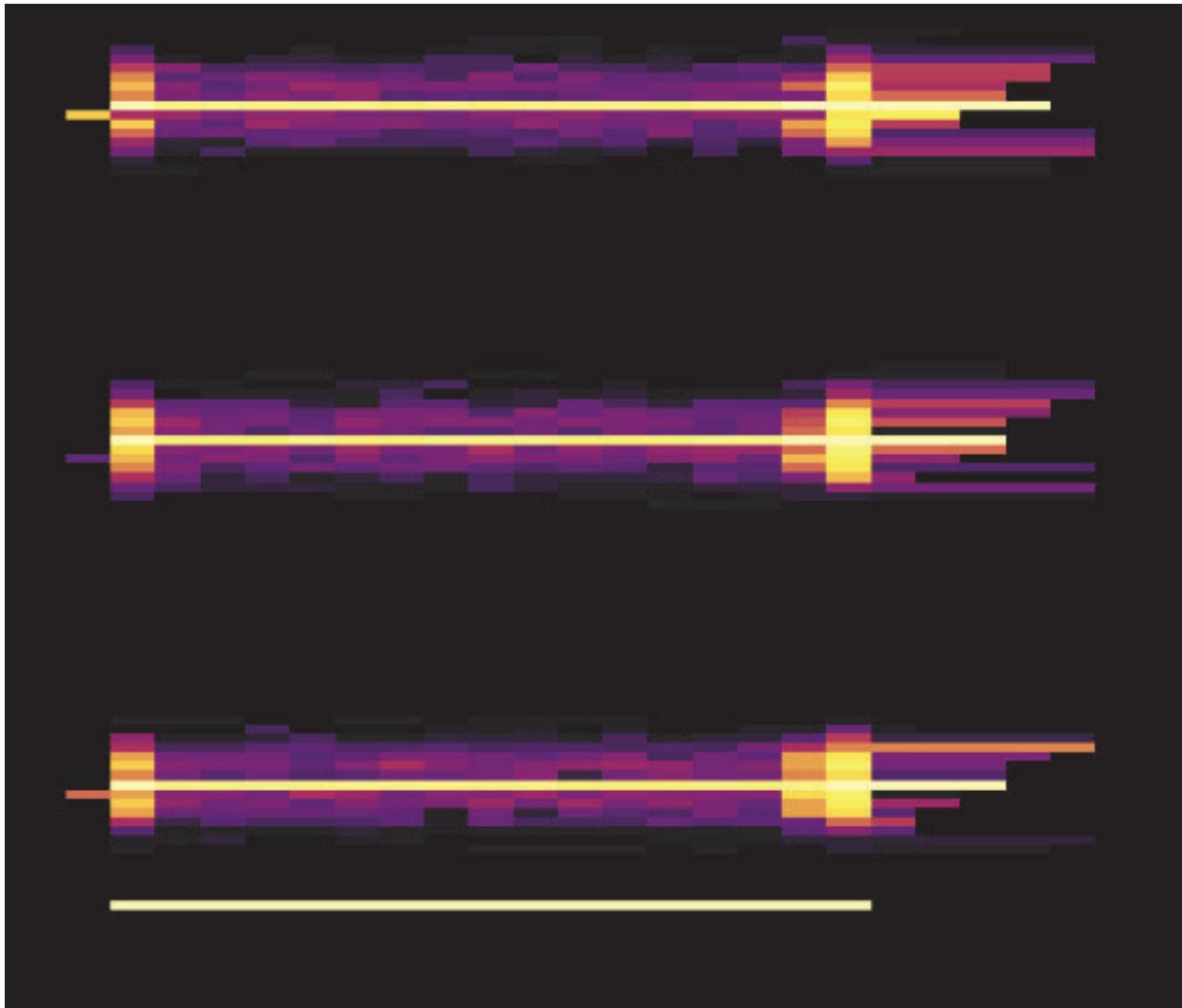


Figure 50: Dashboard Example 3

Hover over the bright block, you can see what table or index has a heavy load. For example:

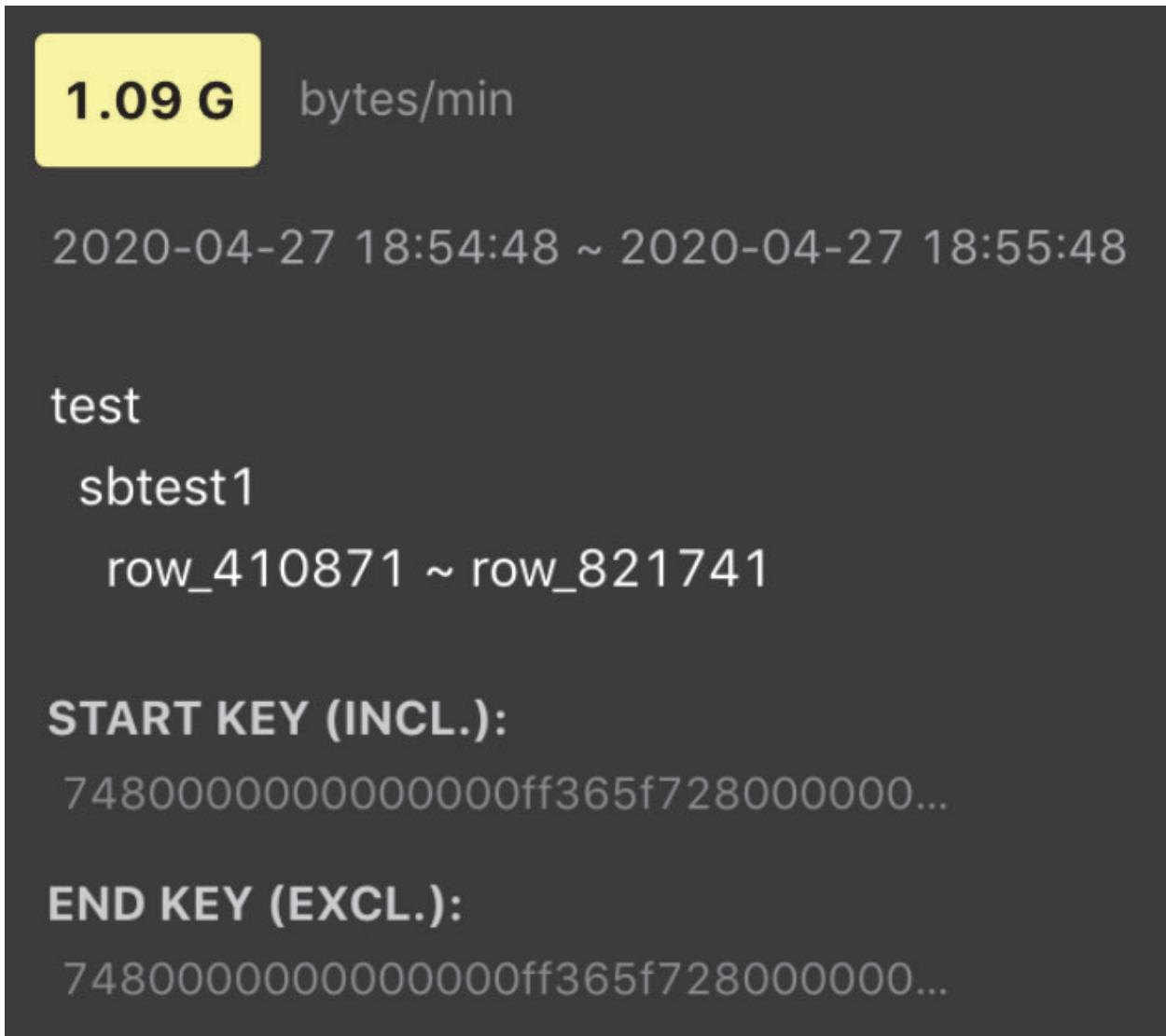


Figure 51: Dashboard Example 4

### 7.7.3 Use SHARD\_ROW\_ID\_BITS to process hotspots

For a non-integer primary key or a table without a primary key or a joint primary key, TiDB uses an implicit auto-increment RowID. When a large number of `INSERT` operations exist, the data is written into a single Region, resulting in a write hotspot.

By setting `SHARD_ROW_ID_BITS`, RowID are scattered and written into multiple Regions, which can alleviates the write hotspot issue. However, if you set `SHARD_ROW_ID_BITS` to an over large value, the number of RPC requests will be enlarged, increasing CPU and network overhead.

```
SHARD_ROW_ID_BITS = 4 # Represents 16 shards.
SHARD_ROW_ID_BITS = 6 # Represents 64 shards.
```

```
SHARD_ROW_ID_BITS = 0 # Represents the default 1 shard.
```

Statement example:

```
CREATE TABLE: CREATE TABLE t (c int) SHARD_ROW_ID_BITS = 4;
ALTER TABLE: ALTER TABLE t SHARD_ROW_ID_BITS = 4;
```

The value of `SHARD_ROW_ID_BITS` can be dynamically modified. The modified value only takes effect for newly written data.

For the table with a primary key of the `CLUSTERED` type, TiDB uses the primary key of the table as the RowID. At this time, the `SHARD_ROW_ID_BITS` option cannot be used because it changes the RowID generation rules. For the table with the primary key of the `NONCLUSTERED` type, TiDB uses an automatically allocated 64-bit integer as the RowID. In this case, you can use the `SHARD_ROW_ID_BITS` feature. For more details about the primary key of the `CLUSTERED` type, refer to [clustered index](#).

The following two load diagrams shows the case where two tables without primary keys use `SHARD_ROW_ID_BITS` to scatter hotspots. The first diagram shows the situation before scattering hotspots, while the second one shows the situation after scattering hotspots.

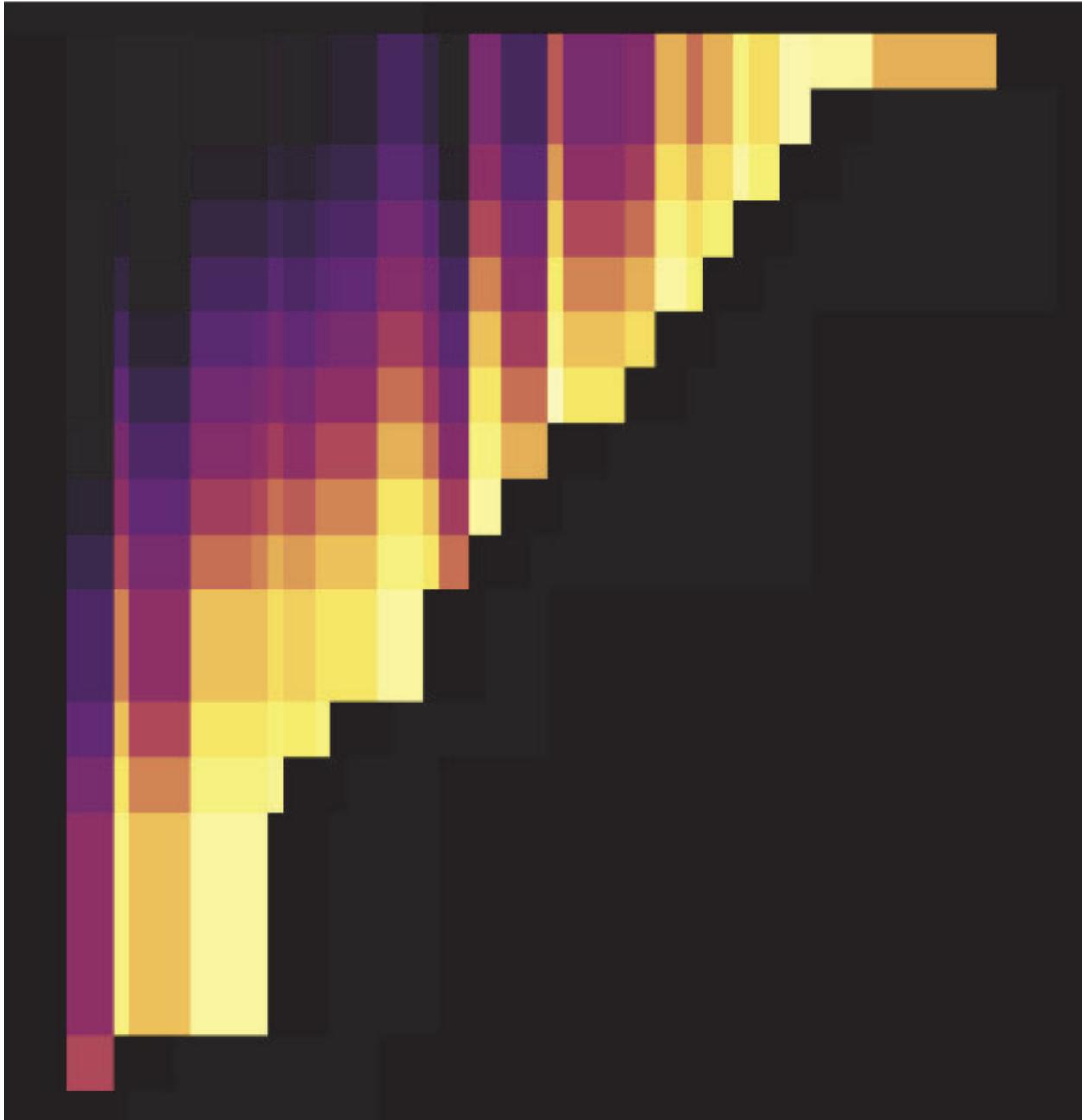


Figure 52: Dashboard Example 5

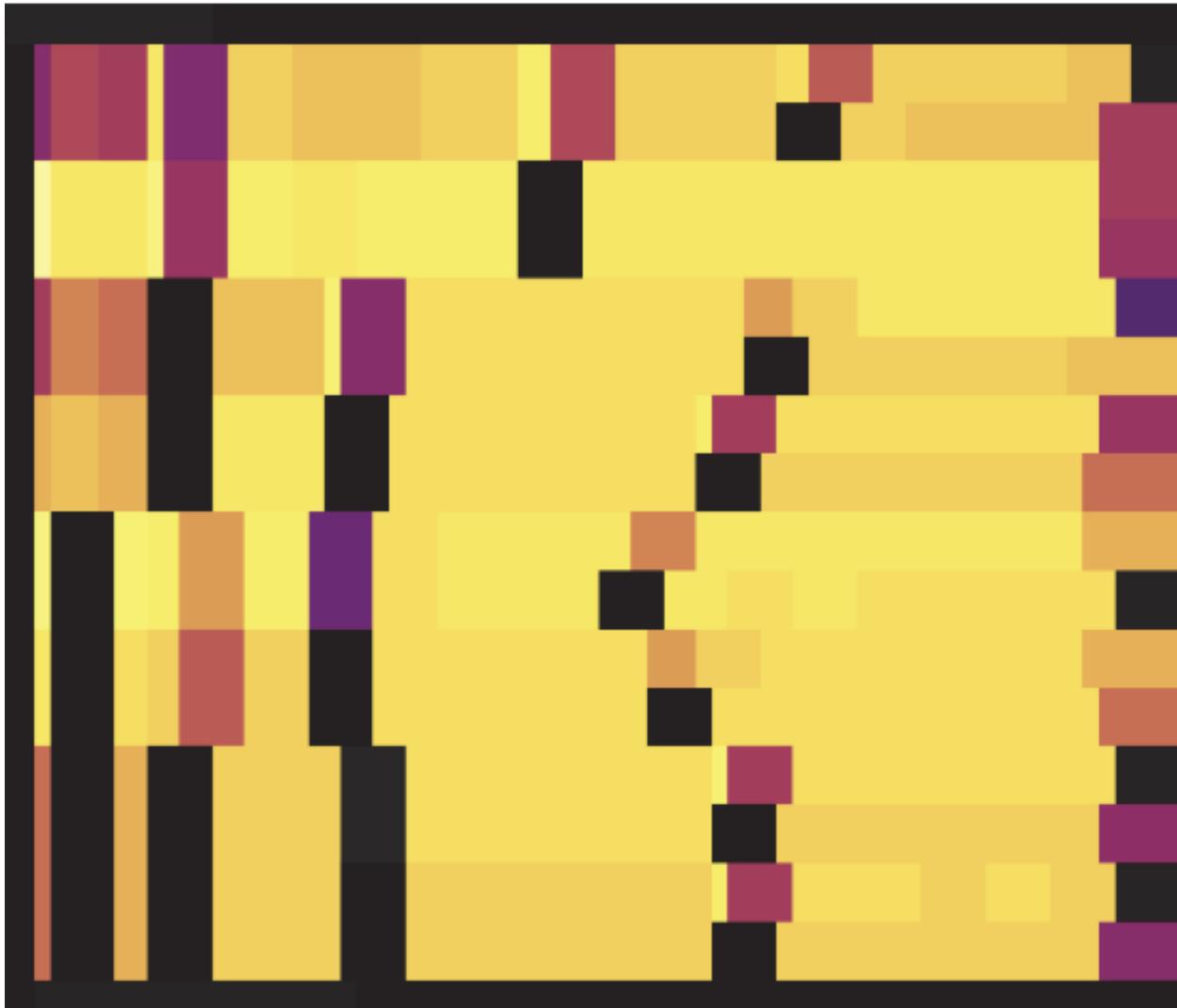


Figure 53: Dashboard Example 6

As shown in the load diagrams above, before setting `SHARD_ROW_ID_BITS`, load hotspots are concentrated on a single Region. After setting `SHARD_ROW_ID_BITS`, load hotspots become scattered.

#### 7.7.4 Handle auto-increment primary key hotspot tables using `AUTO_RANDOM`

To resolve the write hotspots brought by auto-increment primary keys, use `AUTO_RANDOM` to handle hotspot tables that have auto-increment primary keys.

If this feature is enabled, TiDB generates randomly distributed and non-repeated (before the space is used up) primary keys to achieve the purpose of scattering write hotspots.

Note that the primary keys generated by TiDB are no longer auto-increment primary keys and you can use `LAST_INSERT_ID()` to obtain the primary key value assigned last time.

To use this feature, modify AUTO\_INCREMENT to AUTO\_RANDOM in the CREATE TABLE statement. This feature is suitable for non-application scenarios where the primary keys only need to guarantee uniqueness.

For example:

```
CREATE TABLE t (a BIGINT PRIMARY KEY AUTO_RANDOM, b varchar(255));
INSERT INTO t (b) VALUES ("foo");
SELECT * FROM t;
```

| a          | b   |
|------------|-----|
| 1073741825 | foo |

```
SELECT LAST_INSERT_ID();
```

| LAST_INSERT_ID() |
|------------------|
| 1073741825       |

The following two load diagrams shows the situations both before and after modifying AUTO\_INCREMENT to AUTO\_RANDOM to scatter hotspots. The first one uses AUTO\_INCREMENT, while the second one uses AUTO\_RANDOM.

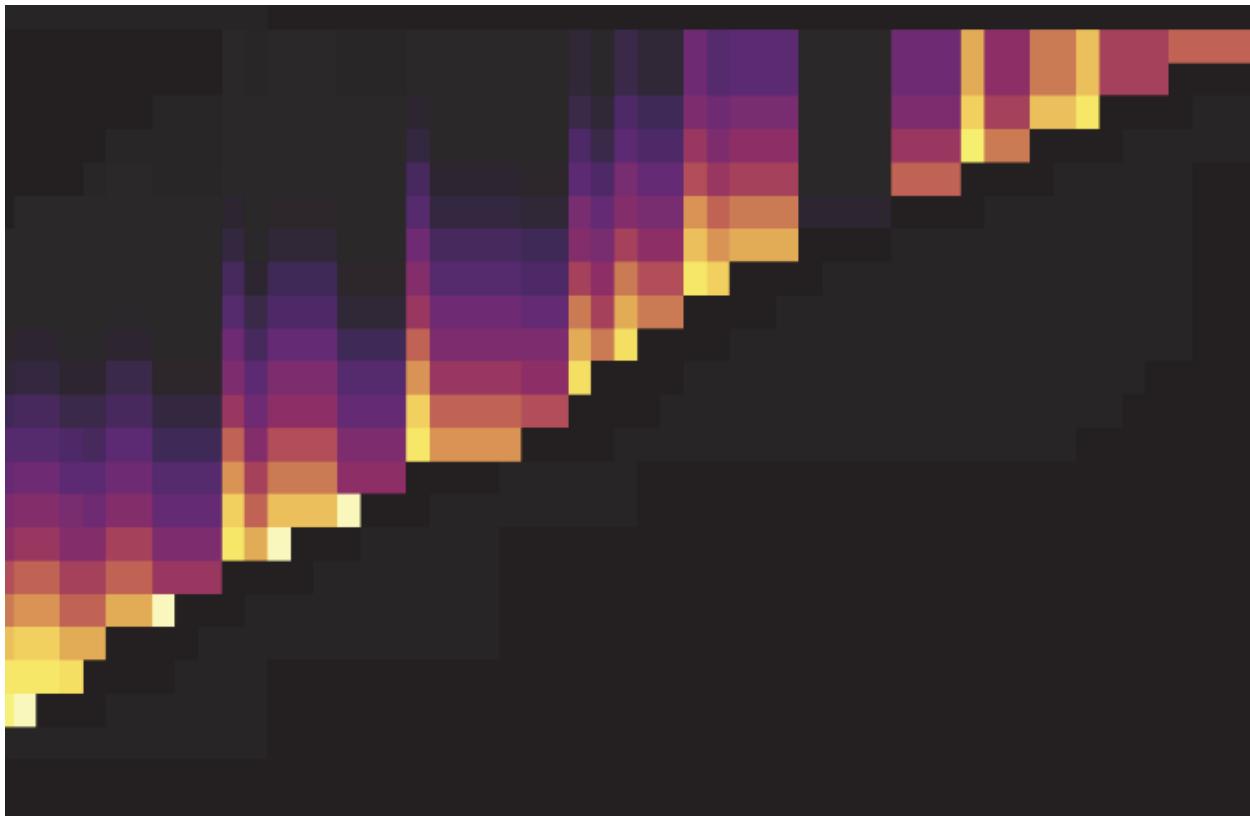


Figure 54: Dashboard Example 7

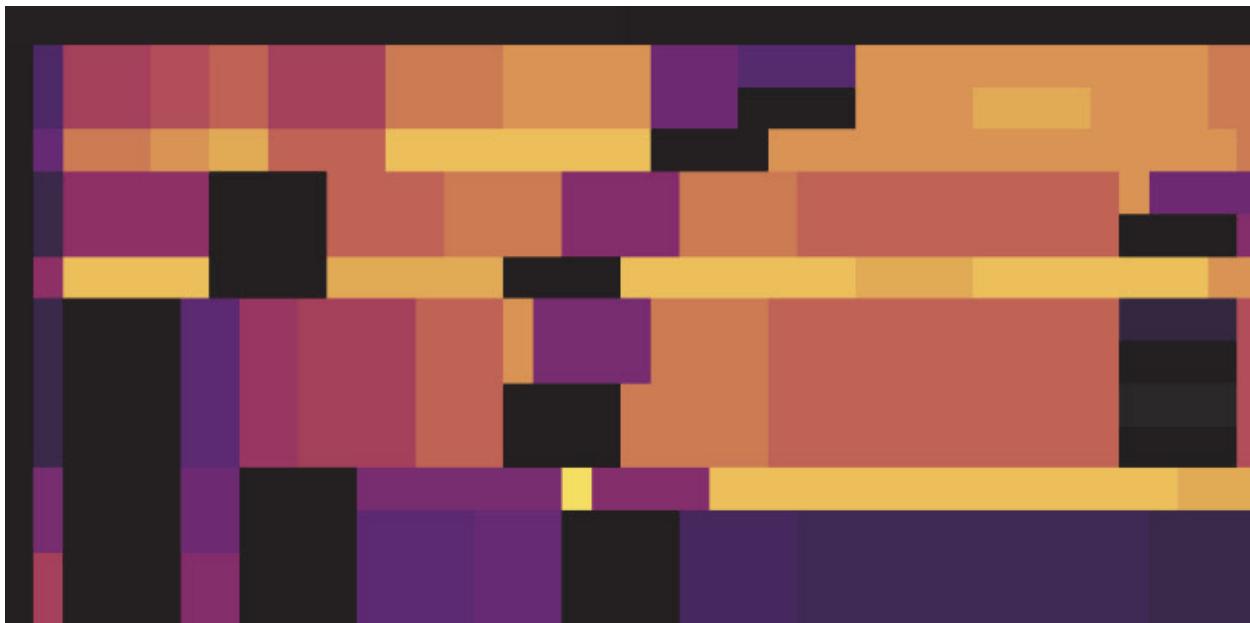


Figure 55: Dashboard Example 8

As shown in the load diagrams above, using `AUTO_RANDOM` to replace `AUTO_INCREMENT` can well scatter hotspots.

For more details, see [AUTO\\_RANDOM](#).

### 7.7.5 Optimization of small table hotspots

The Coprocessor Cache feature of TiDB supports pushing down computing result caches. After this feature is enabled, TiDB caches the computing results that will be pushed down to TiKV. This feature works well for read hotspots of small tables.

For more details, see [Coprocessor Cache](#).

**See also:**

- [Highly Concurrent Write Best Practices](#)
- [Split Region](#)

## 7.8 Troubleshoot Increased Read and Write Latency

This document introduces the possible causes of read and write latency and jitters, and how to troubleshoot these issues.

### 7.8.1 Common causes

#### 7.8.1.1 Incorrect TiDB execution plan

The execution plan of queries is unstable and might select the incorrect index, which causes higher latency.

##### 7.8.1.1.1 Phenomenon

- If the query execution plan is output in the slow log, you can directly view the plan. Execute the `select tidb_decode_plan('xxx...')` statement to parse the detailed execution plan.
- The number of scanned keys in the monitor abnormally increases; in the slow log, the number of `Scan Keys` are large.
- The SQL execution duration in TiDB is greatly different than that in other databases such as MySQL. You can compare the execution plan of other databases (for example, whether `Join Order` is different).

##### 7.8.1.1.2 Possible reason

The statistics is inaccurate.

### 7.8.1.1.3 Troubleshooting methods

- Update the statistical information
  - Execute `analyze table` manually and execute `analyze` periodically with the `cronjob` command to keep the statistics accurate.
  - Execute `auto analyze` automatically. Lower the threshold value of `analyze → ratio`, increase the frequency of information collection, and set the start and end time of the execution. See the following examples:
 

```
* set global tidb_auto_analyze_ratio=0.2;
* set global tidb_auto_analyze_start_time='00:00 +0800';
* set global tidb_auto_analyze_end_time='06:00 +0800';
```
- Bind the execution plan
  - Modify the application SQL statements and execute `use index` to consistently use the index of the column.
  - In 3.0 versions, you do not need to modify the application SQL statements. Use `create global binding` to create the binding SQL statement of `force index`.
  - In 4.0 versions, **SQL Plan Management** is supported, which avoids the performance decrease caused by unstable execution plans.

## 7.8.1.2 PD anomalies

### 7.8.1.2.1 Phenomenon

There is an abnormal increase of the `wait duration` metric for the PD TSO. This metric represents the duration of waiting for PD to return requests.

### 7.8.1.2.2 Possible reasons

- Disk issue. The disk where the PD node is located has full I/O load. Investigate whether PD is deployed with other components with high I/O demand and the health of the disk. You can verify the cause by viewing the monitor metrics in **Grafana -> disk performance -> latency/load**. You can also use the FIO tool to run a check on the disk if necessary.
- Network issues between PD peers. The PD log shows `lost the TCP streaming → connection`. You need to check whether there is a problem with the network between PD nodes and verify the cause by viewing `round trip` in the monitor **Grafana -> PD -> etcd**.
- High server load. The log shows `server is likely overloaded`.
- PD cannot elect a Leader: The PD log shows `lease is not expired`. This issue has been fixed in v3.0.x and v2.1.19.

- The leader election is slow. The Region loading duration is long. You can check this issue by running `grep "regions cost"` in the PD log. If the result is in seconds, such as `load 460927 regions cost 11.77099s`, it means the Region loading is slow. You can enable the `region storage` feature in v3.0 by setting `use-region-storage` to `true`, which significantly reduce the Region loading duration.
- The network issue between TiDB and PD. Check whether the network from TiDB to PD Leader is running normally by accessing the monitor **Grafana** -> **black-box\_exporter** -> **ping latency**.
- PD reports the FATAL error, and the log shows `range failed to find revision ↪ pair`. This issue has been fixed in v3.0.8 ([#2040](#)).
- When the `/api/v1/regions` interface is used, too many Regions might cause PD OOM. This issue has been fixed in v3.0.8 ([#1986](#)).
- PD OOM during the rolling upgrade. The size of gRPC messages is not limited, and the monitor shows that TCP InSegs is relatively large. This issue has been fixed in v3.0.6 ([#1952](#)).
- PD panics. [Report a bug](#).
- Other causes. Get goroutine by running `curl http://127.0.0.1:2379/debug/pprof ↪ /goroutine?debug=2` and [report a bug](#).

### 7.8.1.3 TiKV anomalies

#### 7.8.1.3.1 Phenomenon

The KV Cmd Duration metric in the monitor increases abnormally. This metric represents the duration between the time that TiDB sends a request to TiKV and the time that TiDB receives the response.

#### 7.8.1.3.2 Possible reasons

- Check the gRPC duration metric. This metric represents the total duration of a gRPC request in TiKV. You can find out the potential network issue by comparing gRPC → duration of TiKV and KV duration of TiDB. For example, the gRPC duration is short but the KV duration of TiDB is long, which indicates that the network latency between TiDB and TiKV might be high, or that the NIC bandwidth between TiDB and TiKV is fully occupied.
- Re-election because TiKV is restarted.
  - After TiKV panics, it is pulled up by `systemd` and runs normally. You can check whether panic has occurred by viewing the TiKV log. Because this issue is unexpected, [report a bug](#) if it happens.

- TiKV is stopped or killed by a third party and then pulled up by `systemd`. Check the cause by viewing `dmesg` and the TiKV log.
- TiKV is OOM, which causes restart.
- TiKV is hung because of dynamically adjusting THP (Transparent Hugepage).
- Check monitor: TiKV RocksDB encounters write stall and thus results in re-election. You can check if the monitor **Grafana -> TiKV-details -> errors** shows `server ↗ is busy`.
- Re-election because of network isolation.
- If the `block-cache` configuration is too large, it might cause TiKV OOM. To verify the cause of the problem, check the `block cache size` of RocksDB by selecting the corresponding instance in the monitor **Grafana -> TiKV-details**. Meanwhile, check whether the `[storage.block-cache] capacity = # "1GB"` parameter is set properly. By default, TiKV's `block-cache` is set to 45% of the total memory of the machine. You need to explicitly specify this parameter when you deploy TiKV in the container, because TiKV obtains the memory of the physical machine, which might exceed the memory limit of the container.
- Coprocessor receives many large queries and returns a large volume of data. gRPC fails to send data as quickly as the coprocessor returns data, which results in OOM. To verify the cause, you can check whether `response size` exceeds the `network outbound traffic` by viewing the monitor **Grafana -> TiKV-details -> coprocessor overview**.

#### 7.8.1.4 Bottleneck of a single TiKV thread

There are some single threads in TiKV that might become the bottleneck.

- Too many Regions in a TiKV instance causes a single gRPC thread to be the bottleneck (Check the **Grafana -> TiKV-details -> Thread CPU/gRPC CPU Per Thread** metric). In v3.x or later versions, you can enable `Hibernate Region` to resolve the issue.
- For versions earlier than v3.0, when the raftstore thread or the apply thread becomes the bottleneck (**Grafana -> TiKV-details -> Thread CPU/raft store CPU** and **Async apply CPU** metrics exceed 80%), you can scale out TiKV (v2.x) instances or upgrade to v3.x with multi-threading.

#### 7.8.1.5 CPU load increases

##### 7.8.1.5.1 Phenomenon

The usage of CPU resources becomes the bottleneck.

### 7.8.1.5.2 Possible reasons

- Hotspot issue
- High overall load. Check the slow queries and expensive queries of TiDB. Optimize the executing queries by adding indexes or executing queries in batches. Another solution is to scale out the cluster.

## 7.8.2 Other causes

### 7.8.2.1 Cluster maintenance

Most of each online cluster has three or five nodes. If the machine to be maintained has the PD component, you need to determine whether the node is the leader or the follower. Disabling a follower has no impact on the cluster operation. Before disabling a leader, you need to switch the leadership. During the leadership change, performance jitter of about 3 seconds will occur.

### 7.8.2.2 Minority of replicas are offline

By default, each TiDB cluster has three replicas, so each Region has three replicas in the cluster. These Regions elect the leader and replicate data through the Raft protocol. The Raft protocol ensures that TiDB can still provide services without data loss even when the nodes (that are fewer than half of replicas) fail or are isolated. For the cluster with three replicas, the failure of one node might cause performance jitter but the usability and correctness in theory are not affected.

### 7.8.2.3 New indexes

Creating indexes consumes a huge amount of resources when TiDB scans tables and backfills indexes. Index creation might even conflict with the frequently updated fields, which affects the application. Creating indexes on a large table often takes a long time, so you must try to balance the index creation time and the cluster performance (for example, creating indexes at the off-peak time).

#### Parameter adjustment:

Currently, you can use `tidb_ddl_reorg_worker_cnt` and `tidb_ddl_reorg_batch_size` to dynamically adjust the speed of index creation. Usually, the smaller the values, the smaller the impact on the system, with longer execution time though.

In general cases, you can first keep their default values (4 and 256), observe the resource usage and response speed of the cluster, and then increase the value of `tidb_ddl_reorg_worker_cnt` to increase the concurrency. If no obvious jitter is observed in the monitor, increase the value of `tidb_ddl_reorg_batch_size`. If the columns involved in the index creation are frequently updated, the many resulting conflicts will cause the index creation to fail and be retried.

In addition, you can also set the value of `tidb_ddl_reorg_priority` to `PRIORITY_HIGH` to prioritize the index creation and speed up the process. But in the general OLTP system, it is recommended to keep its default value.

#### 7.8.2.4 High GC pressure

The transaction of TiDB adopts the Multi-Version Concurrency Control (MVCC) mechanism. When the newly written data overwrites the old data, the old data is not replaced, and both versions of data are stored. Timestamps are used to mark different versions. The task of GC is to clear the obsolete data.

- In the phase of Resolve Locks, a large amount of `scan_lock` requests are created in TiKV, which can be observed in the gRPC-related metrics. These `scan_lock` requests call all Regions.
- In the phase of Delete Ranges, a few (or no) `unsafe_destroy_range` requests are sent to TiKV, which can be observed in the gRPC-related metrics and the **GC tasks** panel.
- In the phase of Do GC, each TiKV by default scans the leader Regions on the machine and performs GC to each leader, which can be observed in the **GC tasks** panel.

### 7.9 Use PLAN REPLAYER to Save and Restore the On-Site Information of a Cluster

When you locate and troubleshoot the issues of a TiDB cluster, you often need to provide information on the system and the execution plan. To help you get the information and troubleshoot cluster issues in a more convenient and efficient way, the `PLAN REPLAY ↴` command is introduced in TiDB v5.3.0. This command enables you to easily save and restore the on-site information of a cluster, improves the efficiency of troubleshooting, and helps you more easily archive the issue for management.

The features of `PLAN REPLAYER` are as follows:

- Exports the information of a TiDB cluster at an on-site troubleshooting to a ZIP-formatted file for storage.
- Imports into a cluster the ZIP-formatted file exported from another TiDB cluster. This file contains the information of the latter TiDB cluster at an on-site troubleshooting.

#### 7.9.1 Use PLAN REPLAER to export cluster information

You can use `PLAN REPLAYER` to save the on-site information of a TiDB cluster. The export interface is as follows:

```
PLAN REPLAYER DUMP EXPLAIN [ANALYZE] sql-statement;
```

Based on `sql-statement`, TiDB sorts out and exports the following on-site information:

- TiDB version
- TiDB configuration
- TiDB session variables
- TiDB SQL bindings
- The table schema in `sql-statement`
- The statistics of the table in `sql-statement`
- The result of `EXPLAIN [ANALYZE] sql-statement`

**Note:**

PLAN REPLAYER **DOES NOT** export any table data.

#### 7.9.1.1 Examples of exporting cluster information

```
use test;
create table t(a int, b int);
insert into t values(1,1), (2, 2), (3, 3);
analyze table t;

plan replayer dump explain select * from t;
```

PLAN REPLAYER DUMP packages the table information above into a ZIP file and returns the file identifier as the execution result. This file is a one-time file. After the file is downloaded, TiDB will delete it.

**Note:**

The ZIP file is stored in a TiDB cluster for at most one hour. After one hour, TiDB will delete it.

```
MySQL [test]> plan replayer dump explain select * from t;
+-----+
| Dump_link
+-----+
| replayer_single_J0Gvpu4t7dssySqJfTtS4A==_1635750890568691080.zip |
+-----+
1 row in set (0.015 sec)
```

Because the file cannot be downloaded on MySQL Client, you need to use the TiDB HTTP interface and the file identifier to download the file:

```
http://${tidb-server-ip}:${tidb-server-status-port}/plan_replayer/dump/${
 ↪ file_token}
```

`${tidb-server-ip}:${tidb-server-status-port}` is the address of any TiDB server in the cluster. For example:

```
curl http://127.0.0.1:10080/plan_replayer/dump/
 ↪ replayer_single_JOGvpu4t7dssySqJfTtS4A==_1635750890568691080.zip >
 ↪ plan_replayer.zip
```

### 7.9.2 Use PLAN REPLAYER to import cluster information

#### Warning:

When you import the on-site information of a TiDB cluster to another cluster, the TiDB session variables, SQL bindings, table schemas and statistics of the latter cluster are modified.

With an existing ZIP file exported using PLAN REPLAYER, you can use the PLAN REPLAYER import interface to restore the on-site information of a cluster to any other TiDB cluster. The syntax is as follows:

```
PLAN REPLAYER LOAD 'file_name';
```

In the statement above, `file_name` is the name of the ZIP file to be imported.

For example:

```
PLAN REPLAYER LOAD 'plan_replayer.zip';
```

## 7.10 TiDB Cluster Troubleshooting Guide

You can use this guide to help you diagnose and solve basic problems while using TiDB. If your problem is not resolved, please collect the following information and [create an issue](#):

- The exact error message and the operations while the error occurs
- The state of all the components
- The `error/fatal/panic` information in the log of the component that reports the error
- The configuration and deployment topology
- The TiDB component related issue in `dmesg`

For other information, see [Frequently Asked Questions \(FAQ\)](#).

### 7.10.1 Cannot connect to the database

1. Make sure all the services are started, including `tidb-server`, `pd-server`, and `tikv-  
→ server`.
  2. Use the `ps` command to check if all the processes are running.
    - If a certain process is not running, see the following corresponding sections to diagnose and solve the issue.
    - If all the processes are running, check the `tidb-server` log to see if the following messages are displayed:
      - InformationSchema is out of date: This message is displayed if the `tikv-  
→ server` cannot be connected. Check the state and log of `pd-server` and `tikv-server`.
      - panic: This message is displayed if there is an issue with the program. Please provide the detailed panic log and [create an issue](#).
3. If the data is cleared and the services are re-deployed, make sure that:
  - All the data in `tikv-server` and `pd-server` are cleared. The specific data is stored in `tikv-server` and the metadata is stored in `pd-server`. If only one of the two servers is cleared, the data will be inconsistent.
  - After the data in `pd-server` and `tikv-server` are cleared and the `pd-server` and `tikv-server` are restarted, the `tidb-server` must be restarted too. The cluster ID is randomly allocated when the `pd-server` is initialized. So when the cluster is re-deployed, the cluster ID changes and you need to restart the `tidb-server` to get the new cluster ID.

### 7.10.2 Cannot start `tidb-server`

See the following for the situations when the `tidb-server` cannot be started:

- Error in the startup parameters.

See the [TiDB configuration and options](#).

- The port is occupied.

Use the `lsof -i:port` command to show all the networking related to a given port and make sure the port to start the `tidb-server` is not occupied.

- Cannot connect to `pd-server`.

- Check if the network between TiDB and PD is running smoothly, including whether the network can be pinged or if there is any issue with the Firewall configuration.
- If there is no issue with the network, check the state and log of the `pd-server` process.

### 7.10.3 Cannot start `tikv-server`

See the following for the situations when the `tikv-server` cannot be started:

- Error in the startup parameters: See the [TiKV configuration and options](#).
- The port is occupied: Use the `lsof -i:port` command to show all the networking related to a given port and make sure the port to start the `tikv-server` is not occupied.
- Cannot connect to `pd-server`.
  - Check if the network between TiDB and PD is running smoothly, including whether the network can be pinged or if there is any issue with the Firewall configuration.
  - If there is no issue with the network, check the state and log of the `pd-server` process.
- The file is occupied.

Do not open two TiKV files on one database file directory.

### 7.10.4 Cannot start `pd-server`

See the following for the situations when the `pd-server` cannot be started:

- Error in the startup parameters.  
See the [PD configuration and options](#).
- The port is occupied.

Use the `lsof -i:port` command to show all the networking related to a given port and make sure the port to start the `pd-server` is not occupied.

### 7.10.5 The TiDB/TiKV/PD process aborts unexpectedly

- Is the process started on the foreground? The process might exit because the client aborts.
- Is `nohup +&` run in the command line? This might cause the process to abort because it receives the hup signal. It is recommended to write and run the startup command in a script.

### 7.10.6 TiDB panic

Please provide the panic log and [create an issue](#).

### 7.10.7 The connection is rejected

Make sure the network parameters of the operating system are correct, including but not limited to:

- The port in the connection string is consistent with the `tidb-server` starting port.
- The firewall is configured correctly.

### 7.10.8 Open too many files

Before starting the process, make sure the result of `ulimit -n` is large enough. It is recommended to set the value to `unlimited` or larger than `1000000`.

### 7.10.9 Database access times out and the system load is too high

First, check the `slow query log` and see if it is because of some inappropriate SQL statement.

If you failed to solve the problem, provide the following information:

- The deployment topology
  - How many `tidb-server/pd-server/tikv-server` instances are deployed?
  - How are these instances distributed in the machines?
- The hardware configuration of the machines where these instances are deployed:
  - The number of CPU cores
  - The size of the memory
  - The type of the disk (SSD or Hard Drive Disk)
  - Are they physical machines or virtual machines?
- Are there other services besides the TiDB cluster?
- Are the `pd-servers` and `tikv-servers` deployed separately?
- What is the current operation?
- Check the CPU thread name using the `top -H` command.
- Are there any exceptions in the network or IO monitoring data recently?

## 7.11 Troubleshoot High Disk I/O Usage in TiDB

This document introduces how to locate and address the issue of high disk I/O usage in TiDB.

### 7.11.1 Check the current I/O metrics

If TiDB's response slows down after you have troubleshooted the CPU bottleneck and the bottleneck caused by transaction conflicts, you need to check I/O metrics to help determine the current system bottleneck.

#### 7.11.1.1 Locate I/O issues from monitor

The quickest way to locate I/O issues is to view the overall I/O status from the monitor, such as the Grafana dashboard which is deployed by default by TiUP. The dashboard panels related to I/O include **Overview**, **Node\_exporter**, and **Disk-Performance**.

##### 7.11.1.1.1 The first type of monitoring panels

In **Overview**> **System Info**> **IO Util**, you can see the I/O status of each machine in the cluster. This metric is similar to `util` in the Linux `iostat` monitor. The higher percentage represents higher disk I/O usage:

- If there is only one machine with high I/O usage in the monitor, currently there might be read and write hotspots on this machine.
- If the I/O usage of most machines in the monitor is high, the cluster now has high I/O loads.

For the first situation above (only one machine with high I/O usage), you can further observe I/O metrics from the **Disk-Performance Dashboard** such as **Disk Latency** and **Disk Load** to determine whether any anomaly exists. If necessary, use the `fio` tool to check the disk.

##### 7.11.1.1.2 The second type of monitoring panels

The main storage component of the TiDB cluster is TiKV. One TiKV instance contains two RocksDB instances: one for storing Raft logs, located in `data/raft`, and the other for storing real data, located in `data/db`.

In **TiKV-Details > Raft IO**, you can see the metrics related to disk writes of these two instances:

- **Append log duration**: This metric indicates the response time of writes into RockDB that stores Raft logs. The .99 response time should be within 50 ms.
- **Apply log duration**: This metric indicates the response time of writes into RockDB that stores real data. The .99 response should be within 100 ms.

These two metrics also have the `.. per server` monitoring panel to help you view the write hotspots.

### 7.11.1.1.3 The third type of monitoring panels

In **TiKV-Details > Storage**, there are monitoring metrics related to storage:

- **Storage command total**: Indicates the number of different commands received.
- **Storage async write duration**: Includes monitoring metrics such as `disk sync ↵ duration`, which might be related to Raft I/O. If you encounter an abnormal situation, check the working statuses of related components by checking logs.

### 7.11.1.1.4 Other panels

In addition, some other panel metrics might help you determine whether the bottleneck is I/O, and you can try to set some parameters. By checking the prewrite/commit/raw-put (for raw key-value clusters only) of TiKV gRPC duration, you can determine that the bottleneck is indeed the slow TiKV write. The common situations of slow TiKV writes are as follows:

- `append log` is slow. TiKV Grafana's Raft I/O and `append log duration` metrics are relatively high, which is often due to slow disk writes. You can check the value of `WAL Sync Duration max` in **RocksDB-raft** to determine the cause of slow `append ↵ log`. Otherwise, you might need to report a bug.
- The `raftstore` thread is busy. In TiKV Grafana, `Raft Propose/propose wait ↵ duration` is significantly higher than `append log duration`. Check the following aspects for troubleshooting:
  - Whether the value of `store-pool-size` of `[raftstore]` is too small. It is recommended to set this value between `[1, 5]` and not too large.
  - Whether the CPU resource of the machine is insufficient.
- `append log` is slow. TiKV Grafana's Raft I/O and `append log duration` metrics are relatively high, which might usually occur along with relatively high `Raft Propose ↵ /apply wait duration`. The possible causes are as follows:
  - The value of `apply-pool-size` of `[raftstore]` is too small. It is recommended to set this value between `[1, 5]` and not too large. The value of `Thread CPU ↵ /apply cpu` is also relatively high.
  - Insufficient CPU resources on the machine.
  - Write hotspot issue of a single Region (Currently, the solution to this issue is still on the way). The CPU usage of a single `apply` thread is high (which can be viewed by modifying the Grafana expression, appended with `by (instance, name)`).
  - Slow write into RocksDB, and `RocksDB kv/max write duration` is high. A single Raft log might contain multiple key-value pairs (kv). 128 kvs are written to RocksDB in a batch, so one `apply` log might involve multiple RocksDB writes.
  - For other causes, report them as bugs.

- raft commit log is slow. In TiKV Grafana, Raft I/O and commit log duration (only available in Grafana 4.x) metrics are relatively high. Each Region corresponds to an independent Raft group. Raft has a flow control mechanism similar to the sliding window mechanism of TCP. To control the size of a sliding window, adjust the [raftstore] raft-max-inflight-msgs parameter. If there is a write hotspot and commit log duration is high, you can properly set this parameter to a larger value, such as 1024.

#### 7.11.1.2 Locate I/O issues from log

- If the client reports errors such as `server is busy` or especially `raftstore is busy`, the errors might be related to I/O issues.

You can check the monitoring panel (**Grafana -> TiKV -> errors**) to confirm the specific cause of the `busy` error. `server is busy` is TiKV's flow control mechanism. In this way, TiKV informs `tidb/ti-client` that the current pressure of TiKV is too high, and the client should try later.

- Write stall appears in TiKV RocksDB logs.

It might be that too many level-0 SST files cause the write stall. To address the issue, you can add the [rocksdb] max-sub-compactions = 2 (or 3) parameter to speed up the compaction of level-0 SST files. This parameter means that the compaction tasks of level-0 to level-1 can be divided into `max-sub-compactions` subtasks for multi-threaded concurrent execution.

If the disk's I/O capability fails to keep up with the write, it is recommended to scale up the disk. If the throughput of the disk reaches the upper limit (for example, the throughput of SATA SSD is much lower than that of NVMe SSD), which results in write stall, but the CPU resource is relatively sufficient, you can try to use a compression algorithm of higher compression ratio to relieve the pressure on the disk, that is, use CPU resources to make up for disk resources.

For example, when the pressure of default cf compaction is relatively high, you can change the parameter [rocksdb.defaultcf] compression-per-level = ["no", ↵ "no", "lz4", "lz4", "lz4", "zstd", "zstd"] to compression-per-level ↵ = ["no", "no", "zstd", "zstd", "zstd", "zstd", "zstd"].

#### 7.11.1.3 I/O issues found in alerts

The cluster deployment tool (TiUP) deploys the cluster with alert components by default that have built-in alert items and thresholds. The following alert items are related to I/O:

- TiKV\_write\_stall
- TiKV\_raft\_log\_lag
- TiKV\_async\_request\_snapshot\_duration\_seconds
- TiKV\_async\_request\_write\_duration\_seconds
- TiKV\_raft\_append\_log\_duration\_secs
- TiKV\_raft\_apply\_log\_duration\_secs

### 7.11.2 Handle I/O issues

- When an I/O hotspot issue is confirmed to occur, you need to refer to Handle TiDB Hotspot Issues to eliminate the I/O hotspots.
- When it is confirmed that the overall I/O performance has become the bottleneck, and you can determine that the I/O performance will keep falling behind in the application side, then you can take advantage of the distributed database's capability of scaling and increase the number of TiKV nodes to have greater overall I/O throughput.
- Adjust some of the parameters as described above, and use computing/memory resources to make up for disk storage resources.

## 7.12 Troubleshoot Lock Conflicts

TiDB supports complete distributed transactions. Starting from v3.0, TiDB provides optimistic transaction mode and pessimistic transaction mode. This document introduces how to troubleshoot and resolve lock conflicts in TiDB.

### 7.12.1 Optimistic transaction mode

Transactions in TiDB use two-phase commit (2PC) that includes the Prewrite phase and the Commit phase. The procedure is as follows:

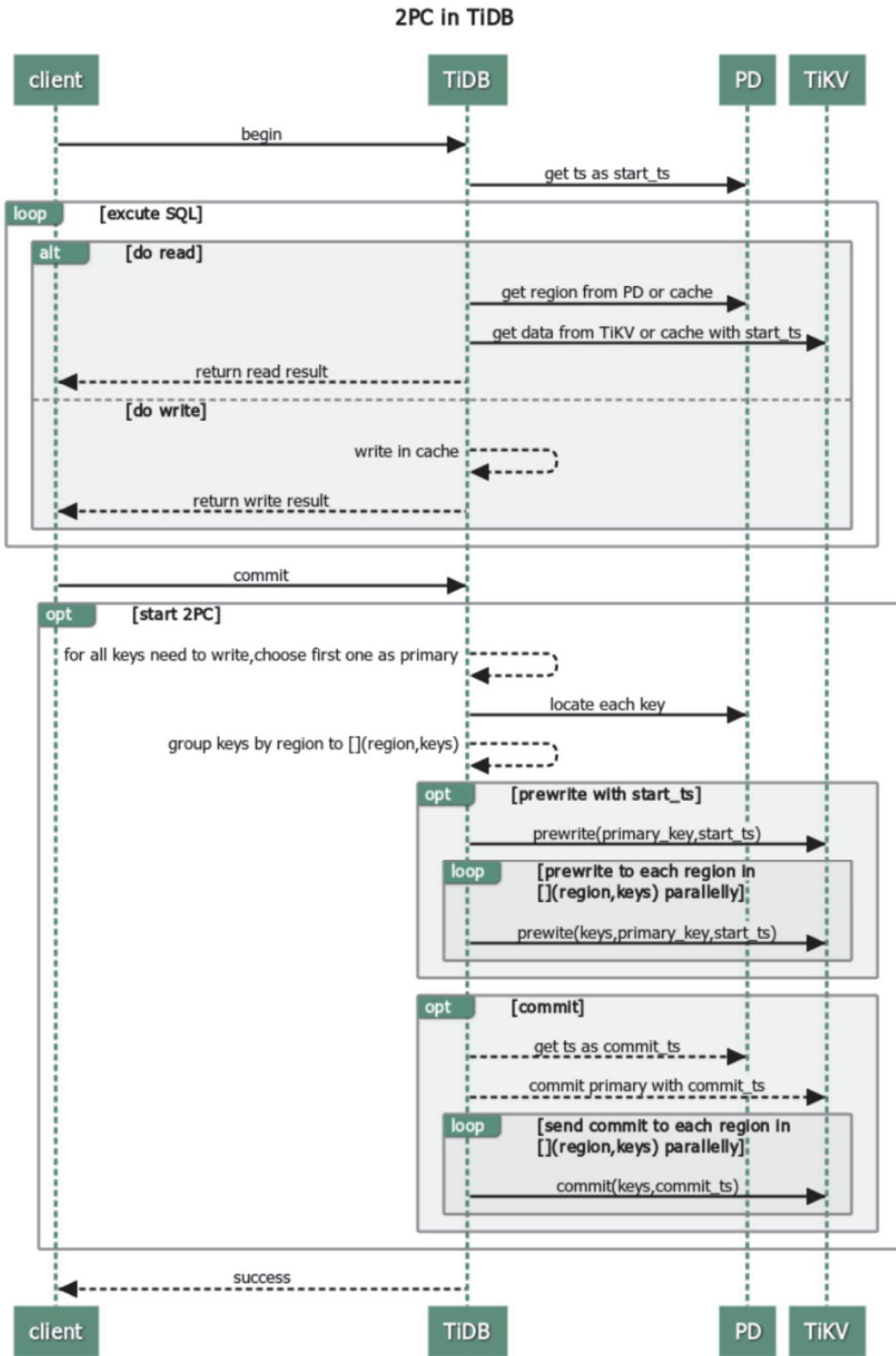


Figure 56: two-phase commit in the optimistic transaction mode  
479

For details of Percolator and TiDB's algorithm of the transactions, see [Google's Percolator](#).

### 7.12.1.1 Prewrite phase (optimistic)

In the Prewrite phase, TiDB adds a primary lock and a secondary lock to target keys. If there are lots of requests for adding locks to the same target key, TiDB prints an error such as write conflict or `keyislocked` to the log and reports it to the client. Specifically, the following errors related to locks might occur in the Prewrite phase.

#### 7.12.1.1.1 Read-write conflict (optimistic)

As the TiDB server receives a read request from a client, it gets a globally unique and increasing timestamp at the physical time as the `start_ts` of the current transaction. The transaction needs to read the latest data before `start_ts`, that is, the target key of the latest `commit_ts` that is smaller than `start_ts`. When the transaction finds that the target key is locked by another transaction, and it cannot know which phase the other transaction is in, a read-write conflict happens. The diagram is as follows:

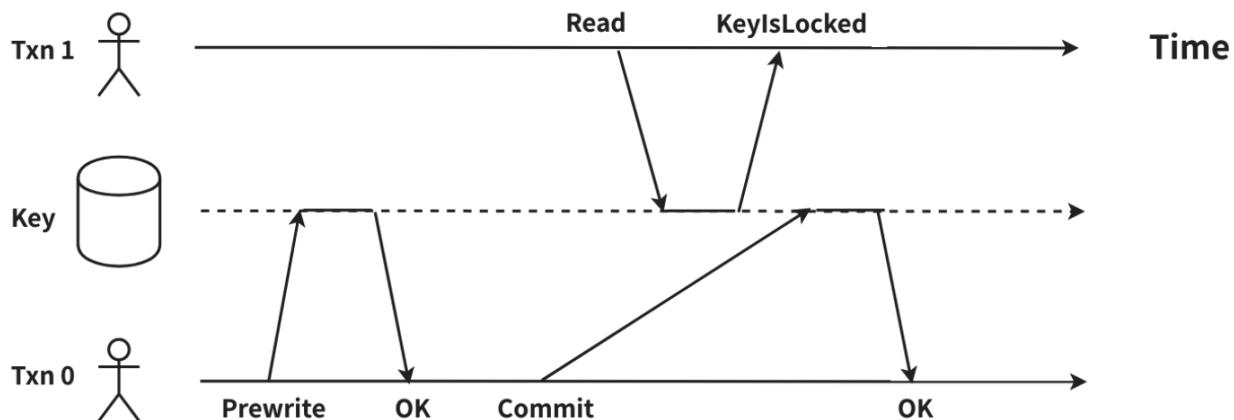


Figure 57: read-write conflict

Txn0 completes the Prewrite phase and enters the Commit phase. At this time, Txn1 requests to read the same target key. Txn1 needs to read the target key of the latest `commit_ts` that is smaller than its `start_ts`. Because Txn1's `start_ts` is larger than Txn0's `lock_ts`, Txn1 must wait for the target key's lock to be cleared, but it hasn't been done. As a result, Txn1 cannot confirm whether Txn0 has been committed or not. Thus, a read-write conflict between Txn1 and Txn0 happens.

You can detect the read-write conflict in your TiDB cluster by the following ways:

1. Monitoring metrics and logs of the TiDB server

- Monitoring data through Grafana

On the KV Errors panel in the TiDB dashboard, there are two monitoring metrics Lock Resolve OPS and KV Backoff OPS which can be used to check read-write conflicts in the transactions. If the values of both `not_expired` and `resolve` under Lock Resolve OPS increase, there might be many read-write conflicts. The `not_expired` item means that the transaction's lock has not timed out. The `resolve` item means that the other transaction tries to clean up the locks. If the value of another `txLockFast` item under KV Backoff OPS increases, there might also be read-write conflicts.



- Logs of the TiDB server

If there is any read-write conflict, you can see the following message in the TiDB log:

```
[INFO] [coprocessor.go:743] "[TIME_COP_PROCESS] resp_time
 ↪ :406.038899ms txnStartTS:416643508703592451 region_id:8297
 ↪ store_addr:10.8.1.208:20160 backoff_ms:255 backoff_types:[
 ↪ txLockFast,txLockFast] kv_process_ms:333 scan_total_write
 ↪ :0 scan_processed_write:0 scan_total_data:0
```

```

 ↵ scan_processed_data:0 scan_total_lock:0 scan_processed_lock
 ↵ :0"]

```

- txnsStartTS: The start\_ts of the transaction that is sending the read request. In the above log, 416643508703592451 is the start\_ts.
- backoff\_types: If a read-write conflict happens, and the read request performs backoff and retry, the type of retry is TxnLockFast.
- backoff\_ms: The time that the read request spends in the backoff and retry, and the unit is milliseconds. In the above log, the read request spends 255 milliseconds in the backoff and retry.
- region\_id: Region ID corresponding to the target key of the read request.

## 2. Logs of the TiKV server

If there is any read-write conflict, you can see the following message in the TiKV log:

```

[ERROR] [endpoint.rs:454] [error-response] [err=""locked primary_lock
 ↵ :74800000000000004
 ↵ D35F6980000000000000000010380000000004C788E0380000000004C0748
 ↵ lock_version: 411402933858205712 key: 7480000000000004
 ↵ D35F7280000000004C0748 lock_ttl: 3008 txn_size: 1""]

```

This message indicates that a read-write conflict occurs in TiDB. The target key of the read request has been locked by another transaction. The locks are from the uncommitted optimistic transaction and the uncommitted pessimistic transaction after the prewrite phase.

- primary\_lock: Indicates that the target key is locked by the primary lock.
- lock\_version: The start\_ts of the transaction that owns the lock.
- key: The target key that is locked.
- lock\_ttl: The lock's TTL (Time To Live)
- txn\_size: The number of keys that are in the Region of the transaction that owns the lock.

Solutions:

- A read-write conflict triggers an automatic backoff and retry. As in the above example, Txn1 has a backoff and retry. The first time of the retry is 100 ms, the longest retry is 3000 ms, and the total time is 20000 ms at maximum.
- You can use the sub-command **decoder** of TiDB Control to view the table id and rowid of the row corresponding to the specified key:

```

./tidb-ctl decoder -f table_row -k "t\x00\x00\x00\x00\x00\x00\x00\x00\x1c_r
 ↵ \x00\x00\x00\x00\x00\x00\x00\xfa"

```

```

table_id: -9223372036854775780
row_id: -9223372036854775558

```

### 7.12.1.1.2 KeyIsLocked error

In the Prewrite phase of a transaction, TiDB checks whether there is any write-write conflict, and then checks whether the target key has been locked by another transaction. If the key is locked, the TiKV server outputs a “KeyIsLocked” error. At present, the error message is not printed in the logs of TiDB and TiKV. Same as read-write conflicts, when “KeyIsLocked” occurs, TiDB automatically performs backoff and retry for the transaction.

You can check whether there's any “KeyIsLocked” error in the TiDB monitoring on Grafana:

The KV Errors panel in the TiDB dashboard has two monitoring metrics `Lock Resolve OPS` and `KV Backoff OPS` which can be used to check write-write conflicts caused by a transaction. If the `resolve` item under `Lock Resolve OPS` and the `txnLock` item under `KV Backoff OPS` have a clear upward trend, a “KeyIsLocked” error occurs. `resolve` refers to the operation that attempts to clear the lock, and `txnLock` represents a write conflict.



Solutions:

- If there is a small amount of txnLock in the monitoring, no need to pay too much attention. The backoff and retry is automatically performed in the background. The first time of the retry is 200 ms and the maximum time is 3000 ms for a single retry.
- If there are too many “txnLock” operations in the KV Backoff OPS, it is recommended that you analyze the reasons to the write conflicts from the application side.
- If your application is a write-write conflict scenario, it is strongly recommended to use the pessimistic transaction mode.

### 7.12.1.2 Commit phase (optimistic)

After the Prewrite phase completes, the client obtains commit\_ts, and then the transaction is going to the next phase of 2PC - the Commit phase.

#### 7.12.1.2.1 LockNotFound error

The error log of “TxnLockNotFound” means that transaction commit time is longer than the TTL time, and when the transaction is going to commit, its lock has been rolled back by other transactions. If the TiDB server enables transaction commit retry, this transaction is re-executed according to `tidb_retry_limit`. (Note about the difference between explicit and implicit transactions.)

You can check whether there is any “LockNotFound” error in the following ways:

1. View the logs of the TiDB server

If a “TxnLockNotFound” error occurs, the TiDB log message is like this:

```
[WARN] [session.go:446] ["commit failed"] [conn=149370] ["finished txn
 ↳ "=Txn{state=invalid}"] [error=[kv:6]Error: KV error safe to
 ↳ retry tikv restarts txn: Txn(Mvcc(TxnLockNotFound{ start_ts:
 ↳ 412720515987275779, commit_ts: 412720519984971777, key: [116,
 ↳ 128, 0, 0, 0, 0, 1, 111, 16, 95, 114, 128, 0, 0, 0, 0, 0, 0, 2
 ↳ }}) [try again later]]"]
```

- start\_ts: The start\_ts of the transaction that outputs the TxnLockNotFound error because its lock has been rolled back by other transactions. In the above log, 412720515987275779 is the start\_ts.
- commit\_ts: The commit\_ts of the transaction that outputs the TxnLockNotFound error. In the above log, 412720519984971777 is the commit\_ts.

2. View the logs of the TiKV server

If a “TxnLockNotFound” error occurs, the TiKV log message is like this:

```
Error: KV error safe to retry restarts txn: Txn(Mvcc(TxnLockNotFound))
 ↳ [ERROR [Kv.rs:708] ["KvService::batch_raft send response fail"] [
 ↳ err=RemoteStoped]
```

Solutions:

- By checking the time interval between start\_ts and commit\_ts, you can confirm whether the commit time exceeds the TTL time.

Checking the time interval using the PD control tool:

```
tiup ctl pd tso [start_ts]
tiup ctl pd tso [commit_ts]
```

- It is recommended to check whether the write performance is slow, which might cause that the efficiency of transaction commit is poor, and thus the lock is cleared.
- In the case of disabling the TiDB transaction retry, you need to catch the exception on the application side and try again.

### 7.12.2 Pessimistic transaction mode

Before v3.0.8, TiDB uses the optimistic transaction mode by default. In this mode, if there is a transaction conflict, the latest transaction will fail to commit. Therefore, the application needs to support retrying transactions. The pessimistic transaction mode resolves this issue, and the application does not need to modify any logic for the workaround.

The commit phase of the pessimistic transaction mode and the optimistic transaction mode in TiDB has the same logic, and both commits are in the 2PC mode. The important adaptation of pessimistic transactions is DML execution.

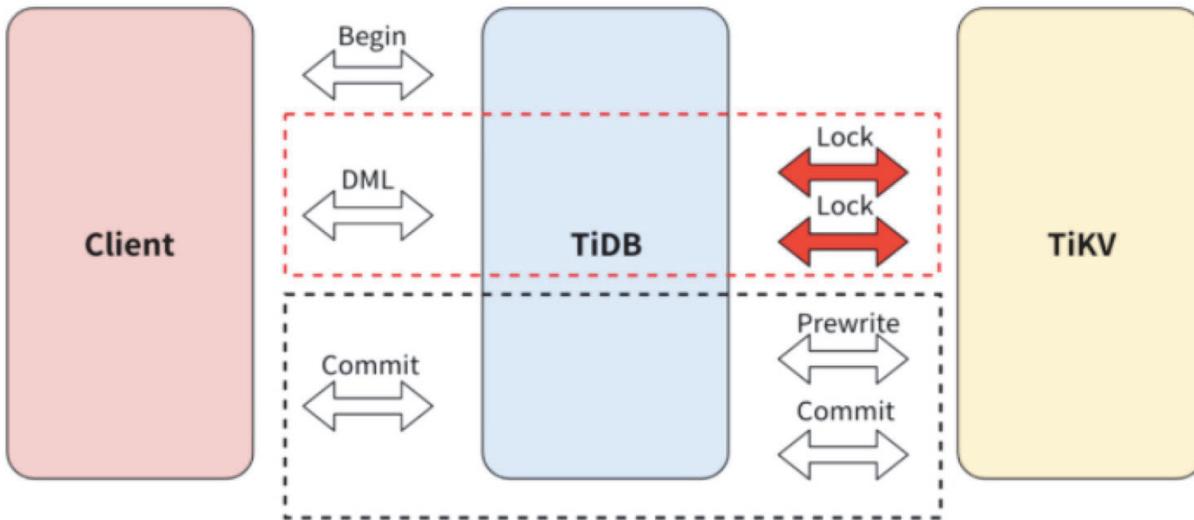


Figure 58: TiDB pessimistic transaction commit logic

The pessimistic transaction adds an `Acquire Pessimistic Lock` phase before 2PC. This phase includes the following steps:

1. (same as the optimistic transaction mode) Receive the `begin` request from the client, and the current timestamp is this transaction's `start_ts`.
2. When the TiDB server receives an `update` request from the client, the TiDB server initiates a pessimistic lock request to the TiKV server, and the lock is persisted to the TiKV server.
3. (same as the optimistic transaction mode) When the client sends the `commit` request, TiDB starts to perform the 2PC similar to the optimistic transaction mode.

### Pessimistic Transaction in TiDB

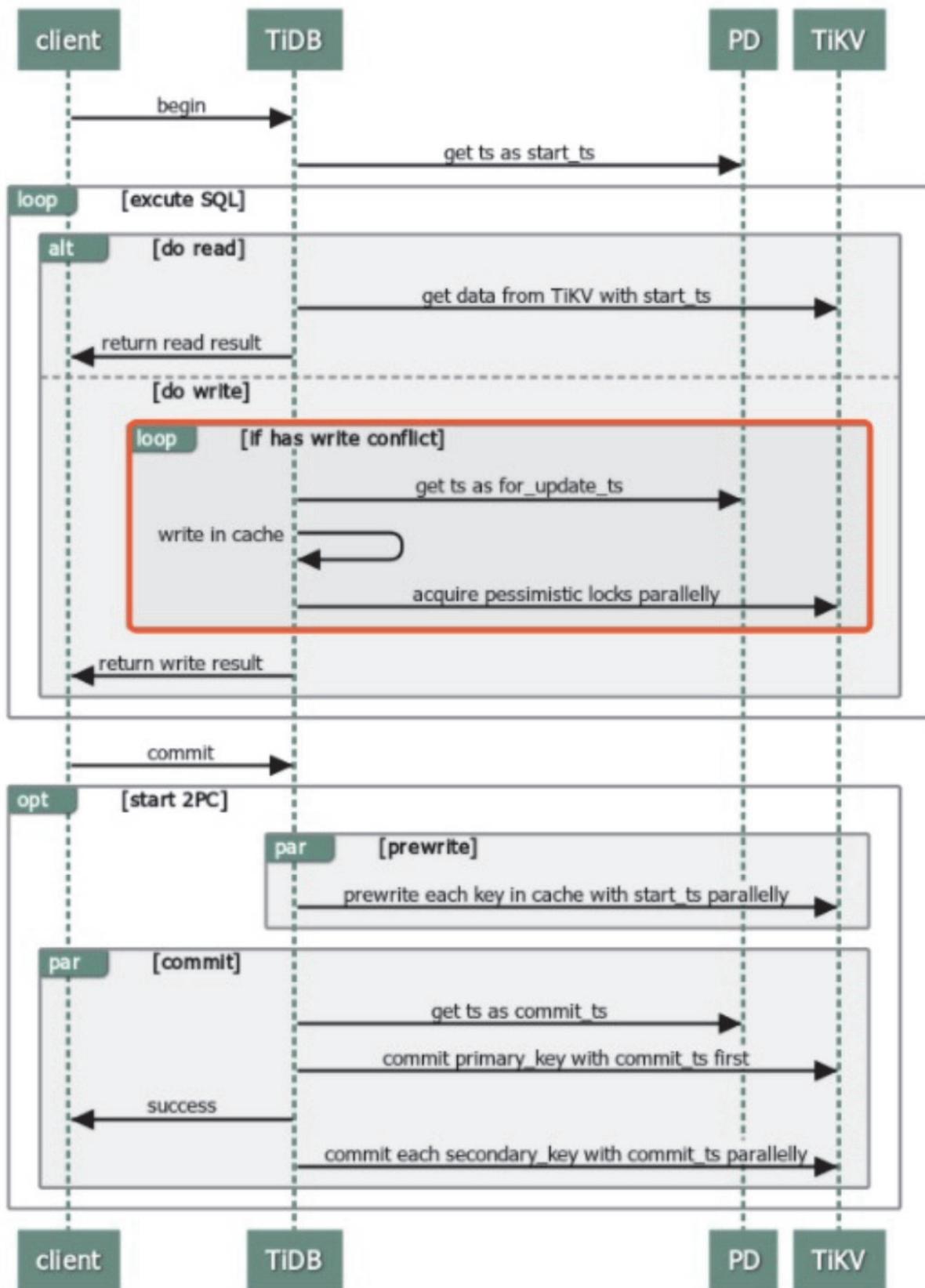


Figure 59: Pessimistic transactions in TiDB

For details, see [Pessimistic transaction mode](#).

### 7.12.2.1 Prewrite phase (pessimistic)

In the transaction pessimistic mode, the commit phase is the same as the 2PC. Therefore, the read-write conflict also exists as in the optimistic transaction mode.

#### 7.12.2.1.1 Read-write conflict (pessimistic)

Same as [Read-write conflict \(optimistic\)](#).

### 7.12.2.2 Commit phase (pessimistic)

In the pessimistic transaction mode, there will be no `TxnLockNotFound` error. Instead, the pessimistic lock will automatically update the TTL of the transaction through `txnheartbeat` to ensure that the second transaction does not clear the lock of the first transaction.

### 7.12.2.3 Other errors related to locks

#### 7.12.2.3.1 Pessimistic lock retry limit reached

When the transaction conflict is very serious or a write conflict occurs, the optimistic transaction will be terminated directly, and the pessimistic transaction will retry the statement with the latest data from storage until there is no write conflict.

Because TiDB's locking operation is a write operation, and the process of the operation is to read first and then write, there are two RPC requests. If a write conflict occurs in the middle of a transaction, TiDB will try again to lock the target keys, and each retry will be printed to the TiDB log. The number of retries is determined by `pessimistic-txn.max-retry-count`.

In the pessimistic transaction mode, if a write conflict occurs and the number of retries reaches the upper limit, an error message containing the following keywords appears in the TiDB log:

```
err="pessimistic lock retry limit reached"
```

Solutions:

- If the above error occurs frequently, it is recommended to adjust from the application side.

#### 7.12.2.3.2 Lock wait timeout exceeded

In the pessimistic transaction mode, transactions wait for locks of each other. The timeout for waiting a lock is defined by the `innodb_lock_wait_timeout` parameter of TiDB. This is the maximum wait lock time at the SQL statement level, which is the expectation of a SQL statement Locking, but the lock has never been acquired. After this time, TiDB will not try to lock again and will return the corresponding error message to the client.

When a wait lock timeout occurs, the following error message will be returned to the client:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

Solutions:

- If the above error occurs frequently, it is recommended to adjust the application logic.

#### 7.12.2.3.3 TTL manager has timed out

The transaction execution time can not exceed the GC time limit. In addition, the TTL time of pessimistic transactions has an upper limit, whose default value is 1 hour. Therefore, a pessimistic transaction executed for more than 1 hour will fail to commit. This timeout threshold is controlled by the TiDB parameter `performance.max-txn-ttl`.

When the execution time of a pessimistic transaction exceeds the TTL time, the following error message occurs in the TiDB log:

```
TTL manager has timed out, pessimistic locks may expire, please commit or
→ rollback this transaction
```

Solutions:

- First, confirm whether the application logic can be optimized. For example, large transactions may trigger TiDB's transaction size limit, which can be split into multiple small transactions.
- Also, you can adjust the related parameters properly to meet the application transaction logic.

#### 7.12.2.3.4 Deadlock found when trying to get lock

Due to resource competition between two or more transactions, a deadlock occurs. If you do not handle it manually, transactions that block each other cannot be executed successfully and will wait for each other forever. To resolve dead locks, you need to manually terminate one of the transactions to resume other transaction requests.

When a pessimistic transaction has a deadlock, one of the transactions must be terminated to unlock the deadlock. The client will return the same `Error 1213` error as in MySQL, for example:

```
[err="[executor:1213]Deadlock found when trying to get lock; try restarting
↪ transaction"]
```

Solutions:

- If it is difficult to confirm the cause of the deadlock, for v5.1 and later versions, you are recommended to try to query the `INFORMATION_SCHEMA.DEADLOCKS` or `INFORMATION_SCHEMA.CLUSTER_DEADLOCKS` system table to get the information of deadlock waiting chain. For details, see the [Deadlock errors](#) section and the [DEADLOCKS table](#) document.
- If the deadlock occurs frequently, you need to adjust the transaction query logic in your application to reduce such occurrences.

#### 7.12.2.4 Use Lock View to troubleshoot issues related to pessimistic locks

Since v5.1, TiDB supports the Lock View feature. This feature has several system tables built in `information_schema` that provide more information about the pessimistic lock conflicts and pessimistic lock waitings. For the detailed introduction of these tables, see the following documents:

- **`TIDB_TRX` and `CLUSTER_TIDB_TRX`**: Provides information of all running transactions on the current TiDB node or in the entire cluster, including whether the transaction is in the lock-waiting state, the lock-waiting time, and the digests of statements that have been executed in the transaction.
- **`DATA_LOCK_WAITS`**: Provides the pessimistic lock-waiting information in TiKV, including the `start_ts` of the blocking and blocked transaction, the digest of the blocked SQL statement, and the key on which the waiting occurs.
- **`DEADLOCKS` and `CLUSTER_DEADLOCKS`**: Provides the information of several deadlock events that have recently occurred on the current TiDB node or in the entire cluster, including the waiting relationship among transactions in the deadlock loops, the digest of the statement currently being executed in the transaction, and the key on which the waiting occurs.

#### Note:

The SQL statements shown in the Lock View-related system tables are normalized SQL statements (that is, SQL statements without formats and arguments), which are obtained by internal queries according to SQL digests, so the tables cannot obtain the complete statements that include the format and arguments. For the detailed description of SQL digests and normalized SQL statement, see [Statement Summary Tables](#).

The following sections show the examples of troubleshooting some issues using these tables.

#### 7.12.2.4.1 Deadlock errors

To get the information of the recent deadlock errors, you can query the DEADLOCKS or CLUSTER\_DEADLOCKS table. For example:

```
select * from information_schema.deadlocks;
```

| DEADLOCK_ID                                                         | OCCUR_TIME                                                       | TRY_LOCK_TRX_ID        |
|---------------------------------------------------------------------|------------------------------------------------------------------|------------------------|
| TRX_HOLDING_LOCK                                                    | KEY                                                              | KEY_INFO               |
| 1                                                                   | 2021-08-05 11:09:03.230341                                       | 0   426812829645406216 |
| 22230766411edb40f27a68dадefc63c6c6970d5827f1e5e22fc97be2c4d8350d    | update `t` set `v` = ? where `id` = ? ;   7480000000000000000355 |                        |
| F72800000000000000002   {"db_id":1,"db_name":"test","table_id":53," | table_name":"t","handle_type":"int","handle_value":2}            |                        |
| 426812829645406217                                                  |                                                                  |                        |
| 1   2021-08-05 11:09:03.230341   0   426812829645406217             |                                                                  |                        |
| 22230766411edb40f27a68dадefc63c6c6970d5827f1e5e22fc97be2c4d8350d    | update `t` set `v` = ? where `id` = ? ;   7480000000000000000355 |                        |
| F72800000000000000001   {"db_id":1,"db_name":"test","table_id":53," | table_name":"t","handle_type":"int","handle_value":1}            |                        |
| 426812829645406216                                                  |                                                                  |                        |

The query result above shows the waiting relationship among multiple transactions in the deadlock error, the normalized form of the SQL statements currently being executed in each transaction (statements without formats and arguments), the key on which the conflict occurs, and the information of the key.

For example, in the above example, the first row means that the transaction with the ID of 426812829645406216 is executing a statement like `update `t` set `v` =? Where `id` =? ;` but is blocked by another transaction with the ID of 426812829645406217. The

transaction with the ID of 426812829645406217 is also executing a statement that is in the form of `update `t` set `v` =? Where `id` =? ;` but is blocked by the transaction with the ID of 426812829645406216. The two transactions thus form a deadlock.

#### 7.12.2.4.2 A few hot keys cause queueing locks

The `DATA_LOCK_WAITS` system table provides the lock-waiting status on the TiKV nodes. When you query this table, TiDB automatically obtains the real-time lock-waiting information from all TiKV nodes. If a few hot keys are frequently locked and block many transactions, you can query the `DATA_LOCK_WAITS` table and aggregate the results by key to try to find the keys on which issues frequently occur:

```
select `key`, count(*) as `count` from information_schema.data_lock_waits
→ group by `key` order by `count` desc;
```

| key                                      | count |
|------------------------------------------|-------|
| 7480000000000000415F7280000000000000001  | 2     |
| 7480000000000000415F72800000000000000002 | 1     |

To avoid contingency, you might need to make multiple queries.

If you know the key that frequently has issues occurred, you can try to get the information of the transaction that tries to lock the key from the `TIDB_TRX` or `CLUSTER_TIDB_TRX` table.

Note that the information displayed in the `TIDB_TRX` and `CLUSTER_TIDB_TRX` tables is also the information of the transactions that are running at the time the query is performed. These tables do not display the information of the completed transactions. If there is a large number of concurrent transactions, the result set of the query might also be large. You can use the `limit` clause or the `where` clause to filter out transactions with a long lock-waiting time. Note that when you join multiple tables in Lock View, the data in different tables might not be obtained at the same time, so the information in different tables might not be consistent.

```
select trx.* from information_schema.data_lock_waits as l left join
→ information_schema.tidb_trx as trx on l.trx_id = trx.id where l.key =
→ "7480000000000000415F7280000000000000001"\G
```

```
***** 1. row *****
ID: 426831815660273668
START_TIME: 2021-08-06 07:16:00.081000
CURRENT_SQL_DIGEST: 06
→ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
CURRENT_SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ? ;
STATE: LockWaiting
```

```

WAITING_START_TIME: 2021-08-06 07:16:00.087720
 MEM_BUFFER_KEYS: 0
 MEM_BUFFER_BYTES: 0
 SESSION_ID: 77
 USER: root
 DB: test
 ALL_SQL_DIGESTS: ["0
 ↳ fdc781f19da1c6078c9de7eadef8a307889c001e05f107847bee4cf8f3cdf3
 ↳ ", "06
 ↳ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
 ↳ "]"
***** 2. row *****
 ID: 426831818019569665
 START_TIME: 2021-08-06 07:16:09.081000
 CURRENT_SQL_DIGEST: 06
 ↳ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
CURRENT_SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ? ;
 STATE: LockWaiting
 WAITING_START_TIME: 2021-08-06 07:16:09.290271
 MEM_BUFFER_KEYS: 0
 MEM_BUFFER_BYTES: 0
 SESSION_ID: 75
 USER: root
 DB: test
 ALL_SQL_DIGESTS: ["0
 ↳ fdc781f19da1c6078c9de7eadef8a307889c001e05f107847bee4cf8f3cdf3
 ↳ ", "06
 ↳ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
 ↳ "]"
2 rows in set (0.00 sec)

```

#### 7.12.2.4.3 A transaction is blocked for a long time

If a transaction is known to be blocked by another transaction (or multiple transactions) and the `start_ts` (transaction ID) of the current transaction is known, you can use the following method to obtain the information of the blocking transaction. Note that when you join multiple tables in Lock View, the data in different tables might not be obtained at the same time, so the information in different tables might not be consistent.

```

select l.key, trx.* , tidb_decode_sql_digests(trx.all_sql_digests) as sqls
 ↳ from information_schema.data_lock_waits as l join information_schema.
 ↳ cluster_tidb_trx as trx on l.current_holding_trx_id = trx.id where l.
 ↳ trx_id = 426831965449355272\G

```

```
***** 1. row *****
```

```

key: 74800000000000004D5F72800000000000000001
INSTANCE: 127.0.0.1:10080
ID: 426832040186609668
START_TIME: 2021-08-06 07:30:16.581000
CURRENT_SQL_DIGEST: 06
 ↳ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
CURRENT_SQL_DIGEST_TEXT: update `t` set `v` = `v` + ? where `id` = ? ;
STATE: LockWaiting
WAITING_START_TIME: 2021-08-06 07:30:16.592763
MEM_BUFFER_KEYS: 1
MEM_BUFFER_BYTES: 19
SESSION_ID: 113
USER: root
DB: test
ALL_SQL_DIGESTS: ["0
 ↳ fdc781f19da1c6078c9de7eadef8a307889c001e05f107847bee4cf8f3cdf3
 ↳ ","
 ↳ a4e28cc182bdd18288e2a34180499b9404cd0ba07e3cc34b6b3be7b7c2de7fe9
 ↳ ","
 ↳ "06
 ↳ da614b93e62713bd282d4685fc5b88d688337f36e88fe55871726ce0eb80d7
 ↳ "
sqls: ["begin ;","select * from `t` where `id` = ? for
 ↳ update ;","update `t` set `v` = `v` + ? where `id` =
 ↳ ? ;"]
1 row in set (0.01 sec)

```

In the above query, the `TIDB_DECODE_SQL_DIGESTS` function is used on the `ALL_SQL_DIGESTS` column of the `CLUSTER_TIDB_TRX` table. This function tries to convert this column (the value is a set of SQL digests) to the normalized SQL statements, which improves readability.

If the `start_ts` of the current transaction is unknown, you can try to find it out from the information in the `TIDB_TRX` / `CLUSTER_TIDB_TRX` table or in the `PROCESSLIST` / `CLUSTER_PROCESSLIST` table.

## 7.13 Troubleshoot a TiFlash Cluster

This section describes some commonly encountered issues when using TiFlash, the reasons, and the solutions.

### 7.13.1 TiFlash fails to start

The issue might occur due to different reasons. It is recommended that you troubleshoot it following the steps below:

1. Check whether your system is CentOS8.

CentOS8 does not have the `libnsl.so` system library. You can manually install it via the following command:

```
shell dnf install libnsl
```

2. Check your system's `ulimit` parameter setting.

```
shell ulimit -n 1000000
```

3. Use the PD Control tool to check whether there is any TiFlash instance that failed to go offline on the node (same IP and Port) and force the instance(s) to go offline. For detailed steps, refer to [Scale in a TiFlash cluster](#).

If the above methods cannot resolve your issue, save the TiFlash log files and email to [info@pingcap.com](mailto:info@pingcap.com) for more information.

### 7.13.2 TiFlash replica is always unavailable

This is because TiFlash is in an abnormal state caused by configuration errors or environment issues. Take the following steps to identify the faulty component:

1. Check whether PD enables the Placement Rules feature:

```
echo 'config show replication' | /path/to/pd-ctl -u http://<pd-ip>:<pd-
→ port>
```

The expected result is "enable-placement-rules": "true". If not enabled, [enable the Placement Rules feature](#).

2. Check whether the TiFlash process is working correctly by viewing UpTime on the TiFlash-Summary monitoring panel.
3. Check whether the TiFlash proxy status is normal through pd-ctl.

```
echo "store" | /path/to/pd-ctl -u http://<pd-ip>:<pd-port>
```

The TiFlash proxy's `store.labels` includes information such as `{"key": "engine", "value": "tiflash"}`. You can check this information to confirm a TiFlash proxy.

4. Check whether `pd_buddy` can correctly print the logs (the log path is the value of `log` in the `[flash.flash_cluster]` configuration item; the default log path is under the `tmp` directory configured in the TiFlash configuration file).
5. Check whether the number of configured replicas is less than or equal to the number of TiKV nodes in the cluster. If not, PD cannot replicate data to TiFlash:

```
echo 'config placement-rules show' | /path/to/pd-ctl -u http://<pd-ip>
↪ >:<pd-port>
```

Reconfirm the value of `default: count`.

**Note:**

After the `placement rules` feature is enabled, the previously configured `max-replicas` and `location-labels` no longer take effect. To adjust the replica policy, use the interface related to placement rules.

6. Check whether the remaining disk space of the machine (where `store` of the TiFlash node is) is sufficient. By default, when the remaining disk space is less than 20% of the `store` capacity (which is controlled by the `low-space-ratio` parameter), PD cannot schedule data to this TiFlash node.

#### 7.13.3 TiFlash query time is unstable, and the error log prints many Lock Exception messages

This is because large amounts of data are written to the cluster, which causes that the TiFlash query encounters a lock and requires query retry.

You can set the query timestamp to one second earlier in TiDB. For example, if the current time is ‘2020-04-08 20:15:01’, you can execute `set @@tidb_snapshot='2020-04-08 20:15:00'`; before you execute the query. This makes less TiFlash queries encounter a lock and mitigates the risk of unstable query time.

#### 7.13.4 Some queries return the Region Unavailable error

If the load pressure on TiFlash is too heavy and it causes that TiFlash data replication falls behind, some queries might return the `Region Unavailable` error.

In this case, you can balance the load pressure by adding more TiFlash nodes.

#### 7.13.5 Data file corruption

Take the following steps to handle the data file corruption:

1. Refer to [Take a TiFlash node down](#) to take the corresponding TiFlash node down.
2. Delete the related data of the TiFlash node.
3. Redeploy the TiFlash node in the cluster.

## 7.14 Troubleshoot Write Conflicts in Optimistic Transactions

This document introduces the reason of and solutions to write conflicts in optimistic transactions.

Before TiDB v3.0.8, TiDB uses the optimistic transaction model by default. In this model, TiDB does not check conflicts during transaction execution. Instead, while the transaction is finally committed, the two-phase commit (2PC) is triggered and TiDB checks write conflicts. If a write conflict exists and the auto-retry mechanism is enabled, then TiDB retries the transaction within limited times. If the retry succeeds or has reached the upper limit on retry times, TiDB returns the result of transaction execution to the client. Therefore, if a lot of write conflicts exist in the TiDB cluster, the duration can be longer.

### 7.14.1 The reason of write conflicts

TiDB implements its transactions by using the [Percolator](#) transaction model. `percolator` is generally an implementation of 2PC. For the detailed 2PC process, see [TiDB Optimistic Transaction Model](#).

After the client sends a `COMMIT` request to TiDB, TiDB starts the 2PC process:

1. TiDB chooses one key from all keys in the transaction as the primary key of the transaction.
2. TiDB sends the `prewrite` request to all the TiKV Regions involved in this commit. TiKV judges whether all keys can preview successfully.
3. TiDB receives the result that all `prewrite` requests are successful.
4. TiDB gets the `commit_ts` from PD.
5. TiDB sends the `commit` request to the TiKV Region that contains the primary key of the transaction. After TiKV receives the `commit` request, it checks the validity of the data and clears the locks left in the `prewrite` stage.
6. After the `commit` request returns successfully, TiDB returns success to the client.

The write conflict occurs in the `prewrite` stage. When the transaction finds that another transaction is writing the current key (`data.commit_ts > txn.start_ts`), a write conflict occurs.

### 7.14.2 Detect write conflicts

In the TiDB Grafana panel, check the following monitoring metrics under **KV Errors**:

- **KV Backoff OPS** indicates the count of error messages per second returned by TiKV.

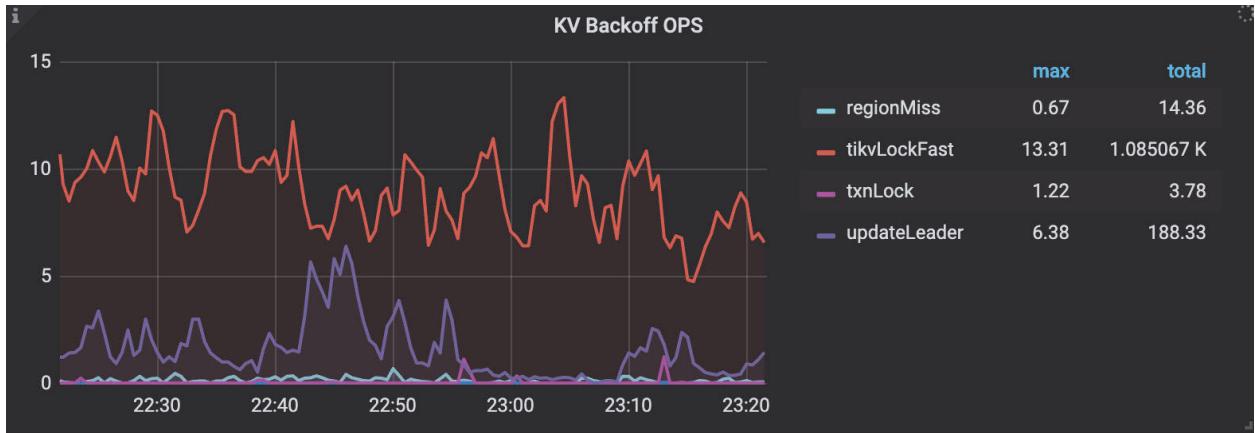


Figure 60: kv-backoff-ops

The `txnlock` metric indicates the write-write conflict. The `txnLockFast` metric indicates the read-write conflict.

- **Lock Resolve OPS** indicates the count of items related to transaction conflicts per second:



Figure 61: lock-resolve-ops

- `not_expired` indicates the TTL of the lock was not expired. The conflict transaction cannot resolve locks until the TTL is expired.
- `wait_expired` indicates that the transaction needs to wait the lock to expire.
- `expired` indicates the TTL of the lock was expired. Then the conflict transaction can resolve this lock.
- **KV Retry Duration** indicates the duration of re-sends the KV request:

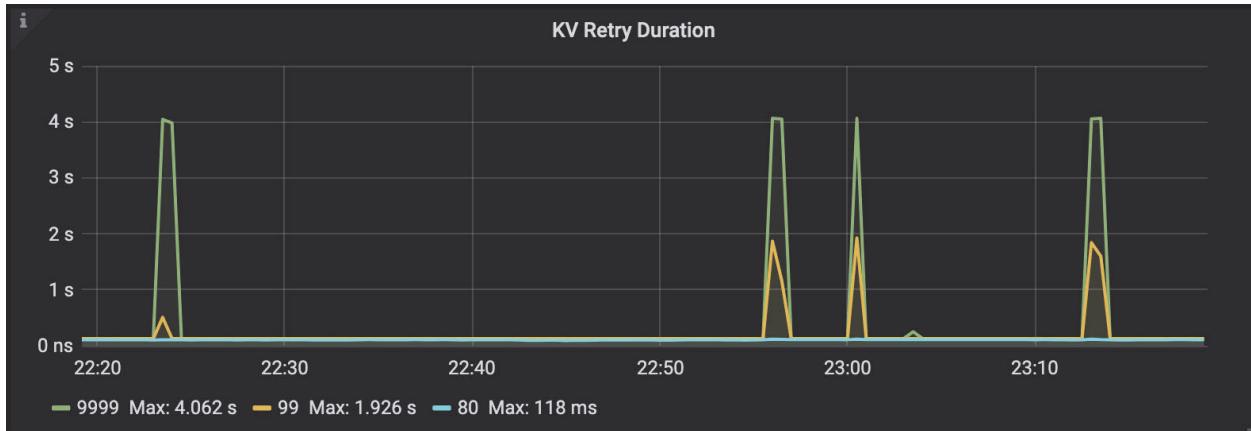


Figure 62: kv-retry-duration

You can also use `[kv:9007]Write conflict` as the keywords to search in the TiDB log. The keywords also indicate the write conflict exists in the cluster.

#### 7.14.3 Resolve write conflicts

If many write conflicts exist in the cluster, it is recommended to find out the write conflict key and the reason, and then try to change the application logic to avoid write conflicts. When the write conflict exists in the cluster, you can see the log similar to the following one in the TiDB log file:

```
[2020/05/12 15:17:01.568 +08:00] [WARN] [session.go:446] ["commit failed"] [
 ↳ conn=3] ["finished txn"="Txn{state=invalid}"] [error=" [kv:9007]Write
 ↳ conflict, txnStartTS=416617006551793665, conflictStartTS
 ↳ =416617018650001409, conflictCommitTS=416617023093080065, key={
 ↳ tableID=47, indexID=1, indexValues={string, }} primary={tableID=47,
 ↳ indexID=1, indexValues={string, }} [try again later] "]
```

The explanation of the log above is as follows:

- `[kv:9007]Write conflict`: indicates the write-write conflict.
- `txnStartTS=416617006551793665`: indicates the `start_ts` of the current transaction. You can use the `pd-ctl` tool to convert `start_ts` to physical time.
- `conflictStartTS=416617018650001409`: indicates the `start_ts` of the write conflict transaction.
- `conflictCommitTS=416617023093080065`: indicates the `commit_ts` of the write conflict transaction.
- `key={tableID=47, indexID=1, indexValues={string, }}`: indicates the write conflict key. `tableID` indicates the ID of the write conflict table. `indexID` indicates the ID of write conflict index. If the write conflict key is a record key, the log prints `handle=x`,

indicating which record(row) has a conflict. `indexValues` indicates the value of the index that has a conflict.

- `primary={tableID=47, indexID=1, indexValues={string, }}:` indicates the primary key information of the current transaction.

You can use the `pd-ctl` tool to convert the timestamp to readable time:

```
tiup ctl pd -u https://127.0.0.1:2379 tso {TIMESTAMP}
```

You can use `tableID` to find the name of the related table:

```
curl http://{TiDBIP}:10080/db-table/{tableID}
```

You can use `indexID` and the table name to find the name of the related index:

```
SELECT * FROM INFORMATION_SCHEMA.TIDB_INDEXES WHERE TABLE_SCHEMA='{db_name}'
 ↪ AND TABLE_NAME='{table_name}' AND INDEX_ID={indexID};
```

In addition, in TiDB v3.0.8 and later versions, the pessimistic transaction becomes the default mode. The pessimistic transaction mode can avoid write conflicts during the transaction prewrite stage, so you do not need to modify the application any more. In the pessimistic transaction mode, each DML statement writes a pessimistic lock to the related keys during execution. This pessimistic lock can prevent other transactions from modifying the same keys, thus ensuring no write conflicts exist in the prewrite stage of the transaction 2PC.

## 8 Performance Tuning

### 8.1 System Tuning

#### 8.1.1 Operating System Tuning

This document introduces how to tune each subsystem of CentOS 7.

##### Note:

- The default configuration of the CentOS 7 operating system is suitable for most services running under moderate workloads. Adjusting the performance of a particular subsystem might negatively affects other subsystems. Therefore, before tuning the system, back up all the user data and configuration information.
- Fully test all the changes in the test environment before applying them to the production environment.

### 8.1.1.1 Performance analysis methods

System tuning must be based on the results of system performance analysis. This section lists common methods for performance analysis.

#### 8.1.1.1.1 In 60 seconds

*Linux Performance Analysis in 60,000 Milliseconds* is published by the author Brendan Gregg and the Netflix Performance Engineering team. All tools used can be obtained from the official release of Linux. You can analyze outputs of the following list items to troubleshoot most common performance issues.

- `uptime`
- `dmesg | tail`
- `vmstat 1`
- `mpstat -P ALL 1`
- `pidstat 1`
- `iostat -xz 1`
- `free -m`
- `sar -n DEV 1`
- `sar -n TCP,ETCP 1`
- `top`

For detailed usage, see the corresponding `man` instructions.

#### 8.1.1.1.2 perf

`perf` is an important performance analysis tool provided by the Linux kernel, which covers hardware level (CPU/PMU, performance monitoring unit) features and software features (software counters, trace points). For detailed usage, see [perf Examples](#).

#### 8.1.1.1.3 BCC/bpftrace

Starting from CentOS 7.6, the Linux kernel has supported Berkeley Packet Filter (BPF). Therefore, you can choose proper tools to conduct an in-depth analysis based on the results in [In 60 seconds](#). Compared with `perf/ftrace`, BPF provides programmability and smaller performance overhead. Compared with kprobe, BPF provides higher security and is more suitable for the production environments. For detailed usage of the BCC toolkit, see [BPF Compiler Collection \(BCC\)](#).

### 8.1.1.2 Performance tuning

This section introduces performance tuning based on the classified kernel subsystems.

#### 8.1.1.2.1 CPU—frequency scaling

`cpufreq` is a module that dynamically adjusts the CPU frequency. It supports five modes. To ensure service performance, select the performance mode and fix the CPU frequency at the highest supported operating frequency without dynamic adjustment. The command for this operation is `cpupower frequency-set --governor performance`.

#### 8.1.1.2.2 CPU—interrupt affinity

- Automatic balance can be implemented through the `irqbalance` service.
- Manual balance:
  - Identify the devices that need to balance interrupts. Starting from CentOS 7.5, the system automatically configures the best interrupt affinity for certain devices and their drivers, such as devices that use the `be2iscsi` driver and NVMe settings. You can no longer manually configure interrupt affinity for such devices.
  - For other devices, check the chip manual to see whether these devices support distributing interrupts.
    - \* If they do not, all interrupts of these devices are routed to the same CPU and cannot be modified.
    - \* If they do, calculate the `smp_affinity` mask and set the corresponding configuration file. For details, see the [kernel document](#).

#### 8.1.1.2.3 NUMA CPU binding

To avoid accessing memory across Non-Uniform Memory Access (NUMA) nodes as much as possible, you can bind a thread/process to certain CPU cores by setting the CPU affinity of the thread. For ordinary programs, you can use the `numactl` command for the CPU binding. For detailed usage, see the Linux manual pages. For network interface card (NIC) interrupts, see [tune network](#).

#### 8.1.1.2.4 Memory—transparent huge page (THP)

It is **NOT** recommended to use THP for database applications, because databases often have sparse rather than continuous memory access patterns. If high-level memory fragmentation is serious, a higher latency will occur when THP pages are allocated. If the direct compaction is enabled for THP, the CPU usage will surge. Therefore, it is recommended to disable THP.

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

#### 8.1.1.2.5 Memory—virtual memory parameters

- **dirty\_ratio** percentage ratio. When the total amount of dirty page caches reach this percentage ratio of the total system memory, the system starts to use the `pdflush` operation to write the dirty page caches to disk. The default value of `dirty_ratio` is 20% and usually does not need adjustment. For high-performance SSDs such as NVMe devices, lowering this value helps improve the efficiency of memory reclamation.
- **dirty\_background\_ratio** percentage ratio. When the total amount of dirty page caches reach this percentage ratio of the total system memory, the system starts to write the dirty page caches to the disk in the background. The default value of `dirty_ratio` is 10% and usually does not need adjustment. For high-performance SSDs such as NVMe devices, setting a lower value helps improve the efficiency of memory reclamation.

#### 8.1.1.2.6 Storage and file system

The core I/O stack link is long, including the file system layer, the block device layer, and the driver layer.

##### I/O scheduler

The I/O scheduler determines when and how long I/O operations run on the storage device. It is also called I/O elevator. For SSD devices, it is recommended to set the I/O scheduling policy to `noop`.

```
echo noop > /sys/block/${SSD_DEV_NAME}/queue/scheduler
```

##### Formatting parameters—block size

Blocks are the working units of the file system. The block size determines how much data can be stored in a single block, and thus determines the minimum amount of data to be written or read each time.

The default block size is suitable for most scenarios. However, if the block size (or the size of multiple blocks) is the same or slightly larger than the amount of data normally read or written each time, the file system performs better and the data storage efficiency is higher. Small files still use the entire block. Files can be distributed among multiple blocks, but this will increase runtime overhead.

When using the `mkfs` command to format a device, specify the block size as a part of the file system options. The parameters that specify the block size vary with the file system. For details, see the corresponding `mkfs` manual pages, such as using `man mkfs.ext4`.

##### mount parameters

If the `noatime` option is enabled in the `mount` command, the update of metadata is disabled when files are read. If the `nodiratime` behavior is enabled, the update of metadata is disabled when the directory is read.

#### 8.1.1.2.7 Network tuning

The network subsystem consists of many different parts with sensitive connections. The CentOS 7 network subsystem is designed to provide the best performance for most workloads and automatically optimizes the performance of these workloads. Therefore, usually you do not need to manually adjust network performance.

Network issues are usually caused by issues of hardware or related devices. So before tuning the protocol stack, rule out hardware issues.

Although the network stack is largely self-optimizing, the following aspects in the network packet processing might become the bottleneck and affect performance:

- NIC hardware cache: To correctly observe the packet loss at the hardware level, use the `ethtool -S ${NIC_DEV_NAME}` command to observe the `drops` field. When packet loss occurs, it might be that the processing speed of the hard/soft interrupts cannot catch up with the receiving speed of NIC. If the received buffer size is less than the upper limit, you can also try to increase the RX buffer to avoid packet loss. The query command is: `ethtool -g ${NIC_DEV_NAME}`, and the modification command is `ethtool -G ${NIC_DEV_NAME}`.
- Hardware interrupts: If the NIC supports the Receive-Side Scaling (RSS, also called multi-NIC receiving) feature, observe the `/proc/interrupts` NIC interrupts. If the interrupts are uneven, see [CPU—frequency scaling](#), [CPU—interrupt affinity](#), and [NUMA CPU binding](#). If the NIC does not support RSS or the number of RSS is much smaller than the number of physical CPU cores, configure Receive Packet Steering (RPS, which can be regarded as the software implementation of RSS), and the RPS extension Receive Flow Steering (RFS). For detailed configuration, see the [kernel document](#).
- Software interrupts: Observe the monitoring of `/proc/net/softnet_stat`. If the values of the other columns except the third column are increasing, properly adjust the value of `net.core.netdev_budget` or `net.core.dev_weight` for `softirq` to get more CPU time. In addition, you also need to check the CPU usage to determine which tasks are frequently using the CPU and whether they can be optimized.
- Receive queue of application sockets: Monitor the `Resv-q` column of `ss -nmp`. If the queue is full, consider increasing the size of the application socket cache or use the automatic cache adjustment method. In addition, consider whether you can optimize the architecture of the application layer and reduce the interval between reading sockets.
- Ethernet flow control: If the NIC and switch support the flow control feature, you can use this feature to leave some time for the kernel to process the data in the NIC queue, to avoid the issue of NIC buffer overflow.
- Interrupts coalescing: Too frequent hardware interrupts reduces system performance, and too late hardware interrupts causes packet loss. Newer NICs support the interrupt coalescing feature and allow the driver to automatically adjust the number of hardware interrupts. You can execute `ethtool -c ${NIC_DEV_NAME}` to check and `ethtool -c ${NIC_DEV_NAME}` to enable this feature. The adaptive mode allows the NIC

to automatically adjust the interrupt coalescing. In this mode, the driver checks the traffic mode and kernel receiving mode, and evaluates the coalescing settings in real time to prevent packet loss. NICs of different brands have different features and default configurations. For details, see the NIC manuals.

- Adapter queue: Before processing the protocol stack, the kernel uses this queue to buffer the data received by the NIC, and each CPU has its own backlog queue. The maximum number of packets that can be cached in this queue is `netdev_max_backlog` ↗ . Observe the second column of `/proc/net/softnet_stat`. When the second column of a row continues to increase, it means that the CPU [row-1] queue is full and the data packet is lost. To resolve this problem, continue to double the `net.core.netdev_max_backlog` value.
- Send queue: The length value of a send queue determines the number of packets that can be queued before sending. The default value is 1000, which is sufficient for 10 Gbps. But if you have observed the value of TX errors from the output of `ip -s link`, you can try to double it: `ip link set dev ${NIC_DEV_NAME} txqueuelen 2000`.
- Driver: NIC drivers usually provide tuning parameters. See the device hardware manual and its driver documentation.

## 8.2 Software Tuning

### 8.2.1 Configuration

#### 8.2.1.1 TiDB Memory Control

Currently, TiDB can track the memory quota of a single SQL query and take actions to prevent OOM (out of memory) or troubleshoot OOM when the memory usage exceeds a specific threshold value. In the TiDB configuration file, you can configure the options as below to control TiDB behaviors when the memory quota exceeds the threshold value:

```
Valid options: ["log", "cancel"]
oom-action = "cancel"
```

- If the configuration item above uses “log”, when the memory quota of a single SQL query exceeds the threshold value which is controlled by the `tidb_mem_quota_query` variable, TiDB prints an entry of log. Then the SQL query continues to be executed. If OOM occurs, you can find the corresponding SQL query in the log.
- If the configuration item above uses “cancel”, when the memory quota of a single SQL query exceeds the threshold value, TiDB stops executing the SQL query immediately and returns an error to the client. The error information clearly shows the memory usage of each physical execution operator that consumes much memory in the SQL execution process.

### 8.2.1.1.1 Configure the memory quota of a query

In the configuration file, you can set the default Memory Quota for each Query. The following example sets it to 32GB:

```
mem-quota-query = 34359738368
```

In addition, you can control the memory quota of a query using the following session variables. Generally, you only need to configure `tidb_mem_quota_query`. Other variables are used for advanced configuration which most users do not need to care about.

| Variable Name                               | Description                                       | Default Unit | Value               |
|---------------------------------------------|---------------------------------------------------|--------------|---------------------|
| <code>tidb_mem_quota_query</code>           | Control the memory quota of a query               | Byte         | $1 << 30$<br>(1 GB) |
| <code>tidb_mem_quota_hashjoin</code>        | Control the memory quota of “HashJoinExec”        | Byte         | $32 << 30$          |
| <code>tidb_mem_quota_mergejoin</code>       | Control the memory quota of “MergeJoinExec”       | Byte         | $32 << 30$          |
| <code>tidb_mem_quota_sort</code>            | Control the memory quota of “SortExec”            | Byte         | $32 << 30$          |
| <code>tidb_mem_quota_topn</code>            | Control the memory quota of “TopNExec”            | Byte         | $32 << 30$          |
| <code>tidb_mem_quota_indexlookupjoin</code> | Control the memory quota of “IndexLookUpExecutor” | Byte         | $32 << 30$          |
| <code>tidb_mem_quota_indexlookjoin</code>   | Control the memory quota of “IndexLookUpJoin”     | Byte         | $32 << 30$          |
| <code>tidb_mem_quota_nestedloopjoin</code>  | Control the memory quota of “NestedLoopApplyExec” | Byte         | $32 << 30$          |

Some usage examples:

```
-- Set the threshold value of memory quota for a single SQL query to 8GB:
set @@tidb_mem_quota_query = 8 << 30;
```

```
-- Set the threshold value of memory quota for a single SQL query to 8MB:
set @@tidb_mem_quota_query = 8 << 20;
```

```
-- Set the threshold value of memory quota for a single SQL query to 8KB:
set @@tidb_mem_quota_query = 8 << 10;
```

### 8.2.1.1.2 Configure the memory usage threshold of a tidb-server instance

In the TiDB configuration file, you can set the memory usage threshold of a tidb-server instance by configuring `server-memory-quota`.

The following example sets the total memory usage of a tidb-server instance to 32 GB:

```
[performance]
server-memory-quota = 34359738368
```

In this configuration, when the memory usage of a tidb-server instance reaches 32 GB, the instance starts to kill running SQL statements randomly until the memory usage drops below 32 GB. SQL operations that are forced to terminate return an `Out Of Global Memory ↩ Limit!` error message to the client.

#### Warning:

- `server-memory-quota` is still an experimental feature. It is **NOT** recommended that you use it in a production environment.
- The default value of `server-memory-quota` is 0, which means no memory limit.

#### 8.2.1.1.3 Trigger the alarm of excessive memory usage

In the default configuration, a tidb-server instance prints an alarm log and records associated status files when the machine memory usage reaches 80% of its total memory. You can set the memory usage ratio threshold by configuring `memory-usage-alarm-ratio`. For detailed alarm rules, refer to the description of `memory-usage-alarm-ratio`.

Note that after the alarm is triggered once, it will be triggered again only if the memory usage rate has been below the threshold for more than ten seconds and reaches the threshold again. In addition, to avoid storing excessive status files generated by alarms, currently, TiDB only retains the status files generated during the recent five alarms.

The following example constructs a memory-intensive SQL statement that triggers the alarm:

1. Set `memory-usage-alarm-ratio` to 0.8:

```
mem-quota-query = 34359738368 // Increases the memory limit of each
 ↩ query to construct SQL statements that take up larger memory.
[performance]
memory-usage-alarm-ratio = 0.8
```

2. Execute `CREATE TABLE t(a int);` and insert 1000 rows of data.
3. Execute `select * from t t1 join t t2 join t t3 order by t1.a.` This SQL statement outputs one billion records, which consumes a large amount of memory and therefore triggers the alarm.

4. Check the `tidb.log` file which records the total system memory, current system memory usage, memory usage of the tidb-server instance, and the directory of status files.

```
[2020/11/30 15:25:17.252 +08:00] [WARN] [memory_usage_alarm.go:141] [
 ↪ tidb-server has the risk of OOM. Running SQLs and heap profile
 ↪ will be recorded in record path"] ["is server-memory-quota set"=
 ↪ false] ["system memory total"=33682427904] ["system memory usage
 ↪ "=27142864896] ["tidb-server memory usage"=22417922896] [memory-
 ↪ usage-alarm-ratio=0.8] ["record path"="/tmp/1000_tidb/
 ↪ MC4wLjAuMDo0MDAwLzAuMC4wLjA6MTAwODA=/tmp-storage/record"]
```

The fields of the example log file above are described as follows:

- `is server-memory-quota set` indicates whether `server-memory-quota` is set.
- `system memory total` indicates the total memory of the current system.
- `system memory usage` indicates the current system memory usage.
- `tidb-server memory usage` indicates the memory usage of the tidb-server instance.
- `memory-usage-alarm-ratio` indicates the value of `memory-usage-alarm-ratio`.
- `record path` indicates the directory of status files.

5. You can see a set of files in the directory of status files (In the above example, the directory is `/tmp/1000_tidb/MC4wLjAuMDo0MDAwLzAuMC4wLjA6MTAwODA=/tmp-storage/record`), including `goroutine`, `heap`, and `running_sql`. These three files are suffixed with the time when status files are logged. They respectively record goroutine stack information, the usage status of heap memory, and the running SQL information when the alarm is triggered. For the format of log content in `running_sql`, refer to [expensive-queries](#).

#### 8.2.1.1.4 Other memory control behaviors of tidb-server

##### Flow control

- TiDB supports dynamic memory control for the operator that reads data. By default, this operator uses the maximum number of threads that `tidb_disql_scan_concurrency` allows to read data. When the memory usage of a single SQL execution exceeds `tidb_mem_quota_query` each time, the operator that reads data stops one thread.
- This flow control behavior is controlled by the system variable `tidb_enable_rate_limit_action`.
- When the flow control behavior is triggered, TiDB outputs a log containing the keywords `memory exceeds quota`, `destroy one token now`.

##### Disk spill

TiDB supports disk spill for execution operators. When the memory usage of a SQL execution exceeds the memory quota, tidb-server can spill the intermediate data of execution operators to the disk to relieve memory pressure. Operators supporting disk spill include Sort, MergeJoin, HashJoin, and HashAgg.

- The disk spill behavior is jointly controlled by the `mem-quota-query`, `oom-use-tmp-storage`, `tmp-storage-path`, and `tmp-storage-quota` parameters.
- When the disk spill is triggered, TiDB outputs a log containing the keywords `memory exceeds quota, spill to disk now` or `memory exceeds quota, set aggregate mode to spill-mode`.
- Disk spill for the Sort, MergeJoin, and HashJoin operator is introduced in v4.0.0; disk spill for the HashAgg operator is introduced in v5.2.0.
- When the SQL executions containing Sort, MergeJoin, or HashJoin cause OOM, TiDB triggers disk spill by default. When SQL executions containing HashAgg cause OOM, TiDB does not trigger disk spill by default. You can configure the system variable `tidb_executor_concurrency = 1` to trigger disk spill for HashAgg.

#### Note:

The disk spill for HashAgg does not support SQL executions containing the `DISTINCT` aggregate function. When a SQL execution containing a `DISTINCT` aggregate function uses too much memory, the disk spill does not apply.

The following example uses a memory-consuming SQL statement to demonstrate the disk spill feature for HashAgg:

1. Configure the memory quota of a SQL statement to 1GB (1 GB by default):

```
set tidb_mem_quota_query = 1 << 30;
```

2. Create a single table `CREATE TABLE t(a int)`; and insert 256 rows of different data.
3. Execute the following SQL statement:

```
[tidb]> explain analyze select /*+ HASH_AGG() */ count(*) from t t1
 ↪ join t t2 join t t3 group by t1.a, t2.a, t3.a;
```

Because executing this SQL statement occupies too much memory, the following “Out of Memory Quota” error message is returned:

```
ERROR 1105 (HY000): Out Of Memory Quota! [conn_id=3]
```

4. Configure the system variable `tidb_executor_concurrency` to 1. With this configuration, when out of memory, HashAgg automatically tries to trigger disk spill.

```
set tidb_executor_concurrency = 1;
```

5. Execute the same SQL statement. You can find that this time, the statement is successfully executed and no error message is returned. From the following detailed execution plan, you can see that HashAgg has used 600 MB of hard disk space.

```
[tidb]> explain analyze select /*+ HASH_AGG() */ count(*) from t t1
 ↪ join t t2 join t t3 group by t1.a, t2.a, t3.a;
```

| Execution Plan         |                                                               |             |                                                                 |           |
|------------------------|---------------------------------------------------------------|-------------|-----------------------------------------------------------------|-----------|
| id                     | object                                                        | estRows     | actRows                                                         | task      |
|                        | execution info                                                |             |                                                                 | access    |
|                        | operator info                                                 |             |                                                                 |           |
|                        | memory                                                        | disk        |                                                                 |           |
| HashAgg_11             |                                                               | 204.80      | 16777216                                                        | root      |
|                        |                                                               |             | time:1m37.4s, loops:16385                                       |           |
|                        | group by:test.t.a, test.t.a, test.t.a, funcs:count(1)->Column |             |                                                                 |           |
|                        | #7                                                            | 1.13 GB     | 600.0 MB                                                        |           |
| -HashJoin_12           |                                                               | 16777216.00 | 16777216                                                        | root      |
|                        |                                                               |             | time:21.5s, loops:16385, build_hash_table:{total                |           |
|                        |                                                               |             | :267.2μs, fetch:228.9μs, build:38.2μs}, probe:{concurrency:1,   |           |
|                        |                                                               |             | total:35s, max:35s, probe:35s, fetch:962.2μs}   CARTESIAN inner |           |
|                        | join                                                          | 8.23 KB     | 4 KB                                                            |           |
| -TableReader_21(Build) |                                                               | 256.00      | 256                                                             | root      |
|                        |                                                               |             | time:87.2μs, loops:2, cop_task: {num: 1, max: 150               |           |
|                        |                                                               |             | μs, proc_keys: 0, rpc_num: 1, rpc_time: 145.1μs,                |           |
|                        |                                                               |             | copr_cache_hit_ratio: 0.00}   data:                             |           |
|                        | TableFullScan_20                                              |             |                                                                 | 885 Bytes |
|                        | N/A                                                           |             |                                                                 |           |
| -TableFullScan_20      |                                                               | 256.00      | 256                                                             | cop[tikv] |
|                        | table:t3                                                      |             | tikv_task:{time:23.2μs, loops:256}                              |           |
|                        |                                                               |             |                                                                 |           |
|                        | keep order:false, stats:pseudo                                |             |                                                                 | N/A       |
|                        | N/A                                                           |             |                                                                 |           |
| -HashJoin_14(Probe)    |                                                               | 65536.00    | 65536                                                           | root      |
|                        |                                                               |             | time:728.1μs, loops:65, build_hash_table:{total                 |           |

```

 ↵ :307.5µs, fetch:277.6µs, build:29.9µs}, probe:{concurrency:1,
 ↵ total:34.3s, max:34.3s, probe:34.3s, fetch:278µs} | CARTESIAN
 ↵ inner join | 8.23 KB | 4 KB |
| - TableReader_19(Build) | 256.00 | 256 | root |
 ↵ | time:126.2µs, loops:2, cop_task: {num: 1, max:
 ↵ 308.4µs, proc_keys: 0, rpc_num: 1, rpc_time: 295.3µs,
 ↵ copr_cache_hit_ratio: 0.00} | data:TableFullScan_18
 ↵ | 885 Bytes | N/A |
| - TableFullScan_18 | 256.00 | 256 | cop[tikv] |
 ↵ table:t2 | tikv_task:{time:79.2µs, loops:256}
 ↵
 ↵ | keep order:false, stats:pseudo | N/A
 ↵ | N/A |
| - TableReader_17(Probe) | 256.00 | 256 | root |
 ↵ | time:211.1µs, loops:2, cop_task: {num: 1, max:
 ↵ 295.5µs, proc_keys: 0, rpc_num: 1, rpc_time: 279.7µs,
 ↵ copr_cache_hit_ratio: 0.00} | data:TableFullScan_16
 ↵ | 885 Bytes | N/A |
| - TableFullScan_16 | 256.00 | 256 | cop[tikv] |
 ↵ table:t1 | tikv_task:{time:71.4µs, loops:256}
 ↵
 ↵ | keep order:false, stats:pseudo | N/A
 ↵ | N/A |
+--+
 ↵ -----
 ↵
 ↵ 9 rows in set (1 min 37.428 sec)

```

### 8.2.1.2 Tune TiKV Thread Pool Performance

This document introduces TiKV internal thread pools and how to tune their performance.

#### 8.2.1.2.1 Thread pool introduction

The TiKV thread pool is mainly composed of gRPC, Scheduler, UnifyReadPool, Raftstore, StoreWriter, Apply, RocksDB, and some scheduled tasks and detection components that do not consume much CPU. This document mainly introduces a few CPU-intensive thread pools that affect the performance of read and write requests.

- The gRPC thread pool: it handles all network requests and forwards requests of different task types to different thread pools.
- The Scheduler thread pool: it detects write transaction conflicts, converts requests like the two-phase commit, pessimistic locking, and transaction rollbacks into key-value pair arrays, and then sends them to the Raftstore thread for Raft log replication.

- The Raftstore thread pool:
  - It processes all Raft messages and the proposal to add a new log.
  - It writes Raft logs to the disk. If the value of `store-io-pool-size` is 0, the Raftstore thread writes the logs to the disk; if the value is not 0, the Raftstore thread sends the logs to the StoreWriter thread.
  - When Raft logs in the majority of replicas are consistent, the Raftstore thread sends the logs to the Apply thread.
- The StoreWriter thread pool: it writes all Raft logs to the disk and returns the result to the Raftstore thread.
- The Apply thread pool: it receives the submitted log sent from the Raftstore thread pool, parses it as a key-value request, then writes it to RocksDB, calls the callback function to notify the gRPC thread pool that the write request is complete, and returns the result to the client.
- The RocksDB thread pool: it is a thread pool for RocksDB to compact and flush tasks. For RocksDB's architecture and `Compact` operation, refer to [RocksDB: A Persistent Key-Value Store for Flash and RAM Storage](#).
- The UnifyReadPool thread pool: it is a combination of the Coprocessor thread pool and Storage Read Pool. All read requests such as kv get, kv batch get, raw kv get, and coprocessor are executed in this thread pool.

#### 8.2.1.2.2 TiKV read-only requests

TiKV's read requests are divided into the following types:

- Simple queries that specify a certain row or several rows, running in the Storage Read Pool.
- Complex aggregate calculation and range queries, running in the Coprocessor Read Pool.

Starting from TiKV v5.0, all read requests use the unified thread pool for queries by default. If your TiKV cluster is upgraded from TiKV v4.0 and the `use-unified-pool` configuration of `readpool.storage` was set to `false` before the upgrade, all read requests continue using different thread pools after the upgrade. In this scenario, to make all read requests use the unified thread pool for queries, you can set the value of `readpool.storage`  $\hookrightarrow$  `.use-unified-pool` to `true`.

#### 8.2.1.2.3 Performance tuning for TiKV thread pools

- The gRPC thread pool.

The default size (configured by `server.grpc-concurrency`) of the gRPC thread pool is 5. This thread pool has almost no computing overhead and is mainly responsible for network I/O and deserialization requests, so generally you do not need to adjust the default configuration.

- If the machine deployed with TiKV has a small number (less than or equal to 8) of CPU cores, consider setting the `server.grpc-concurrency` configuration item to 2.
- If the machine deployed with TiKV has very high configuration, TiKV undertakes a large number of read and write requests, and the value of `gRPC poll CPU` that monitors Thread CPU on Grafana exceeds 80% of `server.grpc-concurrency` → , then consider increasing the value of `server.grpc-concurrency` to keep the thread pool usage rate below 80% (that is, the metric on Grafana is lower than `80% * server.grpc-concurrency`).
- The Scheduler thread pool.

When TiKV detects that the number of machine CPU cores is larger than or equal to 16, the default size (configured by `storage.scheduler-worker-pool-size`) of the Scheduler thread pool is 8; when TiKV detects that the number of machine CPU cores is smaller than 16, the default size is 4.

This thread pool is mainly used to convert complex transaction requests into simple key-value read and write requests. However, **the Scheduler thread pool itself does not perform any write operation.**

- If it detects a transaction conflict, then this thread pool returns the conflict result to the client in advance.
- If no conflict is detected, then this thread pool merges the key-value requests that perform write operations into a Raft log and sends it to the Raftstore thread for Raft log replication.

Generally speaking, to avoid excessive thread switching, it is best to ensure that the utilization rate of the Scheduler thread pool is between 50% and 75%. If the thread pool size is 8, then it is recommended to keep `TiKV-Details.Thread CPU.scheduler worker` CPU on Grafana between 400% and 600%.

- The Raftstore thread pool.

The Raftstore thread pool is the most complex thread pool in TiKV. The default size (configured by `raftstore.store-pool-size`) of this thread pool is 2. For the StoreWriter thread pool, the default size (configured by `raftstore.store-io-pool-size`) is 0.

- When the size of the StoreWriter thread pool is 0, all write requests are written into RocksDB in the way of `fsync` by the Raftstore thread. In this case, it is recommended to tune the performance as follows:

- \* Keep the overall CPU usage of the Raftstore thread below 60%. When the number of Raftstore threads is 2, keep the **TiKV-Details, Thread CPU, Raft store CPU** on Grafana below 120%. Due to I/O requests, the CPU usage of Raftstore threads in theory is always lower than 100%.
- \* Do not increase the size of the Raftstore thread pool to improve write performance without careful consideration, because this might increase the disk burden and degrade performance.
- When the size of the StoreWriter thread pool is not 0, all write requests are written into RocksDB in the way of `fsync` by the StoreWriter thread. In this case, it is recommended to tune the performance as follows:
  - \* Enable the StoreWriter thread pool ONLY when the overall CPU resources are sufficient. When the StoreWriter thread pool is enabled, keep the CPU usage of the StoreWriter thread and the Raftstore thread below 80%.
- Compared with the case that the write requests are processed by the Raftstore thread, in theory, when the write requests are processed by the StoreWriter thread, write latency and the tail latency of data read are significantly reduced. However, as the write speed grows faster, the number of Raft logs increases accordingly. This can cause the CPU overhead of the Raftstore threads, the Apply threads, and the gRPC threads to increase. In this case, insufficient CPU resources might offset the tuning effect, and as a result, the write speed might become slower than before. Therefore, if the CPU resources are not sufficient, it is not recommended to enable the StoreWriter thread. Because the Raftstore thread sends most of the I/O requests to the StoreWriter thread, you need to keep the CPU usage of the Raftstore thread below 80%.
- In most cases, set the size of the StoreWriter thread pool to 1 or 2. This is because the size of the StoreWriter thread pool affects the number of Raft logs, so the value of the thread pool size should not be too large. If the CPU usage is higher than 80%, consider increasing the thread pool size.
- Pay attention to the impact of increasing Raft logs on the CPU overhead of other thread pools. If necessary, you need to increase the number of Raftstore threads, Apply threads, and gRPC threads accordingly.
- The UnifyReadPool thread pool.

The UnifyReadPool is responsible for handling all read requests. The default size (configured by `readpool.unified.max-thread-count`) is 80% of the number of the machine's CPU cores. For example, if the machine CPU has 16 cores, the default thread pool size is 12. It is recommended to adjust the CPU usage rate according to the application workloads and keep it between 60% and 90% of the thread pool size.

If the peak value of the `TiKV-Details.Thread CPU.Unified read pool` CPU on Grafana does not exceed 800%, then it is recommended to set `readpool.unified → .max-thread-count` to 10. Too many threads can cause more frequent thread switching, and take up resources of other thread pools.

- The RocksDB thread pool.

The RocksDB thread pool is a thread pool for RocksDB to compact and flush tasks. Usually, you do not need to configure it.

- If the machine has a small number of CPU cores, set both `rocksdb.max-  
background-jobs` and `raftdb.max-background-jobs` to 4.
- If you encounter write stall, go to Write Stall Reason in **RocksDB-kv** on Grafana and check on the metrics that are not 0.
  - \* If it is caused by reasons related to pending compaction bytes, set `rocksdb.  
.max-sub-compactions` to 2 or 3. This configuration item indicates the number of sub-threads allowed for a single compaction job. Its default value is 3 in TiKV 4.0 and 1 in TiKV 3.0.
  - \* If the reason is related to memtable count, it is recommended to increase the `max-write-buffer-number` of all columns (5 by default).
  - \* If the reason is related to the level0 file limit, it is recommended to increase values of the following parameters to 64 or a larger number:

```
rocksdb.defaultcf.level0-slowdown-writes-trigger
rocksdb.writecf.level0-slowdown-writes-trigger
rocksdb.lockcf.level0-slowdown-writes-trigger
rocksdb.defaultcf.level0-stop-writes-trigger
rocksdb.writecf.level0-stop-writes-trigger
rocksdb.lockcf.level0-stop-writes-trigger
```

#### 8.2.1.3 Tune TiKV Memory Parameter Performance

This document describes how to tune the TiKV parameters for optimal performance.

TiKV uses RocksDB for persistent storage at the bottom level of the TiKV architecture. Therefore, many of the performance parameters are related to RocksDB. TiKV uses two RocksDB instances: the default RocksDB instance stores KV data, the Raft RocksDB instance (RaftDB) stores Raft logs.

TiKV implements **Column Families** (CF) from RocksDB.

- The default RocksDB instance stores KV data in the `default`, `write` and `lock` CFs.
  - The `default` CF stores the actual data. The corresponding parameters are in `[rocksdb.defaultcf]`.
  - The `write` CF stores the version information in Multi-Version Concurrency Control (MVCC) and index-related data. The corresponding parameters are in `[  
rocksdb.writecf]`.
  - The `lock` CF stores the lock information. The system uses the default parameters.
- The Raft RocksDB (RaftDB) instance stores Raft logs.

- The default CF stores the Raft log. The corresponding parameters are in [  
→ `raftdb.defaultcf`].

After TiKV 3.0, by default, all CFs share one block cache instance. You can configure the size of the cache by setting the `capacity` parameter under `[storage.block-cache]`. The bigger the block cache, the more hot data can be cached, and the easier to read data, in the meantime, the more system memory is occupied. To use a separate block cache instance for each CF, set `shared=false` under `[storage.block-cache]`, and configure individual block cache size for each CF. For example, you can configure the size of `write` CF by setting the `block-cache-size` parameter under `[rocksdb.writecf]`.

Before TiKV 3.0, shared block cache is not supported, and you need to configure block cache for each CF individually.

Each CF also has a separate `write buffer`. You can configure the size by setting the `write-buffer-size` parameter.

#### 8.2.1.3.1 Parameter specification

```
Log level: trace, debug, warn, error, info, off.
log-level = "info"

[server]
Set listening address
addr = "127.0.0.1:20160"

Size of thread pool for gRPC
grpc-concurrency = 4
The number of gRPC connections between each TiKV instance
grpc-raft-conn-num = 10

Most read requests from TiDB are sent to the coprocessor of TiKV. This
 → parameter is used to set the number of threads
of the coprocessor. If many read requests exist, add the number of
 → threads and keep the number within that of the
system CPU cores. For example, for a 32-core machine deployed with TiKV
 → , you can even set this parameter to 30 in
repeatable read scenarios. If this parameter is not set, TiKV
 → automatically sets it to CPU cores * 0.8.
end-point-concurrency = 8

Tag the TiKV instances to schedule replicas.
labels = {zone = "cn-east-1", host = "118", disk = "ssd"}

[storage]
The data directory
```

```

data-dir = "/tmp/tikv/store"

In most cases, you can use the default value. When importing data, it
 ↪ is recommended to set the parameter to 1024000.
scheduler-concurrency = 102400
This parameter controls the number of write threads. When write
 ↪ operations occur frequently, set this parameter value
higher. Run `top -H -p tikv-pid` and if the threads named `sched-worker
 ↪ -pool` are busy, set the value of parameter
`scheduler-worker-pool-size` higher and increase the number of write
 ↪ threads.
scheduler-worker-pool-size = 4

[storage.block-cache]
Whether to create a shared block cache for all RocksDB column families
 ↪ .
Block cache is used by RocksDB to cache uncompressed blocks. Big
 ↪ block cache can speed up read.
It is recommended to turn on shared block cache. Since only the total
 ↪ cache size need to be
set, it is easier to configure. In most cases, it should be able to
 ↪ auto-balance cache usage
between column families with standard LRU algorithm.
The rest of config in the storage.block-cache session is effective
 ↪ only when shared block cache
is on.
shared = true

Size of the shared block cache. Normally it should be tuned to 30%-50%
 ↪ of system's total memory.
When the config is not set, it is decided by the sum of the following
 ↪ fields or their default
value:
* rocksdb.defaultcf.block-cache-size or 25% of system's total memory
* rocksdb.writecf.block-cache-size or 15% of system's total memory
* rocksdb.lockcf.block-cache-size or 2% of system's total memory
* raftdb.defaultcf.block-cache-size or 2% of system's total memory
To deploy multiple TiKV nodes on a single physical machine,
 ↪ configure this parameter explicitly.
Otherwise, the OOM problem might occur in TiKV.
capacity = "1GB"

[pd]
PD address
endpoints = ["127.0.0.1:2379", "127.0.0.2:2379", "127.0.0.3:2379"]

```

```

[metric]
The interval of pushing metrics to Prometheus Pushgateway
interval = "15s"
Prometheus Pushgateway address
address = ""
job = "tikv"

[raftstore]
Raft RocksDB directory. The default value is Raft subdirectory of [
 ↪ storage.data-dir].
If there are multiple disks on the machine, store the data of Raft
 ↪ RocksDB on different disks to improve TiKV performance.
raftdb-path = "/tmp/tikv/store/raft"

region-max-size = "384MB"
The threshold value of Region split
region-split-size = "256MB"
When the data size change in a Region is larger than the threshold
 ↪ value, TiKV checks whether this Region needs split.
To reduce the costs of scanning data in the checking process, set the
 ↪ value to 32MB during checking and set it to
the default value in normal operation.
region-split-check-diff = "32MB"

[rocksdb]
The maximum number of threads of RocksDB background tasks. The
 ↪ background tasks include compaction and flush.
For detailed information why RocksDB needs to implement compaction, see
 ↪ RocksDB-related materials. When write
traffic (like the importing data size) is big, it is recommended to
 ↪ enable more threads. But set the number of the enabled
threads smaller than that of CPU cores. For example, when importing
 ↪ data, for a machine with a 32-core CPU,
set the value to 28.
max-background-jobs = 8

The maximum number of file handles RocksDB can open
max-open-files = 40960

The file size limit of RocksDB MANIFEST. For more details, see https://
 ↪ github.com/facebook/rocksdb/wiki/MANIFEST
max-manifest-file-size = "20MB"

The directory of RocksDB write-ahead logs. If there are two disks on

```

```

 ↪ the machine, store the RocksDB data and WAL logs
on different disks to improve TiKV performance.
wal-dir = "/tmp/tikv/store"

Use the following two parameters to deal with RocksDB archiving WAL.
For more details, see https://github.com/facebook/rocksdb/wiki/How-to-
 ↪ persist-in-memory-RocksDB-database%3F
wal-ttl-seconds = 0
wal-size-limit = 0

In most cases, set the maximum total size of RocksDB WAL logs to the
 ↪ default value.
max-total-wal-size = "4GB"

Use this parameter to enable or disable the statistics of RocksDB.
enable-statistics = true

Use this parameter to enable the readahead feature during RocksDB
 ↪ compaction. If you are using mechanical disks, it is recommended to
 ↪ set the value to 2MB at least.
compaction-readahead-size = "2MB"

[rocksdb.defaultcf]
The data block size. RocksDB compresses data based on the unit of block
 ↪ .
Similar to page in other databases, block is the smallest unit cached
 ↪ in block-cache.
block-size = "64KB"

The compaction mode of each layer of RocksDB data. The optional values
 ↪ include no, snappy, zlib,
bzip2, lz4, lz4hc, and zstd.
"no:no:lz4:lz4:zstd:zstd" indicates there is no compaction of
 ↪ level0 and level1; lz4 compaction algorithm is used
from level2 to level4; zstd compaction algorithm is used from level5 to
 ↪ level6.
"no" means no compaction. "lz4" is a compaction algorithm with moderate
 ↪ speed and compaction ratio. The
compaction ratio of zlib is high. It is friendly to the storage space,
 ↪ but its compaction speed is slow. This
compaction occupies many CPU resources. Different machines deploy
 ↪ compaction modes according to CPU and I/O resources.
For example, if you use the compaction mode of "no:no:lz4:lz4:zstd:
 ↪ zstd" and find much I/O pressure of the
system (run the iostat command to find %util lasts 100%, or run the top

```

```

 ↵ command to find many iowait) when writing
(importing) a lot of data while the CPU resources are adequate, you can
 ↵ compress level0 and level1 and exchange CPU
resources for I/O resources. If you use the compaction mode of "no:no:
 ↵ lz4:lz4:lz4:zstd:zstd" and you find the I/O
pressure of the system is not big when writing a lot of data, but CPU
 ↵ resources are inadequate. Then run the top
command and choose the -H option. If you find a lot of bg threads (
 ↵ namely the compaction thread of RocksDB) are
running, you can exchange I/O resources for CPU resources and change
 ↵ the compaction mode to "no:no:no:lz4:lz4:zstd:zstd".
In a word, it aims at making full use of the existing resources of the
 ↵ system and improving TiKV performance
in terms of the current resources.
compression-per-level = ["no", "no", "lz4", "lz4", "lz4", "zstd", "zstd"]

The RocksDB memtable size
write-buffer-size = "128MB"

The maximum number of the memtables. The data written into RocksDB is
 ↵ first recorded in the WAL log, and then inserted
into memtables. When the memtable reaches the size limit of `write-
 ↵ buffer-size`, it turns into read only and generates
a new memtable receiving new write operations. The flush threads of
 ↵ RocksDB will flush the read only memtable to the
disks to become an sst file of level0. `max-background-flushes`
 ↵ controls the maximum number of flush threads. When the
flush threads are busy, resulting in the number of the memtables
 ↵ waiting to be flushed to the disks reaching the limit
of `max-write-buffer-number`, RocksDB stalls the new operation.
"Stall" is a flow control mechanism of RocksDB. When importing data,
 ↵ you can set the `max-write-buffer-number` value
higher, like 10.
max-write-buffer-number = 5

When the number of sst files of level0 reaches the limit of `level0-
 ↵ slowdown-writes-trigger`, RocksDB
tries to slow down the write operation, because too many sst files of
 ↵ level0 can cause higher read pressure of
RocksDB. `level0-slowdown-writes-trigger` and `level0-stop-writes-
 ↵ trigger` are for the flow control of RocksDB.
When the number of sst files of level0 reaches 4 (the default value),
 ↵ the sst files of level0 and the sst files
of level1 which overlap those of level0 implement compaction to relieve
 ↵ the read pressure.

```

```

level0-slowdown-writes-trigger = 20

When the number of sst files of level0 reaches the limit of `level0-
 ↪ stop-writes-trigger`, RocksDB stalls the new
write operation.
level0-stop-writes-trigger = 36

When the level1 data size reaches the limit value of `max-bytes-for-
 ↪ level-base`, the sst files of level1
and their overlap sst files of level2 implement compaction. The golden
 ↪ rule: the first reference principle
of setting `max-bytes-for-level-base` is guaranteeing that the `max-
 ↪ bytes-for-level-base` value is roughly equal to the
data volume of level0. Thus unnecessary compaction is reduced. For
 ↪ example, if the compaction mode is
"no:no:lz4:lz4:lz4:lz4", the `max-bytes-for-level-base` value is
 ↪ write-buffer-size * 4, because there is no
compaction of level0 and level1 and the trigger condition of compaction
 ↪ for level0 is that the number of the
sst files reaches 4 (the default value). When both level0 and level1
 ↪ adopt compaction, it is necessary to analyze
RocksDB logs to know the size of an sst file compressed from an
 ↪ memtable. For example, if the file size is 32MB,
the proposed value of `max-bytes-for-level-base` is 32MB * 4 = 128MB.
max-bytes-for-level-base = "512MB"

The sst file size. The sst file size of level0 is influenced by the
 ↪ compaction algorithm of `write-buffer-size`
and level0. `target-file-size-base` is used to control the size of a
 ↪ single sst file of level1-level16.
target-file-size-base = "32MB"

[rocksdb.writecf]
Set it the same as `rocksdb.defaultcf.compression-per-level`.
compression-per-level = ["no", "no", "lz4", "lz4", "lz4", "zstd", "zstd"]

Set it the same as `rocksdb.defaultcf.write-buffer-size`.
write-buffer-size = "128MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1

Set it the same as `rocksdb.defaultcf.max-bytes-for-level-base`.
max-bytes-for-level-base = "512MB"
target-file-size-base = "32MB"

```

```
[raftdb]
The maximum number of the file handles RaftDB can open
max-open-files = 40960

Configure this parameter to enable or disable the RaftDB statistics
 ↪ information.
enable-statistics = true

Enable the readahead feature in RaftDB compaction. If you are using
 ↪ mechanical disks, it is recommended to set
this value to 2MB at least.
compaction-readahead-size = "2MB"

[raftdb.defaultcf]
Set it the same as `rocksdb.defaultcf.compression-per-level`.
compression-per-level = ["no", "no", "lz4", "lz4", "lz4", "zstd", "zstd"]

Set it the same as `rocksdb.defaultcf.write-buffer-size`.
write-buffer-size = "128MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1

Set it the same as `rocksdb.defaultcf.max-bytes-for-level-base`.
max-bytes-for-level-base = "512MB"
target-file-size-base = "32MB"
```

#### 8.2.1.3.2 TiKV memory usage

Besides `block cache` and `write buffer` which occupy the system memory, the system memory is occupied in the following scenarios:

- Some of the memory is reserved as the system's page cache.
- When TiKV processes large queries such as `select * from ...`, it reads data, generates the corresponding data structure in the memory, and returns this structure to TiDB. During this process, TiKV occupies some of the memory.

#### 8.2.1.3.3 Recommended configuration of TiKV

- In production environments, it is not recommended to deploy TiKV on the machine whose CPU cores are less than 8 or the memory is less than 32GB.
- If you demand a high write throughput, it is recommended to use a disk with good throughput capacity.

- If you demand a very low read-write latency, it is recommended to use SSD with high IOPS.

#### 8.2.1.4 Follower Read

When a read hotspot appears in a Region, the Region leader can become a read bottleneck for the entire system. In this situation, enabling the Follower Read feature can significantly reduce the load of the leader, and improve the throughput of the whole system by balancing the load among multiple followers. This document introduces the use and implementation mechanism of Follower Read.

##### 8.2.1.4.1 Overview

The Follower Read feature refers to using any follower replica of a Region to serve a read request under the premise of strongly consistent reads. This feature improves the throughput of the TiDB cluster and reduces the load of the leader. It contains a series of load balancing mechanisms that offload TiKV read loads from the leader replica to the follower replica in a Region. TiKV's Follower Read implementation provides users with strongly consistent reads.

##### Note:

To achieve strongly consistent reads, the follower node currently needs to request the current execution progress from the leader node (that is `ReadIndex`), which causes an additional network request overhead. Therefore, the main benefits of Follower Read are to isolate read requests from write requests in the cluster and to increase overall read throughput.

##### 8.2.1.4.2 Usage

To enable TiDB's Follower Read feature, modify the value of the `tidb_replica_read` session variable:

```
set @@tidb_replica_read = '<target value>';
```

Scope: SESSION

Default: leader

This variable is used to set the data read mode expected by the current session.

- When the value of `tidb_replica_read` is set to `leader` or an empty string, TiDB maintains its original behavior and sends all read operations to the leader replica to perform.

- When the value of `tidb_replica_read` is set to `follower`, TiDB selects a follower replica of the Region to perform all read operations.
- When the value of `tidb_replica_read` is set to `leader-and-follower`, TiDB can select any replicas to perform read operations.

#### 8.2.1.4.3 Implementation mechanism

Before the Follower Read feature was introduced, TiDB applied the strong leader principle and submitted all read and write requests to the leader node of a Region to handle. Although TiKV can distribute Regions evenly on multiple physical nodes, for each Region, only the leader can provide external services. The other followers can do nothing to handle read requests but receive the data replicated from the leader at all times and prepare for voting to elect a leader in case of a failover.

To allow data reading in the follower node without violating linearizability or affecting Snapshot Isolation in TiDB, the follower node needs to use `ReadIndex` of the Raft protocol to ensure that the read request can read the latest data that has been committed on the leader. At the TiDB level, the Follower Read feature simply needs to send the read request of a Region to a follower replica based on the load balancing policy.

##### Strongly consistent reads

When the follower node processes a read request, it first uses `ReadIndex` of the Raft protocol to interact with the leader of the Region, to obtain the latest commit index of the current Raft group. After the latest commit index of the leader is applied locally to the follower, the processing of a read request starts.

##### Follower replica selection strategy

Because the Follower Read feature does not affect TiDB's Snapshot Isolation transaction isolation level, TiDB adopts the round-robin strategy to select the follower replica. Currently, for the coprocessor requests, the granularity of the Follower Read load balancing policy is at the connection level. For a TiDB client connected to a specific Region, the selected follower is fixed, and is switched only when it fails or the scheduling policy is adjusted.

However, for the non-coprocessor requests, such as a point query, the granularity of the Follower Read load balancing policy is at the transaction level. For a TiDB transaction on a specific Region, the selected follower is fixed, and is switched only when it fails or the scheduling policy is adjusted.

#### 8.2.1.5 Tune TiFlash Performance

This document introduces how to tune the performance of TiFlash, including planning machine resources and tuning TiDB parameters.

##### 8.2.1.5.1 Plan resources

If you want to save machine resources and have no requirement on isolation, you can use the method that combines the deployment of both TiKV and TiFlash. It is recommended that you save enough resources for TiKV and TiFlash respectively, and do not share disks.

### 8.2.1.5.2 Tune TiDB parameters

- For the TiDB node dedicated to OLAP/TiFlash, it is recommended that you increase the value of the `tidb_distsql_scan_concurrency` configuration item for this node to 80:

```
set @@tidb_distsql_scan_concurrency = 80;
```

- Enable the super batch feature:

You can use the `tidb_allow_batch_cop` variable to set whether to merge Region requests when reading from TiFlash.

When the number of Regions involved in the query is relatively large, try to set this variable to 1 (effective for coprocessor requests with `aggregation` operators that are pushed down to TiFlash), or set this variable to 2 (effective for all coprocessor requests that are pushed down to TiFlash).

```
set @@tidb_allow_batch_cop = 1;
```

- Enable the optimization of pushing down aggregate functions before TiDB operators such as `JOIN` or `UNION`:

You can use the `tidb_opt_agg_push_down` variable to control the optimizer to execute this optimization. When the aggregate operations are quite slow in the query, try to set this variable to 1.

```
set @@tidb_opt_agg_push_down = 1;
```

- Enable the optimization of pushing down aggregate functions with `Distinct` before TiDB operators such as `JOIN` or `UNION`:

You can use the `tidb_opt_distinct_agg_push_down` variable to control the optimizer to execute this optimization. When the aggregate operations with `Distinct` are quite slow in the query, try to set this variable to 1.

```
set @@tidb_opt_distinct_agg_push_down = 1;
```

## 8.2.2 Coprocessor Cache

Starting from v4.0, the TiDB instance supports caching the results of the calculation that is pushed down to TiKV (the Coprocessor Cache feature), which can accelerate the calculation process in some scenarios.

### 8.2.2.1 Configuration

You can configure Coprocessor Cache via the `tikv-client.copr-cache` configuration items in the TiDB configuration file. For details about how to enable and configure Coprocessor Cache, see [TiDB Configuration File](#).

### 8.2.2.2 Feature description

- When a SQL statement is executed on a single TiDB instance for the first time, the execution result is not cached.
- Calculation results are cached in the memory of TiDB. If the TiDB instance is restarted, the cache becomes invalid.
- The cache is not shared among TiDB instances.
- Only push-down calculation result is cached. Even if cache is hit, TiDB still need to perform subsequent calculation.
- The cache is in the unit of Region. Writing data to a Region causes the Region cache to be invalid. For this reason, the Coprocessor Cache feature mainly takes effect on the data that rarely changes.
- When push-down calculation requests are the same, the cache is hit. Usually in the following scenarios, the push-down calculation requests are the same or partially the same:
  - The SQL statements are the same. For example, the same SQL statement is executed repeatedly.  
In this scenario, all the push-down calculation requests are consistent, and all requests can use the push-down calculation cache.
  - The SQL statements contain a changing condition, and the other parts are consistent. The changing condition is the primary key of the table or the partition.  
In this scenario, some of the push-down calculation requests are the same with some previous requests, and these calculation requests can use the cached (previous) push-down calculation result.
  - The SQL statements contain multiple changing conditions and the other parts are consistent. The changing conditions exactly match a compound index column.  
In this scenario, some of the push-down calculation requests are the same with some previous requests, and these calculation requests can use the cached (previous) push-down calculation result.
- This feature is transparent to users. Enabling or disabling this feature does not affect the calculation result and only affects the SQL execution time.

### 8.2.2.3 Check the cache effect

You can check the cache effect of Coprocessor by executing `EXPLAIN ANALYZE` or viewing the Grafana monitoring panel.

#### 8.2.2.3.1 Use EXPLAIN ANALYZE

You can view the cache hit rate in [Operators for accessing tables](#) by using the `EXPLAIN ANALYZE` statement. See the following example:

```

EXPLAIN ANALYZE SELECT * FROM t USE INDEX(a);
+--+
| id | estRows | actRows | task | access object
| | execution info
| |
| | operator info | memory | disk |
+--+
| IndexLookUp_6 | 262400.00 | 262400 | root |
| | time:620.513742ms, loops:258, cop_task: {num:
| | 4, max: 5.530817ms, min: 1.51829ms, avg: 2.70883ms, p95: 5.530817ms,
| | max_proc_keys: 2480, p95_proc_keys: 2480, tot_proc: 1ms, tot_wait: 1
| | ms, rpc_num: 4, rpc_time: 10.816328ms, copr_cache_hit_rate: 0.75} ||
| | 6.685169219970703 MB | N/A |
| -IndexFullScan_4(Build) | 262400.00 | 262400 | cop[tikv] | table:t,
| index:a(a, c) | proc max:93ms, min:1ms, p80:93ms, p95:93ms, iters
| :275, tasks:4
|
| | keep order:false, stats:pseudo | 1.7549400329589844 MB | N/A |
| -TableRowIDScan_5(Probe) | 262400.00 | 0 | cop[tikv] | table:t
|
| | time:0ns, loops:0
|
| | keep order:false, stats:pseudo | N/A | N/A |
+--+
| |
| |
| |
3 rows in set (0.62 sec)

```

The column `execution info` of the execution result gives the `copr_cache_hit_ratio` information, which indicates the hit rate of the Coprocessor Cache. The 0.75 in the above example means that the hit rate is about 75%.

#### 8.2.2.3.2 View the Grafana monitoring panel

In Grafana, you can see the `copr-cache` panel in the `distsql` subsystem under the `tidb` namespace. This panel monitors the number of hits, misses, and cache discards of the Coprocessor Cache in the entire cluster.

## 8.3 SQL Tuning

### 8.3.1 SQL Tuning Overview

SQL is a declarative language. That is, an SQL statement describes *what the final result should look like* and not a set of steps to execute in sequence. TiDB will optimize the execution, and is semantically permitted to execute parts of the query in any order provided that it correctly returns the final result as described.

A useful comparison to SQL optimization, is to describe what happens when you use GPS navigation. From your provided address, *2955 Campus Drive San Mateo CA 94403*, the GPS software plans the most time-efficient way to route you. It may make use of various statistics such as previous trips, meta data such as speed limits, and in modern cases, a live feed of traffic information. Several of these analogies translate to TiDB.

This section introduces several concepts about query execution:

- [Understanding the Query Execution Plan](#) introduces how to use the EXPLAIN statement to understand how TiDB has decided to execute a statement.
- [SQL Optimization Process](#) introduces what optimizations TiDB is capable of using to improve query execution performance.
- [Control Execution Plans](#) introduces ways to control the generation of the execution plan. This can be useful in cases where the execution plan decided by TiDB is suboptimal.

### 8.3.2 Understanding the Query Execution Plan

#### 8.3.2.1 EXPLAIN Overview

##### Note:

When you use the MySQL client to connect to TiDB, to read the output result in a clearer way without line wrapping, you can use the `pager less ↵ -S` command. Then, after the EXPLAIN result is output, you can press the right arrow → button on your keyboard to horizontally scroll through the output.

SQL is a declarative language. It describes what the results of a query should look like, **not the methodology** to actually retrieve those results. TiDB considers all the possible ways in which a query could be executed, including using what order to join tables and whether any potential indexes can be used. The process of *considering query execution plans* is known as SQL optimization.

The EXPLAIN statement shows the selected execution plan for a given statement. That is, after considering hundreds or thousands of ways in which the query could be executed, TiDB believes that this *plan* will consume the least resources and execute in the shortest amount of time:

```
CREATE TABLE t (id INT NOT NULL PRIMARY KEY auto_increment, a INT NOT NULL,
 → pad1 VARCHAR(255), INDEX(a));
INSERT INTO t VALUES (1, 1, 'aaa'),(2,2, 'bbb');
EXPLAIN SELECT * FROM t WHERE a = 1;
```

```
Query OK, 0 rows affected (0.96 sec)
```

```
Query OK, 2 rows affected (0.02 sec)
```

```
Records: 2 Duplicates: 0 Warnings: 0
```

```
+--+
→ -----+-----+-----+-----+
→
| id | estRows | task | access object |
| operator info | |
+--+
→ -----+-----+-----+-----+
→
IndexLookUp_10	10.00	root	
-IndexRangeScan_8(Build)	10.00	cop[tikv]	table:t, index:a(a)
→ range:[1,1], keep order:false, stats:pseudo			
-TableRowIDScan_9(Probe)	10.00	cop[tikv]	table:t
→ order:false, stats:pseudo			
+--+
→ -----+-----+-----+-----+
→
3 rows in set (0.00 sec)
```

EXPLAIN does not execute the actual query. EXPLAIN ANALYZE can be used to execute the query and show EXPLAIN information. This can be useful in diagnosing cases where the execution plan selected is suboptimal. For additional examples of using EXPLAIN, see the following documents:

- [Indexes](#)
- [Joins](#)
- [Subqueries](#)
- [Aggregation](#)
- [Views](#)
- [Partitions](#)

### 8.3.2.1.1 Understand EXPLAIN output

The following describes the output of the EXPLAIN statement above:

- `id` describes the name of an operator, or sub-task that is required to execute the SQL statement. See [Operator overview](#) for additional details.
- `estRows` shows an estimate of the number of rows TiDB expects to process. This number might be based on dictionary information, such as when the access method is based on a primary or unique key, or it could be based on statistics such as a CMSketch or histogram.
- `task` shows where an operator is performing the work. See [Task overview](#) for additional details.
- `access object` shows the table, partition and index that is being accessed. The parts of the index are also shown, as in the case above that the column `a` from the index was used. This can be useful in cases where you have composite indexes.
- `operator info` shows additional details about the access. See [Operator info overview](#) for additional details.

#### Operator overview

An operator is a particular step that is executed as part of returning query results. The operators that perform table scans (of the disk or the TiKV Block Cache) are listed as follows:

- **TableFullScan**: Full table scan
- **TableRangeScan**: Table scans with the specified range
- **TableRowIDScan**: Scans the table data based on the RowID. Usually follows an index read operation to retrieve the matching data rows.
- **IndexFullScan**: Similar to a “full table scan”, except that an index is scanned, rather than the table data.
- **IndexRangeScan**: Index scans with the specified range.

TiDB aggregates the data or calculation results scanned from TiKV/TiFlash. The data aggregation operators can be divided into the following categories:

- **TableReader**: Aggregates the data obtained by the underlying operators like `TableFullScan` or `TableRangeScan` in TiKV.
- **IndexReader**: Aggregates the data obtained by the underlying operators like `IndexFullScan` or `IndexRangeScan` in TiKV.
- **IndexLookUp**: First aggregates the RowID (in TiKV) scanned by the `Build` side. Then at the `Probe` side, accurately reads the data from TiKV based on these RowIDs. At the `Build` side, there are operators like `IndexFullScan` or `IndexRangeScan`; at the `Probe` side, there is the `TableRowIDScan` operator.

- **IndexMerge**: Similar to `IndexLookUp`. `IndexMerge` can be seen as an extension of `IndexLookupReader`. `IndexMerge` supports reading multiple indexes at the same time. There are many `Builds` and one `Probe`. The execution process of `IndexMerge` is the same as that of `IndexLookUp`.

While the structure appears as a tree, executing the query does not strictly require the child nodes to be completed before the parent nodes. TiDB supports intra-query parallelism, so a more accurate way to describe the execution is that the child nodes *flow into* their parent nodes. Parent, child and sibling operators *might* potentially be executing parts of the query in parallel.

In the previous example, the `-IndexRangeScan_8(Build)` operator finds the internal `RowID` for rows that match the `a(a)` index. The `-TableRowIDScan_9(Probe)` operator then retrieves these rows from the table.

### Range query

In the `WHERE/HAVING/ON` conditions, the TiDB optimizer analyzes the result returned by the primary key query or the index key query. For example, these conditions might include comparison operators of the numeric and date type, such as `>`, `<`, `=`, `>=`, `<=`, and the character type such as `LIKE`.

#### Note:

- In order to use an index, the condition must be *sargable*. For example, the condition `YEAR(date_column) < 1992` can not use an index, but `date_column < '1992-01-01'` can.
- It is recommended to compare data of the same type and **character set and collation**. Mixing types may require additional `cast` operations, or prevent indexes from being used.
- You can also use `AND` (intersection) and `OR` (union) to combine the range query conditions of one column. For a multi-dimensional composite index, you can use conditions in multiple columns. For example, regarding the composite index (`a`, `b`, `c`):
  - When `a` is an equivalent query, continue to figure out the query range of `b`; when `b` is also an equivalent query, continue to figure out the query range of `c`.
  - Otherwise, if `a` is a non-equivalent query, you can only figure out the range of `a`.

### Task overview

Currently, calculation tasks of TiDB can be divided into two categories: cop tasks and root tasks. A `cop[tikv]` task indicates that the operator is performed inside the TiKV

coprocessor. A `root` task indicates that it will be completed inside of TiDB.

One of the goals of SQL optimization is to push the calculation down to TiKV as much as possible. The Coprocessor in TiKV supports most of the built-in SQL functions (including the aggregate functions and the scalar functions), SQL `LIMIT` operations, index scans, and table scans. However, all `Join` operations can only be performed as root tasks in TiDB.

### Operator info overview

The `operator info` can show useful information such as which conditions were able to be pushed down:

- `range: [1,1]` shows that the predicate from the where clause of the query (`a = 1`) was pushed right down to TiKV (the task is of `cop[tikv]`).
- `keep order:false` shows that the semantics of this query did not require TiKV to return the results in order. If the query were to be modified to require an order (such as `SELECT * FROM t WHERE a = 1 ORDER BY id`), then this condition would be `keep ↪ order:true`.
- `stats:pseudo` shows that the estimates shown in `estRows` might not be accurate. TiDB periodically updates statistics as part of a background operation. A manual update can also be performed by running `ANALYZE TABLE t`.

Different operators output different information after the `EXPLAIN` statement is executed. You can use optimizer hints to control the behavior of the optimizer, and thereby controlling the selection of the physical operators. For example, `/*+ HASH_JOIN(t1, t2)*/` means that the optimizer uses the Hash Join algorithm. For more details, see [Optimizer Hints](#).

#### 8.3.2.2 EXPLAIN Walkthrough

Because SQL is a declarative language, you cannot automatically tell whether a query is executed efficiently. You must first use the `EXPLAIN` statement to learn the current execution plan.

The following statement from the [bikeshare example database](#) counts how many trips were taken on the July 1, 2017:

```
EXPLAIN SELECT count(*) FROM trips WHERE start_date BETWEEN '2017-07-01
↪ 00:00:00' AND '2017-07-01 23:59:59';
```

| +--  | → | →       | →    | →             | →        | → |
|------|---|---------|------|---------------|----------|---|
| id   |   | estRows | task | access object | operator |   |
| info |   |         |      |               |          |   |
|      |   |         |      |               |          |   |
| +--  |   |         |      |               |          |   |
| id   |   |         |      |               |          |   |
| info |   |         |      |               |          |   |
|      |   |         |      |               |          |   |

```

+--+
| StreamAgg_20 | 1.00 | root | | funcs:count(
| ↳ Column#13->Column#11
| ↳
| ↳ |
| -TableReader_21 | 1.00 | root | | data:
| ↳ StreamAgg_9
| ↳
| ↳ |
| -StreamAgg_9 | 1.00 | cop[tikv] | | funcs:count
| ↳ (1)->Column#13
| ↳
| ↳ |
| -Selection_19 | 250.00 | cop[tikv] | | ge(
| ↳ bikeshare.trips.start_date, 2017-07-01 00:00:00.000000), le(bikeshare
| ↳ .trips.start_date, 2017-07-01 23:59:59.000000) |
| -TableFullScan_18 | 10000.00 | cop[tikv] | table:trips | keep
| ↳ order:false, stats:pseudo
| ↳
| ↳ |
+--+
| 5 rows in set (0.00 sec)

```

From the child operator `-TableFullScan_18` back, you can see its execution process as follows, which is currently suboptimal:

1. The coprocessor (TiKV) reads the entire `trips` table as a `TableFullScan` operation. It then passes the rows that it reads to the `Selection_19` operator, which is still within TiKV.
2. The `WHERE start_date BETWEEN ..` predicate is then filtered in the `Selection_19` operator. Approximately 250 rows are estimated to meet this selection. Note that this number is estimated according to the statistics and the operator's logic. The `-TableFullScan_18` operator shows `stats:pseudo`, which means that the table does not have the actual statistical information. After running `ANALYZE TABLE trips` to collect statistical information, the statistics are expected to be more accurate.
3. The rows that meet the selection criteria then have a `count` function applied to them. This is also completed inside the `StreamAgg_9` operator, which is still inside TiKV (`cop[tikv]`). The TiKV coprocessor can execute a number of MySQL built-in functions, `count` being one of them.
4. The results from `StreamAgg_9` are then sent to the `TableReader_21` operator which is now inside the TiDB server (the task of `root`). The `estRows` column value for

this operator is 1, which means that the operator will receive one row from each of the TiKV Regions to be accessed. For more information about these requests, see [EXPLAIN ANALYZE](#).

5. The StreamAgg\_20 operator then applies a count function to each of the rows from the -TableReader\_21 operator, which you can see from `SHOW TABLE REGIONS` and will be about 56 rows. Because this is the root operator, it then returns results to the client.

**Note:**

For a general view of the Regions that a table contains, execute `SHOW TABLE REGIONS`.

#### 8.3.2.2.1 Assess the current performance

`EXPLAIN` only returns the query execution plan but does not execute the query. To get the actual execution time, you can either execute the query or use `EXPLAIN ANALYZE`:

```
EXPLAIN ANALYZE SELECT count(*) FROM trips WHERE start_date BETWEEN '2017-07-01 00:00:00' AND '2017-07-01 23:59:59';
```

| +--                                                                   | id             | estRows | actRows | task | access object |  |
|-----------------------------------------------------------------------|----------------|---------|---------|------|---------------|--|
|                                                                       | execution info |         |         |      |               |  |
|                                                                       | operator info  |         |         |      |               |  |
|                                                                       | memory         |         | disk    |      |               |  |
| +--                                                                   |                |         |         |      |               |  |
| StreamAgg_20                                                          | 1.00           | 1       | root    |      |               |  |
| time:1.031417203s, loops:2                                            |                |         |         |      |               |  |
| funcs:count(Column#13)->Column#11                                     |                |         |         |      |               |  |
| 632 Bytes   N/A                                                       |                |         |         |      |               |  |
| -TableReader_21                                                       | 1.00           | 56      | root    |      |               |  |
| time:1.031408123s, loops:2, cop_task: {num: 56, max: 782.147269ms,    |                |         |         |      |               |  |
| min: 5.759953ms, avg: 252.005927ms, p95: 609.294603ms, max_proc_keys: |                |         |         |      |               |  |
| 910371, p95_proc_keys: 704775, tot_proc: 11.524s, tot_wait: 580ms,    |                |         |         |      |               |  |

```

 ↗ rpc_num: 56, rpc_time: 14.111932641s} | data:StreamAgg_9
 ↗
 ↗ N/A |
| - StreamAgg_9 | 1.00 | 56 | cop[tikv] | |
 ↗ proc max:640ms, min:8ms, p80:276ms, p95:480ms, iters:18695, tasks:56
 ↗
 ↗ | funcs:count(1)->Column#13
 ↗
 ↗ | N/A | N/A |
| - Selection_19 | 250.00 | 11409 | cop[tikv] | |
 ↗ proc max:640ms, min:8ms, p80:276ms, p95:476ms, iters:18695, tasks:56
 ↗
 ↗ | ge(bikeshare.trips.start_date, 2017-07-01 00:00:00.000000), le(
 ↗ bikeshare.trips.start_date, 2017-07-01 23:59:59.000000) | N/A | N/A |
| - TableFullScan_18 | 10000.00 | 19117643 | cop[tikv] | table:trips
 ↗ proc max:612ms, min:8ms, p80:248ms, p95:460ms, iters:18695, tasks
 ↗ :56
 ↗
 ↗ | keep order:false, stats:pseudo
 ↗
 ↗ | N/A | N/A |
+--+
 ↗ -----+-----+-----+-----+
 ↗
5 rows in set (1.03 sec)

```

The example query above takes 1.03 seconds to execute, which is an ideal performance.

From the result of EXPLAIN ANALYZE above, actRows indicates that some of the estimates (estRows) are inaccurate (expecting 10 thousand rows but finding 19 million rows), which is already indicated in the operator info (stats:pseudo) of -TableFullScan\_18. If you run **ANALYZE TABLE** first and then EXPLAIN ANALYZE again, you can see that the estimates are much closer:

```

ANALYZE TABLE trips;
EXPLAIN ANALYZE SELECT count(*) FROM trips WHERE start_date BETWEEN '
 ↗ 2017-07-01 00:00:00' AND '2017-07-01 23:59:59';

```

Query OK, 0 rows affected (10.22 sec)

```

+--+
 ↗ -----+-----+-----+-----+
 ↗
 ↗ | id | estRows | actRows | task | access object
 ↗ | execution info
 ↗

```

```

 ↵ | operator info
 ↵
 ↵ | memory | disk |
+--+
 ↵ -----
 ↵
 ↵
| StreamAgg_20 | 1.00 | 1 | root | |
 ↵ time:926.393612ms, loops:2
 ↵
 ↵ | funcs:count(Column#13)->Column#11
 ↵
 ↵ | 632 Bytes | N/A |
| -TableReader_21 | 1.00 | 56 | root | |
 ↵ time:926.384792ms, loops:2, cop_task: {num: 56, max: 850.94424ms,
 ↵ min: 6.042079ms, avg: 234.987725ms, p95: 495.474806ms, max_proc_keys:
 ↵ 910371, p95_proc_keys: 704775, tot_proc: 10.656s, tot_wait: 904ms,
 ↵ rpc_num: 56, rpc_time: 13.158911952s} | data:StreamAgg_9
 ↵
 ↵ | 328 Bytes |
 ↵ | N/A |
| -StreamAgg_9 | 1.00 | 56 | cop[tikv] | |
 ↵ proc max:592ms, min:4ms, p80:244ms, p95:480ms, iters:18695, tasks:56
 ↵
 ↵
 ↵ | funcs:count(1)->Column#13
 ↵
 ↵ | N/A | N/A |
| -Selection_19 | 432.89 | 11409 | cop[tikv] | |
 ↵ proc max:592ms, min:4ms, p80:244ms, p95:480ms, iters:18695, tasks:56
 ↵
 ↵
 ↵ | ge(bikeshare.trips.start_date, 2017-07-01 00:00:00.000000), le(
 ↵ bikeshare.trips.start_date, 2017-07-01 23:59:59.000000) | N/A | N/A |
| -TableFullScan_18 | 19117643.00 | 19117643 | cop[tikv] | table:
 ↵ trips | proc max:564ms, min:4ms, p80:228ms, p95:456ms, iters:18695,
 ↵ tasks:56
 ↵
 ↵ | keep order:false
 ↵
 ↵ | N/A | N/A |
+--+
 ↵ -----
 ↵
5 rows in set (0.93 sec)

```

After ANALYZE TABLE is executed, you can see that the estimated rows for the -

→ TableFullScan\_18 operator is accurate and the estimate for `-Selection_19` is now also much closer. In the two cases above, although the execution plan (the set of operators TiDB uses to execute this query) has not changed, quite frequently sub-optimal plans are caused by outdated statistics.

In addition to `ANALYZE TABLE`, TiDB automatically regenerates statistics as a background operation after the threshold of `tidb_auto_analyze_ratio` is reached. You can see how close TiDB is to this threshold (how healthy TiDB considers the statistics to be) by executing the `SHOW STATS_HEALTHY` statement:

```
SHOW STATS_HEALTHY;
```

| Db_name   | Table_name | Partition_name | Healthy |
|-----------|------------|----------------|---------|
| bikeshare | trips      |                | 100     |

1 row in set (0.00 sec)

### 8.3.2.2 Identify optimizations

The current execution plan is efficient in the following aspects:

- Most of the work is handled inside the TiKV coprocessor. Only 56 rows need to be sent across the network back to TiDB for processing. Each of these rows is short and contains only the count that matches the selection.
- Aggregating the count of rows both in TiDB (`StreamAgg_20`) and in TiKV ( → `StreamAgg_9`) uses the stream aggregation, which is very efficient in its memory usage.

The biggest issue with the current execution plan is that the predicate `start_date` → `BETWEEN '2017-07-01 00:00:00' AND '2017-07-01 23:59:59'` does not apply immediately. All rows are read first with a TableFullScan operator, and then a selection is applied afterwards. You can find out the cause from the output of `SHOW CREATE TABLE trips`:

```
SHOW CREATE TABLE trips\G
```

```
***** 1. row *****
Table: trips
Create Table: CREATE TABLE `trips` (
 `trip_id` bigint(20) NOT NULL AUTO_INCREMENT,
 `duration` int(11) NOT NULL,
 `start_date` datetime DEFAULT NULL,
 `end_date` datetime DEFAULT NULL,
 `start_station_number` int(11) DEFAULT NULL,
```

```
`start_station` varchar(255) DEFAULT NULL,
`end_station_number` int(11) DEFAULT NULL,
`end_station` varchar(255) DEFAULT NULL,
`bike_number` varchar(255) DEFAULT NULL,
`member_type` varchar(255) DEFAULT NULL,
PRIMARY KEY (`trip_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin AUTO_INCREMENT
→ =20477318
1 row in set (0.00 sec)
```

There is **NO** index on `start_date`. You would need an index in order to push this predicate into an index reader operator. Add an index as follows:

```
ALTER TABLE trips ADD INDEX (start date);
```

Query OK, 0 rows affected (2 min 10.23 sec)

Note:

You can monitor the progress of DDL jobs using the `ADMIN SHOW DDL` → `JOB`s command. The defaults in TiDB are carefully chosen so that adding an index does not impact production workloads too much. For testing environments, consider increasing the `tidb_ddl_reorg_batch_size` and `tidb_ddl_reorg_worker_cnt` values. On a reference system, a batch size of 10240 and worker count of 32 can achieve a 10x performance improvement over the defaults.

After adding an index, you can then repeat the query in EXPLAIN. In the following output, you can see that a new execution plan is chosen, and the TableFullScan and Selection operators have been eliminated:

```
EXPLAIN SELECT count(*) FROM trips WHERE start_date BETWEEN '2017-07-01
→ 00:00:00' AND '2017-07-01 23:59:59';
```

```
+---
→-----+-----+-----+
→
→
| id | estRows | task | access object
| | | operator info |
| |
→-----+-----+-----+
+---
```

```

| StreamAgg_17 | 1.00 | root |
| ↳ | | funcs:count(Column#13)->Column
↳ #11		
↳		
-IndexReader_18	1.00	root
↳		index:StreamAgg_9
↳		
↳		
-StreamAgg_9	1.00	cop[tikv]
↳		funcs:count(1)->Column#13
↳		
↳		
- IndexRangeScan_16	8471.88	cop[tikv]
↳ start_date(start_date)	range:[2017-07-01 00:00:00,2017-07-01	
↳ 23:59:59], keep order:false		
+--+		
↳ -----+-----+-----+		
↳		
4 rows in set (0.00 sec)

```

To compare the actual execution time, you can again use `EXPLAIN ANALYZE`:

```
EXPLAIN ANALYZE SELECT count(*) FROM trips WHERE start_date BETWEEN '2017-07-01 00:00:00' AND '2017-07-01 23:59:59';
```

```

+--+
| id | estRows | actRows | task | access object
| | | | | execution info
| | | | | memory
| | operator info
| | disk |
+--+
| StreamAgg_17 | 1.00 | 1 | root | time:4.516728ms, loops:2
| | | | | 372
| | funcs:count(Column#13)->Column#11
| | Bytes | N/A |
| -IndexReader_18 | 1.00 | 1 | root | time:4.514278ms, loops:2,
| | | | | cop_task: {num: 1, max:4.462288ms, proc_keys: 11409, rpc_num: 1,
| | | | | rpc_time: 4.457148ms} | index:StreamAgg_9
| | 238 Bytes | N/A |
| -StreamAgg_9 | 1.00 | 1 | cop[tikv] |
| | | | | time:4ms, loops:12

```

```

 ↵
 ↵ | funcs:count(1)->Column#13 | N/A
 ↵ | N/A |
| - IndexRangeScan_16 | 8471.88 | 11409 | cop[tikv] | table:trips,
 ↵ index:start_date(start_date) | time:4ms, loops:12
 ↵
 ↵ | range:[2017-07-01 00:00:00,2017-07-01 23:59:59], keep order:false |
 ↵ N/A | N/A |
+--+
 ↵ -----+-----+-----+-----+
 ↵
4 rows in set (0.00 sec)

```

From the result above, the query time has reduced from 1.03 seconds to 0.0 seconds.

#### Note:

Another optimization that applies here is the coprocessor cache. If you are unable to add indexes, consider enabling the [coprocessor cache](#). When it is enabled, as long as the Region has not been modified since the operator is last executed, TiKV will return the value from the cache. This will also help reduce much of the cost of the expensive `TableFullScan` and `Selection` operators.

#### 8.3.2.3 Explain Statements That Use Indexes

TiDB supports several operators which make use of indexes to speed up query execution:

- [IndexLookup](#)
- [IndexReader](#)
- [Point\\_Get and Batch\\_Point\\_Get](#)
- [IndexFullScan](#)

The examples in this document are based on the following sample data:

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY auto_increment,
 intkey INT NOT NULL,
 pad1 VARBINARY(1024),
 INDEX (intkey)
);
```

```

INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM
 ↪ dual;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM t1
 ↪ a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM t1
 ↪ a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, FLOOR(RAND()*1024), RANDOM_BYTES(1024) FROM t1
 ↪ a JOIN t1 b JOIN t1 c LIMIT 10000;

```

### 8.3.2.3.1 IndexLookup

TiDB uses the IndexLookup operator when retrieving data from a secondary index. In this case, the following queries will all use the IndexLookup operator on the intkey index:

```

EXPLAIN SELECT * FROM t1 WHERE intkey = 123;
EXPLAIN SELECT * FROM t1 WHERE intkey < 10;
EXPLAIN SELECT * FROM t1 WHERE intkey BETWEEN 300 AND 310;
EXPLAIN SELECT * FROM t1 WHERE intkey IN (123,29,98);
EXPLAIN SELECT * FROM t1 WHERE intkey >= 99 AND intkey <= 103;

```

```

+--+
| id | estRows | task | access object
| | | operator info |
+--+
| IndexLookUp_10 | 1.00 | root |
| | | |
| -IndexRangeScan_8(Build) | 1.00 | cop[tikv] | table:t1, index:intkey(
| ↪ intkey) | range:[123,123], keep order:false |
| -TableRowIDScan_9(Probe) | 1.00 | cop[tikv] | table:t1
| ↪ | keep order:false |
+--+
| | | |
3 rows in set (0.00 sec)

+--+
| id | estRows | task | access object
| | | operator info |
+--+

```

```
+--+
→ -----+-----+-----+
→
| IndexLookUp_10 | 3.60 | root |
→ | |
| -IndexRangeScan_8(Build) | 3.60 | cop[tikv] | table:t1, index:intkey(
→ intkey) | range:[-inf,10), keep order:false |
| -TableRowIDScan_9(Probe) | 3.60 | cop[tikv] | table:t1
→ | keep order:false |
+--+
→ -----+-----+-----+
→
3 rows in set (0.00 sec)

+--+
→ -----+-----+-----+
→
| id | estRows | task | access object
→ | operator info |
+--+
→ -----+-----+-----+
→
| IndexLookUp_10 | 5.67 | root |
→ | |
| -IndexRangeScan_8(Build) | 5.67 | cop[tikv] | table:t1, index:intkey(
→ intkey) | range:[300,310], keep order:false |
| -TableRowIDScan_9(Probe) | 5.67 | cop[tikv] | table:t1
→ | keep order:false |
+--+
→ -----+-----+-----+
→
3 rows in set (0.00 sec)

+--+
→ -----+-----+-----+
→
| id | estRows | task | access object
→ | operator info |
+--+
→ -----+-----+-----+
→
| IndexLookUp_10 | 5.67 | root |
→ | |
| -IndexRangeScan_8(Build) | 5.67 | cop[tikv] | table:t1, index:intkey(
→ intkey) | range:[300,310], keep order:false |
```

```

| -TableRowIDScan_9(Probe) | 5.67 | cop[tikv] | table:t1
| | keep order:false | |
+--+
| | |
| | |
3 rows in set (0.00 sec)

+--+
| | |
| | |
| id | estRows | task | access object
| | operator info | |
+--+
IndexLookUp_10	4.00	root	
-IndexRangeScan_8(Build)	4.00	cop[tikv]	table:t1, index:intkey(
	intkey)	range:[29,29], [98,98], [123,123], keep order:false	
-TableRowIDScan_9(Probe)	4.00	cop[tikv]	table:t1
	keep order:false		
+--+			
3 rows in set (0.00 sec)

+--+
| | |
| | |
| id | estRows | task | access object
| | operator info | |
+--+
IndexLookUp_10	6.00	root	
-IndexRangeScan_8(Build)	6.00	cop[tikv]	table:t1, index:intkey(
	intkey)	range:[99,103], keep order:false	
-TableRowIDScan_9(Probe)	6.00	cop[tikv]	table:t1
	keep order:false		
+--+			
3 rows in set (0.00 sec)

```

The IndexLookup operator has two child nodes:

- The `-IndexRangeScan_8(Build)` operator performs a range scan on the `intkey` index and retrieves the values of the internal RowID (for this table, the primary key).
- The `-TableRowIDScan_9(Probe)` operator then retrieves the full row from the table data.

Because an IndexLookup task requires two steps, the SQL Optimizer might choose the `TableFullScan` operator based on `statistics` in scenarios where a large number of rows match. In the following example, a large number of rows match the condition of `intkey > 100`, and a `TableFullScan` is chosen:

```
EXPLAIN SELECT * FROM t1 WHERE intkey > 100;
```

| +--                      |                                                            |  |  |  |
|--------------------------|------------------------------------------------------------|--|--|--|
| →                        | -----+-----+-----+-----+                                   |  |  |  |
| →                        |                                                            |  |  |  |
| id                       | estRows   task     access object   operator info           |  |  |  |
| →                        |                                                            |  |  |  |
| +--                      | -----+-----+-----+-----+                                   |  |  |  |
| →                        |                                                            |  |  |  |
| TableRowReader_7         | 898.50   root                         data:Selection_6     |  |  |  |
| →                        |                                                            |  |  |  |
| -Selection_6             | 898.50   cop[tikv]                       gt(test.t1.intkey |  |  |  |
| → , 100 )                |                                                            |  |  |  |
| -TableFullScan_5         | 1010.00   cop[tikv]   table:t1   keep order:false          |  |  |  |
| →                        |                                                            |  |  |  |
| +--                      | -----+-----+-----+-----+                                   |  |  |  |
| →                        |                                                            |  |  |  |
| 3 rows in set (0.00 sec) |                                                            |  |  |  |

The IndexLookup operator can also be used to efficiently optimize `LIMIT` on an indexed column:

```
EXPLAIN SELECT * FROM t1 ORDER BY intkey DESC LIMIT 10;
```

| +-- |                                  |  |  |  |
|-----|----------------------------------|--|--|--|
| →   | -----+-----+-----+-----+         |  |  |  |
| →   |                                  |  |  |  |
| id  | estRows   task     access object |  |  |  |
| →   |                                  |  |  |  |

```
+--+
→ -----+-----+-----+
→
| IndexLookUp_21 | 10.00 | root |
→ | limit embedded(offset:0, count:10) |
| -Limit_20(Build) | 10.00 | cop[tikv] |
→ | offset:0, count:10 |
| -IndexFullScan_18 | 10.00 | cop[tikv] | table:t1, index:intkey(
→ intkey) | keep order:true, desc |
| -TableRowIDScan_19(Probe) | 10.00 | cop[tikv] | table:t1
→ | keep order:false, stats:pseudo |
+--+
→ -----+-----+-----+
→
4 rows in set (0.00 sec)
```

In the above example, the last 20 rows are read from the index `intkey`. These RowID values are then retrieved from the table data.

### 8.3.2.3.2 IndexReader

TiDB supports the *covering index optimization*. If all rows can be retrieved from an index, TiDB will skip the second step that is usually required in an `IndexLookup`. Consider the following two examples:

```
EXPLAIN SELECT * FROM t1 WHERE intkey = 123;
EXPLAIN SELECT id FROM t1 WHERE intkey = 123;
```

```
+--+
→ -----+-----+-----+
→
| id | estRows | task | access object
→ | operator info |
+--+
→ -----+-----+-----+
→
| IndexLookUp_10 | 1.00 | root |
→ | |
| -IndexRangeScan_8(Build) | 1.00 | cop[tikv] | table:t1, index:intkey(
→ intkey) | range:[123,123], keep order:false |
| -TableRowIDScan_9(Probe) | 1.00 | cop[tikv] | table:t1
→ | keep order:false |
+--+
→ -----+-----+-----+
→
```

```
3 rows in set (0.00 sec)

+--+
| id | estRows | task | access object |
| operator info | |
+--+
Projection_4	1.00	root	
↳ test.t1.id			
↳ IndexReader_6	1.00	root	
↳ index:IndexRangeScan_5			
↳ -IndexRangeScan_5	1.00	cop[tikv]	table:t1, index:intkey(
↳ intkey)	range:[123,123], keep order:false		
+--+			
id	estRows	task	access object
operator info			
+--+			
id	estRows	task	access object
operator info			
+--+
3 rows in set (0.00 sec)
```

Because `id` is also the internal RowID, it is stored in the `intkey` index. After using the `intkey` index as part of `-IndexRangeScan_5`, the value of the RowID can be returned directly.

#### 8.3.2.3.3 Point\_Get and Batch\_Point\_Get

TiDB uses the `Point_Get` or `Batch_Point_Get` operator when retrieving data directly from a primary key or unique key. These operators are more efficient than `IndexLookup`. For example:

```
EXPLAIN SELECT * FROM t1 WHERE id = 1234;
EXPLAIN SELECT * FROM t1 WHERE id IN (1234,123);

ALTER TABLE t1 ADD unique_key INT;
UPDATE t1 SET unique_key = id;
ALTER TABLE t1 ADD UNIQUE KEY (unique_key);

EXPLAIN SELECT * FROM t1 WHERE unique_key = 1234;
EXPLAIN SELECT * FROM t1 WHERE unique_key IN (1234, 123);
```

```
+-----+-----+-----+-----+
| id | estRows | task | access object | operator info |
+-----+-----+-----+-----+
| Point_Get_1 | 1.00 | root | table:t1 | handle:1234 |
```

```
+-----+-----+-----+-----+
1 row in set (0.00 sec)

+--+
→ -----+-----+-----+
→
| id | estRows | task | access object | operator info
→
+--+
→ -----+-----+-----+-----+
→
| Batch_Point_Get_1 | 2.00 | root | table:t1 | handle:[1234 123], keep
→ order:false, desc:false |
+--+
→ -----+-----+-----+-----+
→
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.27 sec)

Query OK, 1010 rows affected (0.06 sec)
Rows matched: 1010 Changed: 1010 Warnings: 0

Query OK, 0 rows affected (0.37 sec)

+--+
→ -----+-----+-----+
→
| id | estRows | task | access object | operator
→ info |
+--+
→ -----+-----+-----+
→
| Point_Get_1 | 1.00 | root | table:t1, index:unique_key(unique_key) |
→
+--+
→ -----+-----+-----+
→
1 row in set (0.00 sec)

+--+
→ -----+-----+-----+
→
| id | estRows | task | access object |
→ operator info |
+--+
```

```
+--+
→ -----+-----+-----+
→
| Batch_Point_Get_1 | 2.00 | root | table:t1, index:unique_key(unique_key)
→ | keep order:false, desc:false |
+--+
→ -----+-----+-----+
→
1 row in set (0.00 sec)
```

#### 8.3.2.3.4 IndexFullScan

Because indexes are ordered, the `IndexFullScan` operator can be used to optimize common queries such as the MIN or MAX values for an indexed value:

```
EXPLAIN SELECT MIN(intkey) FROM t1;
EXPLAIN SELECT MAX(intkey) FROM t1;
```

```
+--+
→ -----+-----+-----+
→
| id | estRows | task | access object
→ | operator info | |
+--+
→ -----+-----+-----+
→
| StreamAgg_12 | 1.00 | root |
→ | funcs:min(test.t1.intkey)->Column#4 |
| -Limit_16 | 1.00 | root |
→ | offset:0, count:1 |
| -IndexReader_29 | 1.00 | root |
→ | index:Limit_28 |
| -Limit_28 | 1.00 | cop[tikv] |
→ | offset:0, count:1 |
| -IndexFullScan_27 | 1.00 | cop[tikv] | table:t1, index:intkey(
→ intkey) | keep order:true |
+--+
→ -----+-----+-----+
→
5 rows in set (0.00 sec)
```

| id                                      | estRows | task                  | access object           |
|-----------------------------------------|---------|-----------------------|-------------------------|
| ↳   operator info                       |         |                       |                         |
| +--                                     |         |                       |                         |
| ↳                                       |         |                       |                         |
| ↳                                       |         |                       |                         |
| StreamAgg_12                            | 1.00    | root                  |                         |
| ↳   funcs:max(test.t1.intkey)->Column#4 |         |                       |                         |
| -Limit_16                               | 1.00    | root                  |                         |
| ↳                                       |         | offset:0, count:1     |                         |
| -IndexReader_29                         | 1.00    | root                  |                         |
| ↳                                       |         | index:Limit_28        |                         |
| -Limit_28                               | 1.00    | cop[tikv]             |                         |
| ↳                                       |         | offset:0, count:1     |                         |
| -IndexFullScan_27                       | 1.00    | cop[tikv]             | table:t1, index:intkey( |
| ↳ intkey)                               |         | keep order:true, desc |                         |
| +--                                     |         |                       |                         |
| ↳                                       |         |                       |                         |
| ↳                                       |         |                       |                         |
| 5 rows in set (0.00 sec)                |         |                       |                         |

In the above statements, an IndexFullScan task is performed on each TiKV Region. Despite the name FullScan, only the first row needs to be read ( -Limit\_28). Each TiKV Region returns its MIN or MAX value to TiDB, which then performs Stream Aggregation to filter for a single row. Stream Aggregation with the aggregation function MAX or MIN also ensures that NULL is returned if the table is empty.

By contrast, executing the MIN function on an unindexed value will result in TableFullScan. The query will require all rows to be scanned in TiKV, but a TopN calculation is performed to ensure each TiKV Region only returns one row to TiDB. Although TopN prevents excessive rows from being transferred between TiKV and TiDB, this statement is still considered far less efficient than the above example where MIN is able to make use of an index.

```
EXPLAIN SELECT MIN(pad1) FROM t1;
```

| +--                       |         |      |               |            |
|---------------------------|---------|------|---------------|------------|
| ↳                         |         |      |               |            |
| ↳                         |         |      |               |            |
| id                        | estRows | task | access object | operator   |
| ↳ info                    |         |      |               |            |
| +--                       |         |      |               |            |
| ↳                         |         |      |               |            |
| ↳                         |         |      |               |            |
| StreamAgg_13              | 1.00    | root |               | funcs:min( |
| ↳ test.t1.pad1)->Column#4 |         |      |               |            |

```

| -TopN_14 | 1.00 | root | | test.t1.
| ↵ pad1, offset:0, count:1 |
| -TableReader_23 | 1.00 | root | | data:
| ↵ TopN_22 |
| -TopN_22 | 1.00 | cop[tikv] | | test.t1.
| ↵ pad1, offset:0, count:1 |
| -Selection_21 | 1008.99 | cop[tikv] | | not(isnull(
| ↵ test.t1.pad1)) |
| -TableFullScan_20 | 1010.00 | cop[tikv] | table:t1 | keep order:
| ↵ false |
+--+
 +-----+
 ↵
 ↵
6 rows in set (0.00 sec)

```

The following statements will use the `IndexFullScan` operator to scan every row in the index:

```

EXPLAIN SELECT SUM(intkey) FROM t1;
EXPLAIN SELECT AVG(intkey) FROM t1;

```

```

+--+
 +-----+
 ↵
 ↵
| id | estRows | task | access object |
| ↵ operator info | |
+--+
 +-----+
 ↵
 ↵
| StreamAgg_20 | 1.00 | root | |
| ↵ funcs:sum(Column#6)->Column#4 |
| -IndexReader_21 | 1.00 | root | |
| ↵ index:StreamAgg_8 |
| -StreamAgg_8 | 1.00 | cop[tikv] | |
| ↵ funcs:sum(test.t1.intkey)->Column#6 |
| -IndexFullScan_19 | 1010.00 | cop[tikv] | table:t1, index:intkey(
| ↵ intkey) | keep order:false |
+--+
 +-----+
 ↵
 ↵
4 rows in set (0.00 sec)

```

```

| id | estRows | task | access object |
| ↗ operator info
+--+
| ↗
| ↗
| StreamAgg_20 | 1.00 | root | |
| ↗ funcs:avg(Column#7, Column#8)->Column#4
| -IndexReader_21 | 1.00 | root | |
| ↗ index:StreamAgg_8
| -StreamAgg_8 | 1.00 | cop[tikv] | |
| ↗ funcs:count(test.t1.intkey)->Column#7, funcs:sum(test.t1.intkey)->
| ↗ Column#8
| -IndexFullScan_19 | 1010.00 | cop[tikv] | table:t1, index:intkey(
| ↗ intkey) | keep order:false
+--+
| ↗
| ↗
4 rows in set (0.00 sec)

```

In the above examples, IndexFullScan is more efficient than TableFullScan because the width of the value in the (intkey + RowID) index is less than the width of the full row.

The following statement does not support using an IndexFullScan operator because additional columns are required from the table:

```
EXPLAIN SELECT AVG(intkey), ANY_VALUE(pad1) FROM t1;
```

```

+--+
| ↗
| ↗
| id | estRows | task | access object | operator
| ↗ info
| ↗
| ↗ |
+--+
| ↗
| ↗
| Projection_4 | 1.00 | root | | Column#4,
| ↗ any_value(test.t1.pad1)->Column#5
| ↗
| -StreamAgg_16 | 1.00 | root | | funcs:avg(
| ↗ Column#10, Column#11)->Column#4, funcs:firstrow(Column#12)->test.t1.
| ↗ pad1
| -TableReader_17 | 1.00 | root | | data:
| ↗ StreamAgg_8
| ↗

```

```

 ↵ |
| - StreamAgg_8 | 1.00 | cop[tikv] | | funcs:count(
 ↵ test.t1.intkey)->Column#10, funcs:sum(test.t1.intkey)->Column#11,
 ↵ funcs:firstrow(test.t1.pad1)->Column#12 |
| - TableFullScan_15 | 1010.00 | cop[tikv] | table:t1 | keep order:
 ↵ false
 ↵
 ↵ |
+--+
 ↵ -----+-----+-----+-----+
 ↵
5 rows in set (0.00 sec)

```

#### 8.3.2.4 Explain Statements That Use Joins

In TiDB, the SQL Optimizer needs to decide in which order tables should be joined and what is the most efficient join algorithm for a particular SQL statement. The examples in this document are based on the following sample data:

```

CREATE TABLE t1 (id BIGINT NOT NULL PRIMARY KEY auto_increment, pad1 BLOB,
 ↵ pad2 BLOB, pad3 BLOB, int_col INT NOT NULL DEFAULT 0);
CREATE TABLE t2 (id BIGINT NOT NULL PRIMARY KEY auto_increment, t1_id
 ↵ BIGINT NOT NULL, pad1 BLOB, pad2 BLOB, pad3 BLOB, INDEX(t1_id));
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM dual;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↵ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;

```

```

INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
UPDATE t1 SET int_col = 1 WHERE pad1 = (SELECT pad1 FROM t1 ORDER BY RAND()
 ↪ LIMIT 1);
SELECT SLEEP(1);
ANALYZE TABLE t1, t2;

```

#### 8.3.2.4.1 Index Join

If the number of estimated rows that need to be joined is small (typically less than 10000 rows), it is preferable to use the index join method. This method of join works similar to the primary method of join used in MySQL. In the following example, the operator `-> TableReader_28(Build)` first reads the table `t1`. For each row that matches, TiDB will probe the table `t2`:

```

EXPLAIN SELECT /*+ INL_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON t1.id =
 ↪ t2.t1_id;

```

| +--                                        |  |           |      |  |               |
|--------------------------------------------|--|-----------|------|--|---------------|
| +--                                        |  |           |      |  |               |
| id                                         |  | estRows   | task |  | access object |
|                                            |  |           |      |  |               |
| operator info                              |  |           |      |  |               |
| +--                                        |  |           |      |  |               |
| IndexJoin_10                               |  | 180000.00 | root |  |               |
|                                            |  |           |      |  |               |
| inner join, inner:IndexLookUp_9, outer key |  |           |      |  |               |

```

 ↗ :test.t1.id, inner key:test.t2.t1_id |
| -TableReader_28(Build) | 142020.00 | root |
 ↗ | data:TableFullScan_27
 ↗
| -TableFullScan_27 | 142020.00 | cop[tikv] | table:t1
 ↗ | keep order:false
 ↗
| -IndexLookUp_9(Probe) | 1.27 | root |
 ↗
 ↗
 ↗ |
| -IndexRangeScan_7(Build) | 1.27 | cop[tikv] | table:t2, index:
 ↗ t1_id(t1_id) | range: decided by [eq(test.t2.t1_id, test.t1.id)],
 ↗ keep order:false |
| -TableRowIDScan_8(Probe) | 1.27 | cop[tikv] | table:t2
 ↗ | keep order:false
 ↗
+--+
 ↗ -----
 ↗
6 rows in set (0.00 sec)

```

Index join is efficient in memory usage, but might be slower to execute than other join methods when a large number of probe operations are required. Consider also the following query:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.id=t2.t1_id WHERE t1.pad1 = 'value'
 ↗ and t2.pad1='value';
```

In an inner join operation, TiDB implements join reordering and might access either `t1` or `t2` first. Assume that TiDB selects `t1` as the first table to apply the `build` step, and then TiDB is able to filter on the predicate `t1.col = 'value'` before probing the table `t2`. The filter for the predicate `t2.col='value'` will be applied on each probe of table `t2`, which might be less efficient than other join methods.

Index join is effective if the build side is small and the probe side is pre-indexed and large. Consider the following query where an index join performs worse than a hash join and is not chosen by the SQL Optimizer:

```
-- DROP previously added index
ALTER TABLE t2 DROP INDEX t1_id;

EXPLAIN ANALYZE SELECT /*+ INL_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
 ↗ t1.id = t2.t1_id WHERE t1.int_col = 1;
EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
 ↗ t1.id = t2.t1_id WHERE t1.int_col = 1;
```

```
EXPLAIN ANALYZE SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.t1_id WHERE t1
 ↪ .int_col = 1;
```

Query OK, 0 rows affected (0.29 sec)

| +--                                                                                                                                                                                                                                                                                                                                                                         | id           | estRows  | actRows | task | access object | memory |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|----------|---------|------|---------------|--------|
| +--                                                                                                                                                                                                                                                                                                                                                                         | IndexJoin_13 | 90000.00 | 20000   | root | time          |        |
| ↳ :613.19955ms, loops:21, <b>inner</b> :{total:42.494047ms, concurrency:5, task:12, construct:33.149671ms, fetch:9.322956ms, build:8.66µs}, probe:32.435355ms                                                                                                                                                                                                               |              |          |         |      |               |        |
| ↳   <b>inner join</b> , <b>inner</b> :TableReader_9, <b>outer key</b> :test.t2.t1_id, <b>inner key</b> :test.t1.id   269.63341903686523 MB   N/A                                                                                                                                                                                                                            |              |          |         |      |               |        |
| -TableReader_19(Build)   90000.00   90000   root     time                                                                                                                                                                                                                                                                                                                   |              |          |         |      |               |        |
| ↳ :586.613252ms, loops:95, cop_task: {num: 3, <b>max</b> : 205.893949ms, <b>min</b> : 185.051354ms, <b>avg</b> : 194.878702ms, p95: 205.893949ms, max_proc_keys: 31715, p95_proc_keys: 31715, tot_proc: 332ms, tot_wait: 4ms, rpc_num: 4, rpc_time: 584.907774ms, copr_cache_hit_ratio: 0.00}, backoff{regionMiss: 2ms}   data:TableFullScan_18   182.624906539917 MB   N/A |              |          |         |      |               |        |
| -TableFullScan_18   90000.00   90000   cop[tikv]   <b>table</b> :t2   time                                                                                                                                                                                                                                                                                                  |              |          |         |      |               |        |
| ↳ :0ns, loops:0, tikv_task:{proc <b>max</b> :72ms, <b>min</b> :64ms, p80:72ms, p95:72ms, , iters:102, tasks:3}                                                                                                                                                                                                                                                              |              |          |         |      |               |        |
| ↳   keep <b>order:false</b>   N/A                                                                                                                                                                                                                                                                                                                                           |              |          |         |      |               |        |
| -TableReader_9(Probe)   0.00   5   root     time                                                                                                                                                                                                                                                                                                                            |              |          |         |      |               |        |
| ↳ :8.432051ms, loops:14, cop_task: {num: 14, <b>max</b> : 629.805µs, <b>min</b> : 226.129µs, <b>avg</b> : 420.979µs, p95: 629.805µs, max_proc_keys: 4, p95_proc_keys: 4, rpc_num: 15, rpc_time: 5.953229ms, copr_cache_hit_ratio: 0.00}   data:Selection_8   N/A                                                                                                            |              |          |         |      |               |        |

```

 ↗ | N/A |
| - Selection_8 | 0.00 | 5 | cop[tikv] | | time
 ↗ :0ns, loops:0, tikv_task:{proc max:0s, min:0s, p80:0s, p95:0s, iters
 ↗ :14, tasks:14}
 ↗
 ↗ | eq(test.t1.int_col, 1)
 ↗ | N/A | N/A
 ↗ /A |
| - TableRangeScan_7 | 1.00 | 25 | cop[tikv] | table:t1 | time
 ↗ :0ns, loops:0, tikv_task:{proc max:0s, min:0s, p80:0s, p95:0s, iters
 ↗ :14, tasks:14}
 ↗
 ↗ | range: decided by [test.t2.t1_id], keep order:false
 ↗ | N/A | N/A
+--+
 ↗ -----
 ↗
6 rows in set (0.61 sec)

+--+
 ↗ -----
 ↗
| id | estRows | actRows | task | access object |
 ↗ execution info
 ↗
 ↗ | operator info | memory |
 ↗ disk |
+--+
 ↗ -----
 ↗
| HashJoin_19 | 90000.00 | 20000 | root | | time
 ↗ :406.098528ms, loops:22, build_hash_table:{total:148.574644ms, fetch
 ↗ :146.843636ms, build:1.731008ms}, probe:{concurrency:5, total
 ↗ :2.026547436s, max:406.039309ms, probe:205.337813ms, fetch
 ↗ :1.821209623s} | inner join, equal:[eq(test.t1.id, test.t2.
 ↗ t1_id)] | 30.00731658935547 MB | 0 Bytes |
| -TableReader_22(Build) | 71.01 | 10000 | root | |
 ↗ time:147.072725ms, loops:12, cop_task: {num: 3, max: 145.847743ms,
 ↗ min: 50.932527ms, avg: 113.009029ms, p95: 145.847743ms, max_proc_keys
 ↗ : 31724, p95_proc_keys: 31724, tot_proc: 284ms, rpc_num: 3, rpc_time:
 ↗ 338.950488ms, copr_cache_hit_ratio: 0.00} | data:Selection_21 |
 ↗ 29.679713249206543 MB | N/A |
| - Selection_21 | 71.01 | 10000 | cop[tikv] | |
 ↗ time:0ns, loops:0, tikv_task:{proc max:132ms, min:48ms, p80:132ms,
 ↗ p95:132ms, iters:83, tasks:3}

```

```

 ↗
 ↗ | eq(test.t1.int_col, 1) | N/A | N
 ↗ /A |
| -TableFullScan_20 | 71010.00 | 71010 | cop[tikv] | table:t1 |
 ↗ time:0ns, loops:0, tikv_task:{proc max:128ms, min:48ms, p80:128ms,
 ↗ p95:128ms, iters:83, tasks:3}
 ↗
 ↗ | keep order:false | N/A | N
 ↗ /A |
| -TableReader_24(Probe) | 90000.00 | 90000 | root | |
 ↗ time:365.918504ms, loops:91, cop_task: {num: 3, max: 398.62145ms, min
 ↗ : 338.460345ms, avg: 358.732721ms, p95: 398.62145ms, max_proc_keys:
 ↗ 31715, p95_proc_keys: 31715, tot_proc: 536ms, rpc_num: 3, rpc_time:
 ↗ 1.076128895s, copr_cache_hit_ratio: 0.00} | data:TableFullScan_23 |
 ↗ 182.62489891052246 MB | N/A |
| -TableFullScan_23 | 90000.00 | 90000 | cop[tikv] | table:t2 |
 ↗ time:0ns, loops:0, tikv_task:{proc max:100ms, min:40ms, p80:100ms,
 ↗ p95:100ms, iters:102, tasks:3}
 ↗
 ↗ | keep order:false | N/A | N
 ↗ /A |
+--+
 ↗ -----+-----+-----+-----+
 ↗
6 rows in set (0.41 sec)

+--+
 ↗ -----+-----+-----+-----+
 ↗
| id | estRows | actRows | task | access object |
 ↗ execution info
 ↗
 ↗ | operator info | memory | |
 ↗ disk |
+--+
 ↗ -----+-----+-----+-----+
 ↗
| HashJoin_20 | 90000.00 | 20000 | root | time
 ↗ :441.897092ms, loops:21, build_hash_table:{total:138.600864ms, fetch
 ↗ :136.353899ms, build:2.246965ms}, probe:{concurrency:5, total
 ↗ :2.207403854s, max:441.850032ms, probe:148.01937ms, fetch:2.059384484
 ↗ s} | inner join, equal:[eq(test.t1.id, test.t2.t1_id)] |
 ↗ 30.00731658935547 MB | 0 Bytes |
| -TableReader_25(Build) | 71.01 | 10000 | root | |
 ↗ time:138.081807ms, loops:12, cop_task: {num: 3, max: 134.702901ms,

```

```

 ↵ min: 53.356202ms, avg: 93.372186ms, p95: 134.702901ms, max_proc_keys:
 ↵ 31724, p95_proc_keys: 31724, tot_proc: 236ms, rpc_num: 3, rpc_time:
 ↵ 280.017658ms, copr_cache_hit_ratio: 0.00} | data:Selection_24 |
 ↵ 29.680171966552734 MB | N/A |
| - Selection_24 | 71.01 | 10000 | cop[tikv] | |
 ↵ time:0ns, loops:0, tikv_task:{proc max:80ms, min:52ms, p80:80ms, p95
 ↵ :80ms, iters:83, tasks:3}
 ↵
 ↵ | eq(test.t1.int_col, 1) | N/A | N
 ↵ /A |
| - TableFullScan_23 | 71010.00 | 71010 | cop[tikv] | table:t1 |
 ↵ time:0ns, loops:0, tikv_task:{proc max:80ms, min:52ms, p80:80ms, p95
 ↵ :80ms, iters:83, tasks:3}
 ↵
 ↵ | keep order:false | N/A | N
 ↵ /A |
| - TableReader_22(Probe) | 90000.00 | 90000 | root | |
 ↵ time:413.560548ms, loops:91, cop_task: {num: 3, max: 432.938474ms,
 ↵ min: 231.263355ms, avg: 365.710741ms, p95: 432.938474ms,
 ↵ max_proc_keys: 31715, p95_proc_keys: 31715, tot_proc: 488ms, rpc_num:
 ↵ 3, rpc_time: 1.097021983s, copr_cache_hit_ratio: 0.00} | data:
 ↵ TableFullScan_21 | 182.62489891052246 MB | N/A |
| - TableFullScan_21 | 90000.00 | 90000 | cop[tikv] | table:t2 |
 ↵ time:0ns, loops:0, tikv_task:{proc max:80ms, min:80ms, p80:80ms, p95
 ↵ :80ms, iters:102, tasks:3}
 ↵
 ↵ | keep order:false | N/A | N
 ↵ /A |
+--+
 ↵ -----+-----+-----+-----+
 ↵
6 rows in set (0.44 sec)

```

In the above example, the index join operation is missing an index on `t1.int_col`. Once this index is added, the performance of the operation improves from `0.61 sec` to `0.14 sec`, as the following result shows:

```

-- Re-add index
ALTER TABLE t2 ADD INDEX (t1_id);

EXPLAIN ANALYZE SELECT /*+ INL_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
 ↵ t1.id = t2.t1_id WHERE t1.int_col = 1;
EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1 INNER JOIN t2 ON
 ↵ t1.id = t2.t1_id WHERE t1.int_col = 1;
EXPLAIN ANALYZE SELECT * FROM t1 INNER JOIN t2 ON t1.id = t2.t1_id WHERE t1

```

```
↪ .int_col = 1;
```

```
Query OK, 0 rows affected (3.65 sec)
```

|     |                        | estRows  | actRows | task      | access object                                                      |
|-----|------------------------|----------|---------|-----------|--------------------------------------------------------------------|
|     |                        |          |         |           | execution info                                                     |
|     |                        |          |         |           | operator info                                                      |
|     |                        |          |         |           | memory                                                             |
|     |                        |          |         |           | disk                                                               |
| +-- |                        |          |         |           |                                                                    |
|     |                        |          |         |           |                                                                    |
|     | IndexJoin_11           | 90000.00 | 0       | root      |                                                                    |
|     |                        |          |         |           | time:136.876686ms, loops:1, inner:{total                           |
|     |                        |          |         |           | :114.948158ms, concurrency:5, task:7, construct:5.329114ms, fetch  |
|     |                        |          |         |           | :109.610054ms, build:2.38μs}, probe:1.699799ms                     |
|     |                        |          |         |           |                                                                    |
|     |                        |          |         |           | inner join, inner:IndexLookUp_10, outer key:test.t1.id, inner key: |
|     |                        |          |         |           | test.t2.t1_id   29.864535331726074 MB   N/A                        |
|     | -TableReader_32(Build) | 10000.00 | 10000   | root      |                                                                    |
|     |                        |          |         |           | time:95.755212ms, loops:12, cop_task: {num                         |
|     |                        |          |         |           | : 3, max: 95.652443ms, min: 30.758712ms, avg: 57.545129ms, p95:    |
|     |                        |          |         |           | 95.652443ms, max_proc_keys: 31724, p95_proc_keys: 31724, tot_proc: |
|     |                        |          |         |           | 124ms, rpc_num: 3, rpc_time: 172.528417ms, copr_cache_hit_ratio:   |
|     |                        |          |         |           | 0.00}   data:Selection_31                                          |
|     |                        |          |         |           |                                                                    |
|     |                        |          |         |           | 29.679298400878906 MB   N/A                                        |
|     | -Selection_31          | 10000.00 | 10000   | cop[tikv] |                                                                    |
|     |                        |          |         |           | time:0ns, loops:0, tikv_task:{proc max:44ms,                       |
|     |                        |          |         |           | min:28ms, p80:44ms, p95:44ms, iters:84, tasks:3}                   |
|     |                        |          |         |           |                                                                    |
|     |                        |          |         |           | eq(test.t1.int_col, 1)   N/A                                       |
|     |                        |          |         |           | N/A                                                                |
|     | -TableFullScan_30      | 71010.00 | 71010   | cop[tikv] | table:t1                                                           |
|     |                        |          |         |           | time:0ns, loops:0, tikv_task:{proc max:44ms, min:28ms              |
|     |                        |          |         |           | , p80:44ms, p95:44ms, iters:84, tasks:3}                           |
|     |                        |          |         |           |                                                                    |
|     |                        |          |         |           | keep order:false   N/A                                             |
|     |                        |          |         |           | N/A                                                                |
|     | -IndexLookUp_10(Probe) | 9.00     | 0       | root      |                                                                    |

```

 ↵ | time:103.93801ms, loops:7
 ↵
 ↵ |
 ↵
 ↵ | 2.1787109375 KB | N/A |
| - IndexRangeScan_8(Build) | 9.00 | 0 | cop[tikv] | table:t2,
 ↵ index:t1_id(t1_id) | time:0s, loops:0, cop_task: {num: 7, max:
 ↵ 23.969244ms, min: 12.003682ms, avg: 14.659066ms, p95: 23.969244ms,
 ↵ tot_proc: 100ms, rpc_num: 7, rpc_time: 102.435966ms,
 ↵ copr_cache_hit_ratio: 0.00}, tikv_task:{proc max:24ms, min:12ms, p80
 ↵ :16ms, p95:24ms, iters:7, tasks:7} | range: decided by [eq(test.t2.
 ↵ t1_id, test.t1.id)], keep order:false | N/A | N/A |
| - TableRowIDScan_9(Probe) | 9.00 | 0 | cop[tikv] | table:t2
 ↵ | time:0ns, loops:0
 ↵
 ↵ | keep order:false
 ↵
 ↵ | N/A |
+--+
 ↵ -----
 ↵
7 rows in set (0.14 sec)

+--+
 ↵ -----
 ↵
| id | estRows | actRows | task | access object |
 ↵ execution info
 ↵
 ↵ | operator info | memory |
 ↵ disk |
+--+
 ↵ -----
 ↵
| HashJoin_31 | 90000.00 | 0 | root | time
 ↵ :402.263795ms, loops:1, build_hash_table:{total:128.467151ms, fetch
 ↵ :126.871282ms, build:1.595869ms}, probe:{concurrency:5, total
 ↵ :2.010969815s, max:402.212295ms, probe:8.924769ms, fetch:2.002045046s
 ↵ } | inner join, equal:[eq(test.t1.id, test.t2.t1_id)] |
 ↵ 29.689788818359375 MB | 0 Bytes |
| -TableReader_34(Build) | 10000.00 | 10000 | root | time
 ↵ :126.765972ms, loops:11, cop_task: {num: 3, max: 126.721293ms,
 ↵ min: 54.375481ms, avg: 84.518849ms, p95: 126.721293ms, max_proc_keys:
 ↵ 31724, p95_proc_keys: 31724, tot_proc: 208ms, rpc_num: 3, rpc_time:
 ↵ 253.478218ms, copr_cache_hit_ratio: 0.00} | data:Selection_33 |

```

```

 ↵ 29.679292678833008 MB | N/A |
| - Selection_33 | 10000.00 | 10000 | cop[tikv] | |
 ↵ time:0ns, loops:0, tikv_task:{proc max:72ms, min:56ms, p80:72ms, p95
 ↵ :72ms, iters:84, tasks:3}
 ↵
 ↵ | eq(test.t1.int_col, 1) | N/A | N
 ↵ /A |
| - TableFullScan_32 | 71010.00 | 71010 | cop[tikv] | table:t1 |
 ↵ time:0ns, loops:0, tikv_task:{proc max:72ms, min:56ms, p80:72ms, p95
 ↵ :72ms, iters:84, tasks:3}
 ↵
 ↵ | keep order:false | N/A | N
 ↵ /A |
| - TableReader_36(Probe) | 90000.00 | 90000 | root | |
 ↵ time:400.447175ms, loops:90, cop_task: {num: 3, max: 400.838264ms,
 ↵ min: 309.474053ms, avg: 341.01943ms, p95: 400.838264ms, max_proc_keys
 ↵ : 31719, p95_proc_keys: 31719, tot_proc: 528ms, rpc_num: 3, rpc_time:
 ↵ 1.02298055s, copr_cache_hit_ratio: 0.00} | data:TableFullScan_35 |
 ↵ 182.62786674499512 MB | N/A |
| - TableFullScan_35 | 90000.00 | 90000 | cop[tikv] | table:t2 |
 ↵ time:0ns, loops:0, tikv_task:{proc max:108ms, min:72ms, p80:108ms,
 ↵ p95:108ms, iters:102, tasks:3}
 ↵
 ↵ | keep order:false | N/A | N
 ↵ /A |
+--+
 ↵ -----
 ↵
 ↵
6 rows in set (0.40 sec)

+--+
 ↵ -----
 ↵
 ↵
| id | estRows | actRows | task | access object |
 ↵ execution info
 ↵
 ↵ | operator info | memory | N
 ↵ disk |
+--+
 ↵ -----
 ↵
 ↵
| HashJoin_32 | 90000.00 | 0 | root | | time
 ↵ :356.282882ms, loops:1, build_hash_table:{total:154.187155ms, fetch
 ↵ :151.259305ms, build:2.92785ms}, probe:{concurrency:5, total
 ↵ :1.781087041s, max:356.238312ms, probe:7.406146ms, fetch:1.773680895s

```

```

 ↵ } | inner join, equal:[eq(test.t1.id, test.t2.t1_id)] |
 ↵ 29.689788818359375 MB | 0 Bytes |
| -TableReader_41(Build) | 10000.00 | 10000 | root | |
 ↵ time:151.190175ms, loops:11, cop_task: {num: 3, max: 151.055697ms,
 ↵ min: 56.214348ms, avg: 96.70463ms, p95: 151.055697ms, max_proc_keys:
 ↵ 31724, p95_proc_keys: 31724, tot_proc: 240ms, rpc_num: 3, rpc_time:
 ↵ 290.019942ms, copr_cache_hit_ratio: 0.00} | data:Selection_40 |
 ↵ 29.679292678833008 MB | N/A |
| -Selection_40 | 10000.00 | 10000 | cop[tikv] | |
 ↵ time:0ns, loops:0, tikv_task:{proc max:80ms, min:56ms, p80:80ms, p95
 ↵ :80ms, iters:84, tasks:3}
 ↵
 ↵ | eq(test.t1.int_col, 1) | N/A | N
 ↵ /A |
| -TableFullScan_39 | 71010.00 | 71010 | cop[tikv] | table:t1 |
 ↵ time:0ns, loops:0, tikv_task:{proc max:80ms, min:56ms, p80:80ms, p95
 ↵ :80ms, iters:84, tasks:3}
 ↵
 ↵ | keep order:false | N/A | N
 ↵ /A |
| -TableReader_43(Probe) | 90000.00 | 90000 | root | |
 ↵ time:354.68523ms, loops:90, cop_task: {num: 3, max: 354.313475ms, min
 ↵ : 328.460762ms, avg: 345.530558ms, p95: 354.313475ms, max_proc_keys:
 ↵ 31719, p95_proc_keys: 31719, tot_proc: 508ms, rpc_num: 3, rpc_time:
 ↵ 1.036492374s, copr_cache_hit_ratio: 0.00} | data:TableFullScan_42 |
 ↵ 182.62786102294922 MB | N/A |
| -TableFullScan_42 | 90000.00 | 90000 | cop[tikv] | table:t2 |
 ↵ time:0ns, loops:0, tikv_task:{proc max:84ms, min:64ms, p80:84ms, p95
 ↵ :84ms, iters:102, tasks:3}
 ↵
 ↵ | keep order:false | N/A | N
 ↵ /A |
+--+
 ↵ -----
 ↵
6 rows in set (0.36 sec)

```

### Note:

In the above example, the SQL Optimizer selects the hash join plan which performs worse than the index join. Query optimization is an [NP-complete problem](#), and less-than-optimal plans might be chosen. If this is a frequent query, it is recommended to use [SQL Plan Management](#) to bind a hint to a

query, which can be easier to manage than inserting hints into queries that your application sends to TiDB.

## Variations of Index Join

An index join operation using the hint `INL_JOIN` creates a hash table of the intermediate results before joining on the outer table. TiDB also supports creating a hash table on the outer table using the hint `INL_HASH_JOIN`. Each of these variations of index join is automatically selected by the SQL Optimizer.

### Configuration

Index join performance is influenced by the following system variables:

- `tidb_index_join_batch_size` (default value: 25000) - the batch size of `index` ↪ `lookup join` operations.
- `tidb_index_lookup_join_concurrency` (default value: 4) - the number of concurrent index lookup tasks.

### 8.3.2.4.2 Hash Join

In a hash join operation, TiDB reads and caches the data on the `Build` side of the join in a hash table, and then reads the data on the `Probe` side of the join, probing the hash table to access required rows. Hash joins require more memory to execute than index joins but execute much faster when there are a lot of rows that need to be joined. The hash join operator is multi-threaded in TiDB and executes in parallel.

An example of hash join is as follows:

```
EXPLAIN SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id = t2.id;
```

| +-- |                                      |           |           |               |             |  |
|-----|--------------------------------------|-----------|-----------|---------------|-------------|--|
| +-- | id                                   | estRows   | task      | access object | operator    |  |
|     | ↳ info                               |           |           |               |             |  |
| +-- |                                      |           |           |               |             |  |
|     | ↳                                    |           |           |               |             |  |
|     | HashJoin_27                          | 142020.00 | root      |               | inner join, |  |
|     | ↳ equal:[eq(test.t1.id, test.t2.id)] |           |           |               |             |  |
|     | -TableReader_29(Build)               | 142020.00 | root      |               | data:       |  |
|     | ↳ TableFullScan_28                   |           |           |               |             |  |
|     | -TableFullScan_28                    | 142020.00 | cop[tikv] | table:t1      | keep order: |  |
|     | ↳ false                              |           |           |               |             |  |

```

| └─TableReader_31(Probe) | 180000.00 | root | | data:
| └─TableFullScan_30 | |
| └─TableFullScan_30 | 180000.00 | cop[tikv] | table:t2 | keep order:
| └─false | |
+---+
| └──────────────────+-----+-----+-----+
| └────────────────+-----+
5 rows in set (0.00 sec)

```

For the execution process of HashJoin\_27, TiDB performs the following operations in order:

1. Cache the data of the Build side in memory.
2. Construct a Hash Table on the Build side based on the cached data.
3. Read the data at the Probe side.
4. Use the data of the Probe side to probe the Hash Table.
5. Return qualified data to the user.

The operator `info` column in the EXPLAIN result table also records other information about HashJoin\_27, including whether the query is Inner Join or Outer Join, and what are the conditions of Join. In the above example, the query is an Inner Join, where the Join condition `equal:[eq(test.t1.id, test.t2.id)]` partly corresponds with the query condition `WHERE t1.id = t2.id`. The operator info of the other Join operators in the following examples is similar to this one.

#### Runtime Statistics

If `tidb_mem_quota_query` (default value: 1GB) is exceeded, TiDB will attempt to use temporary storage on condition that the `oom-use-tmp-storage` value is `true` (default). This means that the Build operator used as part of the hash join might be created on disk. Runtime statistics such as memory usage are visible in the `execution_info` of the EXPLAIN → ANALYZE result table. The following example shows the output of EXPLAIN ANALYZE with a 1GB (default) `tidb_mem_quota_query` quota, and a 500MB quota. At 500MB, disk is used for temporary storage:

```

EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id =
 ↪ t2.id;
SET tidb_mem_quota_query=500 * 1024 * 1024;
EXPLAIN ANALYZE SELECT /*+ HASH_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id =
 ↪ t2.id;

```

```

+---+
| └──────────────────+-----+-----+-----+
| └────────────────+-----+

```

| id                                                                                      | estRows   | actRows | task                         | access object |
|-----------------------------------------------------------------------------------------|-----------|---------|------------------------------|---------------|
| ↗ execution info                                                                        |           |         |                              |               |
| ↗                                                                                       |           |         |                              |               |
| ↗   operator info                                                                       |           |         |                              | memory        |
| ↗ disk                                                                                  |           |         |                              |               |
| +--                                                                                     |           |         |                              |               |
| ↗ -----+-----+-----+-----+                                                              |           |         |                              |               |
| ↗                                                                                       |           |         |                              |               |
| HashJoin_27                                                                             | 142020.00 | 71010   | root                         | <b>time</b>   |
| ↗ :647.508572ms, loops:72, build_hash_table:{total:579.254415ms, fetch                  |           |         |                              |               |
| ↗ :566.91012ms, build:12.344295ms}, probe:{concurrency:5, total                         |           |         |                              |               |
| ↗ :3.23315006s, <b>max</b> :647.520113ms, probe:330.884716ms, fetch:2.902265344         |           |         |                              |               |
| ↗ s}   <b>inner join</b> , equal:[eq(test.t1.id, test.t2.id)]                           |           |         |                              |               |
| ↗ 209.61642456054688 MB   0 Bytes                                                       |           |         |                              |               |
| -TableReader_29(Build)                                                                  | 142020.00 | 71010   | root                         |               |
| ↗ <b>time</b> :567.088247ms, loops:72, cop_task: {num: 2, <b>max</b> : 569.809411ms,    |           |         |                              |               |
| ↗ <b>min</b> : 369.67451ms, <b>avg</b> : 469.74196ms, p95: 569.809411ms, max_proc_keys: |           |         |                              |               |
| ↗ 39245, p95_proc_keys: 39245, tot_proc: 400ms, rpc_num: 2, rpc_time:                   |           |         |                              |               |
| ↗ 939.447231ms, copr_cache_hit_ratio: 0.00}   data:TableFullScan_28                     |           |         |                              |               |
| ↗ 210.2100534439087 MB   N/A                                                            |           |         |                              |               |
| -TableFullScan_28                                                                       | 142020.00 | 71010   | cop[tikv]   <b>table</b> :t1 |               |
| ↗ proc <b>max</b> :64ms, <b>min</b> :48ms, p80:64ms, p95:64ms, iters:79, tasks:2        |           |         |                              |               |
| ↗                                                                                       |           |         |                              |               |
| ↗   <b>keep order:false</b>                                                             |           |         | N/A                          | N/A           |
| ↗                                                                                       |           |         |                              |               |
| -TableReader_31(Probe)                                                                  | 180000.00 | 90000   | root                         |               |
| ↗ <b>time</b> :337.233636ms, loops:91, cop_task: {num: 3, <b>max</b> : 569.790741ms,    |           |         |                              |               |
| ↗ <b>min</b> : 332.758911ms, <b>avg</b> : 421.543165ms, p95: 569.790741ms,              |           |         |                              |               |
| ↗ max_proc_keys: 31719, p95_proc_keys: 31719, tot_proc: 500ms, rpc_num:                 |           |         |                              |               |
| ↗ 3, rpc_time: 1.264570696s, copr_cache_hit_ratio: 0.00}   data:                        |           |         |                              |               |
| ↗ TableFullScan_30   267.1126985549927 MB   N/A                                         |           |         |                              |               |
| -TableFullScan_30                                                                       | 180000.00 | 90000   | cop[tikv]   <b>table</b> :t2 |               |
| ↗ proc <b>max</b> :84ms, <b>min</b> :72ms, p80:84ms, p95:84ms, iters:102, tasks:3       |           |         |                              |               |
| ↗                                                                                       |           |         |                              |               |
| ↗   <b>keep order:false</b>                                                             |           |         | N/A                          | N/A           |
| ↗                                                                                       |           |         |                              |               |
| +--                                                                                     |           |         |                              |               |
| ↗ -----+-----+-----+-----+                                                              |           |         |                              |               |
| ↗                                                                                       |           |         |                              |               |
| 5 rows in set (0.65 sec)                                                                |           |         |                              |               |

Query OK, 0 rows affected (0.00 sec)

+--

  | ↗ -----+-----+-----+-----+

| id                                                                      | estRows   | actRows | task      | access object | memory |
|-------------------------------------------------------------------------|-----------|---------|-----------|---------------|--------|
| ↳ execution info                                                        |           |         |           |               |        |
| ↳ operator info                                                         |           |         |           |               |        |
| ↳ disk                                                                  |           |         |           |               |        |
| +--                                                                     |           |         |           |               |        |
| ↳ HashJoin_27                                                           | 142020.00 | 71010   | root      |               | time   |
| ↳ :963.983353ms, loops:72, build_hash_table:{total:775.961447ms, fetch  |           |         |           |               |        |
| ↳ :503.789677ms, build:272.17177ms}, probe:{concurrency:5, total        |           |         |           |               |        |
| ↳ :4.805454793s, max:963.973133ms, probe:922.156835ms, fetch            |           |         |           |               |        |
| ↳ :3.883297958s}   inner join, equal:[eq(test.t1.id, test.t2.           |           |         |           |               |        |
| ↳ id)]   93.53974533081055 MB   210.7459259033203 MB                    |           |         |           |               |        |
| ↳ -TableReader_29(Build)                                                | 142020.00 | 71010   | root      |               |        |
| ↳ time:504.062018ms, loops:72, cop_task: {num: 2, max: 509.276857ms,    |           |         |           |               |        |
| ↳ min: 402.66386ms, avg: 455.970358ms, p95: 509.276857ms, max_proc_keys |           |         |           |               |        |
| ↳ : 39245, p95_proc_keys: 39245, tot_proc: 384ms, rpc_num: 2, rpc_time: |           |         |           |               |        |
| ↳ 911.893237ms, copr_cache_hit_ratio: 0.00}   data:TableFullScan_28     |           |         |           |               |        |
| ↳ 210.20934200286865 MB   N/A                                           |           |         |           |               |        |
| ↳ -TableFullScan_28                                                     | 142020.00 | 71010   | cop[tikv] | table:t1      |        |
| ↳ proc max:88ms, min:72ms, p80:88ms, p95:88ms, iters:79, tasks:2        |           |         |           |               |        |
| ↳   keep order:false                                                    |           |         |           | N/A           | N/A    |
| +--                                                                     |           |         |           |               |        |
| ↳ -TableReader_31(Probe)                                                | 180000.00 | 90000   | root      |               |        |
| ↳ time:363.058382ms, loops:91, cop_task: {num: 3, max: 412.659191ms,    |           |         |           |               |        |
| ↳ min: 358.489688ms, avg: 391.463008ms, p95: 412.659191ms,              |           |         |           |               |        |
| ↳ max_proc_keys: 31719, p95_proc_keys: 31719, tot_proc: 484ms, rpc_num: |           |         |           |               |        |
| ↳ 3, rpc_time: 1.174326746s, copr_cache_hit_ratio: 0.00}   data:        |           |         |           |               |        |
| ↳ TableFullScan_30   267.11340618133545 MB   N/A                        |           |         |           |               |        |
| ↳ -TableFullScan_30                                                     | 180000.00 | 90000   | cop[tikv] | table:t2      |        |
| ↳ proc max:92ms, min:64ms, p80:92ms, p95:92ms, iters:102, tasks:3       |           |         |           |               |        |
| ↳   keep order:false                                                    |           |         |           | N/A           | N/A    |
| +--                                                                     |           |         |           |               |        |
| ↳ 5 rows in set (0.98 sec)                                              |           |         |           |               |        |

Configuration

Hash join performance is influenced by the following system variables:

- `tidb_mem_quota_query` (default value: 1GB) - if the memory quota for a query is exceeded, TiDB will attempt to spill the Build operator of a hash join to disk to save memory.
- `tidb_hash_join_concurrency` (default value: 5) - the number of concurrent hash join tasks.

#### 8.3.2.4.3 Merge Join

Merge join is a special sort of join that applies when both sides of the join are read in sorted order. It can be described as similar to an *efficient zipper merge*: as data is read on both the Build and the Probe sides of the join, the join operation works like a streaming operation. Merge joins require far less memory than hash join but do not execute in parallel.

The following is an example:

```
EXPLAIN SELECT /*+ MERGE_JOIN(t1, t2) */ * FROM t1, t2 WHERE t1.id = t2.id;
```

|                                             |   | estRows   | task      | access object | operator    |
|---------------------------------------------|---|-----------|-----------|---------------|-------------|
| +--                                         | ↳ |           |           |               |             |
| ↳                                           |   |           |           |               |             |
| id                                          |   |           |           |               |             |
| ↳ info                                      |   |           |           |               |             |
| +--                                         | ↳ |           |           |               |             |
| ↳                                           |   |           |           |               |             |
| MergeJoin_7                                 |   | 142020.00 | root      |               | inner join, |
| ↳ left key:test.t1.id, right key:test.t2.id |   |           |           |               |             |
| -TableReader_12(Build)                      |   | 180000.00 | root      |               | data:       |
| ↳ TableFullScan_11                          |   |           |           |               |             |
| -TableFullScan_11                           |   | 180000.00 | cop[tikv] | table:t2      | keep order: |
| ↳ true                                      |   |           |           |               |             |
| -TableReader_10(Probe)                      |   | 142020.00 | root      |               | data:       |
| ↳ TableFullScan_9                           |   |           |           |               |             |
| -TableFullScan_9                            |   | 142020.00 | cop[tikv] | table:t1      | keep order: |
| ↳ true                                      |   |           |           |               |             |
| +--                                         | ↳ |           |           |               |             |
| ↳                                           |   |           |           |               |             |
| 5 rows in set (0.00 sec)                    |   |           |           |               |             |

For the execution process of the merge join operator, TiDB performs the following operations:

1. Read all the data of a Join Group from the Build side into the memory.
2. Read the data of the Probe side.

3. Compare whether each row of data on the **Probe** side matches a complete Join Group on the **Build** side. Apart from equivalent conditions, there are non-equivalent conditions. Here “match” mainly refers to checking whether non-equivalent conditions are met. Join Group refers to the data with the same value among all Join Keys.

### 8.3.2.5 Explain Statements in the MPP Mode

TiDB supports using the [MPP mode](#) to execute queries. In the MPP mode, the TiDB optimizer generates execution plans for MPP. Note that the MPP mode is only available for tables that have replicas on [TiFlash](#).

The examples in this document are based on the following sample data:

```
CREATE TABLE t1 (id int, value int);
INSERT INTO t1 values(1,2),(2,3),(1,3);
ALTER TABLE t1 set tiflash replica 1;
ANALYZE TABLE t1;
SET tidb_allow_mpp = 1;
```

### 8.3.2.5.1 MPP query fragments and MPP tasks

In the MPP mode, a query is logically sliced into multiple query fragments. Take the following statement as an example:

```
EXPLAIN SELECT COUNT(*) FROM t1 GROUP BY id;
```

This query is divided into two fragments in the MPP mode. One for the first-stage aggregation and the other for the second-stage aggregation, also the final aggregation. When this query is executed, each query fragment is instantiated into one or more MPP tasks.

### 8.3.2.5.2 Exchange operators

`ExchangeReceiver` and `ExchangeSender` are two exchange operators specific for MPP execution plans. The `ExchangeReceiver` operator reads data from downstream query fragments and the `ExchangeSender` operator sends data from downstream query fragments to upstream query fragments. In the MPP mode, the root operator of each MPP query fragment is `ExchangeSender`, meaning that query fragments are delimited by the `ExchangeSender` operator.

The following is a simple MPP execution plan:

```
EXPLAIN SELECT COUNT(*) FROM t1 GROUP BY id;
```

|                                                      |                          |                   |          |  |  |  |
|------------------------------------------------------|--------------------------|-------------------|----------|--|--|--|
| +--                                                  |                          |                   |          |  |  |  |
| ↳                                                    | -----+-----+-----+-----+ |                   |          |  |  |  |
| ↳                                                    |                          |                   |          |  |  |  |
| TableReader_31                                       | 2.00                     | root              |          |  |  |  |
| ↳ data:ExchangeSender_30                             |                          |                   |          |  |  |  |
| -ExchangeSender_30                                   | 2.00                     | batchCop[tiflash] |          |  |  |  |
| ↳ ExchangeType: PassThrough                          |                          |                   |          |  |  |  |
| -Projection_26                                       | 2.00                     | batchCop[tiflash] |          |  |  |  |
| ↳ Column#4                                           |                          |                   |          |  |  |  |
| -HashAgg_27                                          | 2.00                     | batchCop[tiflash] |          |  |  |  |
| ↳ group by:test.t1.id, funcs:sum(Column#7)->Column#4 |                          |                   |          |  |  |  |
| -ExchangeReceiver_29                                 | 2.00                     | batchCop[tiflash] |          |  |  |  |
| ↳                                                    |                          |                   |          |  |  |  |
| - ExchangeSender_28                                  | 2.00                     | batchCop[tiflash] |          |  |  |  |
| ↳ ExchangeType: HashPartition, Hash Cols: test.t1.id |                          |                   |          |  |  |  |
| - HashAgg_9                                          | 2.00                     | batchCop[tiflash] |          |  |  |  |
| ↳ group by:test.t1.id, funcs:count(1)->Column#7      |                          |                   |          |  |  |  |
| - TableFullScan_25                                   | 3.00                     | batchCop[tiflash] | table:t1 |  |  |  |
| ↳ keep order:false                                   |                          |                   |          |  |  |  |
| +--                                                  |                          |                   |          |  |  |  |
| ↳                                                    | -----+-----+-----+-----+ |                   |          |  |  |  |
| ↳                                                    |                          |                   |          |  |  |  |

The above execution plan contains two query fragments:

- The first is [TableFullScan\_25, HashAgg\_9, ExchangeSender\_28], which is mainly responsible for the first-stage aggregation.
- The second is [ExchangeReceiver\_29, HashAgg\_27, Projection\_26, ExchangeSender\_30  
  ↳ ], which is mainly responsible for the second-stage aggregation.

The operator `info` column of the `ExchangeSender` operator shows the exchange type information. Currently, there are three exchange types. See the following:

- HashPartition: The `ExchangeSender` operator firstly partitions data according to the Hash values and then distributes data to the `ExchangeReceiver` operator of upstream MPP tasks. This exchange type is often used for Hash Aggregation and Shuffle Hash Join algorithms.
- Broadcast: The `ExchangeSender` operator distributes data to upstream MPP tasks through broadcast. This exchange type is often used for Broadcast Join.
- PassThrough: The `ExchangeSender` operator sends data to the only upstream MPP task, which is different from the Broadcast type. This exchange type is often used when returning data to TiDB.

In the example execution plan, the exchange type of the operator `ExchangeSender_28` is HashPartition, meaning that it performs the Hash Aggregation algorithm. The exchange

type of the operator `ExchangeSender_30` is `PassThrough`, meaning that it is used to return data to TiDB.

MPP is also often applied to join operations. The MPP mode in TiDB supports the following two join algorithms:

- Shuffle Hash Join: Shuffle the data input from the join operation using the `HashPartition` exchange type. Then, upstream MPP tasks join data within the same partition.
- Broadcast Join: Broadcast data of the small table in the join operation to each node, after which each node joins the data separately.

The following is a typical execution plan for Shuffle Hash Join:

```
SET tidb_opt_broadcast_join=0;
SET tidb_broadcast_join_threshold_count=0;
SET tidb_broadcast_join_threshold_size=0;
EXPLAIN SELECT COUNT(*) FROM t1 a JOIN t1 b ON a.id = b.id;
```

| +--                                                              |         |              |               |  |  |
|------------------------------------------------------------------|---------|--------------|---------------|--|--|
| +--                                                              |         |              |               |  |  |
| id                                                               | estRows | task         | access object |  |  |
| operator info                                                    |         |              |               |  |  |
| +--                                                              |         |              |               |  |  |
| StreamAgg_14                                                     | 1.00    | root         |               |  |  |
| funcs: <code>count(1)-&gt;Column#7</code>                        |         |              |               |  |  |
| -TableReader_48                                                  | 9.00    | root         |               |  |  |
| data: <code>ExchangeSender_47</code>                             |         |              |               |  |  |
| -ExchangeSender_47                                               | 9.00    | cop[tiflash] |               |  |  |
| ExchangeType: <code>PassThrough</code>                           |         |              |               |  |  |
| -HashJoin_44                                                     | 9.00    | cop[tiflash] |               |  |  |
| inner join, equal:[eq(test.t1.id, test.t1.id)]                   |         |              |               |  |  |
| -ExchangeReceiver_19(Build)                                      | 6.00    | cop[tiflash] |               |  |  |
|                                                                  |         |              |               |  |  |
| -ExchangeSender_18                                               | 6.00    | cop[tiflash] |               |  |  |
| ExchangeType: <code>HashPartition</code> , Hash Cols: test.t1.id |         |              |               |  |  |
| -Selection_17                                                    | 6.00    | cop[tiflash] |               |  |  |
| not(isnull(test.t1.id))                                          |         |              |               |  |  |
| -TableFullScan_16                                                | 6.00    | cop[tiflash] | table:a       |  |  |
| keep <code>order:false</code>                                    |         |              |               |  |  |
| -ExchangeReceiver_23(Probe)                                      | 6.00    | cop[tiflash] |               |  |  |
|                                                                  |         |              |               |  |  |
| -ExchangeSender_22                                               | 6.00    | cop[tiflash] |               |  |  |
| ExchangeType: <code>HashPartition</code> , Hash Cols: test.t1.id |         |              |               |  |  |

|     |                           |                     |         |  |
|-----|---------------------------|---------------------|---------|--|
|     | - Selection_21            | 6.00   cop[tiflash] |         |  |
|     | ↳ not(isnull(test.t1.id)) |                     |         |  |
|     | - TableFullScan_20        | 6.00   cop[tiflash] | table:b |  |
|     | ↳ keep order:false        |                     |         |  |
| +-- |                           |                     |         |  |
|     | ↳ -----+-----+-----+      |                     |         |  |
|     | ↳                         |                     |         |  |
|     | 12 rows in set (0.00 sec) |                     |         |  |

In the above execution plan:

- The query fragment [TableFullScan\_20, Selection\_21, ExchangeSender\_22] reads data from table b and shuffles data to upstream MPP tasks.
- The query fragment [TableFullScan\_16, Selection\_17, ExchangeSender\_18] reads data from table a and shuffles data to upstream MPP tasks.
- The query fragment [ExchangeReceiver\_19, ExchangeReceiver\_23, HashJoin\_44 ↳ , ExchangeSender\_47] joins all data and returns it to TiDB.

A typical execution plan for Broadcast Join is as follows:

```
EXPLAIN SELECT COUNT(*) FROM t1 a JOIN t1 b ON a.id = b.id;
```

|                             |                                                  |                     |  |               |
|-----------------------------|--------------------------------------------------|---------------------|--|---------------|
| +--                         |                                                  |                     |  |               |
|                             | ↳ -----+-----+-----+                             |                     |  |               |
|                             | ↳                                                |                     |  |               |
| id                          |                                                  | estRows   task      |  | access object |
|                             | ↳ operator info                                  |                     |  |               |
| +--                         |                                                  |                     |  |               |
|                             | ↳ -----+-----+-----+                             |                     |  |               |
|                             | ↳                                                |                     |  |               |
| StreamAgg_15                |                                                  | 1.00   root         |  |               |
|                             | ↳ funcs:count(1)->Column#7                       |                     |  |               |
| -TableReader_47             |                                                  | 9.00   root         |  |               |
|                             | ↳ data:ExchangeSender_46                         |                     |  |               |
| -ExchangeSender_46          |                                                  | 9.00   cop[tiflash] |  |               |
|                             | ↳ ExchangeType: PassThrough                      |                     |  |               |
| -HashJoin_43                |                                                  | 9.00   cop[tiflash] |  |               |
|                             | ↳ inner join, equal:[eq(test.t1.id, test.t1.id)] |                     |  |               |
| -ExchangeReceiver_20(Build) | 6.00   cop[tiflash]                              |                     |  |               |
|                             |                                                  |                     |  |               |
| -ExchangeSender_19          | 6.00   cop[tiflash]                              |                     |  |               |
|                             | ↳ ExchangeType: Broadcast                        |                     |  |               |
| -Selection_18               | 6.00   cop[tiflash]                              |                     |  |               |
|                             | ↳ not(isnull(test.t1.id))                        |                     |  |               |

|     |                           |      |              |         |  |
|-----|---------------------------|------|--------------|---------|--|
|     | - TableFullScan_17        | 6.00 | cop[tiflash] | table:a |  |
|     | ↳ keep order:false        |      |              |         |  |
|     | - Selection_22(Probe)     | 6.00 | cop[tiflash] |         |  |
|     | ↳ not(isnull(test.t1.id)) |      |              |         |  |
|     | - TableFullScan_21        | 6.00 | cop[tiflash] | table:b |  |
|     | ↳ keep order:false        |      |              |         |  |
| +-- |                           |      | -----+-----+ |         |  |
|     | ↳ -----+-----+            |      |              |         |  |
|     |                           |      |              |         |  |

In the above execution plan:

- The query fragment [TableFullScan\_17, Selection\_18, ExchangeSender\_19] reads data from the small table (table a) and broadcasts the data to each node that contains data from the large table (table b).
- The query fragment [TableFullScan\_21, Selection\_22, ExchangeReceiver\_20, ↳ HashJoin\_43, ExchangeSender\_46] joins all data and returns it to TiDB.

### 8.3.2.5.3 EXPLAIN ANALYZE statements in the MPP mode

The EXPLAIN ANALYZE statement is similar to EXPLAIN, but it also outputs some runtime information.

The following is the output of a simple EXPLAIN ANALYZE example:

```
EXPLAIN ANALYZE SELECT COUNT(*) FROM t1 GROUP BY id;
```

|                    |                                                                      |                              |  |        |        |
|--------------------|----------------------------------------------------------------------|------------------------------|--|--------|--------|
| +--                |                                                                      |                              |  |        |        |
|                    | ↳ -----+-----+-----+-----+                                           |                              |  |        |        |
|                    |                                                                      |                              |  |        |        |
| id                 |                                                                      | estRows   actRows   task     |  | access |        |
|                    | ↳ object   execution info                                            |                              |  |        |        |
|                    |                                                                      |                              |  |        |        |
|                    | ↳ operator info                                                      |                              |  |        | memory |
|                    | ↳ disk                                                               |                              |  |        |        |
| +--                |                                                                      |                              |  |        |        |
|                    | ↳ -----+-----+-----+-----+                                           |                              |  |        |        |
|                    |                                                                      |                              |  |        |        |
| TableReader_31     |                                                                      | 4.00   2   root              |  |        |        |
|                    | ↳   time:44.5ms, loops:2, cop_task: {num: 1, max: 0s,                |                              |  |        |        |
|                    | ↳ proc_keys: 0, copr_cache_hit_ratio: 0.00}   data:ExchangeSender_30 |                              |  |        |        |
|                    | ↳   N/A   N/A                                                        |                              |  |        |        |
| -ExchangeSender_30 |                                                                      | 4.00   2   batchCop[tiflash] |  |        |        |
|                    | ↳   tiflash_task:{time:16.5ms, loops:1, threads:1}                   |                              |  |        |        |
|                    | ↳   ExchangeType: PassThrough, tasks:                                |                              |  |        |        |
|                    | ↳ [2, 3, 4]   N/A   N/A                                              |                              |  |        |        |

```

- Projection_26	4.00	2	batchCop[tiflash]
	tiflash_task:{time:16.5ms, loops:1, threads:1}		
		Column#4	
		N/A	N/A
- HashAgg_27	4.00	2	batchCop[tiflash]
	tiflash_task:{time:16.5ms, loops:1, threads:1}		
		group by:test.t1.id, funcs:sum(
	Column#7)->Column#4	N/A	N/A
- ExchangeReceiver_29	4.00	2	batchCop[tiflash]
	tiflash_task:{time:14.5ms, loops:1, threads:20}		
		N/A	N/A
- ExchangeSender_28	4.00	0	batchCop[tiflash]
	tiflash_task:{time:9.49ms, loops:0, threads:0}		
		ExchangeType: HashPartition, Hash	
	Cols: test.t1.id, tasks: [1]	N/A	N/A
- HashAgg_9	4.00	0	batchCop[tiflash]
	tiflash_task:{time:9.49ms, loops:0, threads:0}		
		group by:test.t1.id, funcs:count	
	(1)->Column#7	N/A	N/A
- TableFullScan_25	6.00	0	batchCop[tiflash]
	table:t1	tiflash_task:{time:9.49ms, loops:0, threads:0}	
		keep order:false	
		N/A	N/A
+--+			
-----+-----+-----+-----+			
-----+-----+-----+-----+			

```

Compared to the output of EXPLAIN, the `operator info` column of the operator `ExchangeSender` also shows `tasks`, which records the id of the MPP task that the query fragment instantiates into. In addition, each MPP operator has a `threads` field in the `execution info` column, which records the concurrency of operations when TiDB executes this operator. If the cluster consists of multiple nodes, this concurrency is the result of adding up the concurrency of all nodes.

### 8.3.2.6 Explain Statements That Use Subqueries

TiDB performs [several optimizations](#) to improve the performance of subqueries. This document describes some of these optimizations for common subqueries and how to interpret the output of EXPLAIN.

The examples in this document are based on the following sample data:

```
CREATE TABLE t1 (id BIGINT NOT NULL PRIMARY KEY auto_increment, pad1 BLOB,
 ↵ pad2 BLOB, pad3 BLOB, int_col INT NOT NULL DEFAULT 0);
```

```

CREATE TABLE t2 (id BIGINT NOT NULL PRIMARY KEY auto_increment, t1_id
 ↪ BIGINT NOT NULL, pad1 BLOB, pad2 BLOB, pad3 BLOB, INDEX(t1_id));
CREATE TABLE t3 (
 id INT NOT NULL PRIMARY KEY auto_increment,
 t1_id INT NOT NULL,
 UNIQUE (t1_id)
);

INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM dual;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024), 0 FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;

```

```

 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t2 SELECT NULL, a.id, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
UPDATE t1 SET int_col = 1 WHERE pad1 = (SELECT pad1 FROM t1 ORDER BY RAND()
 ↪ LIMIT 1);
INSERT INTO t3 SELECT NULL, id FROM t1 WHERE id < 1000;

SELECT SLEEP(1);
ANALYZE TABLE t1, t2, t3;

```

### 8.3.2.6.1 Inner join (non-unique subquery)

In the following example, the IN subquery searches for a list of IDs from the table t2 → . For semantic correctness, TiDB needs to guarantee that the column t1\_id is unique. Using EXPLAIN, you can see the execution plan used to remove duplicates and perform an INNER JOIN operation:

```
EXPLAIN SELECT * FROM t1 WHERE id IN (SELECT t1_id FROM t2);
```

| +---                 | id                                                                    | estRows   | task | access object |
|----------------------|-----------------------------------------------------------------------|-----------|------|---------------|
|                      | operator info                                                         |           |      |               |
| +---                 |                                                                       |           |      |               |
|                      |                                                                       |           |      |               |
| IndexJoin_14         | 5.00                                                                  | root      |      |               |
|                      | inner join, inner:IndexLookUp_13, outer                               |           |      |               |
|                      | key:test.t2.t1_id, inner key:test.t1.id, equal cond:eq(test.t2.t1_id, |           |      |               |
|                      | test.t1.id)                                                           |           |      |               |
| -StreamAgg_49(Build) | 5.00                                                                  | root      |      |               |
|                      | group by:test.t2.t1_id, funcs:firstrow(                               |           |      |               |
|                      | test.t2.t1_id)->test.t2.t1_id                                         |           |      |               |
| -IndexReader_50      | 5.00                                                                  | root      |      |               |
|                      | index:StreamAgg_39                                                    |           |      |               |
|                      |                                                                       |           |      |               |
| - StreamAgg_39       | 5.00                                                                  | cop[tikv] |      |               |
|                      | group by:test.t2.t1_id,                                               |           |      |               |
|                      |                                                                       |           |      |               |

The result above shows that TiDB performs an index join operation that starts by reading the index on `t2.t1_id`. The values of `t1_id` are deduplicated inside TiKV first as a part of the `- HashAgg_31` operator task, and then deduplicated again in TiDB as a part of the `- HashAgg_38(Build)` operator task. The deduplication is performed by the aggregation function `firstrow(test.t2.t1_id)`. The result is then joined against the `t1` table's PRIMARY KEY.

### 8.3.2.6.2 Inner join (unique subquery)

In the previous example, aggregation is required to ensure that the values of `t1_id` are unique before joining against the table `t1`. But in the following example, `t3.t1_id` is already guaranteed unique because of a `UNIQUE` constraint:

```
EXPLAIN SELECT * FROM t1 WHERE id IN (SELECT t1.id FROM t3);
```

```
| IndexJoin_17 | 1978.13 | root |
| | inner join, inner:IndexLookUp_16, outer key
| | :test.t3.t1_id, inner key:test.t1.id, equal cond:eq(test.t3.t1_id,
| | test.t1.id) |
| -TableReader_44(Build) | 1978.00 | root |
| | data:TableFullScan_43
| |
| |
| -TableFullScan_43 | 1978.00 | cop[tikv] | table:t3
| | keep order:false
| |
| |
| -IndexLookUp_16(Probe) | 1.00 | root |
| |
| |
| -IndexRangeScan_14(Build) | 1.00 | cop[tikv] | table:t1, index:
| | PRIMARY(id) | range: decided by [eq(test.t1.id, test.t3.t1_id)], keep
| | order:false
| |
| -TableRowIDScan_15(Probe) | 1.00 | cop[tikv] | table:t1
| | keep order:false
| |
| |
+--+
|-----+-----+-----+-----+
```

Semantically because `t3.t1_id` is guaranteed unique, it can be executed directly as an INNER JOIN.

### 8.3.2.6.3 Semi join (correlated subquery)

In the previous two examples, TiDB is able to perform an `INNER JOIN` operation after the data inside the subquery is made unique (via `HashAgg`) or guaranteed unique. Both joins are performed using an Index Join.

In this example, TiDB chooses a different execution plan:

```
EXPLAIN SELECT * FROM t1 WHERE id IN (SELECT t1_id FROM t2 WHERE t1_id !=
 ↴ t1.int col);
```

A horizontal dashed brown line extending across the width of the frame. At each end of the line, there is a small black arrow pointing towards the center. The line is thin and has a slightly irregular texture.

| id                        | estRows | task          | access object                                                                         |
|---------------------------|---------|---------------|---------------------------------------------------------------------------------------|
| ↳                         |         | operator info |                                                                                       |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| +--                       |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| IndexJoin_14              | 1582.40 | root          |                                                                                       |
| ↳                         |         |               | anti semi <b>join</b> , <b>inner</b> :IndexLookUp_13, <b>outer</b>                    |
| ↳                         |         |               | <b>key</b> :test.t3.t1_id, <b>inner key</b> :test.t1.id, equal cond: eq(test.t3.t1_id |
| ↳                         |         |               | , test.t1.id)                                                                         |
| -TableReader_35(Build)    | 1978.00 | root          |                                                                                       |
| ↳                         |         |               | data:TableFullScan_34                                                                 |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| -TableFullScan_34         | 1978.00 | cop[tikv]     | <b>table</b> :t3                                                                      |
| ↳                         |         |               | keep <b>order</b> :false                                                              |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| -IndexLookUp_13(Probe)    | 1.00    | root          |                                                                                       |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| -IndexRangeScan_10(Build) | 1.00    | cop[tikv]     | <b>table</b> :t1, <b>index</b> :                                                      |
| ↳                         |         |               | <b>PRIMARY</b> (id)   range: decided by [eq(test.t1.id, test.t3.t1_id)], keep         |
| ↳                         |         |               | <b>order</b> :false                                                                   |
| -Selection_12(Probe)      | 1.00    | cop[tikv]     |                                                                                       |
| ↳                         |         |               | lt(test.t1.int_col, 100)                                                              |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| -TableRowIDScan_11        | 1.00    | cop[tikv]     | <b>table</b> :t1                                                                      |
| ↳                         |         |               | keep <b>order</b> :false                                                              |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| +--                       |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| ↳                         |         |               |                                                                                       |
| 7 rows in set (0.00 sec)  |         |               |                                                                                       |

From the result above, you can see that TiDB uses a `Semi Join` algorithm. `Semi-join` differs from `inner join`: `semi-join` only permits the first value on the right key (`t2.t1_id`), which means that the duplicates are eliminated as a part of the join operator task. The join algorithm is also `Merge Join`, which is like an efficient zipper-merge as the operator reads data from both the left and the right side in sorted order.

The original statement is considered a *correlated subquery*, because the subquery refers to a column (`t1.int_col`) that exists outside of the subquery. However, the output of EXPLAIN shows the execution plan after the `subquery decorrelation optimization` has been applied. The condition `t1_id != t1.int_col` is rewritten to `t1.id != t1.int_col`. TiDB can perform this in `- Selection_21` as it is reading data from the table `t1`, so this decorrelation and rewriting make the execution a lot more efficient.

#### 8.3.2.6.4 Anti semi join (NOT IN subquery)

In the following example, the query semantically returns all rows from the table `t3` *unless* `t3.t1_id` is in the subquery:

```
EXPLAIN SELECT * FROM t3 WHERE t1_id NOT IN (SELECT id FROM t1 WHERE
 ↪ int_col < 100);
```

| +--                                                                     |                   |           |          |               |                |  |
|-------------------------------------------------------------------------|-------------------|-----------|----------|---------------|----------------|--|
|                                                                         | id                | estRows   | task     | access object | operator       |  |
| +                                                                       | ↳ info            |           |          |               |                |  |
| +                                                                       | IndexMergeJoin_20 | 1598.40   | root     |               | anti semi join |  |
| ↪ , inner:TableReader_15, outer key:test.t3.t1_id, inner key:test.t1.id |                   |           |          |               |                |  |
| ↪                                                                       |                   |           |          |               |                |  |
| -TableReader_28(Build)                                                  | 1998.00           | root      |          |               | data:          |  |
| ↪ TableFullScan_27                                                      |                   |           |          |               |                |  |
| -TableFullScan_27                                                       | 1998.00           | cop[tikv] | table:t3 | keep order:   |                |  |
| ↪ false                                                                 |                   |           |          |               |                |  |
| -TableReader_15(Probe)                                                  | 1.00              | root      |          |               | data:          |  |
| ↪ Selection_14                                                          |                   |           |          |               |                |  |
| -Selection_14                                                           | 1.00              | cop[tikv] |          | lt(test.t1.   |                |  |
| ↪ int_col, 100)                                                         |                   |           |          |               |                |  |
| -TableRangeScan_13                                                      | 1.00              | cop[tikv] | table:t1 | range:        |                |  |
| ↪ decided by [test.t3.t1_id], keep order:true                           |                   |           |          |               |                |  |
| +                                                                       |                   |           |          |               |                |  |
| ↪                                                                       |                   |           |          |               |                |  |
| 6 rows in set (0.00 sec)                                                |                   |           |          |               |                |  |

This query starts by reading the table `t3` and then probes the table `t1` based on the PRIMARY KEY. The join type is an *anti semi join*; anti because this example is for the non-existence of the value (NOT IN) and semi-join because only the first row needs to match before the join is rejected.

### 8.3.2.7 Explain Statements Using Aggregation

When aggregating data, the SQL Optimizer will select either a Hash Aggregation or Stream Aggregation operator. To improve query efficiency, aggregation is performed at both the coprocessor and TiDB layers. Consider the following example:

```

CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY auto_increment, pad1 BLOB,
 ↪ pad2 BLOB, pad3 BLOB);
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM dual;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
SELECT SLEEP(1);
ANALYZE TABLE t1;

```

From the output of `SHOW TABLE REGIONS`, you can see that this table is split into multiple Regions:

```
SHOW TABLE t1 REGIONS;
```

| REGION_ID | START_KEY    | END_KEY      | LEADER_ID | LEADER_STORE_ID | PEERS | SCATTERING | WRITTEN_BYTES | READ_BYTES | APPROXIMATE_SIZE(MB) | APPROXIMATE_KEYS |
|-----------|--------------|--------------|-----------|-----------------|-------|------------|---------------|------------|----------------------|------------------|
| 64        | t_64_0       | t_64_r_31766 | 1325      | 102033520       | 1     | 52797      | 65            | 98         | 65                   |                  |
| 66        | t_64_r_31766 | t_64_r_63531 | 1325      | 72522521        | 1     | 78495      | 67            | 104        | 67                   |                  |
| 68        | t_64_r_63531 | t_64_r_95296 | 1325      | 0               | 1     | 95433      | 69            | 104        | 69                   |                  |
| 2         | t_64_r_95296 |              | 1501      | 0               | 1     | 63211      | 3             | 81         | 3                    |                  |

4 rows in set (0.00 sec)

Using EXPLAIN with the following aggregation statement, you can see that StreamAgg\_8 is first performed on each Region inside TiKV. Each TiKV Region will then send one row back to TiDB, which aggregates the data from each Region in StreamAgg\_16:

```
EXPLAIN SELECT COUNT(*) FROM t1;
```

| id           | info               | estRows | task | access object | operator      |
|--------------|--------------------|---------|------|---------------|---------------|
| StreamAgg_16 | Column#7->Column#5 | 1.00    | root |               | funcs:count() |

```

| -TableReader_17 | 1.00 | root | | data:
| ↳ StreamAgg_8 | |
| -StreamAgg_8 | 1.00 | cop[tikv] | | funcs:count
| ↳ (1)->Column#7 |
| -TableFullScan_15 | 242020.00 | cop[tikv] | table:t1 | keep order:
| ↳ false |
+--+
| ↳ -----+-----+-----+-----+
| ↳
| ↳
4 rows in set (0.00 sec)

```

This is easiest to observe in EXPLAIN ANALYZE, where the actRows matches the number of Regions from SHOW TABLE REGIONS because a TableFullScan is being used and there are no secondary indexes:

```
EXPLAIN ANALYZE SELECT COUNT(*) FROM t1;
```

```

+--+
| id | estRows | actRows | task | access object |
| execution info
| ↳
| ↳ | operator info | memory | disk |
+--+
| ↳ -----+-----+-----+-----+
| ↳
| ↳
| StreamAgg_16 | 1.00 | 1 | root | | time
| ↳ :12.609575ms, loops:2
| ↳
| ↳ | funcs:count(Column#7)->Column#5 | 372 Bytes | N/A |
| -TableReader_17 | 1.00 | 4 | root | | time
| ↳ :12.605155ms, loops:2, cop_task: {num: 4, max: 12.538245ms, min:
| ↳ 9.256838ms, avg: 10.895114ms, p95: 12.538245ms, max_proc_keys: 31765,
| ↳ p95_proc_keys: 31765, tot_proc: 48ms, rpc_num: 4, rpc_time:
| ↳ 43.530707ms, copr_cache_hit_ratio: 0.00} | data:StreamAgg_8 | 293
| ↳ Bytes | N/A |
| -StreamAgg_8 | 1.00 | 4 | cop[tikv] | | proc
| ↳ max:12ms, min:12ms, p80:12ms, p95:12ms, iters:122, tasks:4
| ↳
| ↳ | funcs:count(1)->Column#7 | N/A | N/A |
| -TableFullScan_15 | 242020.00 | 121010 | cop[tikv] | table:t1 |
| ↳ proc max:12ms, min:12ms, p80:12ms, p95:12ms, iters:122, tasks:4
| ↳
| ↳ | keep order:false | N/A | N/A |

```

```
+--+
→ -----+-----+-----+-----+
→
4 rows in set (0.01 sec)
```

### 8.3.2.7.1 Hash Aggregation

The Hash Aggregation algorithm uses a hash table to store intermediate results while performing aggregation. It executes in parallel using multiple threads but consumes more memory than Stream Aggregation.

The following is an example of the HashAgg operator:

```
EXPLAIN SELECT /*+ HASH_AGG() */ count(*) FROM t1;
```

```
+--+
→ -----+-----+-----+-----+
→
| id | estRows | task | access object | operator
| info | | | | operator
+--+
→ -----+-----+-----+-----+
→
| HashAgg_9 | 1.00 | root | | funcs:count(
| >Column#6->Column#5 |
| -TableReader_10 | 1.00 | root | | data:
| > HashAgg_5 |
| -HashAgg_5 | 1.00 | cop[tikv] | | funcs:count
| >(1)->Column#6 |
| -TableFullScan_8 | 242020.00 | cop[tikv] | table:t1 | keep order:
| > false |
+--+
→ -----+-----+-----+-----+
→
4 rows in set (0.00 sec)
```

The operator `info` shows that the hashing function used to aggregate the data is `funcs :count(1)->Column#6`.

### 8.3.2.7.2 Stream Aggregation

The Stream Aggregation algorithm usually consumes less memory than Hash Aggregation. However, this operator requires that data is sent ordered so that it can *stream* and apply the aggregation on values as they arrive.

Consider the following example:

```
CREATE TABLE t2 (id INT NOT NULL PRIMARY KEY, col1 INT NOT NULL);
INSERT INTO t2 VALUES (1, 9),(2, 3),(3,1),(4,8),(6,3);
EXPLAIN SELECT /*+ STREAM_AGG() */ col1, count(*) FROM t2 GROUP BY col1;
```

Query OK, 0 rows affected (0.11 sec)

Query OK, 5 rows affected (0.01 sec)

Records: 5 Duplicates: 0 Warnings: 0

| +-- | id                                                                   | estRows  | task      | access object | operator               |
|-----|----------------------------------------------------------------------|----------|-----------|---------------|------------------------|
|     | id                                                                   |          |           |               |                        |
|     | info                                                                 |          |           |               |                        |
|     |                                                                      |          |           |               |                        |
| +-- | Projection_4                                                         | 8000.00  | root      |               | test.t2.col1           |
|     | , Column#3                                                           |          |           |               |                        |
|     | -StreamAgg_8                                                         | 8000.00  | root      |               | group by:              |
|     | test.t2.col1, funcs:count(1)->Column#3, funcs:firstrow(test.t2.col1) |          |           |               |                        |
|     | ->test.t2.col1                                                       |          |           |               |                        |
|     | -Sort_13                                                             | 10000.00 | root      |               | test.t2.               |
|     | col1                                                                 |          |           |               |                        |
|     |                                                                      |          |           |               |                        |
|     | -TableReader_12                                                      | 10000.00 | root      |               | data:                  |
|     | TableFullScan_11                                                     |          |           |               |                        |
|     |                                                                      |          |           |               |                        |
|     | -TableFullScan_11                                                    | 10000.00 | cop[tikv] |               | table:t2   keep order: |
|     | false, stats:pseudo                                                  |          |           |               |                        |
|     |                                                                      |          |           |               |                        |
| +-- |                                                                      |          |           |               |                        |
|     |                                                                      |          |           |               |                        |
|     |                                                                      |          |           |               |                        |

5 rows in set (0.00 sec)

In this example, the `- Sort_13` operator can be eliminated by adding an index on `col1`. Once the index is added, the data can be read in order and the `- Sort_13` operator is eliminated:

```
ALTER TABLE t2 ADD INDEX (col1);
EXPLAIN SELECT /*+ STREAM_AGG() */ col1, count(*) FROM t2 GROUP BY col1;
```

```
Query OK, 0 rows affected (0.28 sec)
```

```
+--+
→ -----+-----+-----+
→
| id | estRows | task | access object |
→ operator info
→
→ |
+--+
→ -----+-----+-----+
→
| Projection_4 | 4.00 | root | |
→ test.t2.col1, Column#3
→
→
| -StreamAgg_14 | 4.00 | root | |
→ group by:test.t2.col1, funcs:count(Column#4)->Column#3, funcs:
→ firstrow(test.t2.col1)->test.t2.col1 |
| -IndexReader_15 | 4.00 | root | |
→ index:StreamAgg_8
→
→
| - StreamAgg_8 | 4.00 | cop[tikv] | |
→ group by:test.t2.col1, funcs:count(1)->Column#4
→
→
| - IndexFullScan_13 | 5.00 | cop[tikv] | table:t2, index:col1(col1
→) | keep order:true, stats:pseudo
→
→
+--+
→ -----+-----+-----+
→
5 rows in set (0.00 sec)
```

### 8.3.2.8 EXPLAIN Statements Using Views

EXPLAIN displays the tables and indexes that a `view` references, not the name of the view itself. This is because views are only virtual tables and do not store any data themselves. The definition of the view and the rest of the statement are merged together during SQL optimization.

From the `bikeshare` example database, you can see that the following two queries are executed in a similar manner:

```
ALTER TABLE trips ADD INDEX (duration);
CREATE OR REPLACE VIEW long_trips AS SELECT * FROM trips WHERE duration >
→ 3600;
EXPLAIN SELECT * FROM long_trips;
```

```
EXPLAIN SELECT * FROM trips WHERE duration > 3600;
```

```
Query OK, 0 rows affected (2 min 10.11 sec)
```

```
Query OK, 0 rows affected (0.13 sec)
```

| +-- | id                         | estRows    | task      | access object                                                               |
|-----|----------------------------|------------|-----------|-----------------------------------------------------------------------------|
|     | operator info              |            |           |                                                                             |
| +-- | IndexLookUp_12             | 6372547.67 | root      |                                                                             |
|     | - IndexRangeScan_10(Build) | 6372547.67 | cop[tikv] | table:trips, index:duration(duration)   range:(3600,+inf], keep order:false |
|     | - TableRowIDScan_11(Probe) | 6372547.67 | cop[tikv] | table:trips   keep order:false                                              |
| +-- | IndexLookUp_10             | 833219.37  | root      |                                                                             |
|     | - IndexRangeScan_8(Build)  | 833219.37  | cop[tikv] | table:trips, index:duration(duration)   range:(3600,+inf], keep order:false |
|     | - TableRowIDScan_9(Probe)  | 833219.37  | cop[tikv] | table:trips   keep order:false                                              |
| +-- | 3 rows in set (0.00 sec)   |            |           |                                                                             |
| +-- | id                         | estRows    | task      | access object                                                               |
|     | operator info              |            |           |                                                                             |
| +-- | IndexLookUp_10             | 833219.37  | root      |                                                                             |
|     | - IndexRangeScan_8(Build)  | 833219.37  | cop[tikv] | table:trips, index:duration(duration)   range:(3600,+inf], keep order:false |
|     | - TableRowIDScan_9(Probe)  | 833219.37  | cop[tikv] | table:trips   keep order:false                                              |
| +-- | 3 rows in set (0.00 sec)   |            |           |                                                                             |

Similarly, predicates from the view are pushed down to the base table:

```
EXPLAIN SELECT * FROM long_trips WHERE bike_number = 'W00950';
EXPLAIN SELECT * FROM trips WHERE bike_number = 'W00950';
```

```
+--+
→-----+-----+-----+
→
| id | estRows | task | access object
| | | operator info
+--+
→-----+-----+-----+
→
| IndexLookUp_14 | 3.33 | root |
→
→-----+-----+-----+
| -IndexRangeScan_11(Build) | 3333.33 | cop[tikv] | table:trips, index:
→ duration(duration) | range:(3600,+inf], keep order:false, stats:
→ pseudo |
| -Selection_13(Probe) | 3.33 | cop[tikv] |
→
→-----+-----+-----+
| -TableRowIDScan_12 | 3333.33 | cop[tikv] | table:trips
→
→-----+-----+-----+
| keep order:false, stats:pseudo
+--+
→-----+-----+-----+
→
4 rows in set (0.00 sec)
```

```
+--+
→-----+-----+-----+
→
| id | estRows | task | access object | operator
| | | | |
+--+
→-----+-----+-----+
→
| TableRowIDScan_12 | 3333.33 | cop[tikv] | table:trips | data:
| -TableReader_7 | 43.00 | root | | data:
→ Selection_6 |
| -Selection_6 | 43.00 | cop[tikv] | eq(bikeshare.
→ trips.bike_number, "W00950") |
| -TableFullScan_5 | 19117643.00 | cop[tikv] | table:trips | keep order
→ :false
+--+
→-----+-----+-----+
→
```

```
3 rows in set (0.00 sec)
```

In the first statement above, you can see that the index is used to satisfy the view definition, and then the `bike_number = 'W00950'` is applied when TiDB reads the table row. In the second statement, there are no indexes to satisfy the statement, and a `TableFullScan` is used.

TiDB makes use of indexes that satisfy both the view definition and the statement itself. Consider the following composite index:

```
ALTER TABLE trips ADD INDEX (bike_number, duration);
EXPLAIN SELECT * FROM long_trips WHERE bike_number = 'W00950';
EXPLAIN SELECT * FROM trips WHERE bike_number = 'W00950';
```

```
Query OK, 0 rows affected (2 min 31.20 sec)
```

```
+--+
→ -----+-----+-----+
→
→
| id | estRows | task | access object
 | | operator info
→
+--+
→ -----+-----+-----+
→
→
| IndexLookUp_13 | 63725.48 | root |
→
→
| -IndexRangeScan_11(Build) | 63725.48 | cop[tikv] | table:trips, index:
→ bike_number(bike_number, duration) | range:("W00950" 3600, "W00950" +
→ inf], keep order:false |
| -TableRowIDScan_12(Probe) | 63725.48 | cop[tikv] | table:trips
→
→
→
+--+
→ -----+-----+-----+
```

```
3 rows in set (0.00 sec)
```

```
+--+
→ -----+-----+-----+
→
→
| id | estRows | task | access object
 | | operator info
→
+--+
→ -----+-----+-----+
```

```
+--+
| IndexLookUp_10 | 19117.64 | root |
| |
| |
| -IndexRangeScan_8(Build) | 19117.64 | cop[tikv] | table:trips, index:
| ↵ bike_number(bike_number, duration) | range:["W00950", "W00950"], keep
| ↵ order:false |
| -TableRowIDScan_9(Probe) | 19117.64 | cop[tikv] | table:trips
| | keep order:false
| |
+--+
| |
| |
3 rows in set (0.00 sec)
```

In the first statement, TiDB is able to use both parts of the composite index (`↪ bike_number, duration`). In the second statement, only the first part which is `bike_number` of the index (`bike_number, duration`) is used.

### 8.3.2.9 Explain Statements Using Partitions

The EXPLAIN statement displays the partitions that TiDB needs to access in order to execute a query. Because of [partition pruning](#), the displayed partitions are often only a subset of the overall partitions. This document describes some of the optimizations for common partitioned tables, and how to interpret the output of EXPLAIN.

The sample data used in this document:

```
CREATE TABLE t1 (
 id BIGINT NOT NULL auto_increment,
 d date NOT NULL,
 pad1 BLOB,
 pad2 BLOB,
 pad3 BLOB,
 PRIMARY KEY (id,d)
) PARTITION BY RANGE (YEAR(d)) (
 PARTITION p2016 VALUES LESS THAN (2017),
 PARTITION p2017 VALUES LESS THAN (2018),
 PARTITION p2018 VALUES LESS THAN (2019),
 PARTITION p2019 VALUES LESS THAN (2020),
 PARTITION pmax VALUES LESS THAN MAXVALUE
);

INSERT INTO t1 (d, pad1, pad2, pad3) VALUES
```

```

('2016-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2016-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2016-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2017-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2017-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2017-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2018-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2018-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2018-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2019-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2019-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2019-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2020-01-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2020-06-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024)),
('2020-09-01', RANDOM_BYTES(1024), RANDOM_BYTES(1024), RANDOM_BYTES(1024));

INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;
INSERT INTO t1 SELECT NULL, a.d, RANDOM_BYTES(1024), RANDOM_BYTES(1024),
 ↪ RANDOM_BYTES(1024) FROM t1 a JOIN t1 b JOIN t1 c LIMIT 10000;

SELECT SLEEP(1);
ANALYZE TABLE t1;

```

The following example shows a statement against the newly created partitioned table:

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE d = '2017-06-01';
```

| +---                              | →       | →         | →             | → | → | → | → |
|-----------------------------------|---------|-----------|---------------|---|---|---|---|
| id                                | estRows | task      | access object |   |   |   |   |
| ↪ operator info                   |         |           |               |   |   |   |   |
| +---                              | →       | →         | →             | → | → | → | → |
| StreamAgg_21                      | 1.00    | root      |               |   |   |   |   |
| ↪ funcs:count(Column#8)->Column#6 |         |           |               |   |   |   |   |
| -TableReader_22                   | 1.00    | root      |               |   |   |   |   |
| ↪ data:StreamAgg_10               |         |           |               |   |   |   |   |
| - StreamAgg_10                    | 1.00    | cop[tikv] |               |   |   |   |   |

```

 ↗ funcs:count(1)->Column#8 |
| - Selection_20 | 8.87 | cop[tikv] | | eq
| ↗ (test.t1.d, 2017-06-01 00:00:00.000000) |
| - TableFullScan_19 | 8870.00 | cop[tikv] | table:t1, partition:
| ↗ p2017 | keep order:false |
+--+
 ↗ -----
 ↗
5 rows in set (0.01 sec)

```

Starting from the inner-most ( `-TableFullScan_19`) operator and working back towards the root operator (`StreamAgg_21`):

- TiDB successfully identified that only one partition (p2017) needed to be accessed. This is noted under `access object`.
- The partition itself was scanned in the operator `-TableFullScan_19` and then `-Selection_20` was applied to filter for rows that have a start date of `2017-06-01 00:00:00.000000`.
- The rows that match `-Selection_20` are then stream aggregated in the coprocessor, which natively understands the `count` function.
- Each coprocessor request then sends back one row to `-TableReader_22` inside TiDB, which is then stream aggregated under `StreamAgg_21` and one row is returned to the client.

In the following example, partition pruning does not eliminate any partitions:

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE YEAR(d) = 2017;
```

| +--+               | → -----+-----+-----+            | → | → | → |
|--------------------|---------------------------------|---|---|---|
| id                 | estRows   task   access object  |   |   |   |
| operator info      |                                 |   |   |   |
| +--+               | → -----+-----+-----+            | → | → | → |
| HashAgg_20         | 1.00   root                     |   |   |   |
|                    | funcs:count(Column#7)->Column#6 |   |   |   |
| -PartitionUnion_21 | 5.00   root                     |   |   |   |
|                    |                                 |   |   |   |
| -StreamAgg_36      | 1.00   root                     |   |   |   |
|                    | funcs:count(Column#9)->Column#7 |   |   |   |
| -TableReader_37    | 1.00   root                     |   |   |   |
|                    | data:StreamAgg_25               |   |   |   |

```

- StreamAgg_25	1.00	cop[tikv]
↳	funcs:count(1)->Column#9	
- Selection_35	6000.00	cop[tikv]
↳	eq(year(test.t1.d), 2017)	
- TableFullScan_34	7500.00	cop[tikv]
↳ partition:p2016	keep order:false	
- StreamAgg_55	1.00	root
↳	funcs:count(Column#11)->Column#7	
- TableReader_56	1.00	root
↳	data:StreamAgg_44	
- StreamAgg_44	1.00	cop[tikv]
↳	funcs:count(1)->Column#11	
- Selection_54	14192.00	cop[tikv]
↳	eq(year(test.t1.d), 2017)	
- TableFullScan_53	17740.00	cop[tikv]
↳ partition:p2017	keep order:false	
- StreamAgg_74	1.00	root
↳	funcs:count(Column#13)->Column#7	
- TableReader_75	1.00	root
↳	data:StreamAgg_63	
- StreamAgg_63	1.00	cop[tikv]
↳	funcs:count(1)->Column#13	
- Selection_73	3977.60	cop[tikv]
↳	eq(year(test.t1.d), 2017)	
- TableFullScan_72	4972.00	cop[tikv]
↳ partition:p2018	keep order:false	
- StreamAgg_93	1.00	root
↳	funcs:count(Column#15)->Column#7	
- TableReader_94	1.00	root
↳	data:StreamAgg_82	
- StreamAgg_82	1.00	cop[tikv]
↳	funcs:count(1)->Column#15	
- Selection_92	20361.60	cop[tikv]
↳	eq(year(test.t1.d), 2017)	
- TableFullScan_91	25452.00	cop[tikv]
↳ partition:p2019	keep order:false	
- StreamAgg_112	1.00	root
↳	funcs:count(Column#17)->Column#7	
- TableReader_113	1.00	root
↳	data:StreamAgg_101	
- StreamAgg_101	1.00	cop[tikv]
↳	funcs:count(1)->Column#17	
- Selection_111	8892.80	cop[tikv]
↳	eq(year(test.t1.d), 2017)	
- TableFullScan_110	11116.00	cop[tikv]

```

```

 ↪ partition:pmax | keep order:false
+---+
 ↪
 ↪
27 rows in set (0.00 sec)

```

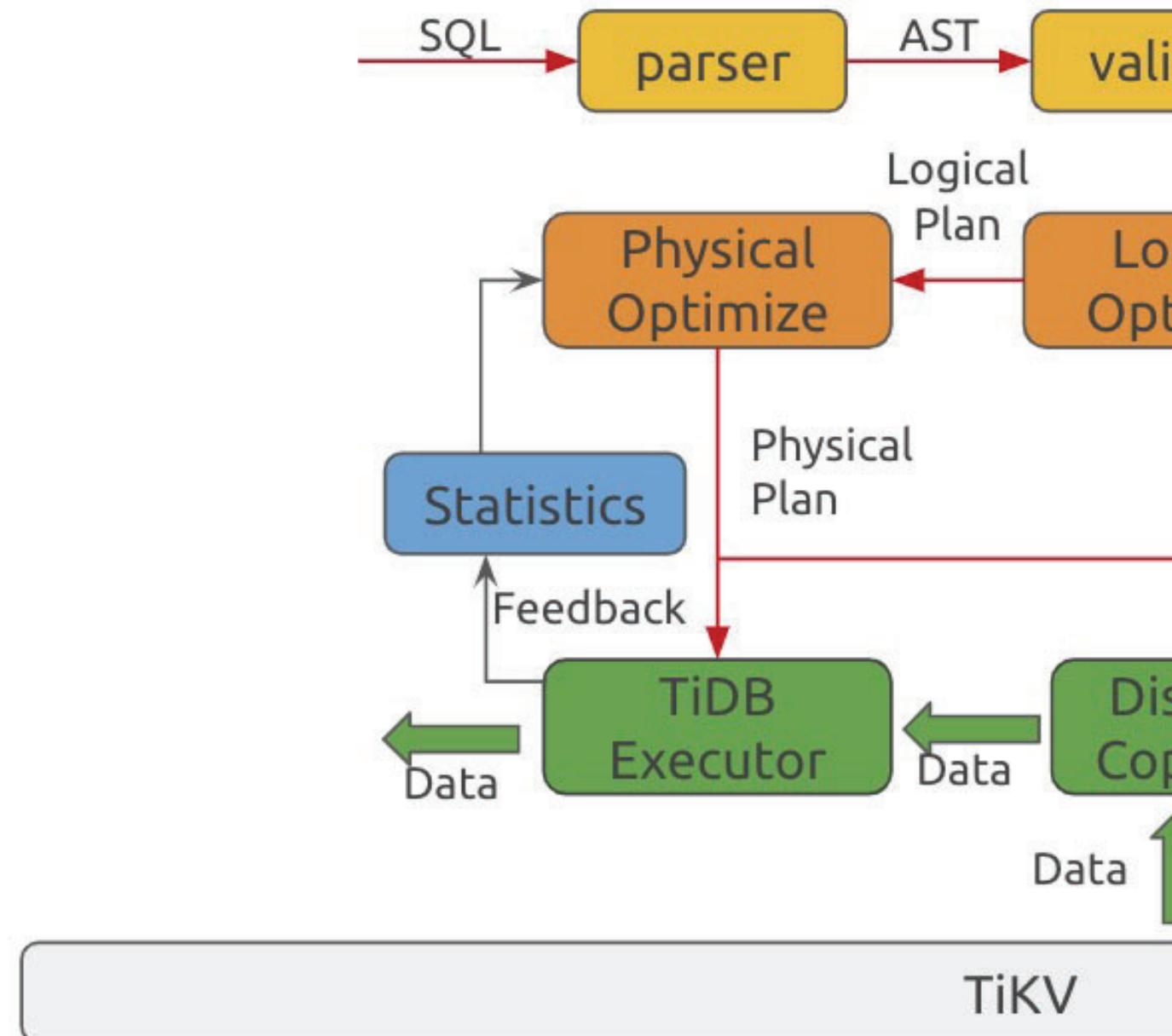


Figure 63: SQL Optimization Process

After parsing the original query text by `parser` and some simple validity checks, TiDB first makes some logically equivalent changes to the query. For detailed changes, see [SQL Logical Optimization](#).

Through these equivalent changes, this query becomes easier to handle in the logical execution plan. After the equivalent change is done, TiDB obtains a query plan structure equivalent to the original query, and then obtains a final execution plan based on the data distribution and the specific execution cost of an operator. For details, see [SQL Physical Optimization](#).

At the same time, when TiDB executes the `PREPARE` statement, you can choose to enable caching to reduce the cost of generating the execution plan in TiDB. For details, see [Execution Plan Cache](#).

### 8.3.3.2 Logic Optimization

#### 8.3.3.2.1 SQL Logical Optimization

This chapter explains some key logic rewrites to help you understand how TiDB generates the final query plan. For example, when you execute the `select * from t where t.a in (select t1.a from t1 where t1.b=t.b)` query in TiDB, you will find that the `IN` sub-query `t.a in (select t1.a from t1 where t1.b=t.b)` does not exist because TiDB has made some rewrites here.

This chapter introduces the following key rewrites:

- Subquery Related Optimizations
- Column Pruning
- Decorrelation of Correlated Subquery
- Eliminate Max/Min
- Predicates Push Down
- Partition Pruning
- TopN and Limit Operator Push Down
- Join Reorder

#### 8.3.3.2.2 Subquery Related Optimizations

This article mainly introduces subquery related optimizations.

Subqueries usually appear in the following situations:

- NOT IN (SELECT ... FROM ...)
- NOT EXISTS (SELECT ... FROM ...)
- IN (SELECT ... FROM ...)
- EXISTS (SELECT ... FROM ...)
- ... >/>=/</<=/!= (SELECT ... FROM ...)

Sometimes a subquery contains non-subquery columns, such as `select * from t where t.a in (select * from t2 where t.b=t2.b)`. The `t.b` column in the subquery does not belong to the subquery, it is introduced from the outside of the subquery. This kind of subquery is usually called a “correlated subquery”, and the externally introduced column is called a “correlated column”. For optimizations about correlated subquery, see [Decorrelation of correlated subquery](#). This article focuses on subqueries that do not involve correlated columns.

By default, subqueries use `semi join` mentioned in [Understanding TiDB Execution Plan](#) as the execution method. For some special subqueries, TiDB do some logical rewrite to get better performance.

```
... < ALL (SELECT ... FROM ...) or ... > ANY (SELECT ... FROM ...)
```

In this case, `ALL` and `ANY` can be replaced by `MAX` and `MIN`. When the table is empty, the result of `MAX(EXPR)` and `MIN(EXPR)` is `NULL`. It works the same when the result of `EXPR` contains `NULL`. Whether the result of `EXPR` contains `NULL` may affect the final result of the expression, so the complete rewrite is given in the following form:

- `t.id < all (select s.id from s)` is rewritten as `t.id < min(s.id) and if(sum(s.id is null)!= 0, null, true)`
- `t.id < any (select s.id from s)` is rewritten as `t.id < max(s.id) or if(sum(s.id is null)!= 0, null, false)`

```
... != ANY (SELECT ... FROM ...)
```

In this case, if all the values from the subquery are distinct, it is enough to compare the query with them. If the number of different values in the subquery is more than one, then there must be inequality. Therefore, such subqueries can be rewritten as follows:

- `select * from t where t.id != any (select s.id from s)` is rewritten as `select t.* from t, (select s.id, count(distinct s.id)as cnt_distinct from s)where (t.id != s.id or cnt_distinct > 1)`

```
... = ALL (SELECT ... FROM ...)
```

In this case, when the number of different values in the subquery is more than one, then the result of this expression must be false. Therefore, such subquery is rewritten into the following form in TiDB:

- `select * from t where t.id = all (select s.id from s)` is rewritten as `select t.* from t, (select s.id, count(distinct s.id)as cnt_distinct from s)where (t.id = s.id and cnt_distinct <= 1)`

```
... IN (SELECT ... FROM ...)
```

In this case, the subquery of `IN` is rewritten into `SELECT ... FROM ... GROUP ...`, and then rewritten into the normal form of `JOIN`.

For example, `select * from t1 where t1.a in (select t2.a from t2)` is rewritten as `select t1.* from t1, (select distinct(a)a from t2)t2 where t1.a = t2.`  
 $\hookrightarrow$  The form of `a`. The `DISTINCT` attribute here can be eliminated automatically if `t2.a` has the `UNIQUE` attribute.

```
explain select * from t1 where t1.a in (select t2.a from t2);
```

| +--                                                                  |         |           |                        |       |
|----------------------------------------------------------------------|---------|-----------|------------------------|-------|
| id                                                                   | estRows | task      | access object          |       |
| +--                                                                  |         |           |                        |       |
| id                                                                   | estRows | task      | access object          |       |
| +--                                                                  |         |           |                        |       |
| IndexJoin_12                                                         | 9990.00 | root      |                        | inner |
| join, inner:TableReader_11, outer key:test.t2.a, inner key:test.t1.a |         |           |                        |       |
|                                                                      |         |           |                        |       |
| -HashAgg_21(Build)                                                   | 7992.00 | root      |                        |       |
| group by:test.t2.a, funcs:firstrow(test.t2.a)->test.t2.a             |         |           |                        |       |
| -IndexReader_28                                                      | 9990.00 | root      |                        |       |
| index:IndexFullScan_27                                               |         |           |                        |       |
| -IndexFullScan_27                                                    | 9990.00 | cop[tikv] | table:t2, index:idx(a) |       |
| keep order:false, stats:pseudo                                       |         |           |                        |       |
| -TableReader_11(Probe)                                               | 1.00    | root      |                        | data  |
| :TableRangeScan_10                                                   |         |           |                        |       |
| -TableRangeScan_10                                                   | 1.00    | cop[tikv] | table:t1               |       |
| range: decided by [test.t2.a], keep order:false, stats:pseudo        |         |           |                        |       |
| +--                                                                  |         |           |                        |       |
| id                                                                   | estRows | task      | access object          |       |
| +--                                                                  |         |           |                        |       |

This rewrite gets better performance when the `IN` subquery is relatively small and the external query is relatively large, because without rewriting, using `index join` with `t2` as the driving table is impossible. However, the disadvantage is that when the aggregation cannot be automatically eliminated during the rewrite and the `t2` table is relatively large, this rewrite affects the performance of the query. Currently, the variable `tidb_opt_insubq_to_join_and_agg` is used to control this optimization. When this optimization is not suitable, you can manually disable it.

`EXISTS` subquery and ... `>/>=/</<=/=/!=` (`SELECT ... FROM ...`)

At present, for a subquery in such scenarios, if the subquery is not a correlated subquery, TiDB evaluates it in advance in the optimization stage, and directly replaces it with a

result set. As shown in the figure below, the `EXISTS` subquery is evaluated to TRUE in the optimization stage in advance, so it does not show in the final execution result.

```
create table t1(a int);
create table t2(a int);
insert into t2 values(1);
explain select * from t where exists (select * from t2);
```

| +--               | id | estRows  | task      | access object | operator info     |
|-------------------|----|----------|-----------|---------------|-------------------|
| +--               |    |          |           |               |                   |
| id                |    |          |           |               |                   |
|                   |    |          |           |               |                   |
| +--               |    |          |           |               |                   |
| TableReader_12    |    | 10000.00 | root      |               | data:             |
|                   |    |          |           |               |                   |
| -TableFullScan_11 |    | 10000.00 | cop[tikv] | table:t       | keep order:false, |
|                   |    |          |           |               |                   |
| +--               |    |          |           |               |                   |
|                   |    |          |           |               |                   |
| +--               |    |          |           |               |                   |
|                   |    |          |           |               |                   |

### 8.3.3.2.3 Column Pruning

The basic idea of column pruning is that for columns not used in the operator, the optimizer does not need to retain them during optimization. Removing these columns reduces the use of I/O resources and facilitates the subsequent optimization. The following is an example of column repetition:

Suppose there are four columns (a, b, c, and d) in table t. You can execute the following statement:

```
select a from t where b > 5
```

In this query, only column a and column b are used, and column c and column d are redundant. Regarding the query plan of this statement, the `Selection` operator uses column b. Then the `DataSource` operator uses columns a and column b. Columns c and column d can be pruned because the `DataSource` operator does not read them.

Therefore, when TiDB performs a top-down scanning during the logic optimization phase, redundant columns are pruned to reduce waste of resources. This scanning process is called “Column Pruning”, corresponding to the `columnPruner` rule. If you want to disable this rule, refer to [The Blocklist of Optimization Rules and Expression Pushdown](#).

#### 8.3.3.2.4 Decorrelation of Correlated Subquery

[Subquery related optimizations](#) describes how TiDB handles subqueries when there are no correlated columns. Because decorrelation of correlated subquery is complex, this article introduces some simple scenarios and the scope to which the optimization rule applies.

## Introduction

Take `select * from t1 where t1.a < (select sum(t2.a)from t2 where t2.b = t1.b)` as an example. The subquery `t1.a < (select sum(t2.a)from t2 where t2.b = t1.b)` here refers to the correlated column in the query condition `t2.b=t1.b`, this condition happens to be an equivalent condition, so the query can be rewritten as `select t1.* from t1, (select b, sum(a)sum_a from t2 group by b)t2 where t1.b = t2.b and t1.a < t2.sum_a;.` In this way, a correlated subquery is rewritten into JOIN.

The reason why TiDB needs to do this rewriting is that the correlated subquery is bound to its external query result every time the subquery is executed. In the above example, if `t1.a` has 10 million values, this subquery would repeat 10 million times, because the condition `t2 → .b=t1.b` varies with the value of `t1.a`. When the correlation is lifted somehow, this subquery would execute only once.

## Restrictions

The disadvantage of this rewriting is that when the correlation is not lifted, the optimizer can use the index on the correlated column. That is, although this subquery may repeat many times, the index can be used to filter data each time. After using the rewriting rule, the position of the correlated column usually changes. Although the subquery is only executed once, the single execution time would be longer than that without decorrelation.

Therefore, when there are few external values, do not perform decorrelation, because it may bring better execution performance. At present, this optimization can be disabled by setting `subquery decorrelation` optimization rules in [blocklist of optimization rules and expression pushdown](#).

## Example

```
create table t1(a int, b int);
create table t2(a int, b int, index idx(b));
explain select * from t1 where t1.a < (select sum(t2.a) from t2 where t2.b
→ = t1.b);
```

| +--             | → | -----+-----+-----+-----+       |
|-----------------|---|--------------------------------|
| id              | → | estRows   task   access object |
| → operator info | → |                                |
| →               |   |                                |
| +--             | → | -----+-----+-----+-----+       |

|                                                                        |                                        |
|------------------------------------------------------------------------|----------------------------------------|
| HashJoin_11                                                            | 9990.00   root   inner                 |
| ↳ join, equal:[eq(test.t1.b, test.t2.b)], other cond:< cast(test.t1.a) |                                        |
| ↳ , Column#7)                                                          |                                        |
| -HashAgg_23(Build)                                                     | 7992.00   root   group by              |
| ↳ :test.t2.b, funcs:sum(Column#8)->Column#7, funcs:firstrow(test.t2.b) |                                        |
| ↳ ->test.t2.b                                                          |                                        |
| -TableReader_24                                                        | 7992.00   root   data:                 |
| ↳ HashAgg_16                                                           |                                        |
| ↳                                                                      |                                        |
| - HashAgg_16                                                           | 7992.00   cop[tikv]   group by         |
| ↳ :test.t2.b, funcs:sum(test.t2.a)->Column#8                           |                                        |
| - Selection_22                                                         | 9990.00   cop[tikv]   not(             |
| ↳ isnull(test.t2.b))                                                   |                                        |
| ↳                                                                      |                                        |
| - TableFullScan_21                                                     | 10000.00   cop[tikv]   table:t2   keep |
| ↳ order:false, stats:pseudo                                            |                                        |
| ↳                                                                      |                                        |
| -TableReader_15(Probe)                                                 | 9990.00   root   data:                 |
| ↳ Selection_14                                                         |                                        |
| ↳                                                                      |                                        |
| - Selection_14                                                         | 9990.00   cop[tikv]   not(             |
| ↳ isnull(test.t1.b))                                                   |                                        |
| ↳                                                                      |                                        |
| - TableFullScan_13                                                     | 10000.00   cop[tikv]   table:t1   keep |
| ↳ order:false, stats:pseudo                                            |                                        |
| ↳                                                                      |                                        |
| +--                                                                    |                                        |
| ↳ -----+-----+-----+-----+                                             |                                        |
| ↳                                                                      |                                        |

The above is an example where the optimization takes effect. HashJoin\_11 is a normal inner join.

Then, turn off the subquery decorrelation rules:

```
insert into mysql.opt_rule_blacklist values("decorrelate");
admin reload opt_rule_blacklist;
explain select * from t1 where t1.a < (select sum(t2.a) from t2 where t2.b
 ↳ = t1.b);
```

| id                         | operator info | estRows | task | access object |
|----------------------------|---------------|---------|------|---------------|
| +--                        |               |         |      |               |
| ↳ -----+-----+-----+-----+ |               |         |      |               |
| ↳                          |               |         |      |               |

```
+--+
→ -----
→
| Projection_10 | 10000.00 | root |
| test.t1.a, test.t1.b |
|
| -Apply_12 | 10000.00 | root |
| CARTESIAN inner join, other cond:lt(cast(test.
| t1.a), Column#7) |
|
| -TableReader_14(Build) | 10000.00 | root |
| data:TableFullScan_13 |
|
| -TableFullScan_13 | 10000.00 | cop[tikv] | table:t1
| keep order:false, stats:pseudo |
|
| -MaxOneRow_15(Probe) | 1.00 | root |
|
|
| -HashAgg_27 | 1.00 | root |
| funcs:sum(Column#10)->Column#7 |
|
| -IndexLookUp_28 | 1.00 | root |
|
|
| -IndexRangeScan_25(Build) | 10.00 | cop[tikv] | table:t2, index
| :idx(b) | range: decided by [eq(test.t2.b, test.t1.b)], keep order:
| false, stats:pseudo |
|
| -HashAgg_17(Probe) | 1.00 | cop[tikv] |
| funcs:sum(test.t2.a)->Column#10 |
|
| -TableRowIDScan_26 | 10.00 | cop[tikv] | table:t2
| keep order:false, stats:pseudo |
|
+--+
→ -----
→
```

After disabling the subquery decorrelation rule, you can see `range: decided by [eq(test.t2.b, test.t1.b)]` in operator info of `IndexRangeScan_25(Build)`. It means that the decorrelation of correlated subquery is not performed and TiDB uses the index range query.

#### 8.3.3.2.5 Eliminate Max/Min

When a SQL statement contains `max/min` functions, the query optimizer tries to convert

the `max/min` aggregate functions to the TopN operator by applying the `max/min` optimization rule. In this way, TiDB can perform the query more efficiently through indexes.

This optimization rule is divided into the following two types according to the number of `max/min` functions in the `select` statement:

- The statement with only one `max/min` function
- The statement with multiple `max/min` functions

One `max/min` function

When a SQL statement meets the following conditions, this rule is applied:

- The statement contains only one aggregate function, which is `max` or `min`.
- The aggregate function has no related `group by` clause.

For example:

```
select max(a) from t
```

The optimization rule rewrites the statement as follows:

```
select max(a) from (select a from t where a is not null order by a desc
 ↪ limit 1) t
```

When column `a` has an index, or when column `a` is the prefix of some composite index, with the help of index, the new SQL statement can find the maximum or minimum value by scanning only one row of data. This optimization avoids full table scan.

The example statement has the following execution plan:

```
mysql> explain select max(a) from t;
+---+
| id | estRows | task | access object |
| operator info | | |
+---+
StreamAgg_13	1.00	root	
funcs:max(test.t.a)->Column#4			
-Limit_17	1.00	root	
offset:0, count:1			
-IndexReader_27	1.00	root	
index:Limit_26			
-Limit_26	1.00	cop[tikv]	
offset:0, count:1			
+---+
```

```

| - IndexFullScan_25 | 1.00 | cop[tikv] | table:t, index:idx_a(a) |
| ↳ keep order:true, desc, stats:pseudo |
+--+
| ↳ -----+-----+-----+
| ↳
5 rows in set (0.00 sec)

```

Multiple `max/min` functions

When a SQL statement meets the following conditions, this rule is applied:

- The statement contains multiple aggregate functions, which are all `max` or `min` functions.
- None of the aggregate functions has a related `group by` clause.
- The columns in each `max/min` function has indexes to preserve the order.

For example:

```
select max(a) - min(a) from t
```

The optimization rule first checks whether column `a` has an index to preserve its order. If yes, the SQL statement is rewritten as the Cartesian product of two subqueries:

```

select max_a - min_a
from
 (select max(a) as max_a from t) t1,
 (select min(a) as min_a from t) t2

```

Through the rewrite, the optimizer can apply the rule for statements with only one `max` /`min` function to the two subqueries respectively. The statement is then rewritten as follows:

```

select max_a - min_a
from
 (select max(a) as max_a from (select a from t where a is not null order
 ↳ by a desc limit 1) t) t1,
 (select min(a) as min_a from (select a from t where a is not null order
 ↳ by a asc limit 1) t) t2

```

Similarly, if column `a` has an index to preserve its order, the optimized execution only scans two rows of data instead of the whole table. However, if column `a` does not have an index to preserve its order, this rule results in two full table scans, but the execution only needs one full table scan if it is not rewritten. Therefore, in such cases, this rule is not applied.

The final execution plan is as follows:

```

mysql> explain select max(a)-min(a) from t;
+---+
| | id | estRows | task | access object |
| | operator info | | |
+---+
	Projection_17	1.00	root		
	↳ minus(Column#4, Column#5)->Column#6				
	↳ HashJoin_18	1.00	root		
	↳	CARTESIAN inner join			
	↳	StreamAgg_45(Build)	1.00	root	
	↳	funcs:min(test.t.a)->Column#5			
	↳	Limit_49	1.00	root	
	↳	offset:0, count:1			
	↳	IndexReader_59	1.00	root	
	↳	index:Limit_58			
	↳	Limit_58	1.00	cop[tikv]	
	↳	offset:0, count:1			
	↳	IndexFullScan_57	1.00	cop[tikv]	table:t, index:
	↳ idx_a(a)	keep order:true, stats:pseudo			
	↳ StreamAgg_24(Probe)	1.00	root		
	↳ funcs:max(test.t.a)->Column#4				
	↳	Limit_28	1.00	root	
	↳	offset:0, count:1			
	↳	IndexReader_38	1.00	root	
	↳	index:Limit_37			
	↳	Limit_37	1.00	cop[tikv]	
	↳	offset:0, count:1			
	↳	IndexFullScan_36	1.00	cop[tikv]	table:t, index:
	↳ idx_a(a)	keep order:true, desc, stats:pseudo			
+---+					
	↳				
	↳				
12 rows in set (0.01 sec)

```

### 8.3.3.2.6 Predicates Push Down (PPD)

This document introduces one of the TiDB's logic optimization rules—Predicate Push Down (PPD). It aims to help you understand the predicate push down and know its applicable and inapplicable scenarios.

PPD pushes down selection operators to data source as close as possible to complete data filtering as early as possible, which significantly reduces the cost of data transmission or computation.

## Examples

The following cases describe the optimization of PPD. Case 1, 2, and 3 are scenarios where PPD is applicable, and Case 4, 5, and 6 are scenarios where PPD is not applicable.

Case 1: push predicates to storage layer

```
create table t(id int primary key, a int);
explain select * from t where a < 1;
+---+
→ -----+-----+-----+
→
→
| id | estRows | task | access object | operator info
→
→
+---+
→ -----+-----+-----+
→
→
| TableReader_7 | 3323.33 | root | | data:Selection_6
→
→
| -Selection_6 | 3323.33 | cop[tikv] | | lt(test.t.a, 1)
→
→
| -TableFullScan_5 | 10000.00 | cop[tikv] | table:t | keep order:false
→ , stats:pseudo |
+---+
→ -----+-----+-----+
→
→
3 rows in set (0.00 sec)
```

In this query, pushing down the predicate  $a < 1$  to the TiKV layer to filter the data can reduce the overhead of network transmission.

Case 2: push predicates to storage layer

```
create table t(id int primary key, a int not null);
explain select * from t where a < substring('123', 1, 1);
+---+
 ↗-----+-----+-----+-----+
 ↗-----+
| id | estRows | task | access object | operator info
 ↗ |
+---+
 ↗-----+-----+-----+-----+
 ↗-----+
| TableReader_7 | 3323.33 | root | | data:Selection_6
 ↗ |

```

|                                                                      |
|----------------------------------------------------------------------|
| -Selection_6   3323.33   cop[tikv]   lt(test.t.a, 1)                 |
| ↳                                                                    |
| -TableFullScan_5   10000.00   cop[tikv]   table:t   keep order:false |
| ↳ , stats:pseudo                                                     |
| +--                                                                  |
| ↳ -----+-----+-----+-----+                                           |
| ↳                                                                    |

This query has the same execution plan as the query in case 1, because the input parameters of the `substring` of the predicate `a < substring('123', 1, 1)` are constants, so they can be calculated in advance. Then the predicate is simplified to the equivalent predicate `a < 1`. After that, TiDB can push `a < 1` down to TiKV.

Case 3: push predicates below join operator

| create table t(id int primary key, a int not null);<br>create table s(id int primary key, a int not null);<br>explain select * from t join s on t.a = s.a where t.a < 1; |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| +--                                                                                                                                                                      |
| ↳ -----+-----+-----+-----+                                                                                                                                               |
| ↳                                                                                                                                                                        |
| id   estRows   task   access object   operator                                                                                                                           |
| ↳ info                                                                                                                                                                   |
| +--                                                                                                                                                                      |
| ↳ -----+-----+-----+-----+                                                                                                                                               |
| ↳                                                                                                                                                                        |
| HashJoin_8   4154.17   root   inner join,                                                                                                                                |
| ↳ equal:[eq(test.t.a, test.s.a)]                                                                                                                                         |
| -TableReader_15(Build)   3323.33   root   data:                                                                                                                          |
| ↳ Selection_14                                                                                                                                                           |
| -Selection_14   3323.33   cop[tikv]   lt(test.s.a                                                                                                                        |
| ↳ , 1)                                                                                                                                                                   |
| -TableFullScan_13   10000.00   cop[tikv]   table:s   keep order:                                                                                                         |
| ↳ false, stats:pseudo                                                                                                                                                    |
| -TableReader_12(Probe)   3323.33   root   data:                                                                                                                          |
| ↳ Selection_11                                                                                                                                                           |
| -Selection_11   3323.33   cop[tikv]   lt(test.t.a                                                                                                                        |
| ↳ , 1)                                                                                                                                                                   |
| -TableFullScan_10   10000.00   cop[tikv]   table:t   keep order:                                                                                                         |
| ↳ false, stats:pseudo                                                                                                                                                    |
| +--                                                                                                                                                                      |
| ↳ -----+-----+-----+-----+                                                                                                                                               |
| ↳                                                                                                                                                                        |
| 7 rows in set (0.00 sec)                                                                                                                                                 |

In this query, the predicate `t.a < 1` is pushed below join to filter in advance, which can

reduce the calculation overhead of join.

In addition, This SQL statement has an inner join executed, and the `ON` condition is `t.a = s.a`. The predicate `s.a < 1` can be derived from `t.a < 1` and pushed down to `s` table below the join operator. Filtering the `s` table can further reduce the calculation overhead of join.

Case 4: predicates that are not supported by storage layers cannot be pushed down

| id               | estRows | task      | access object | operator info                                         |
|------------------|---------|-----------|---------------|-------------------------------------------------------|
|                  |         |           |               |                                                       |
| Selection_7      | 2.00    | root      |               | eq( <code>substring("123", test.t.a, 1)</code> , "1") |
| -TableReader_6   | 2.00    | root      |               | data:                                                 |
| TableFullScan_5  |         |           |               |                                                       |
| -TableFullScan_5 | 2.00    | cop[tikv] | table:t       | keep order:false,                                     |
| stats:pseudo     |         |           |               |                                                       |

In this query, there is a predicate `substring('123', a, 1)= '1'`.

From the `explain` results, we can see that the predicate is not pushed down to TiKV for calculation. This is because the TiKV coprocessor does not support the built-in function `substring`.

Case 5: predicates of inner tables on the outer join can't be pushed down

| id   | estRows | task | access object | operator |
|------|---------|------|---------------|----------|
|      |         |      |               |          |
| info |         |      |               |          |

```

| Selection_7 | 10000.00 | root | | isnull(test
| ↳ .s.a) | | | |
| -HashJoin_8 | 12500.00 | root | | left outer
| ↳ join, equal:[eq(test.t.a, test.s.a)] |
| -TableReader_13(Build) | 10000.00 | root | | data:
| ↳ TableFullScan_12 | | | |
| -TableFullScan_12 | 10000.00 | cop[tikv] | table:s | keep order:
| ↳ false, stats:pseudo | | |
| -TableReader_11(Probe) | 10000.00 | root | | data:
| ↳ TableFullScan_10 | | | |
| -TableFullScan_10 | 10000.00 | cop[tikv] | table:t | keep order:
| ↳ false, stats:pseudo | | |
+--+
| ↳ -----+-----+-----+-----+
| ↳
6 rows in set (0.00 sec)

```

In this query, there is a predicate `s.a is null` on the inner table `s`.

From the `explain` results, we can see that the predicate is not pushed below join operator. This is because the outer join fills the inner table with NULL values when the `on` condition isn't satisfied, and the predicate `s.a is null` is used to filter the results after the join. If it is pushed down to the inner table below join, the execution plan is not equivalent to the original one.

Case 6: the predicates which contain user variables cannot be pushed down

```

create table t(id int primary key, a char);
set @a = 1;
explain select * from t where a < @a;
+--
| id | estRows | task | access object | operator info
| | | | |
+--+
| Selection_5 | 8000.00 | root | | lt(test.t.a,
| ↳ getvar("a")) |
| -TableReader_7 | 10000.00 | root | | data:
| ↳ TableFullScan_6 | | | |
| -TableFullScan_6 | 10000.00 | cop[tikv] | table:t | keep order:false
| ↳ , stats:pseudo | | |
+--+
| ↳ -----+-----+-----+-----+
| ↳

```

```
3 rows in set (0.00 sec)
```

In this query, there is a predicate `a < @a` on table `t`. The `@a` of the predicate is a user variable.

As can be seen from `explain` results, the predicate is not like case 2, which is simplified to `a < 1` and pushed down to TiKV. This is because the value of the user variable `@a` may change during the computation, and TiKV is not aware of the changes. So TiDB does not replace `@a` with 1, and does not push down it to TiKV.

An example to help you understand is as follows:

```
create table t(id int primary key, a int);
insert into t values(1, 1), (2,2);
set @a = 1;
select id, a, @a:=@a+1 from t where a = @a;
+----+----+-----+
| id | a | @a:=@a+1 |
+----+----+-----+
| 1 | 1 | 2 |
| 2 | 2 | 3 |
+----+----+-----+
2 rows in set (0.00 sec)
```

As you can see from this query, the value of `@a` will change during the query. So if you replace `a = @a` with `a = 1` and push it down to TiKV, it's not an equivalent execution plan.

#### 8.3.3.2.7 Partition Pruning

Partition pruning is a performance optimization that applies to partitioned tables. It analyzes the filter conditions in query statements, and eliminates (*prunes*) partitions from consideration when they do not contain any data that will be required. By eliminating the non-required partitions, TiDB is able to reduce the amount of data that needs to be accessed and potentially significantly improving query execution times.

The following is an example:

```
CREATE TABLE t1 (
 id INT NOT NULL PRIMARY KEY,
 pad VARCHAR(100)
)
PARTITION BY RANGE COLUMNS(id) (
 PARTITION p0 VALUES LESS THAN (100),
 PARTITION p1 VALUES LESS THAN (200),
 PARTITION p2 VALUES LESS THAN (MAXVALUE)
);

INSERT INTO t1 VALUES (1, 'test1'),(101, 'test2'), (201, 'test3');
```

```
EXPLAIN SELECT * FROM t1 WHERE id BETWEEN 80 AND 120;
```

| +--                      | id                                             | estRows | task      | access object          | operator info |
|--------------------------|------------------------------------------------|---------|-----------|------------------------|---------------|
|                          | PartitionUnion_8                               | 80.00   | root      |                        |               |
|                          | -TableReader_10                                | 40.00   | root      |                        | data:         |
|                          | TableRangeScan_9                               |         |           |                        |               |
|                          | -TableRangeScan_9                              | 40.00   | cop[tikv] | table:t1, partition:p0 |               |
|                          | range:[80,120], keep order:false, stats:pseudo |         |           |                        |               |
|                          | -TableReader_12                                | 40.00   | root      |                        | data:         |
|                          | TableRangeScan_11                              |         |           |                        |               |
|                          | -TableRangeScan_11                             | 40.00   | cop[tikv] | table:t1, partition:p1 |               |
|                          | range:[80,120], keep order:false, stats:pseudo |         |           |                        |               |
| 5 rows in set (0.00 sec) |                                                |         |           |                        |               |

### Usage scenarios of partition pruning

The usage scenarios of partition pruning are different for the two types of partitioned tables: Range partitioned tables and Hash partitioned tables.

#### Use partition pruning in Hash partitioned tables

This section describes the applicable and inapplicable usage scenarios of partition pruning in Hash partitioned tables.

##### Applicable scenario in Hash partitioned tables

Partition pruning applies only to the query condition of equality comparison in Hash partitioned tables.

```
create table t (x int) partition by hash(x) partitions 4;
explain select * from t where x = 1;
```

| +-- | id | estRows | task | access object | operator info |
|-----|----|---------|------|---------------|---------------|
|     |    |         |      |               |               |

|                                  |                    |           |                       |           |
|----------------------------------|--------------------|-----------|-----------------------|-----------|
| +--                              |                    |           |                       |           |
| ↳                                | -----+-----+-----+ |           |                       | -----+    |
| ↳                                |                    |           |                       |           |
| TableReader_8                    | 10.00              | root      |                       | data:     |
| ↳ Selection_7                    |                    |           |                       |           |
| -Selection_7                     | 10.00              | cop[tikv] |                       | eq(test.t |
| ↳ .x, 1)                         |                    |           |                       |           |
| -TableFullScan_6                 | 10000.00           | cop[tikv] | table:t, partition:p1 |           |
| ↳ keep order:false, stats:pseudo |                    |           |                       |           |
| +--                              |                    |           |                       |           |
| ↳                                | -----+-----+-----+ |           |                       | -----+    |
| ↳                                |                    |           |                       |           |

In the SQL statement above, it can be known from the condition `x = 1` that all results fall in one partition. The value 1 can be confirmed to be in the p1 partition after passing through the Hash partition. Therefore, only the p1 partition needs to be scanned, and there is no need to access the p2, p3, and p4 partitions that will not have matching results. From the execution plan, only one `TableFullScan` operator appears and the p1 partition is specified in `access object`, so it can be confirmed that `partition pruning` takes effect.

#### Inapplicable scenarios in Hash partitioned tables

This section describes two inapplicable usage scenarios of partition pruning in Hash partitioned tables.

##### Scenario one

If you cannot confirm the condition that the query result falls in only one partition (such as `in`, `between`, `>`, `<`, `>=`, `<=`), you cannot use the partition pruning optimization. For example:

```
create table t (x int) partition by hash(x) partitions 4;
explain select * from t where x > 2;
```

|                 |                    |           |               |        |
|-----------------|--------------------|-----------|---------------|--------|
| +--             |                    |           |               |        |
| ↳               | -----+-----+-----+ |           |               | -----+ |
| ↳               |                    |           |               |        |
| id              | estRows            | task      | access object |        |
| ↳ operator info |                    |           |               |        |
| +--             |                    |           |               |        |
| ↳               | -----+-----+-----+ |           |               | -----+ |
| ↳               |                    |           |               |        |
| Union_10        | 13333.33           | root      |               |        |
| ↳               |                    |           |               |        |
| -TableReader_13 | 3333.33            | root      |               | data   |
| ↳ :Selection_12 |                    |           |               |        |
| -Selection_12   | 3333.33            | cop[tikv] |               | gt(    |
| ↳ test.t.x, 2)  |                    |           |               |        |

```

| - TableFullScan_11 | 10000.00 | cop[tikv] | table:t, partition:p0
| ↵ | keep order:false, stats:pseudo |
| - TableReader_16 | 3333.33 | root | data
| ↵ :Selection_15 |
| - Selection_15 | 3333.33 | cop[tikv] | gt(
| ↵ test.t.x, 2) |
| - TableFullScan_14 | 10000.00 | cop[tikv] | table:t, partition:p1
| ↵ | keep order:false, stats:pseudo |
| - TableReader_19 | 3333.33 | root | data
| ↵ :Selection_18 |
| - Selection_18 | 3333.33 | cop[tikv] | gt(
| ↵ test.t.x, 2) |
| - TableFullScan_17 | 10000.00 | cop[tikv] | table:t, partition:p2
| ↵ | keep order:false, stats:pseudo |
| - TableReader_22 | 3333.33 | root | data
| ↵ :Selection_21 |
| - Selection_21 | 3333.33 | cop[tikv] | gt(
| ↵ test.t.x, 2) |
| - TableFullScan_20 | 10000.00 | cop[tikv] | table:t, partition:p3
| ↵ | keep order:false, stats:pseudo |
+--+
| ↵ -----
| ↵
| ↵

```

In this case, partition pruning is inapplicable because the corresponding Hash partition cannot be confirmed by the  $x > 2$  condition.

#### Scenario two

Because the rule optimization of partition pruning is performed during the generation phase of the query plan, partition pruning is not suitable for scenarios where the filter conditions can be obtained only during the execution phase. For example:

```

create table t (x int) partition by hash(x) partitions 4;
explain select * from t2 where x = (select * from t1 where t2.x = t1.x and
 ↵ t2.x < 2);

```

```

+--+
| ↵ -----
| ↵
| ↵
| id | estRows | task | access object
| ↵ | operator info |
+--+
| ↵ -----
| ↵
| Projection_13 | 9990.00 | root |
| ↵ | test.t2.x |

```

```

-Apply_15	9990.00	root
↳	inner join, equal:[eq(test.t2.x, test.t1.x)]	
-TableReader_18(Build)	9990.00	root
↳	data:Selection_17	
-Selection_17	9990.00	cop[tikv]
↳	not(isnull(test.t2.x))	
-TableFullScan_16	10000.00	cop[tikv]
↳	keep order:false, stats:pseudo	
-Selection_19(Probe)	0.80	root
↳	not(isnull(test.t1.x))	
-MaxOneRow_20	1.00	root
↳		
-Union_21	2.00	root
↳		
-TableReader_24	2.00	root
↳	data:Selection_23	
-Selection_23	2.00	cop[tikv]
↳	eq(test.t2.x, test.t1.x), lt(test.t2.x, 2)	
-TableFullScan_22	2500.00	cop[tikv]
↳ partition:p0	keep order:false, stats:pseudo	
-TableReader_27	2.00	root
↳	data:Selection_26	
-Selection_26	2.00	cop[tikv]
↳	eq(test.t2.x, test.t1.x), lt(test.t2.x, 2)	
-TableFullScan_25	2500.00	cop[tikv]
↳ partition:p1	keep order:false, stats:pseudo	
+--+		
↳-----+-----+-----+-----+		
↳		

```

Each time this query reads a row from `t2`, it will query on the `t1` partitioned table. Theoretically, the filter condition of `t1.x = val` is met at this time, but in fact, partition pruning takes effect only in the generation phase of the query plan, not the execution phase.

#### Use partition pruning in Range partitioned tables

This section describes the applicable and inapplicable usage scenarios of partition pruning in Range partitioned tables.

#### Applicable scenarios in Range partitioned tables

This section describes three applicable usage scenarios of partition pruning in Range partitioned tables.

##### Scenario one

Partition pruning applies to the query condition of equality comparison in Range partitioned tables. For example:

```
create table t (x int) partition by range (x) (
 partition p0 values less than (5),
 partition p1 values less than (10),
 partition p2 values less than (15)
);
explain select * from t where x = 3;
```

| +--                              | ↓        | ↓         | ↓                     | ↓         |
|----------------------------------|----------|-----------|-----------------------|-----------|
| id                               | estRows  | task      | access object         | operator  |
| ↵ info                           |          |           |                       |           |
| +--                              |          |           |                       |           |
| ↵                                |          |           |                       |           |
| TableReader_8                    | 10.00    | root      |                       | data:     |
| ↵ Selection_7                    |          |           |                       |           |
| -Selection_7                     | 10.00    | cop[tikv] |                       | eq(test.t |
| ↵ .x, 3)                         |          |           |                       |           |
| -TableFullScan_6                 | 10000.00 | cop[tikv] | table:t, partition:p0 |           |
| ↵ keep order:false, stats:pseudo |          |           |                       |           |
| +--                              |          |           |                       |           |
| ↵                                |          |           |                       |           |

Partition pruning also applies to the equality comparison that uses the `in` query condition. For example:

```
create table t (x int) partition by range (x) (
 partition p0 values less than (5),
 partition p1 values less than (10),
 partition p2 values less than (15)
);
explain select * from t where x in(1,13);
```

| +--             | ↓       | ↓    | ↓             | ↓ |
|-----------------|---------|------|---------------|---|
| id              | estRows | task | access object |   |
| ↵ operator info |         |      |               |   |
| +--             |         |      |               |   |
| ↵               |         |      |               |   |
| Union_8         | 40.00   | root |               |   |
| ↵               |         |      |               |   |

|                                  |          |           |                       |       |
|----------------------------------|----------|-----------|-----------------------|-------|
| -TableReader_11                  | 20.00    | root      |                       | data: |
| ↳ Selection_10                   |          |           |                       |       |
| -Selection_10                    | 20.00    | cop[tikv] |                       | in(   |
| ↳ test.t.x, 1, 13)               |          |           |                       |       |
| -TableFullScan_9                 | 10000.00 | cop[tikv] | table:t, partition:p0 |       |
| ↳ keep order:false, stats:pseudo |          |           |                       |       |
| -TableReader_14                  | 20.00    | root      |                       | data: |
| ↳ Selection_13                   |          |           |                       |       |
| -Selection_13                    | 20.00    | cop[tikv] |                       | in(   |
| ↳ test.t.x, 1, 13)               |          |           |                       |       |
| -TableFullScan_12                | 10000.00 | cop[tikv] | table:t, partition:p2 |       |
| ↳ keep order:false, stats:pseudo |          |           |                       |       |
| +--                              |          |           |                       |       |
| ↳ -----+-----+-----+             |          |           |                       |       |
| ↳                                |          |           |                       |       |

In the SQL statement above, it can be known from the `x in(1,13)` condition that all results fall in a few partitions. After analysis, it is found that all records of `x = 1` are in the `p0` partition, and all records of `x = 13` are in the `p2` partition, so only `p0` and `p2` partitions need to be accessed.

### Scenario two

Partition pruning applies to the query condition of interval comparison, such as `between`, `>`, `<`, `=`, `>=`, `<=`. For example:

```
create table t (x int) partition by range (x) (
 partition p0 values less than (5),
 partition p1 values less than (10),
 partition p2 values less than (15)
);
explain select * from t where x between 7 and 14;
```

|                                  |         |           |               |       |
|----------------------------------|---------|-----------|---------------|-------|
| +--                              |         |           |               |       |
| ↳ -----+-----+-----+-----+       |         |           |               |       |
| ↳                                |         |           |               |       |
| id                               | estRows | task      | access object |       |
| ↳ operator info                  |         |           |               |       |
| +--                              |         |           |               |       |
| ↳ -----+-----+-----+-----+       |         |           |               |       |
| ↳                                |         |           |               |       |
| Union_8                          | 500.00  | root      |               |       |
| ↳                                |         |           |               |       |
| -TableReader_11                  | 250.00  | root      |               | data: |
| ↳ Selection_10                   |         |           |               |       |
| -Selection_10                    | 250.00  | cop[tikv] |               | ge(   |
| ↳ test.t.x, 7), le(test.t.x, 14) |         |           |               |       |

```

| - TableFullScan_9 | 10000.00 | cop[tikv] | table:t, partition:p1 |
| ↳ keep order:false, stats:pseudo |
| - TableReader_14 | 250.00 | root | data:
| ↳ Selection_13 |
| - Selection_13 | 250.00 | cop[tikv] | ge(
| ↳ test.t.x, 7), le(test.t.x, 14) |
| - TableFullScan_12 | 10000.00 | cop[tikv] | table:t, partition:p2 |
| ↳ keep order:false, stats:pseudo |
+--+
| ↳ -----
| ↳

```

### Scenario three

Partition pruning applies to the scenario where the partition expression is in the simple form of `fn(col)`, the query condition is one of `>`, `<`, `=`, `>=`, and `<=`, and the `fn` function is monotonous.

If the `fn` function is monotonous, for any `x` and `y`, if `x > y`, then `fn(x) > fn(y)`. Then this `fn` function can be called strictly monotonous. For any `x` and `y`, if `x > y`, then `fn(x) >= fn(y)`. In this case, `fn` could also be called “monotonous”. Theoretically, all monotonous functions, strictly or not, are supported by partition pruning. Currently, TiDB only supports the following monotonous functions:

```
unix_timestamp
to_days
```

For example, partition pruning takes effect when the partition expression is in the form of `fn(col)`, where the `fn` is monotonous function `to_days`:

```
create table t (id datetime) partition by range (to_days(id)) (
 partition p0 values less than (to_days('2020-04-01')),
 partition p1 values less than (to_days('2020-05-01')));
explain select * from t where id > '2020-04-18';
```

```

+--+
| id | estRows | task | access object | operator |
| ↳ info | | | |
+--+
| TableReader_8 | 3333.33 | root | data:
| ↳ Selection_7 |
| - Selection_7 | 3333.33 | cop[tikv] | gt(test.t
| ↳ .id, 2020-04-18 00:00:00.000000) |

```

```

| - TableFullScan_6 | 10000.00 | cop[tikv] | table:t, partition:p1 |
| ↳ keep order:false, stats:pseudo |
+--+
| ↳ -----
| ↳

```

Inapplicable scenario in Range partitioned tables

Because the rule optimization of partition pruning is performed during the generation phase of the query plan, partition pruning is not suitable for scenarios where the filter conditions can be obtained only during the execution phase. For example:

```

create table t1 (x int) partition by range (x) (
 partition p0 values less than (5),
 partition p1 values less than (10));
create table t2 (x int);
explain select * from t2 where x < (select * from t1 where t2.x < t1.x and
 ↳ t2.x < 2);

```

```

+--+
| id | estRows | task | access object
| ↳ | operator info
+--+
| Projection_13 | 9990.00 | root | test.t2.x
| ↳
| -Apply_15 | 9990.00 | root | CARTESIAN inner join, other cond:lt(test.t2.x,
| ↳ | | test.t1.x) |
| -TableReader_18(Build) | 9990.00 | root | data:Selection_17
| ↳
| -Selection_17 | 9990.00 | cop[tikv] | |
| ↳ | not(isnull(test.t2.x)) |
| ↳
| -TableFullScan_16 | 10000.00 | cop[tikv] | table:t2
↳	keep order:false, stats:pseudo		
-Selection_19(Probe)	0.80	root	
↳	not(isnull(test.t1.x))		
↳			
-MaxOneRow_20	1.00	root	
↳			

```

```

| ↗
| -Union_21 | 2.00 | root |
| ↘
| ↗
| -TableReader_24 | 2.00 | root |
| ↘ | data:Selection_23
| ↘
| -Selection_23 | 2.00 | cop[tikv] |
| ↘ | lt(test.t2.x, 2), lt(test.t2.x, test.t1.x)
| ↘
| -TableFullScan_22 | 2.50 | cop[tikv] | table:t1,
| ↘ partition:p0 | keep order:false, stats:pseudo
| -TableReader_27 | 2.00 | root |
| ↘ | data:Selection_26
| ↘
| -Selection_26 | 2.00 | cop[tikv] |
| ↘ | lt(test.t2.x, 2), lt(test.t2.x, test.t1.x)
| ↘
| -TableFullScan_25 | 2.50 | cop[tikv] | table:t1,
| ↘ partition:p1 | keep order:false, stats:pseudo
+--+
→ -----+-----+-----+
→
14 rows in set (0.00 sec)

```

Each time this query reads a row from `t2`, it will query on the `t1` partitioned table. Theoretically, the `t1.x > val` filter condition is met at this time, but in fact, partition pruning takes effect only in the generation phase of the query plan, not the execution phase.

#### 8.3.3.2.8 TopN and Limit Operator Push Down

This document describes the implementation of TopN and Limit operator pushdown.

In the TiDB execution plan tree, the `LIMIT` clause in SQL corresponds to the Limit operator node, and the `ORDER BY` clause corresponds to the Sort operator node. The adjacent Limit operator and Sort operator are combined as the TopN operator node, which means that the top N records are returned according to a certain sorting rule. That is to say, a Limit operator is equivalent to a TopN operator node with a null sorting rule.

Similar to predicate pushdown, TopN and Limit are pushed down in the execution plan tree to a position as close to the data source as possible so that the required data is filtered at an early stage. In this way, the pushdown significantly reduces the overhead of data transmission and calculation.

To disable this rule, refer to [Optimization Rules and Blocklist for Expression Pushdown](#).

Examples

This section illustrates TopN pushdown through some examples.

Example 1: Push down to the Coprocessors in the storage layer

```
create table t(id int primary key, a int not null);
explain select * from t order by a limit 10;
```

| id                | estRows  | task      | access object | operator                           |
|-------------------|----------|-----------|---------------|------------------------------------|
| TopN_7            | 10.00    | root      |               | test.t.a,<br>offset:0, count:10    |
| -TableReader_15   | 10.00    | root      |               | data:TopN_14                       |
| -TopN_14          | 10.00    | cop[tikv] |               | test.t.a,<br>offset:0, count:10    |
| -TableFullScan_13 | 10000.00 | cop[tikv] | table:t       | keep order:<br>false, stats:pseudo |

4 rows in set (0.00 sec)

In this query, the TopN operator node is pushed down to TiKV for data filtering, and each Coprocessor returns only 10 records to TiDB. After TiDB aggregates the data, the final filtering is performed.

Example 2: TopN can be pushed down into Join (the sorting rule only depends on the columns in the outer table)

```
create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t left join s on t.a = s.a order by t.a limit 10;
```

| id           | estRows | task | access object                                      |
|--------------|---------|------|----------------------------------------------------|
| TopN_12      | 10.00   | root | test.t.a,<br>offset:0, count:10                    |
| -HashJoin_17 | 12.50   | root | left<br>outer join, equal:[eq(test.t.a, test.s.a)] |

```

| - TopN_18(Build) | 10.00 | root | | test.t.a
| ↵ , offset:0, count:10 | | |
| - TableReader_26 | 10.00 | root | | data:
| ↵ TopN_25 | | |
| - TopN_25 | 10.00 | cop[tikv] | | test.t.a
| ↵ , offset:0, count:10 | | |
| - TableFullScan_24 | 10000.00 | cop[tikv] | table:t | keep
| ↵ order:false, stats:pseudo | |
| - TableReader_30(Probe) | 10000.00 | root | | data:
| ↵ TableFullScan_29 | | |
| - TableFullScan_29 | 10000.00 | cop[tikv] | table:s | keep
| ↵ order:false, stats:pseudo | |
+-----+-----+-----+-----+
 ↵
8 rows in set (0.01 sec)

```

In this query, the sorting rule of the TopN operator only depends on the columns in the outer table  $t$ , so a calculation can be performed before pushing down TopN to Join, to reduce the calculation cost of the Join operation. Besides, TiDB also pushes TopN down to the storage layer.

Example 3: TopN cannot be pushed down before Join

```

create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t join s on t.a = s.a order by t.id limit 10;

```

```

+-----+-----+-----+
 ↵
| id | estRows | task | access object | operator
| ↵ info | | | |
+-----+-----+-----+
 ↵
| TopN_12 | 10.00 | root | | test.t.id,
| ↵ offset:0, count:10 | |
| -HashJoin_16 | 12500.00 | root | | inner join,
| ↵ equal:[eq(test.t.a, test.s.a)] |
| -TableReader_21(Build) | 10000.00 | root | | data:
| ↵ TableFullScan_20 | |
| -TableFullScan_20 | 10000.00 | cop[tikv] | table:s | keep order:
| ↵ false, stats:pseudo | |
| -TableReader_19(Probe) | 10000.00 | root | | data:
| ↵ TableFullScan_18 | |
| -TableFullScan_18 | 10000.00 | cop[tikv] | table:t | keep order:
| ↵ false, stats:pseudo | |

```

```
+-----+-----+-----+
 ↗
6 rows in set (0.00 sec)
```

TopN cannot be pushed down before `Inner Join`. Taking the query above as an example, if you get 100 records after Join, then you can have 10 records left after TopN. However, if TopN is performed first to get 10 records, only 5 records are left after Join. In such cases, the pushdown results in different results.

Similarly, TopN can neither be pushed down to the inner table of Outer Join, nor can it be pushed down when its sorting rule is related to columns on multiple tables, such as `t.a+s.a`. Only when the sorting rule of TopN exclusively depends on columns on the outer table, can TopN be pushed down.

Example 4: Convert TopN to Limit

```
create table t(id int primary key, a int not null);
create table s(id int primary key, a int not null);
explain select * from t left join s on t.a = s.a order by t.id limit 10;
```

```
+-----+-----+-----+
 ↗
| id | estRows | task | access object |
| ↗ operator info | | |
+-----+-----+-----+
 ↗
| TopN_12 | 10.00 | root | | test.t.id
| ↗ , offset:0, count:10 | | |
| -HashJoin_17 | 12.50 | root | | left
| ↗ outer join, equal:[eq(test.t.a, test.s.a)] |
| -Limit_21(Build) | 10.00 | root | | offset
| ↗ :0, count:10 | | |
| -TableReader_31 | 10.00 | root | | data:
| ↗ Limit_30 | | |
| -Limit_30 | 10.00 | cop[tikv] | | offset
| ↗ :0, count:10 | | |
| -TableFullScan_29 | 10.00 | cop[tikv] | table:t | keep
| ↗ order:true, stats:pseudo |
| -TableReader_35(Probe) | 10000.00 | root | | data:
| ↗ TableFullScan_34 | | |
| -TableFullScan_34 | 10000.00 | cop[tikv] | table:s | keep
| ↗ order:false, stats:pseudo |
+-----+-----+-----+
 ↗
8 rows in set (0.00 sec)
```

In the query above, TopN is first pushed to the outer table `t`. TopN needs to sort by `t → .id`, which is the primary key and can be directly read in order (`keep order: true`) without extra sorting in TopN. Therefore, TopN is simplified as Limit.

#### 8.3.3.2.9 Introduction to Join Reorder

In real application scenarios, it is common to join multiple tables. The execution efficiency of join is associated with the order in which each table joins.

For example:

```
SELECT * FROM t1, t2, t3 WHERE t1.a=t2.a AND t3.a=t2.a;
```

In this query, tables can be joined in the following two orders:

- `t1` joins `t2`, and then joins `t3`
- `t2` joins `t3`, and then joins `t1`

As `t1` and `t3` have different data volumes and distribution, these two execution orders might show different performances.

Therefore, the optimizer needs an algorithm to determine the join order. Currently, TiDB uses the Join Reorder algorithm, also known as the greedy algorithm.

Instance of Join Reorder algorithm

Take the three tables above (`t1`, `t2`, and `t3`) as an example.

First, TiDB obtains all nodes in the ascending order

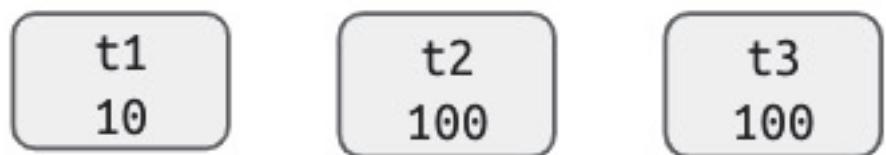
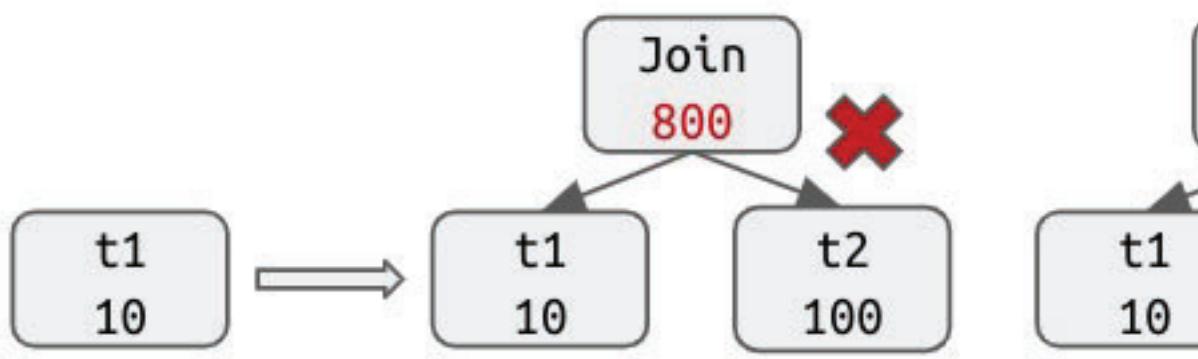


Figure 64: join-reorder-1

After that, the table with the least rows is selected and joined with other two tables respectively. By comparing the sizes of the output result sets, TiDB selects the pair with a smaller result set.



Then TiDB enters the next step and continues to compare the sizes of the intermediate result set.

In this case only three tables

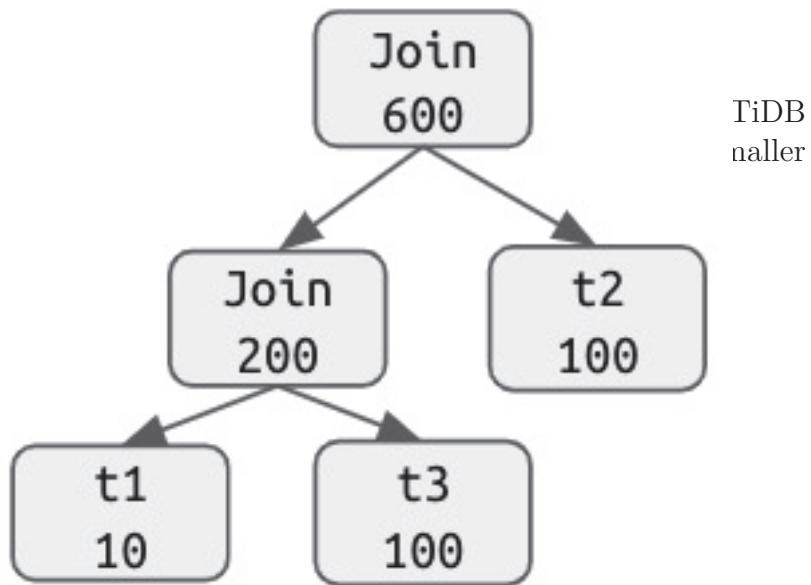


Figure 66: join-reorder-3

The above process is the Join Reorder algorithm currently used in TiDB.

Limitations of Join Reorder algorithm

The current Join Reorder algorithm has the following limitations:

- The Join Reorder for Outer Join is not supported
- Limited by the calculation methods of the result sets, the algorithm cannot ensure it selects the optimum join order.

Currently, the `STRAIGHT_JOIN` syntax is supported in TiDB to force a join order. For more information, refer to [Description of the syntax elements](#).

### 8.3.3.3 Physical Optimization

#### 8.3.3.3.1 SQL Physical Optimization

Physical optimization is cost-based optimization, which makes a physical execution plan for the logical execution plan generated in the previous stage. In this stage, the optimizer selects a specific physical implementation for each operator in the logical execution plan. Different physical implementations of logical operators have different time complexity, resource consumption and physical properties. In this process, the optimizer determines the cost of different physical implementations based on the statistics of the data, and selects the physical execution plan with the smallest overall cost.

[Understand the Query Execution Plan](#) has introduced some physical operators. This chapter focuses on the following aspects:

- In [Index Selection](#), you will learn how to select the optimal index to access tables when TiDB has multiple indexes on a table.
- In [Introduction to Statistics](#), you will learn what statistics TiDB collects to obtain the data distribution of a table.
- [Wrong Index Solution](#) introduces how to use the right index when you find the index is selected wrongly.
- [Distinct Optimization](#) introduces an optimization related to the DISTINCT keyword during physical optimization. In this section, you will learn its advantages and disadvantages and how to use it.

#### 8.3.3.3.2 Index Selection

Reading data from storage engines is one of the most time-consuming steps during the SQL execution. Currently, TiDB supports reading data from different storage engines and different indexes. Query execution performance depends largely on whether you select a suitable index or not.

This document introduces how to select an index to access a table, and some related ways to control index selection.

Access tables

Before introducing index selection, it is important to understand the ways TiDB accesses tables, what triggers each way, what differences each way makes, and what the pros and cons are.

Operators for accessing tables

| Operator                  | Trigger Conditions                                        | Applicable Scenarios | Explanations                                                                                                                                                                         |
|---------------------------|-----------------------------------------------------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PointGet / Batch-PointGet | When accessing tables in one or more single point ranges. | Any scenario         | If triggered, it is usually considered as the fastest operator, since it calls the kvget interface directly to perform the calculations rather than calls the coprocessor interface. |
|                           |                                                           |                      |                                                                                                                                                                                      |

| Operator  | Trigger Conditions | Applicable Scenarios | Explanations                                                                                                                                                                                                                                                                        |
|-----------|--------------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TableRead | None               | Any scenario         | <p>It is generally considered as the least efficient operator that scans table data directly from the TiKV layer. It can be selected only if there is a range query on the <code>_tidb_rowid</code> column, or if there are no other operators for accessing tables to be read.</p> |

| Operator  | Trigger Conditions                         | Applicable Scenarios                                        | Explanations                                                                                                                                               |
|-----------|--------------------------------------------|-------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TableRead | A table has a replica on the TiFlash node. | There are fewer columns to read, but many rows to evaluate. | Tiflash is column-based storage. If you need to calculate a small number of columns and a large number of rows, it is recommended to choose this operator. |

| Operator  | Trigger Conditions                                                                                       | Applicable Scenarios                                                                                           | Explanations                                                                               |
|-----------|----------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| IndexRead | A table has one or more indexes, and the columns needed for the calculation are included in the indexes. | When there is a smaller range query on the indexes, or when there is an order requirement for indexed columns. | When multiple indexes exist, a reason-able index is selected based on the cost estimation. |

| Operator               | Trigger Conditions                                                                                                  | Applicable Scenarios                                                                                                                                                                         | Explanations                                                                                                                                                                                         |
|------------------------|---------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IndexLookupTableReader | Same as has one or more indexes, and the columns needed for calculation are not completely included in the indexes. | IndexReader.index does not completely cover calculated columns, TiDB needs to retrieve rows from a table after reading indexes. There is an extra cost compared to the IndexReader operator. | Since the dexReader.index does not completely cover calculated columns, TiDB needs to retrieve rows from a table after reading indexes. There is an extra cost compared to the IndexReader operator. |

**Note:**

The TableReader operator is based on the `_tidb_rowid` column index, and TiFlash uses a column storage index, so the selection of index is the selection

of an operator for accessing tables.

## Index selection rules

TiDB selects indexes based on rules or cost. The based rules include pre-rules and skyline-pruning. When selecting an index, TiDB tries the pre-rule first. If an index satisfies a pre-rule, TiDB directly selects this index. Otherwise, TiDB uses skyline-pruning to exclude unsuitable indexes, and then selects the index with the lowest cost based on the cost estimation of each operator that accesses tables.

### Rule-based selection

#### Pre-rules

TiDB uses the following heuristic pre-rules to select indexes:

- Rule 1: If an index satisfies “unique index with full match + no need to retrieve rows from a table (which means that the plan generated by the index is the `IndexReader` operator)”, TiDB directly selects this index.
- Rule 2: If an index satisfies “unique index with full match + the need to retrieve rows from a table (which means that the plan generated by the index is the `IndexReader` operator)”, TiDB selects the index with the smallest number of rows to be retrieved from a table as a candidate index.
- Rule 3: If an index satisfies “ordinary index + no need to retrieve rows from a table + the number of rows to be read is less than the value of a certain threshold”, TiDB selects the index with the smallest number of rows to be read as a candidate index.
- Rule 4: If only one candidate index is selected based on rule 2 and 3, select this candidate index. If two candidate indexes are respectively selected based on rule 2 and 3, select the index with the smaller number of rows to be read (the number of rows with index + the number of rows to be retrieved from a table).

The “index with full match” in the above rules means each indexed column has the equal condition. When executing the `EXPLAIN FORMAT = 'verbose' ...` statement, if the pre-rules match an index, TiDB outputs a NOTE-level warning indicating that the index matches the pre-rule.

In the following example, because the index `idx_b` meets the condition “unique index with full match + the need to retrieve rows from a table” in rule 2, TiDB selects the index `idx_b` as the access path, and `SHOW WARNING` returns a note indicating that the index `idx_b` matches the pre-rule.

```
mysql> CREATE TABLE t(a INT PRIMARY KEY, b INT, c INT, UNIQUE INDEX idx_b(b
 ↪));
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> EXPLAIN FORMAT = 'verbose' SELECT b, c FROM t WHERE b = 3 OR b = 6;
```

```
+----+-----+-----+-----+
| id | estRows | estCost | task | access object | operator |
+----+-----+-----+-----+
| 1 | 1 | 8.80 | root | table:t, index:idx_b(b) | keep |
| | | | | order:false, desc:false | |
+----+-----+-----+-----+
```

1 row in set, 1 warning (0.00 sec)

```
mysql> SHOW WARNINGS;
```

```
+----+-----+-----+
| Level | Code | Message
+----+-----+-----+
| Note | 1105 | unique index idx_b of t is selected since the path only has
| | | point ranges with double scan |
+----+-----+-----+
```

1 row in set (0.00 sec)

### Skyline-pruning

Skyline-pruning is a heuristic filtering rule for indexes, which can reduce the probability of wrong index selection caused by wrong estimation. To judge an index, the following three dimensions are needed:

- How many access conditions are covered by the indexed columns. An “access condition” is a where condition that can be converted to a column range. And the more access conditions an indexed column set covers, the better it is in this dimension.
- Whether it needs to retrieve rows from a table when you select the index to access the table (that is, the plan generated by the index is IndexReader operator or In-

dexLookupReader operator). Indexes that do not retrieve rows from a table are better on this dimension than indexes that do. If both indexes need TiDB to retrieve rows from the table, compare how many filtering conditions are covered by the indexed columns. Filtering conditions mean the `where` condition that can be judged based on the index. If the column set of an index covers more access conditions, the smaller the number of retrieved rows from a table, and the better the index is in this dimension.

- Select whether the index satisfies a certain order. Because index reading can guarantee the order of certain column sets, indexes that satisfy the query order are superior to indexes that do not satisfy on this dimension.

For these three dimensions above, if the index `idx_a` performs no worse than the index `idx_b` in all three dimensions and performs better than `idx_b` in one dimension, then `idx_a` is preferred. When executing the `EXPLAIN FORMAT = 'verbose' ...` statement, if skyline-pruning excludes some indexes, TiDB outputs a NOTE-level warning listing the remaining indexes after the skyline-pruning exclusion.

In the following example, the indexes `idx_b` and `idx_e` are both inferior to `idx_b_c`, so they are excluded by skyline-pruning. The returned result of `SHOW WARNING` displays the remaining indexes after skyline-pruning.

```
mysql> CREATE TABLE t(a INT PRIMARY KEY, b INT, c INT, d INT, e INT, INDEX
 ↪ idx_b(b), INDEX idx_b_c(b, c), INDEX idx_e(e));
Query OK, 0 rows affected (0.01 sec)

mysql> EXPLAIN FORMAT = 'verbose' SELECT * FROM t WHERE b = 2 AND c > 4;
+--+
| id | estRows | estCost | task | access object
| | operator info |
+--+
| IndexLookUp_10 | 33.33 | 738.29 | root |
| | | | |
| | -IndexRangeScan_8(Build) | 33.33 | 2370.00 | cop[tikv] | table:t, index
| | :idx_b_c(b, c) | range:(2 4,2 +inf], keep order:false, stats:pseudo |
| | -TableRowIDScan_9(Probe) | 33.33 | 2370.00 | cop[tikv] | table:t
| | | keep order:false, stats:pseudo |
+--+
| | | | |
3 rows in set, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS;
+---+
| Level | Code | Message
+---+
| Note | 1105 | [t, idx_b_c] remain after pruning paths for t given Prop{
 ↵ SortItems: [], TaskTp: rootTask} |
+---+
1 row in set (0.00 sec)
```

### Cost estimation-based selection

After using the skyline-pruning rule to rule out inappropriate indexes, the selection of indexes is based entirely on the cost estimation. The cost estimation of accessing tables requires the following considerations:

- The average length of each row of the indexed data in the storage engine.
- The number of rows in the query range generated by the index.
- The cost for retrieving rows from a table.
- The number of ranges generated by index during the query execution.

According to these factors and the cost model, the optimizer selects an index with the lowest cost to access the table.

### Common tuning problems with cost estimation based selection

1. The estimated number of rows is not accurate?

This is usually due to stale or inaccurate statistics. You can re-execute the `analyze ↵ table` statement or modify the parameters of the `analyze table` statement.

2. Statistics are accurate, and reading from TiFlash is faster, but why does the optimizer choose to read from TiKV?

At present, the cost model of distinguishing TiFlash from TiKV is still rough. You can decrease the value of `tidb_opt_seek_factor` parameter, then the optimizer prefers to choose TiFlash.

3. The statistics are accurate. Index A needs to retrieve rows from tables, but it actually executes faster than Index B that does not retrieve rows from tables. Why does the optimizer choose Index B?

In this case, the cost estimation may be too large for retrieving rows from tables. You can decrease the value of `tidb_opt_network_factor` parameter to reduce the cost of retrieving rows from tables.

### Control index selection

The index selection can be controlled by a single query through [Optimizer Hints](#).

- `USE_INDEX / IGNORE_INDEX` can force the optimizer to use / not use certain indexes.
- `READ_FROM_STORAGE` can force the optimizer to choose the TiKV / TiFlash storage engine for certain tables to execute queries.

#### 8.3.3.3.3 Introduction to Statistics

TiDB uses statistics to decide [which index to choose](#). The `tidb_analyze_version` variable controls the statistics collected by TiDB. Currently, two versions of statistics are supported: `tidb_analyze_version = 1` and `tidb_analyze_version = 2`. In versions before v5.1.0, the default value of this variable is 1. In v5.1, v5.2, and v5.3, the default value of this variable is 2, which serves as an experimental feature.

##### Note:

When `tidb_analyze_version = 2`, if memory overflow occurs after `ANALYZE` is executed, you need to set `tidb_analyze_version = 1` and perform one of the following operations:

- If the `ANALYZE` statement is executed manually, manually analyze every table to be analyzed.

```
sql select distinct(concat('ANALYZE ',table_schema, '.',
→ table_name,';'))from information_schema.tables, mysql
→ .stats_histograms where stats_ver = 2 and table_id =
→ tidb_table_id ;
```

- If TiDB automatically executes the `ANALYZE` statement because the auto-analysis has been enabled, execute the following statement that generates the `DROP STATS` statement:

```
sql select distinct(concat('DROP STATS ',table_schema,
→ '.', table_name,';'))from information_schema.tables,
→ mysql.stats_histograms where stats_ver = 2 and table_id =
→ tidb_table_id ;
```

These two versions include different information in TiDB:

|             | Version | Version |
|-------------|---------|---------|
| Information | 2       |         |
| The         | ✓       | ✓       |
| to-         |         |         |
| tal         |         |         |
| num-        |         |         |
| ber         |         |         |
| of          |         |         |
| rows        |         |         |
| in          |         |         |
| the         |         |         |
| ta-         |         |         |
| ble         |         |         |
| Column      | ✓       | ✗       |
| Count-      |         |         |
| Min         |         |         |
| Sketch      |         |         |
| Index       | ✓       | ✗       |
| Count-      |         |         |
| Min         |         |         |
| Sketch      |         |         |
| Column      | ✓       |         |
| Top-        | (Main-  |         |
| N           | te-     |         |
|             | nance   |         |
|             | meth-   |         |
|             | ods     |         |
|             | and     |         |
|             | pre-    |         |
|             | ci-     |         |
|             | sion    |         |
|             | are     |         |
|             | im-     |         |
|             | proved) |         |

|                  | Version         | Version                                        |
|------------------|-----------------|------------------------------------------------|
| Information      | 2               |                                                |
| Index            | ✓               | ✓                                              |
| Top-N            | (In-sufficient) | (Main-tenance methods and pre-conditions)      |
|                  |                 | might cause inaccuracy)                        |
| Column histogram | ✓               | (The histogram does not include Top-N values.) |

|             | Version | Version |
|-------------|---------|---------|
| Information | 2       |         |
| Index       | ✓       | ✓       |
| his-        |         | (The    |
| togram      |         | his-    |
|             |         | togram  |
|             |         | buck-   |
|             |         | ets     |
|             |         | record  |
|             |         | the     |
|             |         | num-    |
|             |         | ber     |
|             |         | of      |
|             |         | dif-    |
|             |         | fer-    |
|             |         | ent     |
|             |         | val-    |
|             |         | ues     |
|             |         | in      |
|             |         | each    |
|             |         | bucket, |
|             |         | and     |
|             |         | the     |
|             |         | his-    |
|             |         | togram  |
|             |         | does    |
|             |         | not     |
|             |         | in-     |
|             |         | clude   |
|             |         | Top-    |
|             |         | N       |
|             |         | val-    |
|             |         | ues.)   |
| The         | ✓       | ✓       |
| num-        |         |         |
| ber         |         |         |
| of          |         |         |
| NULL        |         |         |
| ↪ s         |         |         |
| in          |         |         |
| the         |         |         |
| col-        |         |         |
| umn         |         |         |

|                                     | Version | Version |
|-------------------------------------|---------|---------|
| Information                         | 2       |         |
| The number of NULL → s in the index | ✓       | ✓       |
| The average length of columns       | ✓       | ✓       |
| The average length of indexes       | ✓       | ✓       |

Compared to Version 1, Version 2 statistics avoids the potential inaccuracy caused by hash collision when the data volume is huge. It also maintains the estimate precision in most scenarios.

This document briefly introduces the histogram, Count-Min Sketch, and Top-N, and details the collection and maintenance of statistics.

### Histogram

A histogram is an approximate representation of the distribution of data. It divides the entire range of values into a series of buckets, and uses simple data to describe each bucket, such as the number of values falling in the bucket. In TiDB, an equal-depth histogram is created for the specific columns of each table. The equal-depth histogram can be used to estimate the interval query.

Here “equal-depth” means that the number of values falling into each bucket is as equal as possible. For example, for a given set {1.6, 1.9, 1.9, 2.0, 2.4, 2.6, 2.7, 2.7, 2.8, 2.9, 3.4, 3.5}, you want to generate 4 buckets. The equal-depth histogram is as follows. It contains

four buckets [1.6, 1.9], [2.0, 2.6], [2.7, 2.8], [2.9, 3.5]. The bucket depth is 3.

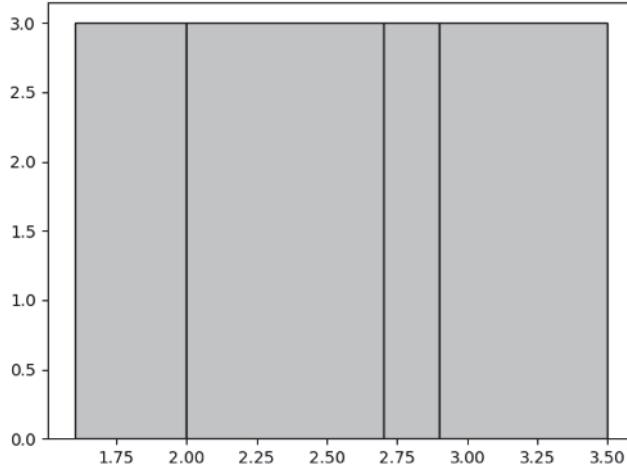


Figure 67: Equal-depth Histogram Example

For details about the parameter that determines the upper limit to the number of histogram buckets, refer to [Manual Collection](#). When the number of buckets is larger, the accuracy of the histogram is higher; however, higher accuracy is at the cost of the usage of memory resources. You can adjust this number appropriately according to the actual scenario.

#### Count-Min Sketch

Count-Min Sketch is a hash structure. When an equivalence query contains `a = 1` or `IN` query (for example, `a in (1, 2, 3)`), TiDB uses this data structure for estimation.

A hash collision might occur since Count-Min Sketch is a hash structure. In the `EXPLAIN` statement, if the estimate of the equivalent query deviates greatly from the actual value, it can be considered that a larger value and a smaller value have been hashed together. In this case, you can take one of the following ways to avoid the hash collision:

- Modify the `WITH NUM TOPN` parameter. TiDB stores the high-frequency (top x) data separately, with the other data stored in Count-Min Sketch. Therefore, to prevent a larger value and a smaller value from being hashed together, you can increase the value of `WITH NUM TOPN`. In TiDB, its default value is 20. The maximum value is 1024. For more information about this parameter, see [Full Collection](#).
- Modify two parameters `WITH NUM CMSKETCH DEPTH` and `WITH NUM CMSKETCH WIDTH`. Both affect the number of hash buckets and the collision probability. You can increase the values of the two parameters appropriately according to the actual scenario to reduce the probability of hash collision, but at the cost of higher memory usage of statistics. In TiDB, the default value of `WITH NUM CMSKETCH DEPTH` is 5, and the

default value of `WITH NUM CMSKETCH WIDTH` is 2048. For more information about the two parameters, see [Full Collection](#).

### Top-N values

Top-N values are values with the top N occurrences in a column or index. TiDB records the values and occurrences of Top-N values.

#### Collect statistics

#### Manual collection

You can run the `ANALYZE` statement to collect statistics.

#### Note:

The execution time of `ANALYZE TABLE` in TiDB is longer than that in MySQL or InnoDB. In InnoDB, only a small number of pages are sampled, while in TiDB a comprehensive set of statistics is completely rebuilt. Scripts that were written for MySQL may naively expect `ANALYZE TABLE` will be a short-lived operation.

For quicker analysis, you can set `tidb_enable_fast_analyze` to 1 to enable the Quick Analysis feature. The default value for this parameter is 0.

After Quick Analysis is enabled, TiDB randomly samples approximately 10,000 rows of data to build statistics. Therefore, in the case of uneven data distribution or a relatively small amount of data, the accuracy of statistical information is relatively poor. It might lead to poor execution plans, such as choosing the wrong index. If the execution time of the normal `ANALYZE` → statement is acceptable, it is recommended to disable the Quick Analysis feature.

`tidb_enable_fast_analyze` is an experimental feature, which currently **does not match exactly** with the statistical information of `tidb_analyze_version=2`. Therefore, you need to set the value of `tidb_analyze_version` to 1 when `tidb_enable_fast_analyze` is enabled.

### Full collection

You can perform full collection using the following syntax.

- To collect statistics of all the tables in `TableNameList`:

```
ANALYZE TABLE TableNameList [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|
 ↪ CMSKETCH WIDTH] | [WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];
```

- `WITH NUM BUCKETS` specifies the maximum number of buckets in the generated histogram.
- `WITH NUM TOPN` specifies the maximum number of the generated TOPNs.
- `WITH NUM CMSKETCH DEPTH` specifies the depth of the CM Sketch.
- `WITH NUM CMSKETCH WIDTH` specifies the width of the CM Sketch.
- `WITH NUM SAMPLES` specifies the number of samples.
- `WITH FLOAT_NUM SAMPLERATE` specifies the sampling rate.

`WITH NUM SAMPLES` and `WITH FLOAT_NUM SAMPLERATE` correspond to two different algorithms of collecting samples.

- `WITH NUM SAMPLES` specifies the size of the sampling set, which is implemented in the reservoir sampling method in TiDB. When a table is large, it is not recommended to use this method to collect statistics. Because the intermediate result set of the reservoir sampling contains redundant results, it causes additional pressure on resources such as memory.
- `WITH FLOAT_NUM SAMPLERATE` is a sampling method introduced in v5.3.0. With the value range  $(0, 1]$ , this parameter specifies the sampling rate. It is implemented in the way of Bernoulli sampling in TiDB, which is more suitable for sampling larger tables and performs better in collection efficiency and resource usage.

Before v5.3.0, TiDB uses the reservoir sampling method to collect statistics. Since v5.3.0, the TiDB Version 2 statistics uses the Bernoulli sampling method to collect statistics by default. To re-use the reservoir sampling method, you can use the `WITH NUM SAMPLES` statement.

#### Note:

The current sampling rate is calculated based on an adaptive algorithm. When you can observe the number of rows in a table using `SHOW STATS_META`, you can use this number of rows to calculate the sampling rate corresponding to 100,000 rows. If you cannot observe this number, you can use the `TABLE_KEYS` column in the `TABLE_STORAGE_STATS` table as another reference to calculate the sampling rate.

Normally, `STATS_META` is more credible than `TABLE_KEYS`. However, after importing data through the methods like `TiDB Lightning`, the result of `STATS_META` is 0. To handle this situation, you can use `TABLE_KEYS` to calculate the sampling rate when the result of `STATS_META` is much smaller than the result of `TABLE_KEYS`.

The following syntax collects statistics for some columns in the `TableName` table:

```
ANALYZE TABLE TableName COLUMNS ColumnNameList [WITH NUM BUCKETS|TOPN|
 ↪ CMSKETCH DEPTH|CMSKETCH WIDTH] | [WITH NUM SAMPLES|WITH FLOATNUM
 ↪ SAMPLERATE] ;
```

This syntax collects statistics on the specified columns and indexes, as well as the statistics on the columns involved in the extended statistics. If the number of columns in the table is large, the columns that require statistics might only be a small subset of the table. In this situation, this syntax can greatly reduce the stress of collecting statistics.

#### Note:

- The syntax above takes effect only when `tidb_analyze_version = 2`.
- In the syntax above, `ColumnNameList` cannot be empty.
- The syntax above collects the full statistics of a table. For example, after collecting the statistics of column a and column b, to further collect the statistics of column c, you need to specify all three columns in the statement `ANALYZE table t columns a, b, c` rather than specifying only the additional column c like `ANALYZE TABLE t COLUMNS c`.

- To collect statistics of the index columns on all `IndexNameLists` in `TableName`:

```
ANALYZE TABLE TableName INDEX [IndexNameList] [WITH NUM BUCKETS|TOPN|
 ↪ CMSKETCH DEPTH|CMSKETCH WIDTH|SAMPLES] | [WITH FLOATNUM SAMPLERATE
 ↪] ;
```

The statement collects statistics of all index columns when `IndexNameList` is empty.

- To collect statistics of partition in all `PartitionNameLists` in `TableName`:

```
ANALYZE TABLE TableName PARTITION PartitionNameList [WITH NUM BUCKETS|
 ↪ TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH] | [WITH NUM SAMPLES|WITH
 ↪ FLOATNUM SAMPLERATE] ;
```

- To collect statistics of some columns for the partitions in all `PartitionNameLists` in `TableName`:

```
ANALYZE TABLE TableName PARTITION PartitionNameList COLUMNS
 ↪ ColumnNameList [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH
 ↪ WIDTH] | [WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE] ;
```

- To collect statistics of index columns for the partitions in all `PartitionNameLists` in `TableName`:

```
ANALYZE TABLE TableName PARTITION PartitionNameList INDEX [
 ↳ IndexNameList] [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH
 ↳ WIDTH] | [WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE] ;
```

**Note:**

To ensure that the statistical information before and after the collection is consistent, when you set `tidb_analyze_version=2`, `ANALYZE TABLE TableName INDEX` will also collect statistics of the whole table instead of the given index.

### Incremental collection

To improve the speed of analysis after full collection, incremental collection could be used to analyze the newly added sections in monotonically non-decreasing columns such as time columns.

**Note:**

- Currently, the incremental collection is only provided for index.
- When using the incremental collection, you must ensure that only `INSERT` operations exist on the table, and that the newly inserted value on the index column is monotonically non-decreasing. Otherwise, the statistical information might be inaccurate, affecting the TiDB optimizer to select an appropriate execution plan.

You can perform incremental collection using the following syntax.

- To incrementally collect statistics for index columns in all `IndexNameLists` in `TableName`:

```
ANALYZE INCREMENTAL TABLE TableName INDEX [IndexNameList] [WITH NUM
 ↳ BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH] | [WITH NUM SAMPLES|
 ↳ WITH FLOATNUM SAMPLERATE] ;
```

- To incrementally collect statistics of index columns for partitions in all `PartitionNameLists` in `TableName`:

```
ANALYZE INCREMENTAL TABLE TableName PARTITION PartitionNameList INDEX [
 ↪ IndexNameList] [WITH NUM BUCKETS|TOPN|CMSSKETCH DEPTH|CMSSKETCH
 ↪ WIDTH] | [WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE] ;
```

## Automatic update

For the `INSERT`, `DELETE`, or `UPDATE` statements, TiDB automatically updates the number of rows and updated rows. TiDB persists this information regularly and the update cycle is  $20 * \text{stats-lease}$ . The default value of `stats-lease` is 3s. If you specify the value as 0, it does not update automatically.

Three system variables related to automatic update of statistics are as follows:

| System                                    | Vari-         | Default     |
|-------------------------------------------|---------------|-------------|
| able                                      | Value         | Description |
| <code>tidb_auto_analyze_ratio</code>      | → thresh-     |             |
|                                           | old           |             |
|                                           | value         |             |
|                                           | of            |             |
|                                           | au-           |             |
|                                           | to-           |             |
|                                           | matic         |             |
|                                           | up-           |             |
|                                           | date          |             |
| <code>tidb_auto_analyze_start_time</code> | →     → start |             |
|                                           | → +00:00:00   |             |
|                                           | → in a        |             |
|                                           | day           |             |
|                                           | when          |             |
|                                           | TiDB          |             |
|                                           | can           |             |
|                                           | per-          |             |
|                                           | form          |             |
|                                           | au-           |             |
|                                           | to-           |             |
|                                           | matic         |             |
|                                           | up-           |             |
|                                           | date          |             |

---

| System Variable                         | Default Value       | Description                                     |
|-----------------------------------------|---------------------|-------------------------------------------------|
| <code>tidb_auto_analyze_end_time</code> | <code>+00:00</code> | in a day when TiDB can perform automatic update |

---

When the ratio of the number of modified rows to the total number of rows of `tbl` in a table is greater than `tidb_auto_analyze_ratio`, and the current time is between `tidb_auto_analyze_start_time` and `tidb_auto_analyze_end_time`, TiDB executes the `ANALYZE TABLE tbl` statement in the background to automatically update the statistics of this table.

#### Note:

Currently, the automatic update does not record the configuration items input at manual `ANALYZE`. Therefore, when you use the `WITH` syntax to control the collecting behavior of `ANALYZE`, you need to manually set scheduled tasks to collect statistics.

Before v5.0, when the query is executed, TiDB collects feedback with the probability of `feedback-probability` and uses it to update the histogram and Count-Min Sketch. **In v5.0, this feature is disabled by default, and it is not recommended to enable this feature.**

#### Control `ANALYZE` concurrency

When you run the `ANALYZE` statement, you can adjust the concurrency using the following parameters, to control its effect on the system.

`tidb_build_stats_concurrency`

Currently, when you run the `ANALYZE` statement, the task is divided into multiple small tasks. Each task only works on one column or index. You can use the `tidb_build_stats_concurrency` parameter to control the number of simultaneous tasks. The default value is 4.

#### `tidb_distsql_scan_concurrency`

When you analyze regular columns, you can use the `tidb_distsql_scan_concurrency` parameter to control the number of Region to be read at one time. The default value is 15.

#### `tidb_index_serial_scan_concurrency`

When you analyze index columns, you can use the `tidb_index_serial_scan_concurrency` parameter to control the number of Region to be read at one time. The default value is 1.

#### View ANALYZE state

When executing the `ANALYZE` statement, you can view the current state of `ANALYZE` using the following SQL statement:

```
SHOW ANALYZE STATUS [ShowLikeOrWhere]
```

This statement returns the state of `ANALYZE`. You can use `ShowLikeOrWhere` to filter the information you need.

Currently, the `SHOW ANALYZE STATUS` statement returns the following 7 columns:

| Syntax | Element        | Description                                                                                 |
|--------|----------------|---------------------------------------------------------------------------------------------|
|        | table_schema   | The database name                                                                           |
|        | table_name     | The table name                                                                              |
|        | partition_name | The partition name                                                                          |
|        | job_info       | The task information.<br>The element includes index names when index analysis is performed. |
|        | row_count      | The number of rows that have been analyzed                                                  |
|        | start_time     | The time at which the task starts                                                           |

| Syntax Element | Description                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------------------|
| state          | The state of a task, including <code>pending</code> , <code>running</code> , <code>finished</code> , and <code>failed</code> |

## View statistics

You can view the statistics status using the following statements.

### Metadata of tables

You can use the `SHOW STATS_META` statement to view the total number of rows and the number of updated rows.

The syntax of `ShowLikeOrWhereOpt` is as follows:

```
SHOW STATS_META [ShowLikeOrWhere]
```

Currently, the `SHOW STATS_META` statement returns the following 6 columns:

| Syntax Element              | Description                 |
|-----------------------------|-----------------------------|
| <code>db_name</code>        | The database name           |
| <code>table_name</code>     | The table name              |
| <code>partition_name</code> | The partition name          |
| <code>update_time</code>    | The time of the update      |
| <code>modify_count</code>   | The number of modified rows |
| <code>row_count</code>      | The total number of rows    |

### Note:

When TiDB automatically updates the total number of rows and the number of modified rows according to DML statements, `update_time` is also updated. Therefore, `update_time` does not necessarily indicate the last time when the `ANALYZE` statement is executed.

## Health state of tables

You can use the `SHOW STATS_HEALTHY` statement to check the health state of tables and roughly estimate the accuracy of the statistics. When `modify_count >= row_count`, the

health state is 0; when `modify_count < row_count`, the health state is  $(1 - \text{modify\_count} / \text{row\_count}) * 100$ .

The synopsis of `SHOW STATS_HEALTHY` is:



Figure 68: ShowStatsHealthy

and the synopsis of the `ShowLikeOrWhereOpt` part is:

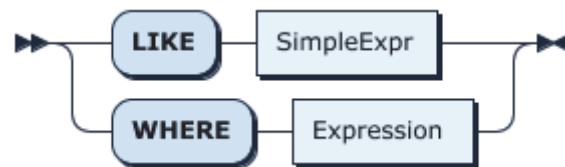


Figure 69: ShowLikeOrWhereOpt

Currently, the `SHOW STATS_HEALTHY` statement returns the following 4 columns:

| Syntax Element              | Description                |
|-----------------------------|----------------------------|
| <code>db_name</code>        | The database name          |
| <code>table_name</code>     | The table name             |
| <code>partition_name</code> | The partition name         |
| <code>healthy</code>        | The health state of tables |

Metadata of columns

You can use the `SHOW STATS_HISTOGRAMS` statement to view the number of different values and the number of NULL in all the columns.

Syntax as follows:

```
SHOW STATS_HISTOGRAMS [ShowLikeOrWhere]
```

This statement returns the number of different values and the number of NULL in all the columns. You can use `ShowLikeOrWhere` to filter the information you need.

Currently, the `SHOW STATS_HISTOGRAMS` statement returns the following 10 columns:

| Syntax Element       | Description       |
|----------------------|-------------------|
| <code>db_name</code> | The database name |

---

| Syntax                      |                                                                                                                                                  |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Element                     | Description                                                                                                                                      |
| <code>table_name</code>     | The table name                                                                                                                                   |
| $\hookrightarrow$           |                                                                                                                                                  |
| <code>partition_name</code> | The partition name                                                                                                                               |
| $\hookrightarrow$           |                                                                                                                                                  |
| <code>column_name</code>    | The column name (when <code>is_index</code> is 0) or the index name (when <code>is_index</code> is 1)                                            |
| $\hookrightarrow$           |                                                                                                                                                  |
| <code>is_index</code>       | Whether it is an index column or not                                                                                                             |
| <code>update_time</code>    | The time of the update                                                                                                                           |
| $\hookrightarrow$           |                                                                                                                                                  |
| <code>distinct_count</code> | The number of different values                                                                                                                   |
| <code>null_count</code>     | The number of NULL                                                                                                                               |
| $\hookrightarrow$           |                                                                                                                                                  |
| <code>avg_col_size</code>   | The average length of columns                                                                                                                    |
| $\hookrightarrow$           |                                                                                                                                                  |
| <code>correlation</code>    | The Pearson correlation coefficient of the column and the integer primary key, which indicates the degree of association between the two columns |

---

### Buckets of histogram

You can use the `SHOW STATS_BUCKETS` statement to view each bucket of the histogram.

The syntax is as follows:

```
SHOW STATS_BUCKETS [ShowLikeOrWhere]
```

The diagram is as follows:



Figure 70: SHOW STATS\_BUCKETS

This statement returns information about all the buckets. You can use `ShowLikeOrWhere` to filter the information you need.

Currently, the `SHOW STATS_BUCKETS` statement returns the following 11 columns:

| Syntax                      |                                                                                                       |
|-----------------------------|-------------------------------------------------------------------------------------------------------|
| Element                     | Description                                                                                           |
| <code>db_name</code>        | The database name                                                                                     |
| <code>table_name</code>     | The table name                                                                                        |
| <code>partition_name</code> | The partition name                                                                                    |
| <code>column_name</code>    | The column name (when <code>is_index</code> is 0) or the index name (when <code>is_index</code> is 1) |
| <code>is_index</code>       | Whether it is an index column or not                                                                  |
| <code>bucket_id</code>      | The ID of a bucket                                                                                    |
| <code>count</code>          | The number of all the values that falls on the bucket and the previous buckets                        |
| <code>repeats</code>        | The occurrence number of the maximum value                                                            |
| <code>lower_bound</code>    | The minimum value                                                                                     |
| <code>upper_bound</code>    | The maximum value                                                                                     |

| Syntax<br>Element | Description                                                                                                                                                   |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ndv               | The number of different values in the bucket.<br>When <code>tidb_analyze_version</code> $\hookrightarrow = 1$ , ndv is always 0, which has no actual meaning. |

### Top-N information

You can use the `SHOW STATS_TOPN` statement to view the Top-N information currently collected by TiDB.

The syntax is as follows:

```
SHOW STATS_TOPN [ShowLikeOrWhere];
```

Currently, the `SHOW STATS_TOPN` statement returns the following 7 columns:

| Syntax<br>Ele-<br>ment      | Description        |
|-----------------------------|--------------------|
| <code>db_name</code>        | The database name  |
| <code>table_name</code>     | The table name     |
| <code>partition_name</code> | The partition name |

---

| Syntax             |             |
|--------------------|-------------|
| Ele-               |             |
| ment               | Description |
| <b>column_name</b> |             |
| ↪ col-             |             |
| umn                |             |
| name               |             |
| (when              |             |
| <b>is_index</b>    |             |
| ↪                  |             |
| is 0)              |             |
| or                 |             |
| the                |             |
| index              |             |
| name               |             |
| (when              |             |
| <b>is_index</b>    |             |
| ↪                  |             |
| is 1)              |             |
| <b>is_index</b>    | Whether     |
| ↪ it is            |             |
| an                 |             |
| index              |             |
| col-               |             |
| umn                |             |
| or                 |             |
| not                |             |
| <b>value</b>       | The         |
| ↪ value            |             |
| of                 |             |
| this               |             |
| col-               |             |
| umn                |             |
| <b>count</b>       | How         |
| ↪ many             |             |
| times              |             |
| the                |             |
| value              |             |
| ap-                |             |
| pears              |             |

---

## Delete statistics

You can run the `DROP STATS` statement to delete statistics.

Syntax as follows:

```
DROP STATS TableName
```

The statement deletes statistics of all the tables in TableName.

Import and export statistics

Export statistics

The interface to export statistics is as follows:

- To obtain the JSON format statistics of the \${table\_name} table in the \${db\_name} database:

```
http://${tidb-server-ip}:${tidb-server-status-port}/stats/dump/${
 ↳ db_name}/${table_name}
```

For example:

```
curl -s http://127.0.0.1:10080/stats/dump/test/t1 -o /tmp/t1.json
```

- To obtain the JSON format statistics of the \${table\_name} table in the \${db\_name} database at specific time:

```
http://${tidb-server-ip}:${tidb-server-status-port}/stats/dump/${
 ↳ db_name}/${table_name}/${yyyyMMddHHmmss}
```

Import statistics

#### Note:

When you start the MySQL client, use the --local-infile=1 option.

Generally, the imported statistics refer to the JSON file obtained using the export interface.

Syntax:

```
LOAD STATS 'file_name'
```

`file_name` is the file name of the statistics to be imported.

See also

- LOAD STATS
- DROP STATS

#### 8.3.3.3.4 Wrong Index Solution

If you find that the execution speed of some query does not reach the expectation, the optimizer might choose the wrong index to run the query.

You can first view the [health state of tables](#) in the statistics, and then solve this issue according to the different health states.

Low health state

The low health state means TiDB has not performed the `ANALYZE` statement for a long time. You can update the statistics by running the `ANALYZE` command. After the update, if the optimizer still uses the wrong index, refer to the next section.

Near 100% health state

The near 100% health state suggests that the `ANALYZE` statement is just completed or was completed a short time ago. In this case, the wrong index issue might be related to TiDB's estimation logic for the number of rows.

For equivalence queries, the cause might be Count-Min Sketch. You can check whether Count-Min Sketch is the cause and take corresponding solutions.

If the cause above does not apply to your problem, you can force-select indexes by using the `USE_INDEX` or `use index` optimizer hint (see [USE\\_INDEX](#) for details). Also, you can change the query behavior by using [SQL Plan Management](#) in a non-intrusive way.

### Other situations

Apart from the aforementioned situations, the wrong index issue might also be caused by data updates which renders all the indexes no longer applicable. In such cases, you need to perform analysis on the conditions and data distribution to see whether new indexes can speed up the query. If so, you can add new indexes by running the `ADD INDEX` command.

### 8.3.3.3.5 Distinct Optimization

This document introduces the `distinct` optimization in the TiDB query optimizer, including `SELECT DISTINCT` and `DISTINCT` in the aggregate functions.

DISTINCT modifier in SELECT statements

The DISTINCT modifier specifies removal of duplicate rows from the result set. SELECT  
→ DISTINCT is transformed to GROUP BY, for example:

```
mysql> explain SELECT DISTINCT a from t;
+---+
| id | estRows | task | access object | operator info
| | | | |
+---+
+---+
```

```

| HashAgg_6 | 2.40 | root | | group by:test.t.a
| ↵ , funcs:firstrow(test.t.a)->test.t.a |
| -TableReader_11 | 3.00 | root | | data:
| ↵ TableFullScan_10
| -TableFullScan_10 | 3.00 | cop[tikv] | table:t | keep order:false
| ↵ , stats:pseudo |
+--+
| ↵ -----+-----+-----+
| ↵
3 rows in set (0.00 sec)

```

DISTINCT option in aggregate functions

Usually, aggregate functions with the DISTINCT option is executed in the TiDB layer in a single-threaded execution model.

The `tidb_opt_distinct_agg_push_down` system variable or the `distinct-agg-push-down` configuration item in TiDB controls whether to rewrite the distinct aggregate queries and push them to the TiKV/TiFlash Coprocessor.

Take the following queries as an example of this optimization. `tidb_opt_distinct_agg_push_down` is disabled by default, which means the aggregate functions are executed in the TiDB layer. After enabling this optimization by setting its value to 1, the `distinct` a part of `count(distinct a)` is pushed to TiKV/TiFlash Coprocessor: there is a HashAgg\_5 to remove the duplicated values on column a in the TiKV Coprocessor. It might reduce the computation overhead of HashAgg\_8 in the TiDB layer.

```

mysql> desc select count(distinct a) from test.t;
+--+
| id | estRows | task | access object | operator info
| | | |
+--+
| StreamAgg_6 | 1.00 | root | | funcs:count(
| ↵ distinct test.t.a)->Column#4 |
| -TableReader_10 | 10000.00 | root | | data:
| ↵ TableFullScan_9
| -TableFullScan_9 | 10000.00 | cop[tikv] | table:t | keep order:false
| ↵ , stats:pseudo |
+--+
| ↵ -----+-----+-----+
| ↵
3 rows in set (0.01 sec)

```

```

mysql> set session tidb_opt_distinct_agg_push_down = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> desc select count(distinct a) from test.t;
+----+-----+-----+-----+-----+
| id | estRows | task | access object | operator info |
+----+-----+-----+-----+-----+
HashAgg_8	1.00	root	funcs:count(
> distinct test.t.a)->Column#3				
-TableReader_9	1.00	root	data:HashAgg_5	
>			group by:test.	
> HashAgg_5	1.00	cop[tikv]		
> t.a,			table:t	
> TableFullScan_7	10000.00	cop[tikv]	keep order:	
> false, stats:pseudo				
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

#### 8.3.3.4 SQL Prepare Execution Plan Cache

TiDB supports execution plan caching for `Prepare` and `Execute` queries. This includes both forms of prepared statements:

- Using the `COM_STMT_PREPARE` and `COM_STMT_EXECUTE` protocol features.
- Using the SQL statements `PREPARE` and `EXECUTE`.

The TiDB optimizer handles these two types of queries in the same way: when preparing, the parameterized query is parsed into an AST (Abstract Syntax Tree) and cached; in later execution, the execution plan is generated based on the stored AST and specific parameter values.

When the execution plan cache is enabled, in the first execution every `Prepare` statement checks whether the current query can use the execution plan cache, and if the query can use it, then put the generated execution plan into a cache implemented by LRU (Least Recently Used) linked list. In the subsequent `Execute` queries, the execution plan is obtained from the cache and checked for availability. If the check succeeds, the step of generating an execution plan is skipped. Otherwise, the execution plan is regenerated and saved in the cache.

In the current version of TiDB, if a `Prepare` statement meets any of the following conditions, the query or the plan is not cached:

- The query contains SQL statements other than `SELECT`, `UPDATE`, `INSERT`, `DELETE`, `Union`, `Intersect`, and `Except`.
- The query accesses partitioned tables or temporary tables, or a table that contains generated columns.
- The query contains sub-queries, such as `select * from t where a > (select ...)`.
- The query contains the `ignore_plan_cache` hint, such as `select /*+ ignore_plan_cache */ (*) * from t`.
- The query contains variables other than `?` (including system variables or user-defined variables), such as `select * from t where a>? and b>@x`.
- The query contains the functions that cannot be cached: `database()`, `current_user`, `current_role`, `user`, `connection_id`, `last_insert_id`, `row_count`, `version`, and like.
- The query contains `?` after `Limit`, such as `Limit ? and Limit 10, ?`. Such queries are not cached because the specific value of `?` has a great impact on query performance.
- The query contains `?` after `Order By`, such as `Order By ?`. Such queries sort data based on the column specified by `?`. If the queries targeting different columns use the same execution plan, the results will be wrong. Therefore, such queries are not cached. However, if the query is a common one, such as `Order By a+?`, it is cached.
- The query contains `?` after `Group By`, such as `Group By ?`. Such queries group data based on the column specified by `?`. If the queries targeting different columns use the same execution plan, the results will be wrong. Therefore, such queries are not cached. However, if the query is a common one, such as `Group By a+?`, it is cached.
- The query contains `?` in the definition of the `Window Frame` window function, such as `(partition by year order by sale rows ? preceding)`. If `?` appears elsewhere in the window function, the query is cached.
- The query contains parameters for comparing `int` and `string`, such as `c_int >= ?` or `c_int in (?, ?)`, in which `?` indicates the string type, such as `set @x='123'`. To ensure that the query result is compatible with MySQL, parameters need to be adjusted in each query, so such queries are not cached.
- The plan attempts to access `TiFlash`.

The LRU linked list is designed as a session-level cache because `Prepare / Execute` cannot be executed across sessions. Each element of the LRU list is a key-value pair. The value is the execution plan, and the key is composed of the following parts:

- The name of the database where `Execute` is executed
- The identifier of the `Prepare` statement, that is, the name after the `PREPARE` keyword
- The current schema version, which is updated after every successfully executed DDL statement
- The SQL mode when executing `Execute`
- The current time zone, which is the value of the `time_zone` system variable

Any change in the above information (for example, switching databases, renaming `Prepare` statement, executing DDL statements, or modifying the value of SQL mode / `time_zone`), or the LRU cache elimination mechanism causes the execution plan cache miss when executing.

After the execution plan cache is obtained from the cache, TiDB first checks whether the execution plan is still valid. If the current `Execute` statement is executed in an explicit transaction, and the referenced table is modified in the transaction pre-order statement, the cached execution plan accessing this table does not contain the `UnionScan` operator, then it cannot be executed.

After the validation test is passed, the scan range of the execution plan is adjusted according to the current parameter values, and then used to perform data querying.

There are several points worth noting about execution plan caching and query performance:

- The first `Execute` (no cache plan yet) is affected by existing SQL Bindings. Cached plans are not affected by new SQL Bindings. Similarly, cached plans are not affected by changes in statistics, optimization rules, and blacklist pushdown by expressions.
- When restarting a TiDB instance (for example, online rolling upgrade of a TiDB cluster), the `Prepare` statement gets lost. `Prepared Statement not found` is returned if executing `execute stmt....`. To address the error, execute `prepare stmt ...` again.
- Considering that the parameters of `Execute` are different, the execution plan cache prohibits some aggressive query optimization methods that are closely related to specific parameter values to ensure adaptability. This causes that the query plan may not be optimal for certain parameter values. For example, the filter condition of the query is `where a > ? And a < ?`, the parameters of the first `Execute` statement are 2 and 1 respectively. Considering that these two parameters maybe be 1 and 2 in the next execution time, the optimizer does not generate the optimal `TableDual` execution plan that is specific to current parameter values;
- If cache invalidation and elimination are not considered, an execution plan cache is applied to various parameter values, which in theory also results in non-optimal execution plans for certain values. For example, if the filter condition is `where a < ?` and the parameter value used for the first execution is 1, then the optimizer generates the optimal `IndexScan` execution plan and puts it into the cache. In the subsequent executions, if the value becomes 10000, the `TableScan` plan might be the better one. But due to the execution plan cache, the previously generated `IndexScan` is used for execution. Therefore, the execution plan cache is more suitable for application scenarios where the query is simple (the ratio of compilation is high) and the execution plan is relatively fixed.

Currently, the execution plan cache is disabled by default. You can enable this feature by enabling the `prepare-plan-cache` in the TiDB configuration file.

**Note:**

The execution plan cache feature applies only to `Prepare / Execute` queries and does not take effect for normal queries.

After the execution plan cache feature is enabled, you can use the session-level system variable `last_plan_from_cache` to see whether the previous `Execute` statement used the cached execution plan, for example:

```
MySQL [test]> create table t(a int);
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> prepare stmt from 'select * from t where a = ?';
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> set @a = 1;
Query OK, 0 rows affected (0.00 sec)

-- The first execution generates an execution plan and saves it in the
-- ↪ cache.
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+-----+
| @@last_plan_from_cache |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)

-- The second execution hits the cache.
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+-----+
| @@last_plan_from_cache |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

If you find that a certain set of `Prepare / Execute` has unexpected behavior due to the execution plan cache, you can use the `ignore_plan_cache()` SQL hint to skip using the execution plan cache for the current statement. Still, use the above statement as an example:

```

MySQL [test]> prepare stmt from 'select /*+ ignore_plan_cache() */ * from t
 ↪ where a = ?';
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> set @a = 1;
Query OK, 0 rows affected (0.00 sec)
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+-----+
| @@last_plan_from_cache |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
MySQL [test]> execute stmt using @a;
Empty set (0.00 sec)
MySQL [test]> select @@last_plan_from_cache;
+-----+
| @@last_plan_from_cache |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)

```

### 8.3.4 Control Execution Plans

#### 8.3.4.1 Control Execution Plan

The first two chapters of SQL Tuning introduce how to understand TiDB's execution plan and how TiDB generates an execution plan. This chapter introduces what methods can be used to control the generation of the execution plan when you determine the problems with the execution plan. This chapter mainly includes the following three aspects:

- In [Optimizer Hints](#), you will learn how to use hints to guide TiDB to generate an execution plan.
- But hints change the SQL statement intrusively. In some scenarios, hints cannot be simply inserted. In [SQL Plan Management](#), you will know how TiDB uses another syntax to non-intrusively control the generation of execution plans, and the methods of automatic execution plan evolution in the background to alleviate the execution plan instability caused by reasons such as version upgrades, which degrades cluster performance.
- Finally, you will learn how to use the blocklist in [Blocklist of Optimization Rules and Expression Pushdown](#).

### 8.3.4.2 Optimizer Hints

TiDB supports optimizer hints, which are based on the comment-like syntax introduced in MySQL 5.7. For example, one of the common syntaxes is `/*+ HINT_NAME([t1_name [, ↪ t2_name] ...])*/`. Use of optimizer hints is recommended in cases where the TiDB optimizer selects a less optimal query plan.

**Note:**

MySQL command-line clients earlier than 5.7.7 strip optimizer hints by default. If you want to use the Hint syntax in these earlier versions, add the `--comments` option when starting the client. For example: `mysql -h ↪ 127.0.0.1 -P 4000 -uroot --comments`.

#### 8.3.4.2.1 Syntax

Optimizer hints are case insensitive and specified within `/*+ ... */` comments following the `SELECT`, `UPDATE` or `DELETE` keyword in a SQL statement. Optimizer hints are not currently supported for `INSERT` statements.

Multiple hints can be specified by separating with commas. For example, the following query uses three different hints:

```
SELECT /*+ USE_INDEX(t1, idx1), HASH_AGG(), HASH_JOIN(t1) */ count(*) FROM
 ↪ t t1, t t2 WHERE t1.a = t2.b;
```

How optimizer hints affect query execution plans can be observed in the output of `EXPLAIN` and `EXPLAIN ANALYZE`.

An incorrect or incomplete hint will not result in a statement error. This is because hints are intended to have only a *hint* (suggestion) semantic to query execution. Similarly, TiDB will at most return a warning if a hint is not applicable.

**Note:**

If the comments do not follow behind the specified keywords, they will be treated as common MySQL comments. The comments do not take effect, and no warning is reported.

Currently, TiDB supports two categories of hints, which are different in scope. The first category of hints takes effect in the scope of query blocks, such as `/*+ HASH_AGG()*/`; the

second category of hints takes effect in the whole query, such as `/*+ MEMORY_QUOTA(1024 → MB)*/`.

Each query or sub-query in a statement corresponds to a different query block, and each query block has its own name. For example:

```
SELECT * FROM (SELECT * FROM t) t1, (SELECT * FROM t) t2;
```

The above query statement has three query blocks: the outermost SELECT corresponds to the first query block, whose name is `sel_1`; the two SELECT sub-queries correspond to the second and the third query block, whose names are `sel_2` and `sel_3`, respectively. The sequence of the numbers is based on the appearance of SELECT from left to right. If you replace the first SELECT with DELETE or UPDATE, then the corresponding query block names are `del_1` or `upd_1`.

#### 8.3.4.2.2 Hints that take effect in query blocks

This category of hints can follow behind **any** SELECT, UPDATE or DELETE keywords. To control the effective scope of the hint, use the name of the query block in the hint. You can make the hint parameters clear by accurately identifying each table in the query (in case of duplicated table names or aliases). If no query block is specified in the hint, the hint takes effect in the current block by default.

For example:

```
SELECT /*+ HASH_JOIN(@sel_1 t1@sel_1, t3) */ * FROM (SELECT t1.a, t1.b
→ FROM t t1, t t2 WHERE t1.a = t2.a) t1, t t3 WHERE t1.b = t3.b;
```

This hint takes effect in the `sel_1` query block, and its parameters are the `t1` and `t3` tables in `sel_1` (`sel_2` also contains a `t1` table).

As described above, you can specify the name of the query block in the hint in the following ways:

- Set the query block name as the first parameter of the hint, and separate it from other parameters with a space. In addition to `QB_NAME`, all the hints listed in this section also have another optional hidden parameter `@QB_NAME`. By using this parameter, you can specify the effective scope of this hint.
- Append `@QB_NAME` to a table name in the parameter to explicitly specify which query block this table belongs to.

#### Note:

You must put the hint in or before the query block where the hint takes effect. If the hint is put after the query block, it cannot take effect.

## QB\_NAME

If the query statement is a complicated statement that includes multiple nested queries, the ID and name of a certain query block might be mistakenly identified. The hint `QB_NAME` can help us in this regard.

`QB_NAME` means Query Block Name. You can specify a new name to a query block. The specified `QB_NAME` and the previous default name are both valid. For example:

```
SELECT /*+ QB_NAME(QB1) */ * FROM (SELECT * FROM t) t1, (SELECT * FROM t)
 ↪ t2;
```

This hint specifies the outer `SELECT` query block's name to `QB1`, which makes `QB1` and the default name `sel_1` both valid for the query block.

### Note:

In the above example, if the hint specifies the `QB_NAME` to `sel_2` and does not specify a new `QB_NAME` for the original second `SELECT` query block, then `sel_2` becomes an invalid name for the second `SELECT` query block.

## MERGE\_JOIN(t1\_name [, tl\_name ...])

The `MERGE_JOIN(t1_name [, tl_name ...])` hint tells the optimizer to use the sort-merge join algorithm for the given table(s). Generally, this algorithm consumes less memory but takes longer processing time. If there is a very large data volume or insufficient system memory, it is recommended to use this hint. For example:

```
select /*+ MERGE_JOIN(t1, t2) */ * from t1, t2 where t1.id = t2.id;
```

### Note:

`TIDB_SMJ` is the alias for `MERGE_JOIN` in TiDB 3.0.x and earlier versions. If you are using any of these versions, you must apply the `TIDB_SMJ(t1_name ↪ [, tl_name ...])` syntax for the hint. For the later versions of TiDB, `TIDB_SMJ` and `MERGE_JOIN` are both valid names for the hint, but `MERGE_JOIN` is recommended.

## INL\_JOIN(t1\_name [, tl\_name ...])

The `INL_JOIN(t1_name [, tl_name ...])` hint tells the optimizer to use the index nested loop join algorithm for the given table(s). This algorithm might consume less system

resources and take shorter processing time in some scenarios and might produce an opposite result in other scenarios. If the result set is less than 10,000 rows after the outer table is filtered by the WHERE condition, it is recommended to use this hint. For example:

```
select /*+ INL_JOIN(t1, t2) */ * from t1, t2 where t1.id = t2.id;
```

The parameter(s) given in `INL_JOIN()` is the candidate table for the inner table when you create the query plan. For example, `INL_JOIN(t1)` means that TiDB only considers using `t1` as the inner table to create a query plan. If the candidate table has an alias, you must use the alias as the parameter in `INL_JOIN()`; if it does not have an alias, use the table's original name as the parameter. For example, in the `select /*+ INL_JOIN(t1) */ * from t t1, t t2 where t1.a = t2.b;` query, you must use the `t` table's alias `t1` or `t2` rather than `t` as `INL_JOIN()`'s parameter.

#### Note:

`TIDB_INLJ` is the alias for `INL_JOIN` in TiDB 3.0.x and earlier versions. If you are using any of these versions, you must apply the `TIDB_INLJ(t1_name → [, t1_name ...])` syntax for the hint. For the later versions of TiDB, `TIDB_INLJ` and `INL_JOIN` are both valid names for the hint, but `INL_JOIN` is recommended.

## INL\_HASH\_JOIN

The `INL_HASH_JOIN(t1_name [, t1_name])` hint tells the optimizer to use the index nested loop hash join algorithm. The conditions for using this algorithm are the same with the conditions for using the index nested loop join algorithm. The difference between the two algorithms is that `INL_JOIN` creates a hash table on the joined inner table, but `INL_HASH_JOIN` creates a hash table on the joined outer table. `INL_HASH_JOIN` has a fixed limit on memory usage, while the memory used by `INL_JOIN` depends on the number of rows matched in the inner table.

`HASH_JOIN(t1_name [, t1_name ...])`

The `HASH_JOIN(t1_name [, t1_name ...])` hint tells the optimizer to use the hash join algorithm for the given table(s). This algorithm allows the query to be executed concurrently with multiple threads, which achieves a higher processing speed but consumes more memory. For example:

```
select /*+ HASH_JOIN(t1, t2) */ * from t1, t2 where t1.id = t2.id;
```

#### Note:

TIDB\_HJ is the alias for HASH\_JOIN in TiDB 3.0.x and earlier versions. If you are using any of these versions, you must apply the TIDB\_HJ(`t1_name ↪ [, t1_name ...]`) syntax for the hint. For the later versions of TiDB, TIDB\_HJ and HASH\_JOIN are both valid names for the hint, but HASH\_JOIN is recommended.

### HASH\_AGG()

The HASH\_AGG() hint tells the optimizer to use the hash aggregation algorithm in all the aggregate functions in the specified query block. This algorithm allows the query to be executed concurrently with multiple threads, which achieves a higher processing speed but consumes more memory. For example:

```
select /*+ HASH_AGG() */ count(*) from t1, t2 where t1.a > 10 group by t1.
↪ id;
```

### STREAM\_AGG()

The STREAM\_AGG() hint tells the optimizer to use the stream aggregation algorithm in all the aggregate functions in the specified query block. Generally, this algorithm consumes less memory but takes longer processing time. If there is a very large data volume or insufficient system memory, it is recommended to use this hint. For example:

```
select /*+ STREAM_AGG() */ count(*) from t1, t2 where t1.a > 10 group by t1
↪ .id;
```

### USE\_INDEX(`t1_name, idx1_name [, idx2_name ...]`)

The USE\_INDEX(`t1_name, idx1_name [, idx2_name ...]`) hint tells the optimizer to use only the given index(es) for a specified `t1_name` table. For example, applying the following hint has the same effect as executing the `select * from t t1 use index(idx1, ↪ idx2);` statement.

```
SELECT /*+ USE_INDEX(t1, idx1, idx2) */ * FROM t1;
```

#### Note:

If you specify only the table name but not index name in this hint, the execution does not consider any index but scan the entire table.

### IGNORE\_INDEX(`t1_name, idx1_name [, idx2_name ...]`)

The IGNORE\_INDEX(`t1_name, idx1_name [, idx2_name ...]`) hint tells the optimizer to ignore the given index(es) for a specified `t1_name` table. For example, applying the

following hint has the same effect as executing the `select * from t t1 ignore index(→ idx1, idx2);` statement.

```
select /*+ IGNORE_INDEX(t1, idx1, idx2) */ * from t t1;
```

### AGG\_TO\_COP()

The `AGG_TO_COP()` hint tells the optimizer to push down the aggregate operation in the specified query block to the coprocessor. If the optimizer does not push down some aggregate function that is suitable for pushdown, then it is recommended to use this hint. For example:

```
select /*+ AGG_TO_COP() */ sum(t1.a) from t t1;
```

### LIMIT\_TO\_COP()

The `LIMIT_TO_COP()` hint tells the optimizer to push down the `Limit` and `TopN` operators in the specified query block to the coprocessor. If the optimizer does not perform such an operation, it is recommended to use this hint. For example:

```
SELECT /*+ LIMIT_TO_COP() */ * FROM t WHERE a = 1 AND b > 10 ORDER BY c
→ LIMIT 1;
```

`READ_FROM_STORAGE(TIFLASH[t1_name [, tl_name ...]], TIKV[t2_name [, tl_name ...]])`

The `READ_FROM_STORAGE(TIFLASH[t1_name [, tl_name ...]], TIKV[t2_name [, tl_name ...]])` hint tells the optimizer to read specific table(s) from specific storage engine(s). Currently, this hint supports two storage engine parameters - `TIKV` and `TIFLASH`. If a table has an alias, use the alias as the parameter of `READ_FROM_STORAGE()`; if the table does not have an alias, use the table's original name as the parameter. For example:

```
select /*+ READ_FROM_STORAGE(TIFLASH[t1], TIKV[t2]) */ t1.a from t t1, t
→ t2 where t1.a = t2.a;
```

### Note:

If you want the optimizer to use a table from another schema, you need to explicitly specify the schema name. For example:

```
SELECT /*+ READ_FROM_STORAGE(TIFLASH[test1.t1,test2.t2]) */ t1
→ .a FROM test1.t t1, test2.t t2 WHERE t1.a = t2.a;
```

### USE\_INDEX\_MERGE(t1\_name, idx1\_name [, idx2\_name ...])

The `USE_INDEX_MERGE(t1_name, idx1_name [, idx2_name ...])` hint tells the optimizer to access a specific table with the index merge method. The given list of indexes are

optional parameters. If you explicitly specify the list, TiDB selects indexes from the list to build index merge; if you do not give the list of indexes, TiDB selects indexes from all available indexes to build index merge. For example:

```
SELECT /*+ USE_INDEX_MERGE(t1, idx_a, idx_b, idx_c) */ * FROM t1 WHERE t1.
 ↪ a > 10 OR t1.b > 10;
```

When multiple `USE_INDEX_MERGE` hints are made to the same table, the optimizer tries to select the index from the union of the index sets specified by these hints.

#### Note:

The parameters of `USE_INDEX_MERGE` refer to index names, rather than column names. The index name of the primary key is `primary`.

This hint takes effect on strict conditions, including:

- If the query can select a single index scan in addition to full table scan, the optimizer does not select index merge.
- If the query is in an explicit transaction, and if the statements before this query has already written data, the optimizer does not select index merge.

#### 8.3.4.2.3 Hints that take effect in the whole query

This category of hints can only follow behind the `first` `SELECT`, `UPDATE` or `DELETE` keyword, which is equivalent to modifying the value of the specified system variable when this query is executed. The priority of the hint is higher than that of existing system variables.

#### Note:

This category of hints also has an optional hidden variable `@QB_NAME`, but the hint takes effect in the whole query even if you specify the variable.

#### NO\_INDEX\_MERGE()

The `NO_INDEX_MERGE()` hint disables the index merge feature of the optimizer.

For example, the following query will not use index merge:

```
select /*+ NO_INDEX_MERGE() */ * from t where t.a > 0 or t.b > 0;
```

In addition to this hint, setting the `tidb_enable_index_merge` system variable also controls whether to enable this feature.

#### Note:

`NO_INDEX_MERGE` has a higher priority over `USE_INDEX_MERGE`. When both hints are used, `USE_INDEX_MERGE` does not take effect.

### USE\_TOJA(boolean\_value)

The `boolean_value` parameter can be TRUE or FALSE. The `USE_TOJA(TRUE)` hint enables the optimizer to convert an `in` condition (containing a sub-query) to join and aggregation operations. Comparatively, the `USE_TOJA(FALSE)` hint disables this feature.

For example, the following query will convert `in (select t2.a from t2)subq` to corresponding join and aggregation operations:

```
select /*+ USE_TOJA(TRUE) */ t1.a, t1.b from t1 where t1.a in (select t2.a
 ↪ from t2) subq;
```

In addition to this hint, setting the `tidb_opt_insubq_to_join_and_agg` system variable also controls whether to enable this feature.

### MAX\_EXECUTION\_TIME(N)

The `MAX_EXECUTION_TIME(N)` hint places a limit N (a timeout value in milliseconds) on how long a statement is permitted to execute before the server terminates it. In the following hint, `MAX_EXECUTION_TIME(1000)` means that the timeout is 1000 milliseconds (that is, 1 second):

```
select /*+ MAX_EXECUTION_TIME(1000) */ * from t1 inner join t2 where t1.id
 ↪ = t2.id;
```

In addition to this hint, the `global.max_execution_time` system variable can also limit the execution time of a statement.

### MEMORY\_QUOTA(N)

The `MEMORY_QUOTA(N)` hint places a limit N (a threshold value in MB or GB) on how much memory a statement is permitted to use. When a statement's memory usage exceeds this limit, TiDB produces a log message based on the statement's over-limit behavior or just terminates it.

In the following hint, `MEMORY_QUOTA(1024 MB)` means that the memory usage is limited to 1024 MB:

```
select /*+ MEMORY_QUOTA(1024 MB) */ * from t;
```

In addition to this hint, the `tidb_mem_quota_query` system variable can also limit the memory usage of a statement.

#### `READ_CONSISTENT_REPLICA()`

The `READ_CONSISTENT_REPLICA()` hint enables the feature of reading consistent data from the TiKV follower node. For example:

```
select /*+ READ_CONSISTENT_REPLICA() */ * from t;
```

In addition to this hint, setting the `tidb_replica_read` environment variable to '`follower`' or '`leader`' also controls whether to enable this feature.

#### `IGNORE_PLAN_CACHE()`

The `IGNORE_PLAN_CACHE()` hint reminds the optimizer not to use the Plan Cache when handling the current `prepare` statement.

This hint is used to temporarily disable the Plan Cache for a certain type of queries when `prepare-plan-cache` is enabled.

In the following example, the Plan Cache is forcibly disabled when executing the `prepare` statement.

```
prepare stmt from 'select /*+ IGNORE_PLAN_CACHE() */ * from t where t.id =
→ ?';
```

#### `NTH_PLAN(N)`

The `NTH_PLAN(N)` hint reminds the optimizer to select the `N`th physical plan found during the physical optimization. `N` must be a positive integer.

If the specified `N` is beyond the search range of the physical optimization, TiDB will return a warning and select the optimal physical plan based on the strategy that ignores this hint.

This hint does not take effect when the cascades planner is enabled.

In the following example, the optimizer is forced to select the third physical plan found during the physical optimization:

```
SELECT /*+ NTH_PLAN(3) */ count(*) from t where a > 5;
```

#### Note:

`NTH_PLAN(N)` is mainly used for testing, and its compatibility is not guaranteed in later versions. Use this hint **with caution**.

### 8.3.4.3 SQL Plan Management (SPM)

SQL Plan Management is a set of functions that execute SQL bindings to manually interfere with SQL execution plans. These functions include SQL binding, baseline capturing, and baseline evolution.

#### 8.3.4.3.1 SQL binding

An SQL binding is the basis of SPM. The [Optimizer Hints](#) document introduces how to select a specific execution plan using hints. However, sometimes you need to interfere with execution selection without modifying SQL statements. With SQL bindings, you can select a specified execution plan without modifying SQL statements.

Create a binding

```
CREATE [GLOBAL | SESSION] BINDING FOR BindableStmt USING BindableStmt
```

This statement binds SQL execution plans at the GLOBAL or SESSION level. Currently, supported bindable SQL statements (BindableStmt) in TiDB include SELECT, DELETE, UPDATE, and INSERT / REPLACE with SELECT subqueries.

Specifically, two types of these statements cannot be bound to execution plans due to syntax conflicts. See the following examples:

```
-- Type one: Statements that get the Cartesian product by using the `join` keyword and not specifying the associated columns with the `using` keyword.
create global binding for
 select * from t t1 join t t2
using
 select * from t t1 join t t2;

-- Type two: `DELETE` statements that contain the `using` keyword.
create global binding for
 delete from t1 using t1 join t2 on t1.a = t2.a
using
 delete from t1 using t1 join t2 on t1.a = t2.a;
```

You can bypass syntax conflicts by using equivalent statements. For example, you can rewrite the above statements in the following ways:

```
-- First rewrite of type one statements: Add a `using` clause for the `join` keyword.
create global binding for
 select * from t t1 join t t2 using (a)
using
 select * from t t1 join t t2 using (a);
```

```
-- Second rewrite of type one statements: Delete the `join` keyword.
create global binding for
 select * from t t1, t t2
using
 select * from t t1, t t2;

-- Rewrite of type two statements: Remove the `using` keyword from the `→ delete` statement.
create global binding for
 delete t1 from t1 join t2 on t1.a = t2.a
using
 delete t1 from t1 join t2 on t1.a = t2.a;
```

### Note:

When creating execution plan bindings for `INSERT / REPLACE` statements with `SELECT` subqueries, you need to specify the optimizer hints you want to bind in the `SELECT` subquery, not after the `INSERT / REPLACE` keyword. Otherwise, the optimizer hints do not take effect as intended.

Here are two examples:

```
-- The hint takes effect in the following statement.
create global binding for
 insert into t1 select * from t2 where a > 1 and b = 1
using
 insert into t1 select /*+ use_index(@sel_1 t2, a) */ * from t2 where a
 → > 1 and b = 1;

-- The hint cannot take effect in the following statement.
create global binding for
 insert into t1 select * from t2 where a > 1 and b = 1
using
 insert /*+ use_index(@sel_1 t2, a) */ into t1 select * from t2 where a
 → > 1 and b = 1;
```

If you do not specify the scope when creating an execution plan binding, the default scope is SESSION. The TiDB optimizer normalizes bound SQL statements and stores them in the system table. When processing SQL queries, if a normalized statement matches one of the bound SQL statements in the system table and the system variable `tidb_use_plan_baselines` is set to `on` (the default value is `on`), TiDB then uses the

corresponding optimizer hint for this statement. If there are multiple matchable execution plans, the optimizer chooses the least costly one to bind.

Normalization is a process that converts a constant in an SQL statement to a variable parameter and explicitly specifies the database for tables referenced in the query, with standardized processing on the spaces and line breaks in the SQL statement. See the following example:

```
select * from t where a > 1
-- After normalization, the above statement is as follows:
select * from test . t where a > ?
```

#### Note:

Multiple constants joined by commas , are normalized as ... instead of ?.

For example:

```
select * from t limit 10
select * from t limit 10, 20
select * from t where a in (1)
select * from t where a in (1,2,3)
-- After normalization, the above statements are as follows:
select * from test . t limit ?
select * from test . t limit ...
select * from test . t where a in (?)
select * from test . t where a in (...)
```

When bindings are created, TiDB treats SQL statements that contain a single constant and SQL statements that contain multiple constants joined by commas differently. Therefore, you need to create bindings for the two SQL types separately.

When a SQL statement has bound execution plans in both GLOBAL and SESSION scopes, because the optimizer ignores the bound execution plan in the GLOBAL scope when it encounters the SESSION binding, the bound execution plan of this statement in the SESSION scope shields the execution plan in the GLOBAL scope.

For example:

```
-- Creates a GLOBAL binding and specifies using `sort merge join` in this
-- → binding.
create global binding for
 select * from t1, t2 where t1.id = t2.id
using
```

```

select /*+ merge_join(t1, t2) */ * from t1, t2 where t1.id = t2.id;

-- The execution plan of this SQL statement uses the `sort merge join`
-- ↪ specified in the GLOBAL binding.
explain select * from t1, t2 where t1.id = t2.id;

-- Creates another SESSION binding and specifies using `hash join` in this
-- ↪ binding.
create binding for
 select * from t1, t2 where t1.id = t2.id
using
 select /*+ hash_join(t1, t2) */ * from t1, t2 where t1.id = t2.id;

-- In the execution plan of this statement, `hash join` specified in the
-- ↪ SESSION binding is used, instead of `sort merge join` specified in
-- ↪ the GLOBAL binding.
explain select * from t1, t2 where t1.id = t2.id;

```

When the first `select` statement is being executed, the optimizer adds the `sm_join(t1, t2)` hint to the statement through the binding in the GLOBAL scope. The top node of the execution plan in the `explain` result is `MergeJoin`. When the second `select` statement is being executed, the optimizer uses the binding in the SESSION scope instead of the binding in the GLOBAL scope and adds the `hash_join(t1, t2)` hint to the statement. The top node of the execution plan in the `explain` result is `HashJoin`.

Each standardized SQL statement can have only one binding created using `CREATE BINDING` at a time. When multiple bindings are created for the same standardized SQL statement, the last created binding is retained, and all previous bindings (created and evolved) are marked as deleted. But session bindings and global bindings can coexist and are not affected by this logic.

In addition, when you create a binding, TiDB requires that the session is in a database context, which means that a database is specified when the client is connected or `use ${database}` is executed.

The original SQL statement and the bound statement must have the same text after normalization and hint removal, or the binding will fail. Take the following examples:

- This binding can be created successfully because the texts before and after parameterization and hint removal are the same: `select * from test . t where a > ?`

```

sql CREATE BINDING FOR SELECT * FROM t WHERE a > 1 USING SELECT * FROM
↪ t use index (idx) WHERE a > 2

```
- This binding will fail because the original SQL statement is processed as `select * from test . t where a > ?`, while the bound SQL statement is processed differently as `select * from test . t where b > ?`.

```
sql CREATE BINDING FOR SELECT * FROM t WHERE a > 1 USING SELECT * FROM
↪ t use index(idx) WHERE b > 2
```

**Note:**

For `PREPARE` / `EXECUTE` statements and for queries executed with binary protocols, you need to create execution plan bindings for the real query statements, not for the `PREPARE` / `EXECUTE` statements.

Remove binding

```
DROP [GLOBAL | SESSION] BINDING FOR BindableStmt;
```

This statement removes a specified execution plan binding at the GLOBAL or SESSION level. The default scope is SESSION.

Generally, the binding in the SESSION scope is mainly used for test or in special situations. For a binding to take effect in all TiDB processes, you need to use the GLOBAL binding. A created SESSION binding shields the corresponding GLOBAL binding until the end of the SESSION, even if the SESSION binding is dropped before the session closes. In this case, no binding takes effect and the plan is selected by the optimizer.

The following example is based on the example in [create binding](#) in which the SESSION binding shields the GLOBAL binding:

```
-- Drops the binding created in the SESSION scope.
drop session binding for select * from t1, t2 where t1.id = t2.id;

-- Views the SQL execution plan again.
explain select * from t1,t2 where t1.id = t2.id;
```

In the example above, the dropped binding in the SESSION scope shields the corresponding binding in the GLOBAL scope. The optimizer does not add the `sm_join(t1, t2)` hint to the statement. The top node of the execution plan in the `explain` result is not fixed to `MergeJoin` by this hint. Instead, the top node is independently selected by the optimizer according to the cost estimation.

**Note:**

Executing `DROP GLOBAL BINDING` drops the binding in the current tidb-server instance cache and changes the status of the corresponding row in the system table to ‘deleted’. This statement does not directly delete the records in the

system table, because other tidb-server instances need to read the ‘deleted’ status to drop the corresponding binding in their cache. For the records in these system tables with the status of ‘deleted’, at every 100 `bind-info-lease` (the default value is 3s, and 300s in total) interval, the background thread triggers an operation of reclaiming and clearing on the bindings of `update_time` before 10 `bind-info-lease` (to ensure that all tidb-server instances have read the ‘deleted’ status and updated the cache).

### View binding

```
SHOW [GLOBAL | SESSION] BINDINGS [ShowLikeOrWhere]
```

This statement outputs the execution plan bindings at the GLOBAL or SESSION level according to the order of binding update time from the latest to earliest. The default scope is SESSION. Currently SHOW BINDINGS outputs eight columns, as shown below:

| Column       |                                                                              |
|--------------|------------------------------------------------------------------------------|
| Name         | Note                                                                         |
| original_sql | Original SQL statement after parameterization                                |
| bind_sql     | Bound SQL statement with hints                                               |
| default_db   | Default database                                                             |
| status       | Status including Using, Deleted, Invalid, Rejected, and Pending verification |
| create_time  | Creating time                                                                |
| update_time  | Updating time                                                                |
| charset      | Character set                                                                |
| collation    | Ordering rule                                                                |

| Column Name | Note                                                                                                                                                                                                                                                                      |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| source      | The way in which a binding is created, including <code>manual</code> (created by the <code>create</code> ↗ [global] ↗ <code>binding</code> SQL statement), <code>capture</code> (captured automatically by TiDB), and <code>evolve</code> (evolved automatically by TiDB) |

### Troubleshoot binding

```
SELECT @@[SESSION.]last_plan_from_binding;
```

This statement uses the system variable `last_plan_from_binding` to show whether the execution plan used by the last executed statement is from the binding.

In addition, when you use the `explain format = 'verbose'` statement to view the query plan of a SQL statement, if the SQL statement uses binding, the `explain` statement will return a warning. In this situation, you can check the warning message to learn which binding is used in the SQL statement.

```
-- Create a global binding.

create global binding for
 select * from t
using
 select * from t;

-- Use the `explain format = 'verbose'` statement to check the SQL
-- execution plan. Check the warning message to view the binding used
-- in the query.

explain format = 'verbose' select * from t;
show warnings;
```

#### 8.3.4.3.2 Baseline capturing

To enable baseline capturing, set `tidb_capture_plan_baselines` to `on`. The default value is `off`.

##### Note:

Because the automatic binding creation function relies on [Statement Summary](#), make sure to enable Statement Summary before using automatic binding.

After automatic binding creation is enabled, the historical SQL statements in the Statement Summary are traversed every `bind-info-lease` (the default value is `3s`), and a binding is automatically created for SQL statements that appear at least twice. For these SQL statements, TiDB automatically binds the execution plan recorded in Statement Summary.

However, TiDB does not automatically capture bindings for the following types of SQL statements:

- `EXPLAIN` and `EXPLAIN ANALYZE` statements.
- SQL statements executed internally in TiDB, such as `SELECT` queries used for automatically loading statistical information.
- SQL statements that are bound to a manually created execution plan.

For `PREPARE` / `EXECUTE` statements and for queries executed with binary protocols, TiDB automatically captures bindings for the real query statements, not for the `PREPARE` / `EXECUTE` statements.

##### Note:

Because TiDB has some embedded SQL statements to ensure the correctness of some features, baseline capturing by default automatically shields these SQL statements.

#### 8.3.4.3.3 Baseline evolution

Baseline evolution is an important feature of SPM introduced in TiDB v4.0.

As data updates, the previously bound execution plan might no longer be optimal. The baseline evolution feature can automatically optimize the bound execution plan.

In addition, baseline evolution, to a certain extent, can also avoid the jitter brought to the execution plan caused by the change of statistical information.

### Usage

Use the following statement to enable automatic binding evolution:

```
set global tidb_evolve_plan_baselines = on;
```

The default value of `tidb_evolve_plan_baselines` is `off`.

### Warning:

- Baseline evolution is an experimental feature. Unknown risks might exist. It is **NOT** recommended that you use it in the production environment.
- This variable is forcibly set to `off` until the baseline evolution feature becomes generally available (GA). If you try to enable this feature, an error is returned. If you have already used this feature in a production environment, disable it as soon as possible. If you find that the binding status is not as expected, contact PingCAP's technical support for help.

After the automatic binding evolution feature is enabled, if the optimal execution plan selected by the optimizer is not among the binding execution plans, the optimizer marks the plan as an execution plan that waits for verification. At every `bind-info-lease` (the default value is `3s`) interval, an execution plan to be verified is selected and compared with the binding execution plan that has the least cost in terms of the actual execution time. If the plan to be verified has shorter execution time (the current criterion for the comparison is that the execution time of the plan to be verified is no longer than 2/3 that of the binding execution plan), this plan is marked as a usable binding. The following example describes the process above.

Assume that table `t` is defined as follows:

```
create table t(a int, b int, key(a), key(b));
```

Perform the following query on table `t`:

```
select * from t where a < 100 and b < 100;
```

In the table defined above, few rows meet the `a < 100` condition. But for some reason, the optimizer mistakenly selects the full table scan instead of the optimal execution plan that uses index `a`. You can first use the following statement to create a binding:

```
create global binding for select * from t where a < 100 and b < 100 using
 ↵ select * from t use index(a) where a < 100 and b < 100;
```

When the query above is executed again, the optimizer selects index **a** (influenced by the binding created above) to reduce the query time.

Assuming that as insertions and deletions are performed on table **t**, an increasing number of rows meet the  $a < 100$  condition and a decreasing number of rows meet the  $b < 100$  condition. At this time, using index **a** under the binding might no longer be the optimal plan.

The binding evolution can address this kind of issues. When the optimizer recognizes data change in a table, it generates an execution plan for the query that uses index **b**. However, because the binding of the current plan exists, this query plan is not adopted and executed. Instead, this plan is stored in the backend evolution list. During the evolution process, if this plan is verified to have an obviously shorter execution time than that of the current execution plan that uses index **a**, index **b** is added into the available binding list. After this, when the query is executed again, the optimizer first generates the execution plan that uses index **b** and makes sure that this plan is in the binding list. Then the optimizer adopts and executes this plan to reduce the query time after data changes.

To reduce the impact that the automatic evolution has on clusters, use the following configurations:

- Set `tidb_evolve_plan_task_max_time` to limit the maximum execution time of each execution plan. The default value is 600s. In the actual verification process, the maximum execution time is also limited to no more than twice the time of the verified execution plan.
- Set `tidb_evolve_plan_task_start_time` (00:00 +0000 by default) and `tidb_evolve_plan_task_end_time` (23:59 +0000 by default) to limit the time window.

## Notes

Because the baseline evolution automatically creates a new binding, when the query environment changes, the automatically created binding might have multiple behavior choices. Pay attention to the following notes:

- Baseline evolution only evolves standardized SQL statements that have at least one global binding.
- Because creating a new binding deletes all previous bindings (for a standardized SQL statement), the automatically evolved binding will be deleted after manually creating a new binding.
- All hints related to the calculation process are retained during the evolution. These hints are as follows:

| Hint                            | Description                                                             |
|---------------------------------|-------------------------------------------------------------------------|
| <code>memory_quota</code>       | The maximum memory that can be used for a query.                        |
| <code>use_toja</code>           | Whether the optimizer transforms sub-queries to Join.                   |
| <code>use_cascades</code>       | Whether to use the cascades optimizer.                                  |
| <code>no_index_merge</code>     | Whether the optimizer uses Index Merge as an option for reading tables. |
| <code>read_consistency</code>   | Whether the replica forcibly enable Follower Read when reading tables.  |
| <code>max_execution_time</code> | The longest duration for a query.                                       |

- `read_from_storage` is a special hint in that it specifies whether to read data from TiKV or from TiFlash when reading tables. Because TiDB provides isolation reads, when the isolation condition changes, this hint has a great influence on the evolved execution plan. Therefore, when this hint exists in the initially created binding, TiDB ignores all its evolved bindings.

#### 8.3.4.3.4 Upgrade checklist

During cluster upgrade, SQL Plan Management (SPM) might cause compatibility issues and make the upgrade fail. To ensure a successful upgrade, you need to include the following list for upgrade precheck:

- When you upgrade from a version earlier than v5.2.0 (that is, v4.0, v5.0, and v5.1) to the current version, make sure that `tidb_evolve_plan_baselines` is disabled before the upgrade. To disable this variable, perform the following steps.

```
-- Check whether `tidb_evolve_plan_baselines` is disabled in the
-- earlier version.
```

```
select @@global.tidb_evolve_plan_baselines;

-- If `tidb_evolve_plan_baselines` is still enabled, disable it.

set global tidb_evolve_plan_baselines = off;
```

- Before you upgrade from v4.0 to the current version, you need to check whether the syntax of all queries corresponding to the available SQL bindings is correct in the new version. If any syntax errors exist, delete the corresponding SQL binding. To do that, perform the following steps.

```
-- Check the query corresponding to the available SQL binding in the
-- ↪ version to be upgraded.

select bind_sql from mysql.bind_info where status = 'using';

-- Verify the result from the above SQL query in the test environment
-- ↪ of the new version.

bind_sql_0;
bind_sql_1;
...

-- In the case of a syntax error (ERROR 1064 (42000): You have an
-- ↪ error in your SQL syntax), delete the corresponding binding.
-- For any other errors (for example, tables are not found), it means
-- ↪ that the syntax is compatible. No other operation is needed.
```

#### 8.3.4.4 The Blocklist of Optimization Rules and Expression Pushdown

This document introduces how to use the blocklist of optimization rules and the blocklist of expression pushdown to control the behavior of TiDB.

##### 8.3.4.4.1 The blocklist of optimization rules

The blocklist of optimization rules is one way to tune optimization rules, mainly used to manually disable some optimization rules.

Important optimization rules

| Optimization Rule | Name         | Description                                                                                          |
|-------------------|--------------|------------------------------------------------------------------------------------------------------|
| Column pruning    | column_prune | operator will prune the column if it is not needed by the upper operator except executor.            |
| Decorrelation     | decorrelate  | This rule tries to rewrite the correlated sub-query to non-correlated join or aggregation operation. |

---

| Optimization Rule       | Name                    | Description                                                                                        |
|-------------------------|-------------------------|----------------------------------------------------------------------------------------------------|
| Aggregation elimination | Aggregation elimination | Aggregation elimination rule eliminates unnecessary aggregation operators from the execution plan. |
| Project elimination     | Project elimination     | Project elimination rule eliminates unnecessary projection operators from the execution plan.      |

| Optimization Rule | Name      | Description                                                             |
|-------------------|-----------|-------------------------------------------------------------------------|
| Max/Min           | max/min   | eliminate some functions in aggregation to the order                    |
| push-down         | push-down | push predicates down to the operator that is closer to the data source. |

---

| Optimizat <u>ion</u>   | Name                 | Description                                                                |
|------------------------|----------------------|----------------------------------------------------------------------------|
| Rule                   |                      |                                                                            |
| Outer join elimination | outer_join_eliminate | to remove the unnecessary left join or right join from the execution plan. |

---

| Optimizat <b>ion</b> | Rule                         | Name      | Description                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|------------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Partitiop <b>n</b>   | partitionPr <b>processor</b> |           | partitionPr <b>processor</b><br>prun-<br>ing<br>which<br>are<br>re-<br>jected<br>by<br>the<br>predi-<br>cates<br>and<br>rewrite<br>parti-<br>tioned<br>table<br>query<br>to<br>the<br>UnionAll<br>$\hookrightarrow$<br>$\hookrightarrow +$<br>$\hookrightarrow$<br>$\hookrightarrow$ Partition<br>$\hookrightarrow$<br>$\hookrightarrow$ Datasource<br>$\hookrightarrow$<br>form. |
| Aggregation          | aggregationRules             | push_down | push_down<br>to<br>push<br>ag-<br>gre-<br>ga-<br>tions<br>down<br>to<br>their<br>chil-<br>dren.                                                                                                                                                                                                                                                                                   |

| Optimization   |               |                                                                       |
|----------------|---------------|-----------------------------------------------------------------------|
| Rule           | Name          | Description                                                           |
| TopN push-down | topn_pushdown | TopN pushes the TopN operator to the place closer to the data source. |
| Join re-order  | join_reorder  | join_reorder decides the order of multi-table joins.                  |

## Disable optimization rules

You can use the blocklist of optimization rules to disable some of them if some rules lead to a sub-optimal execution plan for special queries.

### Usage

#### Note:

All the following operations need the `super privilege` privilege of the database. Each optimization rule has a name. For example, the name of column pruning is `column_prune`. The names of all optimization rules can be found in the second column of the table [Important Optimization Rules](#).

- If you want to disable some rules, write its name to the `mysql.opt_rule_blacklist` table. For example:

```
INSERT INTO mysql.opt_rule_blacklist VALUES("join_reorder"), (
 ↳ topn_push_down);
```

Executing the following SQL statement can make the above operation take effect immediately. The effective range includes all old connections of the corresponding TiDB server:

```
admin reload opt_rule_blacklist;
```

#### Note:

`admin reload opt_rule_blacklist` only takes effect on the TiDB server where the above statement has been run. If you want all TiDB servers of the cluster to take effect, run this command on each TiDB server.

- If you want to re-enable a rule, delete the corresponding data in the table, and then run the `admin reload` statement:

```
DELETE FROM mysql.opt_rule_blacklist WHERE name IN ("join_reorder", "
 ↳ topn_push_down");
```

```
admin reload opt_rule_blacklist;
```

#### 8.3.4.4.2 The blocklist of expression pushdown

The blocklist of expression pushdown is one way to tune the expression pushdown, mainly used to manually disable some expressions of some specific data types.

Expressions which are supported to be pushed down

| Classification                     | Concrete Operations                                                               |
|------------------------------------|-----------------------------------------------------------------------------------|
| Logical operations                 | AND (&&), OR (  ), NOT (!)                                                        |
| Comparison functions and operators | <, <=, =, != (<>), >, >=, <=>, IN(), IS NULL, LIKE, IS TRUE, IS FALSE, COALESCE() |
| Numeric functions and operators    | +, -, *, /, ABS(), CEIL(), CEILING(), FLOOR()                                     |
| Control flow functions             | CASE, IF(), IFNULL()                                                              |

| Expression Classification | Concrete Operations                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| JSON functions            | JSON_TYPE(json_val),<br>JSON_EXTRACT(json_doc, path[, path] ...), JSON_UNQUOTE(json_val),<br>JSON_OBJECT(key, val[, key, val] ...),<br>JSON_ARRAY([val[, val] ...]),<br>JSON_MERGE(json_doc, json_doc[, json_doc] ...), JSON_SET(json_doc, path, val[, path, val] ...),<br>JSON_INSERT(json_doc, path, val[, path, val] ...),<br>JSON_REPLACE(json_doc, path, val[, path, val] ...),<br>JSON_REMOVE(json_doc, path[, path] ...) |
| Date and time functions   | DATE_FORMAT()                                                                                                                                                                                                                                                                                                                                                                                                                   |

Disable the pushdown of specific expressions

When you get wrong results due to the expression pushdown, you can use the blocklist to make a quick recovery for the application. More specifically, you can add some of the supported functions or operators to the `mysql.expr_pushdown_blacklist` table to disable the pushdown of specific expressions.

The schema of `mysql.expr_pushdown_blacklist` is shown as follows:

```
DESC mysql.expr_pushdown_blacklist;
```

| Field      | Type         | Null | Key | Default           | Extra |
|------------|--------------|------|-----|-------------------|-------|
| name       | char(100)    | NO   |     | NULL              |       |
| store_type | char(100)    | NO   |     | tikv,tiflash,tidb |       |
| reason     | varchar(200) | YES  |     | NULL              |       |

3 rows in set (0.00 sec)

Here is the description of each field above:

- **name**: The name of the function that is disabled to be pushed down.
- **store\_type**: To specify the component that you want to prevent the function from being pushed down to for computing. Available components are `tidb`, `tikv`, and `tiflash`. The `store_type` is case-insensitive. If you need to specify multiple components, use a comma to separate each component.

- When `store_type` is `tidb`, it indicates whether the function can be executed in other TiDB servers while the TiDB memory table is being read.
- When `store_type` is `tikv`, it indicates whether the function can be executed in TiKV server's Coprocessor component.
- When `store_type` is `tiflash`, it indicates whether the function can be executed in TiFlash Server's Coprocessor component.
- `reason`: To record the reason why this function is added to the blocklist.

### Usage

This section describes how to use the blocklist of expression pushdown.

#### Add to the blocklist

To add one or more expressions (functions or operators) to the blocklist, perform the following steps:

1. Insert the corresponding function name or operator name, and the set of components you want to disable the pushdown, to the `mysql.expr_pushdown_blacklist` table.
2. Execute `admin reload expr_pushdown_blacklist`.

#### Remove from the blocklist

To remove one or more expressions from the blocklist, perform the following steps:

1. Delete the corresponding function name or operator name, and the set of components you want to disable the pushdown, from the `mysql.expr_pushdown_blacklist` table.
2. Execute `admin reload expr_pushdown_blacklist`.

#### Note:

`admin reload expr_pushdown_blacklist` only takes effect on the TiDB server where this statement is run. If you want all TiDB servers of the cluster to take effect, run this command on each TiDB server.

#### 8.3.4.4.3 Expression blocklist usage example

In the following example, the `<` and `>` operators are added to the blocklist, and then the `>` operator is removed from the blocklist.

To judge whether the blocklist takes effect, observe the results of EXPLAIN (See [Optimize SQL statements using EXPLAIN](#)).

1. The predicates `a < 2` and `a > 2` in the `WHERE` clause of the following SQL statement can be pushed down to TiKV.

```
EXPLAIN SELECT * FROM t WHERE a < 2 AND a > 2;
```

```
+--+
→ -----+-----+-----+
→
| id | estRows | task | access object | operator
| info | | |
+--+
→ -----+-----+-----+
→
| TableReader_7 | 0.00 | root | | data:
| Selection_6 | | |
| -Selection_6 | 0.00 | cop[tikv] | | gt(ssb_1.t.
| | | | a, 2), lt(ssb_1.t.a, 2) |
| -TableFullScan_5 | 10000.00 | cop[tikv] | table:t | keep order:
| | | | false, stats:pseudo |
+--+
→ -----+-----+-----+
→
3 rows in set (0.00 sec)
```

2. Insert the expression to the `mysql.expr_pushdown_blacklist` table and execute `admin reload expr_pushdown_blacklist`.

```
INSERT INTO mysql.expr_pushdown_blacklist VALUES('<', 'tikv', '<'), ('>', 'tikv', '>');
```

```
Query OK, 2 rows affected (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
admin reload expr_pushdown_blacklist;
```

```
Query OK, 0 rows affected (0.00 sec)
```

3. Observe the execution plan again and you will find that both the `<` and `>` operators are not pushed down to TiKV Coprocessor.

```
EXPLAIN SELECT * FROM t WHERE a < 2 AND a > 2;
```

```
+--+
→ -----+-----+-----+
→
```

| id                       | estRows  | task      | access object | operator     |
|--------------------------|----------|-----------|---------------|--------------|
| ↳ info                   |          |           |               |              |
| +--                      |          |           |               |              |
| ↳                        |          |           |               |              |
| Selection_7              | 10000.00 | root      |               | gt(ssb_1.t.a |
| ↳ , 2), lt(ssb_1.t.a, 2) |          |           |               |              |
| --TableReader_6          | 10000.00 | root      |               | data:        |
| ↳ TableFullScan_5        |          |           |               |              |
| --TableFullScan_5        | 10000.00 | cop[tikv] | table:t       | keep order:  |
| ↳ false, stats:pseudo    |          |           |               |              |
| +--                      |          |           |               |              |
| ↳                        |          |           |               |              |
| 3 rows in set (0.00 sec) |          |           |               |              |

4. Remove one expression (here is `>`) from the blocklist and execute `admin reload`  
 ↳ `expr_pushdown_blacklist`.

```
DELETE FROM mysql.expr_pushdown_blacklist WHERE name = '>';
```

```
Query OK, 1 row affected (0.01 sec)
```

```
admin reload expr_pushdown_blacklist;
```

```
Query OK, 0 rows affected (0.00 sec)
```

5. Observe the execution plan again and you will find that `<` is not pushed down while `>` is pushed down to TiKV Coprocessor.

```
EXPLAIN SELECT * FROM t WHERE a < 2 AND a > 2;
```

| id              | estRows | task | access object | operator   |
|-----------------|---------|------|---------------|------------|
| ↳ info          |         |      |               |            |
| +--             |         |      |               |            |
| ↳               |         |      |               |            |
| Selection_8     | 0.00    | root |               | lt(ssb_1.t |
| ↳ .a, 2)        |         |      |               |            |
| --TableReader_7 | 0.00    | root |               | data:      |
| ↳ Selection_6   |         |      |               |            |

```

| -Selection_6 | 0.00 | cop[tikv] | | gt(ssb_1.
| ↳ t.a, 2) |
| -TableFullScan_5 | 10000.00 | cop[tikv] | table:t | keep
| ↳ order:false, stats:pseudo |
+--+
| ↳ -----+-----+-----+-----+
| ↳
4 rows in set (0.00 sec)

```

## 9 Tutorials

### 9.1 Multiple Data Centers in One City Deployment

As a NewSQL database, TiDB combines the best features of the traditional relational database and the scalability of the NoSQL database, and is highly available across data centers (DC). This document introduces the deployment of multiple DCs in one city.

#### 9.1.1 Raft protocol

Raft is a distributed consensus algorithm. Using this algorithm, both PD and TiKV, among components of the TiDB cluster, achieve disaster recovery of data, which is implemented through the following mechanisms:

- The essential role of Raft members is to perform log replication and act as a state machine. Among Raft members, data replication is implemented by replicating logs. Raft members change their own states in different conditions to elect a leader to provide services.
- Raft is a voting system that follows the majority protocol. In a Raft group, if a member gets the majority of votes, its membership changes to leader. In other words, when the majority of nodes remain in the Raft group, a leader can be elected to provide services.

To take advantage of Raft's reliability, the following conditions must be met in a real deployment scenario:

- Use at least three servers in case one server fails.
- Use at least three racks in case one rack fails.
- Use at least three DCs in case one DC fails.
- Deploy TiDB in at least three cities in case data safety issue occurs in one city.

The native Raft protocol does not have a good support for an even number of replicas. Considering the impact of cross-city network latency, three DCs in the same city might be the most suitable solution to a highly available and disaster tolerant Raft deployment.

### 9.1.2 Three DCs in one city deployment

TiDB clusters can be deployed in three DCs in the same city. In this solution, data replication across the three DCs is implemented using the Raft protocol within the cluster. These three DCs can provide read and write services at the same time. Data consistency is not affected even if one DC fails.

#### 9.1.2.1 Simple architecture

TiDB, TiKV and PD are distributed among three DCs, which is the most common deployment with the highest availability.

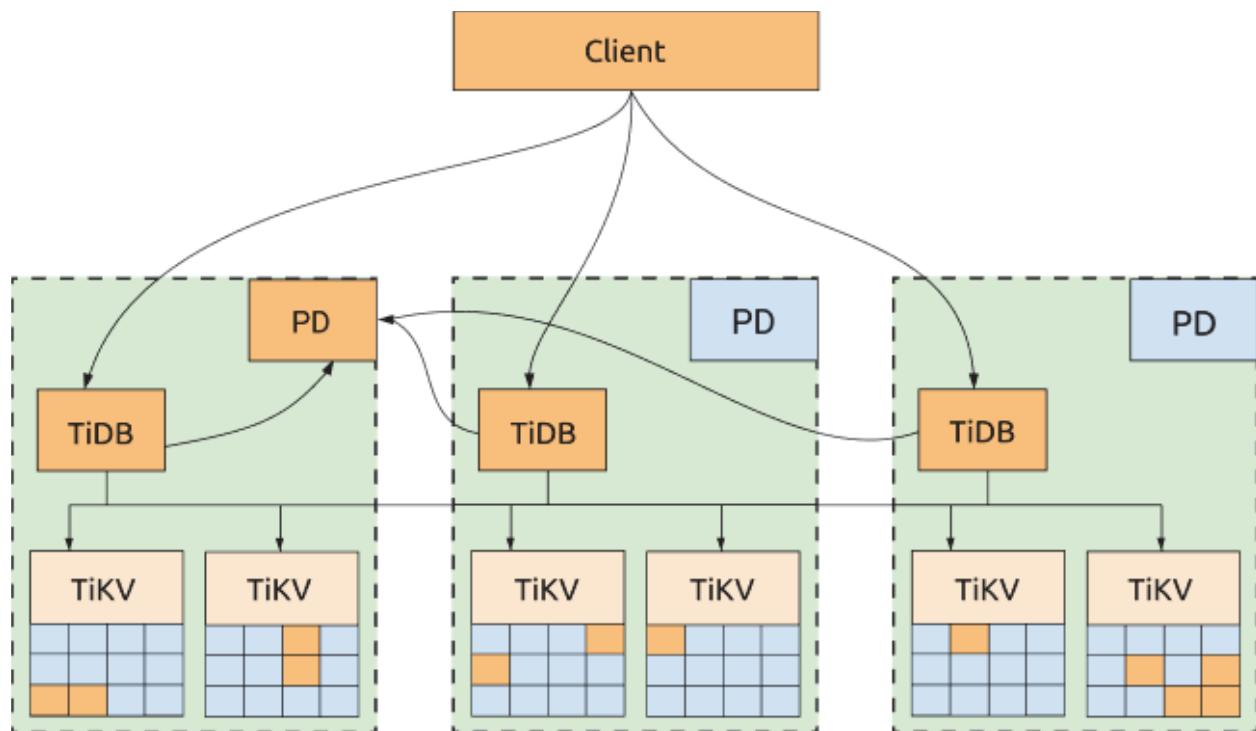


Figure 71: 3-DC Deployment Architecture

#### Advantages:

- All replicas are distributed among three DCs, with high availability and disaster recovery capability.
- No data will be lost if one DC is down ( $RPO = 0$ ).
- Even if one DC is down, the other two DCs will automatically start leader election and automatically resume services within a reasonable amount of time (within 20 seconds in most cases). See the following diagram for more information:

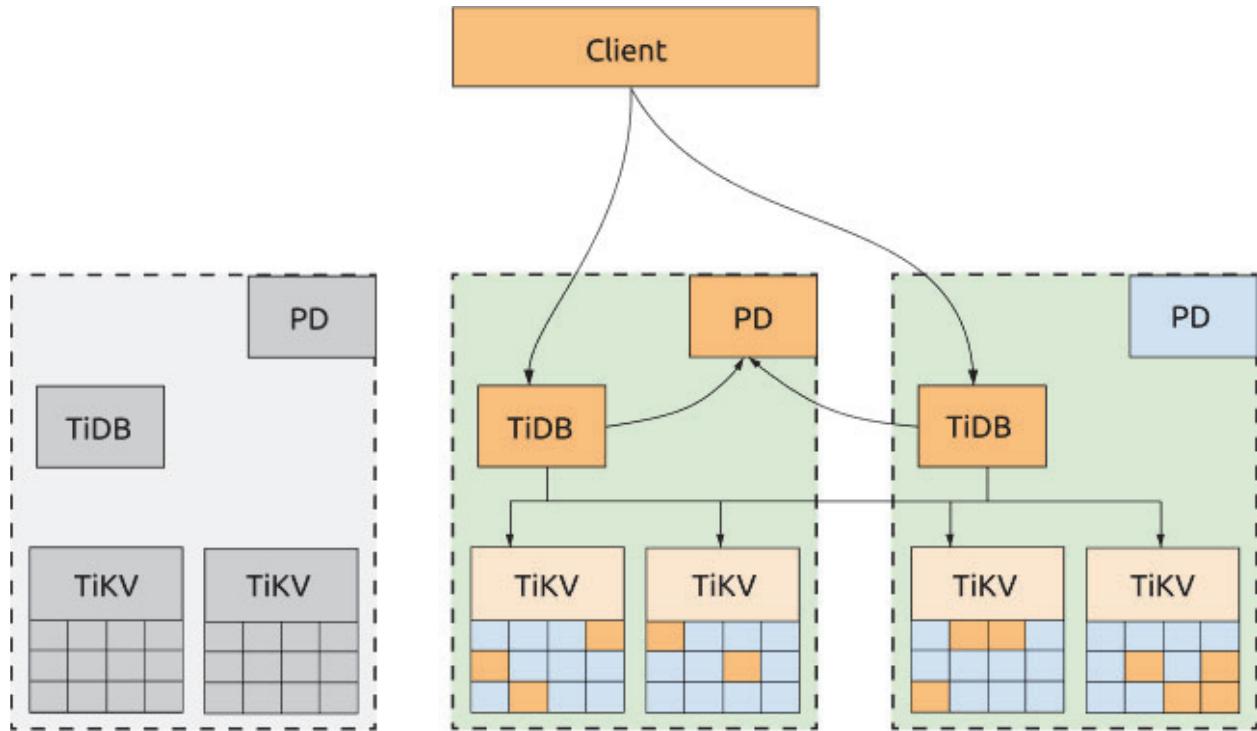


Figure 72: Disaster Recovery for 3-DC Deployment

#### Disadvantages:

The performance can be affected by the network latency.

- For writes, all the data has to be replicated to at least 2 DCs. Because TiDB uses 2-phase commit for writes, the write latency is at least twice the latency of the network between two DCs.
- The read performance will also be affected by the network latency if the leader is not in the same DC with the TiDB node that sends the read request.
- Each TiDB transaction needs to obtain TimeStamp Oracle (TSO) from the PD leader. So if the TiDB and PD leaders are not in the same DC, the performance of the transactions will also be affected by the network latency because each transaction with the write request has to obtain TSO twice.

#### 9.1.2.2 Optimized architecture

If not all of the three DCs need to provide services to the applications, you can dispatch all the requests to one DC and configure the scheduling policy to migrate all the TiKV Region leader and PD leader to the same DC. In this way, neither obtaining TSO nor reading TiKV Regions will be impacted by the network latency across DCs. If this DC is down, the PD leader and TiKV Region leader will be automatically elected in other surviving DCs, and you just need to switch the requests to the DCs that are still alive.

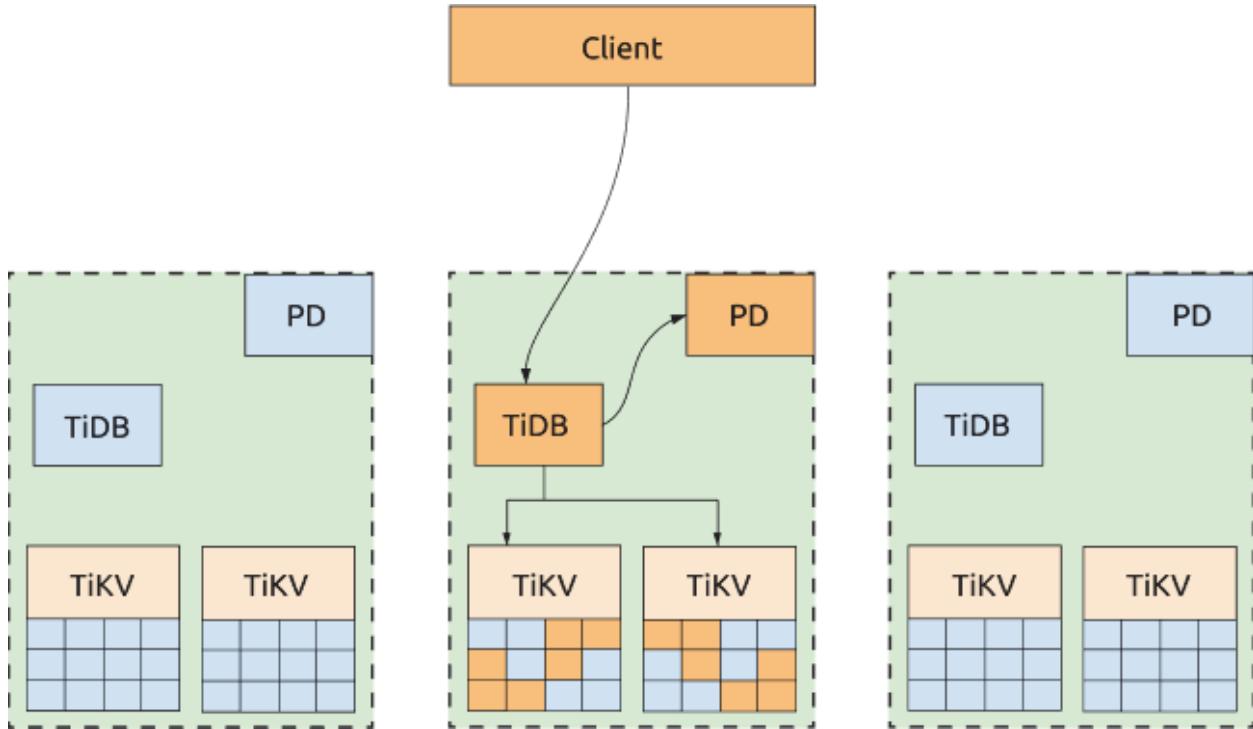


Figure 73: Read Performance Optimized 3-DC Deployment

### Advantages:

The cluster's read performance and the capability to get TSO are improved. A configuration template of scheduling policy is as follows:

```
-- Evicts all leaders of other DCs to the DC that provides services to the
 ↪ application.
config set label-property reject-leader LabelName labelValue

-- Migrates PD leaders and sets priority.
member leader transfer pdName1
member leader_priority pdName1 5
member leader_priority pdName2 4
member leader_priority pdName3 3
```

### Note:

Since TiDB 5.2, the `label-property` configuration is not supported by default. To set the replica policy, use the [placement rules](#).

## Disadvantages:

- Write scenarios are still affected by network latency across DCs. This is because Raft follows the majority protocol and all written data must be replicated to at least two DCs.
- The TiDB server that provides services is only in one DC.
- All application traffic is processed by one DC and the performance is limited by the network bandwidth pressure of that DC.
- The capability to get TSO and the read performance are affected by whether the PD server and TiKV server are up in the DC that processes application traffic. If these servers are down, the application is still affected by the cross-center network latency.

### 9.1.2.3 Deployment example

This section provides a topology example, and introduces TiKV labels and TiKV labels planning.

#### 9.1.2.3.1 Topology example

The following example assumes that three DCs (IDC1, IDC2, and IDC3) are located in one city; each IDC has two sets of racks and each rack has three servers. The example ignores the hybrid deployment or the scenario where multiple instances are deployed on one machine. The deployment of a TiDB cluster (three replicas) on three DCs in one city is as follows:

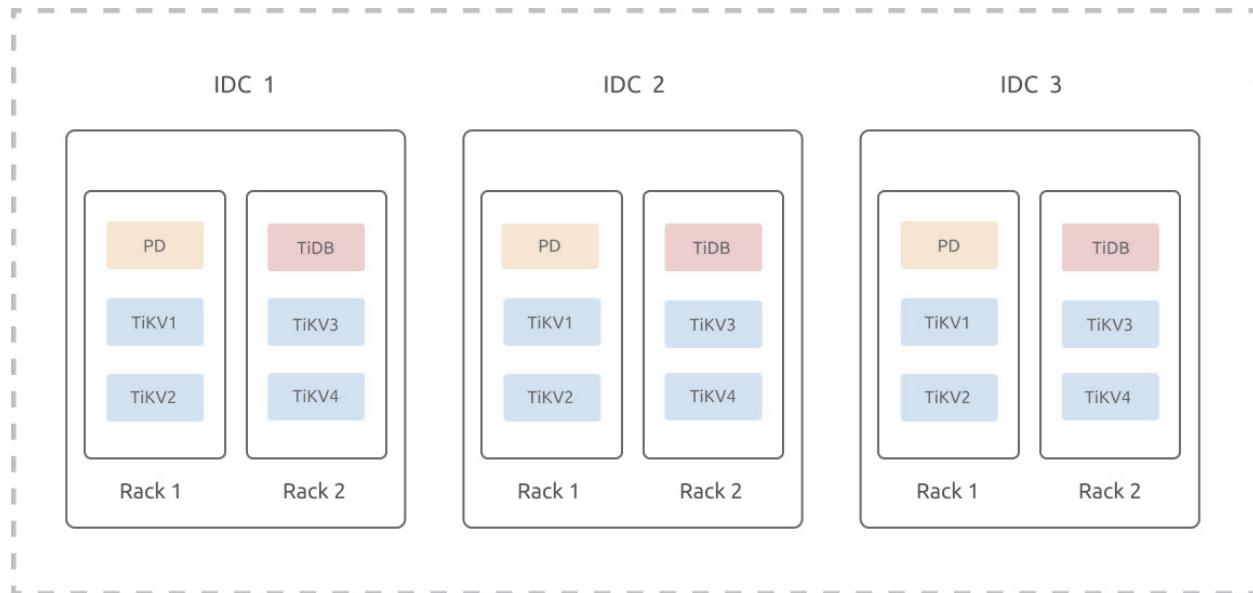


Figure 74: 3-DC in One City

### 9.1.2.3.2 TiKV labels

TiKV is a Multi-Raft system where data is divided into Regions and the size of each Region is 96 MB by default. Three replicas of each Region form a Raft group. For a TiDB cluster of three replicas, because the number of Region replicas is independent of the TiKV instance numbers, three replicas of a Region are only scheduled to three TiKV instances. This means that even if the cluster is scaled out to have N TiKV instances, it is still a cluster of three replicas.

Because a Raft group of three replicas tolerates only one replica failure, even if the cluster is scaled out to have N TiKV instances, this cluster still tolerates only one replica failure. Two failed TiKV instances might cause some Regions to lose replicas and the data in this cluster is no longer complete. SQL requests that access data from these Regions will fail. The probability of two simultaneous failures among N TiKV instances is much higher than the probability of two simultaneous failures among three TiKV instances. This means that the more TiKV instances the Multi-Raft system is scaled out to have, the less the availability of the system.

Because of the limitation described above, `label` is used to describe the location information of TiKV. The label information is refreshed to the TiKV startup configuration file with deployment or rolling upgrade operations. The started TiKV reports its latest label information to PD. Based on the user-registered label name (the label metadata) and the TiKV topology, PD optimally schedules Region replicas and improves the system availability.

### 9.1.2.3.3 TiKV labels planning example

To improve the availability and disaster recovery of the system, you need to design and plan TiKV labels according to your existing physical resources and the disaster recovery capability. You also need to configure in the cluster initialization configuration file according to the planned topology:

```
server_configs:
 pd:
 replication.location-labels: ["zone", "dc", "rack", "host"]

tikv_servers:
 - host: 10.63.10.30
 config:
 server.labels: { zone: "z1", dc: "d1", rack: "r1", host: "30" }
 - host: 10.63.10.31
 config:
 server.labels: { zone: "z1", dc: "d1", rack: "r1", host: "31" }
 - host: 10.63.10.32
 config:
 server.labels: { zone: "z1", dc: "d1", rack: "r2", host: "32" }
 - host: 10.63.10.33
 config:
```

```

 server.labels: { zone: "z1", dc: "d1", rack: "r2", host: "33" }
- host: 10.63.10.34
 config:
 server.labels: { zone: "z2", dc: "d1", rack: "r1", host: "34" }
- host: 10.63.10.35
 config:
 server.labels: { zone: "z2", dc: "d1", rack: "r1", host: "35" }
- host: 10.63.10.36
 config:
 server.labels: { zone: "z2", dc: "d1", rack: "r2", host: "36" }
- host: 10.63.10.37
 config:
 server.labels: { zone: "z2", dc: "d1", rack: "r2", host: "37" }
- host: 10.63.10.38
 config:
 server.labels: { zone: "z3", dc: "d1", rack: "r1", host: "38" }
- host: 10.63.10.39
 config:
 server.labels: { zone: "z3", dc: "d1", rack: "r1", host: "39" }
- host: 10.63.10.40
 config:
 server.labels: { zone: "z3", dc: "d1", rack: "r2", host: "40" }
- host: 10.63.10.41
 config:
 server.labels: { zone: "z3", dc: "d1", rack: "r2", host: "41" }

```

In the example above, `zone` is the logical availability zone layer that controls the isolation of replicas (three replicas in the example cluster).

Considering that the DC might be scaled out in the future, the three-layer label structure (`dc`, `rack`, `host`) is not directly adopted. Assuming that `d2`, `d3`, and `d4` are to be scaled out, you only need to scale out the DCs in the corresponding availability zone and scale out the racks in the corresponding DC.

If this three-layer label structure is directly adopted, after scaling out a DC, you might need to apply new labels and the data in TiKV needs to be rebalanced.

#### 9.1.2.4 High availability and disaster recovery analysis

The multiple DCs in one city deployment can guarantee that if one DC fails, the cluster can automatically recover services without manual intervention. Data consistency is also guaranteed. Note that scheduling policies are used to optimize performance, but when failure occurs, these policies prioritize availability over performance.

## 9.2 Three Data Centers in Two Cities Deployment

This document introduces the architecture and configuration of the three data centers (DC) in two cities deployment.

### 9.2.1 Overview

The architecture of three DCs in two cities is a highly available and disaster tolerant deployment solution that provides a production data center, a disaster recovery center in the same city, and a disaster recovery center in another city. In this mode, the three DCs in two cities are interconnected. If one DC fails or suffers from a disaster, other DCs can still operate well and take over the key applications or all applications. Compared with the multi-DC in one city deployment, this solution has the advantage of cross-city high availability and can survive city-level natural disasters.

The distributed database TiDB natively supports the three-DC-in-two-city architecture by using the Raft algorithm, and guarantees the consistency and high availability of data within a database cluster. Because the network latency across DCs in the same city is relatively low, the application traffic can be dispatched to two DCs in the same city, and the traffic load can be shared by these two DCs by controlling the distribution of TiKV Region leaders and PD leaders.

### 9.2.2 Deployment architecture

This section takes the example of Seattle and San Francisco to explain the deployment mode of three DCs in two cities for the distributed database of TiDB.

In this example, two DCs (IDC1 and IDC2) are located in Seattle and another DC (IDC3) is located in San Francisco. The network latency between IDC1 and IDC2 is lower than 3 milliseconds. The network latency between IDC3 and IDC1/IDC2 in Seattle is about 20 milliseconds (ISP dedicated network is used).

The architecture of the cluster deployment is as follows:

- The TiDB cluster is deployed to three DCs in two cities: IDC1 in Seattle, IDC2 in Seattle, and IDC3 in San Francisco.
- The cluster has five replicas, two in IDC1, two in IDC2, and one in IDC3. For the TiKV component, each rack has a label, which means that each rack has a replica.
- The Raft protocol is adopted to ensure consistency and high availability of data, which is transparent to users.

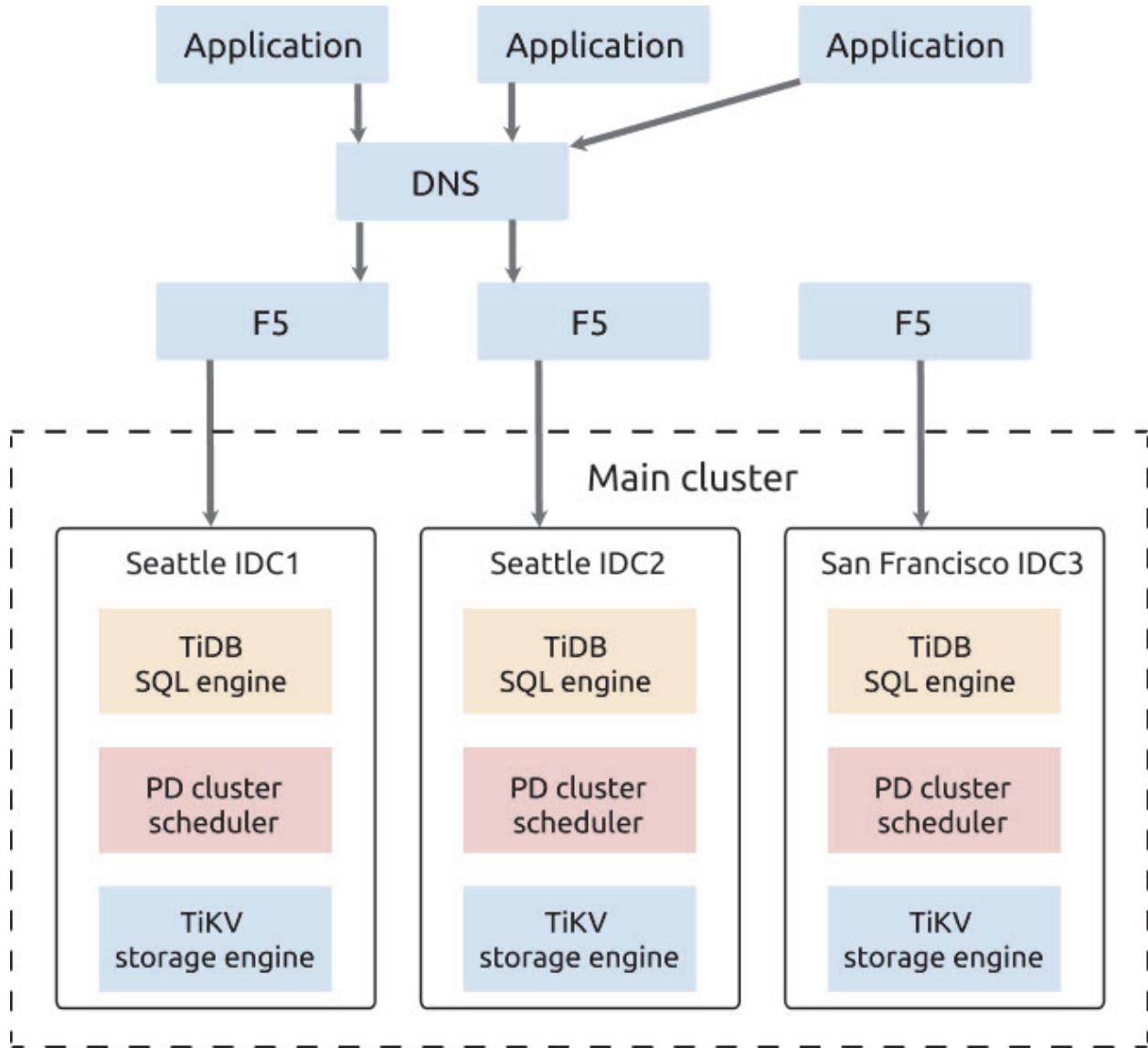


Figure 75: 3-DC-in-2-city architecture

This architecture is highly available. The distribution of Region leaders is restricted to the two DCs (IDC1 and IDC2) that are in the same city (Seattle). Compared with the three-DC solution in which the distribution of Region leaders is not restricted, this architecture has the following advantages and disadvantages:

- **Advantages**

- Region leaders are in DCs of the same city with low latency, so the write is faster.
- The two DCs can provide services at the same time, so the resources usage rate is higher.
- If one DC fails, services are still available and data safety is ensured.

- Disadvantages

- Because the data consistency is achieved by the Raft algorithm, when two DCs in the same city fail at the same time, only one surviving replica remains in the disaster recovery DC in another city (San Francisco). This cannot meet the requirement of the Raft algorithm that most replicas survive. As a result, the cluster can be temporarily unavailable. Maintenance staff needs to recover the cluster from the one surviving replica and a small amount of hot data that has not been replicated will be lost. But this case is a rare occurrence.
- Because the ISP dedicated network is used, the network infrastructure of this architecture has a high cost.
- Five replicas are configured in three DCs in two cities, data redundancy increases, which brings a higher storage cost.

#### 9.2.2.1 Deployment details

The configuration of the three DCs in two cities (Seattle and San Francisco) deployment plan is illustrated as follows:

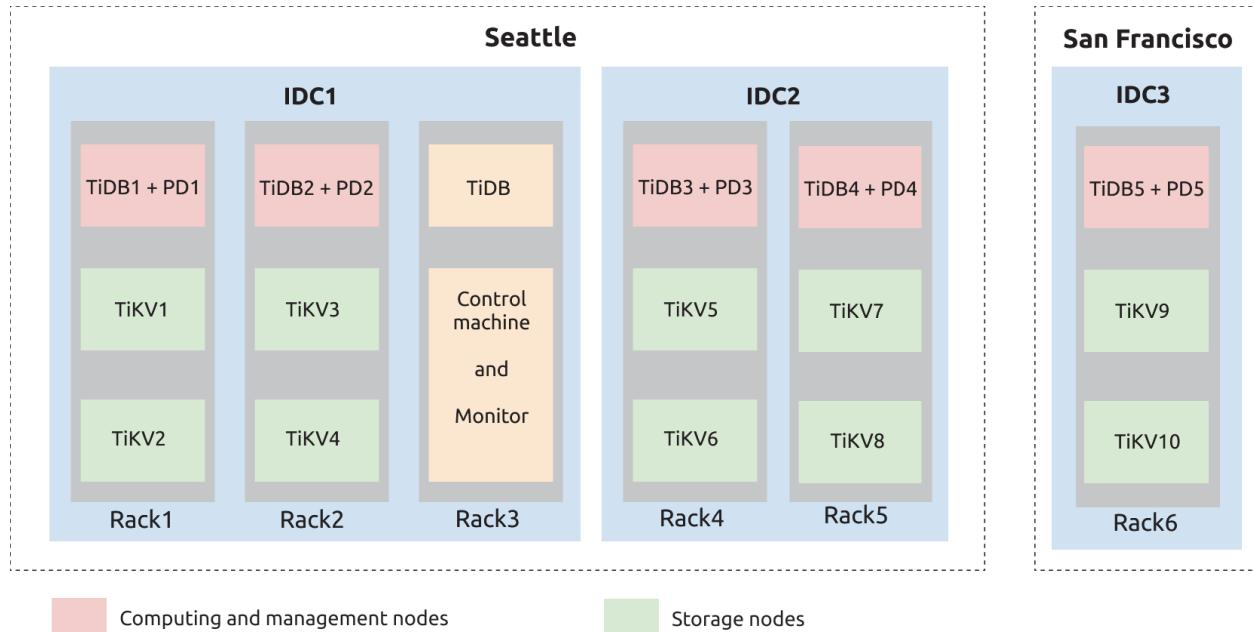


Figure 76: 3-DC-2-city

- From the illustration above, you can see that Seattle has two DCs: IDC1 and IDC2. IDC1 has three sets of racks: RAC1, RAC2, and RAC3. IDC2 has two racks: RAC4 and RAC5. The IDC3 DC in San Francisco has the RAC6 rack.
- From the RAC1 rack illustrated above, TiDB and PD services are deployed on the same server. Each of the two TiKV servers are deployed with two TiKV instances (tikv-server). This is similar to RAC2, RAC4, RAC5, and RAC6.

- The TiDB server, the control machine, and the monitoring server are on RAC3. The TiDB server is deployed for regular maintenance and backup. Prometheus, Grafana, and the restore tools are deployed on the control machine and monitoring machine.
- Another backup server can be added to deploy Drainer. Drainer saves binlog data to a specified location by outputting files, to achieve incremental backup.

### 9.2.3 Configuration

#### 9.2.3.1 Example

See the following `tiup topology.yaml` yaml file for example:

```
Global variables are applied to all deployments and used as the default
value of
the deployments if a specific deployment value is missing.
global
 user: "tidb"
 ssh_port: 22
 deploy_dir: "/data/tidb_cluster/tidb-deploy"
 data_dir: "/data/tidb_cluster/tidb-data"

server_configs:
 tikv:
 server.grpc-compression-type: gzip
 pd:
 replication.location-labels: ["dc", "zone", "rack", "host"]
 schedule.tolerant-size-ratio: 20.0

pd_servers:
 - host: 10.63.10.10
 name: "pd-10"
 - host: 10.63.10.11
 name: "pd-11"
 - host: 10.63.10.12
 name: "pd-12"
 - host: 10.63.10.13
 name: "pd-13"
 - host: 10.63.10.14
 name: "pd-14"

tidb_servers:
 - host: 10.63.10.10
 - host: 10.63.10.11
 - host: 10.63.10.12
 - host: 10.63.10.13
 - host: 10.63.10.14
```

```

tikv_servers:
 - host: 10.63.10.30
 config:
 server.labels: { dc: "1", zone: "1", rack: "1", host: "30" }
 - host: 10.63.10.31
 config:
 server.labels: { dc: "1", zone: "2", rack: "2", host: "31" }
 - host: 10.63.10.32
 config:
 server.labels: { dc: "2", zone: "3", rack: "3", host: "32" }
 - host: 10.63.10.33
 config:
 server.labels: { dc: "2", zone: "4", rack: "4", host: "33" }
 - host: 10.63.10.34
 config:
 server.labels: { dc: "3", zone: "5", rack: "5", host: "34" }
 raftstore.raft-min-election-timeout-ticks: 1000
 raftstore.raft-max-election-timeout-ticks: 1200

monitoring_servers:
 - host: 10.63.10.60

grafana_servers:
 - host: 10.63.10.60

alertmanager_servers:
 - host: 10.63.10.60

```

### 9.2.3.2 Labels design

In the deployment of three DCs in two cities, the label design requires taking availability and disaster recovery into account. It is recommended that you define the four levels (`dc`, `zone`, `rack`, `host`) based on the physical structure of the deployment.

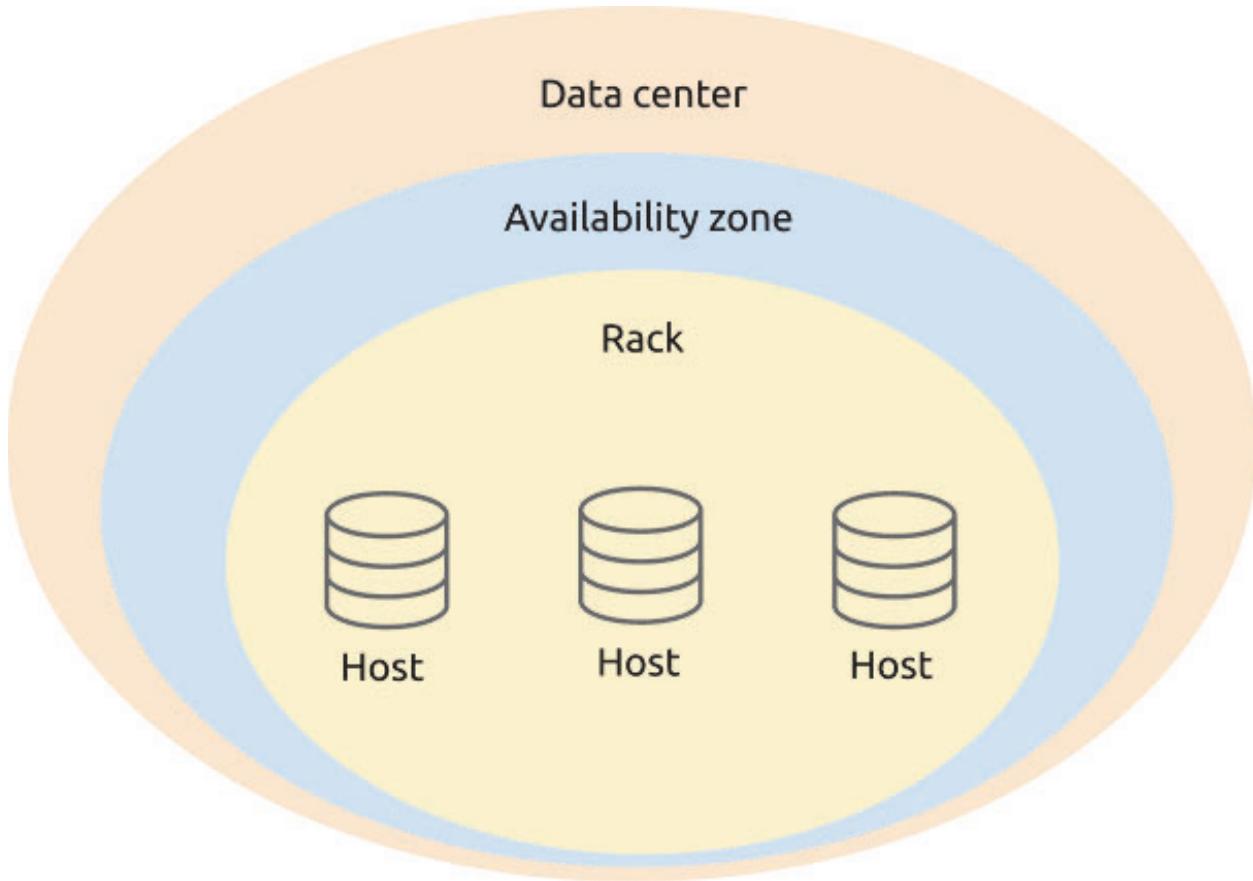


Figure 77: Label logical definition

In the PD configuration, add level information of TiKV labels:

```
server_configs:
 pd:
 replication.location-labels: ["dc", "zone", "rack", "host"]
```

The configuration of `tikv_servers` is based on the label information of the real physical deployment location of TiKV, which makes it easier for PD to perform global management and scheduling.

```
tikv_servers:
 - host: 10.63.10.30
 config:
 server.labels: { dc: "1", zone: "1", rack: "1", host: "30" }
 - host: 10.63.10.31
 config:
 server.labels: { dc: "1", zone: "2", rack: "2", host: "31" }
 - host: 10.63.10.32
 config:
```

```

server.labels: { dc: "2", zone: "3", rack: "3", host: "32" }
- host: 10.63.10.33
 config:
 server.labels: { dc: "2", zone: "4", rack: "4", host: "33" }
- host: 10.63.10.34
 config:
 server.labels: { dc: "3", zone: "5", rack: "5", host: "34" }

```

### 9.2.3.3 Optimize parameter configuration

In the deployment of three DCs in two cities, to optimize performance, you need to not only configure regular parameters, but also adjust component parameters.

- Enable gRPC message compression in TiKV. Because data of the cluster is transmitted in the network, you can enable the gRPC message compression to lower the network traffic.

```
server.grpc-compression-type: gzip
```

- Adjust the PD balance buffer size and increase the tolerance of PD. Because PD calculates the score of each object according to the situation of the node as the basis for scheduling, when the difference between the scores of leaders (or Regions) of two stores is less than the specified multiple of the Region size, PD believes the balance is achieved.

```
schedule.tolerant-size-ratio: 20.0
```

- Optimize the network configuration of the TiKV node in another city (San Francisco). Modify the following TiKV parameters for IDC3 (alone) in San Francisco and try to prevent the replica in this TiKV node from participating in the Raft election.

```
raftstore.raft-min-election-timeout-ticks: 1000
raftstore.raft-max-election-timeout-ticks: 1200
```

- Configure scheduling. After the cluster is enabled, use the `tiup ctl pd` tool to modify the scheduling policy. Modify the number of TiKV Raft replicas. Configure this number as planned. In this example, the number of replicas is five.

```
config set max-replicas 5
```

- Forbid scheduling the Raft leader to IDC3. Scheduling the Raft leader to in another city (IDC3) causes unnecessary network overhead between IDC1/IDC2 in Seattle and IDC3 in San Francisco. The network bandwidth and latency also affect performance of the TiDB cluster.

```
config set label-property reject-leader dc 3
```

**Note:**

Since TiDB 5.2, the `label-property` configuration is not supported by default. To set the replica policy, use the [placement rules](#).

- Configure the priority of PD. To avoid the situation where the PD leader is in another city (IDC3), you can increase the priority of local PD (in Seattle) and decrease the priority of PD in another city (San Francisco). The larger the number, the higher the priority.

```
member leader_priority PD-10 5
member leader_priority PD-11 5
member leader_priority PD-12 5
member leader_priority PD-13 5
member leader_priority PD-14 1
```

## 9.3 Two Data Centers in One City Deployment

This document introduces the deployment mode of two data centers (DCs) in one city, including the architecture, configuration, how to enable this deployment mode, and how to use replicas in this mode.

In an on-premises environment, TiDB usually adopts the multi-data-center deployment solution to ensure high availability and disaster recovery capability. The multi-data-center deployment solution includes multiple deployment modes, such as three data centers in two cities and three data centers in one city. This document introduces the deployment mode of two data centers in one city. Deployed in this mode, TiDB can also meet the requirements of high availability and disaster recovery, with a lower cost. This deployment solution adopts Data Replication Auto Synchronous mode, or the DR Auto-Sync mode.

Under the mode of two data centers in one city, the two data centers are less than 50 kilometers apart. They are usually located in the same city or in two adjacent cities. The network latency between the two data centers is lower than 1.5 milliseconds and the bandwidth is higher than 10 Gbps.

### 9.3.1 Deployment architecture

This section takes the example of a city where two data centers IDC1 and IDC2 are located respectively in the east and west.

The architecture of the cluster deployment is as follows:

- The TiDB cluster is deployed to two DCs in one city: the primary IDC1 in the east, and the disaster recovery (DR) IDC2 in the west.
- The cluster has 4 replicas: 2 Voter replicas in IDC1, 1 Voter replica and 1 Learner replica in IDC2. For the TiKV component, each rack has a proper label.
- The Raft protocol is adopted to ensure consistency and high availability of data, which is transparent to users.

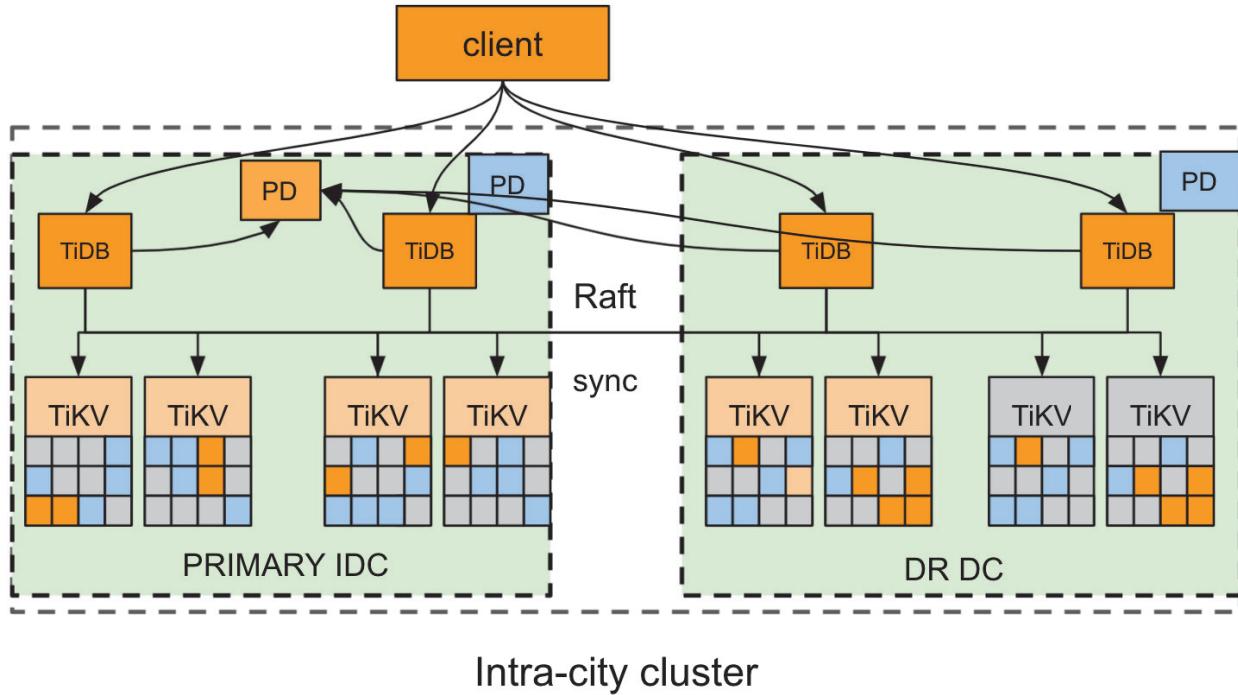


Figure 78: 2-DC-in-1-city architecture

This deployment solution defines three statuses to control and identify the replication status of the cluster, which restricts the replication mode of TiKV. The replication mode of the cluster can automatically and adaptively switch between the three statuses. For details, see the [Status switch](#) section.

- **sync:** Synchronous replication mode. In this mode, at least one replica in the disaster recovery (DR) data center synchronizes with the primary data center. The Raft algorithm ensures that each log is replicated to the DR based on the label.
- **async:** Asynchronous replication mode. In this mode, the DR data center is not fully synchronized with the primary data center. The Raft algorithm follows the majority protocol to replicate logs.
- **sync-recover:** Synchronous recovery mode. In this mode, the DR data center is not fully synchronized with the primary data center. Raft gradually switches to the label replication mode and then reports the label information to PD.

## 9.3.2 Configuration

### 9.3.2.1 Example

The following `tiup topology.yaml` example file is a typical topology configuration for the two data centers in one city deployment mode:

```
Global variables are applied to all deployments and used as the default
→ value of
the deployments if a specific deployment value is missing.
global:
 user: "tidb"
 ssh_port: 22
 deploy_dir: "/data/tidb_cluster/tidb-deploy"
 data_dir: "/data/tidb_cluster/tidb-data"
server_configs:
 pd:
 replication.location-labels: ["zone", "rack", "host"]
pd_servers:
 - host: 10.63.10.10
 name: "pd-10"
 - host: 10.63.10.11
 name: "pd-11"
 - host: 10.63.10.12
 name: "pd-12"
tidb_servers:
 - host: 10.63.10.10
 - host: 10.63.10.11
 - host: 10.63.10.12
tikv_servers:
 - host: 10.63.10.30
 config:
 server.labels: { zone: "east", rack: "east-1", host: "30" }
 - host: 10.63.10.31
 config:
 server.labels: { zone: "east", rack: "east-2", host: "31" }
 - host: 10.63.10.32
 config:
 server.labels: { zone: "west", rack: "west-1", host: "32" }
 - host: 10.63.10.33
 config:
 server.labels: { zone: "west", rack: "west-2", host: "33" }
monitoring_servers:
 - host: 10.63.10.60
grafana_servers:
 - host: 10.63.10.60
```

```
alertmanager_servers:
 - host: 10.63.10.60
```

### 9.3.2.2 Placement Rules

To deploy a cluster based on the planned topology, you need to use [placement rules](#) to determine the locations of the cluster replicas. If 4 replicas and 2 Voter replicas are at the primary center and 1 Voter replica and 1 Learner replica are at the DR center, you can use the placement rules to configure the replicas as follows:

```
cat rule.json
[
 {
 "group_id": "pd",
 "id": "zone-east",
 "start_key": "",
 "end_key": "",
 "role": "voter",
 "count": 2,
 "label_constraints": [
 {
 "key": "zone",
 "op": "in",
 "values": [
 "east"
]
 }
],
 "location_labels": [
 "zone",
 "rack",
 "host",
]
 },
 {
 "group_id": "pd",
 "id": "zone-west",
 "start_key": "",
 "end_key": "",
 "role": "voter",
 "count": 1,
 "label_constraints": [
 {
 "key": "zone",
 "op": "in",
 "values": [
 "west"
]
 }
],
 "location_labels": [
 "zone",
 "rack",
 "host",
]
 }
]
```

```

 "values": [
 "west"
]
 }
],
"location_labels": [
 "zone",
 "rack",
 "host"
]
},
{
 "group_id": "pd",
 "id": "zone-west",
 "start_key": "",
 "end_key": "",
 "role": "learner",
 "count": 1,
 "label_constraints": [
 {
 "key": "zone",
 "op": "in",
 "values": [
 "west"
]
 }
],
 "location_labels": [
 "zone",
 "rack",
 "host"
]
}
]
]

```

### 9.3.2.3 Enable the DR Auto-Sync mode

The replication mode is controlled by PD. When deploying a cluster, you can configure the replication mode in the PD configuration file. For example:

```
[replication-mode]
replication-mode = "dr-auto-sync"
[replication-mode.dr-auto-sync]
label-key = "zone"
primary = "east"
```

```

dr = "west"
primary-replicas = 2
dr-replicas = 1
wait-store-timeout = "1m"
wait-sync-timeout = "1m"

```

In the configuration above:

- `replication-mode` is the replication mode to be enabled. In the above example, it is set to `dr-auto-sync`. By default, the majority protocol is used.
- `label-key` is used to distinguish different data centers and needs to match placement rules. In this example, the primary data center is “east” and the DR data center is “west”.
- `primary-replicas` is the number of Voter replicas in the primary data center.
- `dr-replicas` is the number of Voter replicas in the DR data center.
- `wait-store-timeout` is the waiting time for switching to asynchronous replication mode when network isolation or failure occurs. If the time of network failure exceeds the waiting time, asynchronous replication mode is enabled. The default waiting time is 60 seconds.

To check the current replication status of the cluster, use the following API:

```
curl http://pd_ip:pd_port/pd/api/v1/replication_mode/status
```

```
{
 "mode": "dr-auto-sync",
 "dr-auto-sync": {
 "label-key": "zone",
 "state": "sync"
 }
}
```

#### 9.3.2.3.1 Status switch

The replication mode of a cluster can automatically and adaptively switch between three statuses:

- When the cluster is normal, the synchronous replication mode is enabled to maximize the data integrity of the disaster recovery data center.
- When the network connection between the two data centers fails or the DR data center breaks down, after a pre-set protective interval, the cluster enables the asynchronous replication mode to ensure the availability of the application.
- When the network reconnects or the DR data center recovers, the TiKV node joins the cluster again and gradually replicates the data. Finally, the cluster switches to the synchronous replication mode.

The details for the status switch are as follows:

1. **Initialization:** At the initialization stage, the cluster is in the synchronous replication mode. PD sends the status information to TiKV, and all TiKV nodes strictly follow the synchronous replication mode to work.
2. **Switch from sync to async:** PD regularly checks the heartbeat information of TiKV to judge whether the TiKV node fails or is disconnected. If the number of failed nodes exceeds the number of replicas of the primary data center (**primary-replicas**) and the DR data center (**dr-replicas**), the synchronous replication mode can no longer serve the data replication and it is necessary to switch the status. When the failure or disconnect time exceeds the time set by `wait-store-timeout`, PD switches the status of the cluster to the async mode. Then PD sends the status of async to all TiKV nodes, and the replication mode for TiKV switches from two-center replication to the native Raft majority.
3. **Switch from async to sync:** PD regularly checks the heartbeat information of TiKV to judge whether the TiKV node is reconnected. If the number of failed nodes is less than the number of replicas of the primary data center (**primary-replicas**) and the DR data center (**dr-replicas**), the synchronous replication mode can be enabled again. PD first switches the status of the cluster to sync-recover and sends the status information to all TiKV nodes. All Regions of TiKV gradually switch to the two-data-center synchronous replication mode and then report the heartbeat information to PD. PD records the status of TiKV Regions and calculates the recovery progress. When all TiKV Regions finish the switching, PD switches the replication mode to sync.

#### 9.3.2.4 Disaster recovery

This section introduces the disaster recovery solution of the two data centers in one city deployment.

When a disaster occurs to a cluster in the synchronous replication mode, you can perform data recovery with  $RPO = 0$ :

- If the primary data center fails and most of the Voter replicas are lost, but complete data exists in the DR data center, the lost data can be recovered from the DR data center. At this time, manual intervention is required with professional tools. You can contact the TiDB team for a recovery solution.
- If the DR center fails and a few Voter replicas are lost, the cluster automatically switches to the asynchronous replication mode.

When a disaster occurs to a cluster that is not in the synchronous replication mode and you cannot perform data recovery with  $RPO = 0$ :

- If most of the Voter replicas are lost, manual intervention is required with professional tools. You can contact the TiDB team for a recovery solution.

## 9.4 Read Historical Data

### 9.4.1 Use Stale Read (Recommended)

#### 9.4.1.1 Usage Scenarios of Stale Read

This document describes the usage scenarios of Stale Read. Stale Read is a mechanism that TiDB applies to read historical versions of data stored in TiDB. Using this mechanism, you can read the corresponding historical data of a specific point in time or within a specified time range, and thus save the latency brought by data replication between storage nodes.

When you are using Stale Read, TiDB will randomly select a replica for data reading, which means that all replicas are available for data reading. If your application cannot tolerate reading non-real-time data, do not use Stale Read; otherwise, the data read from the replica might not be the latest data written into TiDB.

##### 9.4.1.1.1 Scenario examples

- Scenario one: If a transaction only involves read operations and is tolerant of data staleness to some extent, you can use Stale Read to get historical data. Using Stale Read, TiDB makes the query requests sent to any replica at the expense of some real-time performance, and thus increases the throughput of query executions. Especially in some scenarios where small tables are queried, if strongly consistent reads are used, leader might be concentrated on a certain storage node, causing the query pressure to be concentrated on that node as well. Therefore, that node might become a bottleneck for the whole query. Stale Read, however, can improve the overall query throughput and significantly improve the query performance.
- Scenario two: In some scenarios of geo-distributed deployment, if strongly consistent follower reads are used, to make sure that the data read from the Followers is consistent with that stored in the Leader, TiDB requests `Readindex` from different data centers for verification, which increases the access latency for the whole query process. With Stale Read, TiDB accesses the replica in the current data center to read the corresponding data at the expense of some real-time performance, which avoids network latency brought by cross-center connection and reduces the access latency for the entire query. For more information, see [Local Read under Three Data Centers Deployment](#).

##### 9.4.1.1.2 Usages

In TiDB, you can specify either an exact point in time or a time range when performing stale reads:

- Specifying an exact point in time (recommended): If you need TiDB to read data that follows the global transaction consistency from a specific point in time without damaging the isolation level, you can specify the corresponding timestamp of that point in time in the query statement. For detailed usage, see [AS OF TIMESTAMP Clause](#).

- Specifying a time range: If you need TiDB to read data as new as possible within a time range without damaging the isolation level, you can specify the time range in the query statement. Then, TiDB selects a suitable timestamp within the specified time range to read the corresponding data. “Suitable” means there are no transactions that start before this timestamp and have not been committed on the accessed replica, that is, TiDB can perform read operations on the accessed replica and the read operations are not blocked. For detailed usage, refer to the introduction of the [AS OF TIMESTAMP clause](#) and the [TiDB\\_BOUNDED\\_STALENESS function](#).

#### 9.4.1.2 Read Historical Data Using the AS OF TIMESTAMP Clause

This document describes how to perform the [Stale Read](#) feature using the AS OF → TIMESTAMP clause to read historical data in TiDB, including specific usage examples and strategies for saving historical data.

##### Warning:

Currently, you cannot use Stale Read together with TiFlash. If your SQL query contains the AS OF TIMESTAMP clause and TiDB might read data from TiFlash replicas, you might encounter an error with a message like `ERROR → 1105 (HY000): stale requests require tikv backend`.

To fix the problem, disable TiFlash replicas for your Stale Read query. To do that, perform one of the following operations:

- Use the `set session tidb_isolation_read_engines='tidb,tikv'` variable.
- Use the [hint](#) to enforce TiDB to read data from TiKV.

TiDB supports reading historical data through a standard SQL interface, which is the AS OF TIMESTAMP SQL clause, without the need for special clients or drivers. After data is updated or deleted, you can read the historical data before the update or deletion using this SQL interface.

##### Note:

When reading historical data, TiDB returns the data with the old table structure even if the current table structure is different.

#### 9.4.1.2.1 Syntax

You can use the AS OF TIMESTAMP clause in the following three ways:

- `SELECT ... FROM ... AS OF TIMESTAMP`
- `START TRANSACTION READ ONLY AS OF TIMESTAMP`
- `SET TRANSACTION READ ONLY AS OF TIMESTAMP`

If you want to specify an exact point of time, you can set a datetime value or use a time function in the `AS OF TIMESTAMP` clause. The format of datetime is like “2016-10-08 16:45:26.999”, with millisecond as the minimum time unit, but for most of the time, the time unit of second is enough for specifying a datetime, such as “2016-10-08 16:45:26”. You can also get the current time to the millisecond using the `NOW(3)` function. If you want to read the data of several seconds ago, it is **recommended** to use an expression such as `NOW() - INTERVAL 10 SECOND`.

If you want to specify a time range, you can use the `TIDB_BOUNDED_STALENESS()` function in the clause. When this function is used, TiDB selects a suitable timestamp within the specified time range. “Suitable” means there are no transactions that start before this timestamp and have not been committed on the accessed replica, that is, TiDB can perform read operations on the accessed replica and the read operations are not blocked. You need to use `TIDB_BOUNDED_STALENESS(t1, t2)` to call this function. `t1` and `t2` are the two ends of the time range, which can be specified using either datetime values or time functions.

Here are some examples of the `AS OF TIMESTAMP` clause:

- `AS OF TIMESTAMP '2016-10-08 16:45:26'`: Tells TiDB to read the latest data stored at 16:45:26 on October 8, 2016.
- `AS OF TIMESTAMP NOW() - INTERVAL 10 SECOND`: Tells TiDB to read the latest data stored 10 seconds ago.
- `AS OF TIMESTAMP TIDB_BOUNDED_STALENESS('2016-10-08 16:45:26', '2016-10-08 → 16:45:29')`: Tells TiDB to read the data as new as possible within the time range of 16:45:26 to 16:45:29 on October 8, 2016.
- `AS OF TIMESTAMP TIDB_BOUNDED_STALENESS(NOW() - INTERVAL 20 SECOND, NOW() → )`: Tells TiDB to read the data as new as possible within the time range of 20 seconds ago to the present.

#### Note:

In addition to specifying a timestamp, the most common use of the `AS OF → TIMESTAMP` clause is to read data that is several seconds old. If this approach is used, it is recommended to read historical data older than 5 seconds.

You need to deploy the NTP service for your TiDB and PD nodes when you use Stale Read. This avoids the situation where the specified timestamp used by TiDB goes ahead of the latest TSO allocating progress (such as a timestamp several seconds ahead), or is later than the GC safe point timestamp. When the specified timestamp goes beyond the service scope, TiDB returns an error.

#### 9.4.1.2.2 Usage examples

This section describes different ways to use the AS OF TIMESTAMP clause with several examples. It first introduces how to prepare the data for recovery, and then shows how to use AS OF TIMESTAMP in SELECT, START TRANSACTION READ ONLY AS OF TIMESTAMP, and SET TRANSACTION READ ONLY AS OF TIMESTAMP respectively.

Prepare data sample

To prepare data for recovery, create a table first and insert several rows of data:

```
create table t (c int);
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
insert into t values (1), (2), (3);
```

```
Query OK, 3 rows affected (0.00 sec)
```

View the data in the table:

```
select * from t;
```

```
+---+
| c |
+---+
| 1 |
| 2 |
| 3 |
+---+
3 rows in set (0.00 sec)
```

View the current time:

```
select now();
```

```
+-----+
| now() |
+-----+
| 2021-05-26 16:45:26 |
+-----+
1 row in set (0.00 sec)
```

Update the data in a row:

```
update t set c=22 where c=2;
```

```
Query OK, 1 row affected (0.00 sec)
```

Confirm that the data of the row is updated:

```
select * from t;
```

```
+----+
| c |
+----+
| 1 |
| 22 |
| 3 |
+----+
3 rows in set (0.00 sec)
```

Read historical data using the SELECT statement

You can use the `SELECT ... FROM ... AS OF TIMESTAMP` statement to read data from a time point in the past.

```
select * from t as of timestamp '2021-05-26 16:45:26';
```

```
+----+
| c |
+----+
| 1 |
| 2 |
| 3 |
+----+
3 rows in set (0.00 sec)
```

### Note:

When reading multiple tables using one `SELECT` statement, you need to make sure that the format of `TIMESTAMP EXPRESSIONS` is consistent. For example, `select * from t as of timestamp NOW() - INTERVAL 2 SECOND, ↪ c as of timestamp NOW() - INTERVAL 2 SECOND;`. In addition, you must specify the `AS OF` information for the relevant table in the `SELECT` statement; otherwise, the `SELECT` statement reads the latest data by default.

Read historical data using the `START TRANSACTION READ ONLY AS OF TIMESTAMP` statement

You can use the `START TRANSACTION READ ONLY AS OF TIMESTAMP` statement to start a read-only transaction based on a time point in the past. The transaction reads historical data of the given time.

```
start transaction read only as of timestamp '2021-05-26 16:45:26';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select * from t;
```

```
+---+
| c |
+---+
| 1 |
| 2 |
| 3 |
+---+
3 rows in set (0.00 sec)
```

```
commit;
```

```
Query OK, 0 rows affected (0.00 sec)
```

After the transaction is committed, you can read the latest data.

```
select * from t;
```

```
+---+
| c |
+---+
| 1 |
| 22 |
| 3 |
+---+
3 rows in set (0.00 sec)
```

### Note:

If you start a transaction with the statement `START TRANSACTION READ ONLY AS OF TIMESTAMP`, it is a read-only transaction. Write operations are rejected in this transaction.

Read historical data using the `SET TRANSACTION READ ONLY AS OF TIMESTAMP` statement

You can use the `SET TRANSACTION READ ONLY AS OF TIMESTAMP` statement to set the next transaction as a read-only transaction based on a specified time point in the past. The transaction reads historical data of the given time.

```
set transaction read only as of timestamp '2021-05-26 16:45:26';
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
begin;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
select * from t;
```

```
+----+
| c |
+----+
| 1 |
| 2 |
| 3 |
+----+
3 rows in set (0.00 sec)
```

```
commit;
```

```
Query OK, 0 rows affected (0.00 sec)
```

After the transaction is committed, you can read the latest data.

```
select * from t;
```

```
+----+
| c |
+----+
| 1 |
| 22 |
| 3 |
+----+
3 rows in set (0.00 sec)
```

### Note:

If you start a transaction with the statement `SET TRANSACTION READ ONLY → AS OF TIMESTAMP`, it is a read-only transaction. Write operations are rejected in this transaction.

## 9.4.2 Read Historical Data Using the System Variable `tidb_snapshot`

This document describes how to read data from the history versions using the system variable `tidb_snapshot`, including specific usage examples and strategies for saving historical data.

### Note:

You can also use the **Stale Read** feature to read historical data, which is more recommended.

### 9.4.2.1 Feature description

TiDB implements a feature to read history data using the standard SQL interface directly without special clients or drivers.

### Note:

- Even when data is updated or removed, its history versions can be read using the SQL interface.
- When reading historical data, TiDB returns the data with the old table structure even if the current table structure is different.

### 9.4.2.2 How TiDB reads data from history versions

The `tidb_snapshot` system variable is introduced to support reading history data. About the `tidb_snapshot` variable:

- The variable is valid in the `SESSION` scope.
- Its value can be modified using the `SET` statement.
- The data type for the variable is `text`.
- The variable accepts TSO (Timestamp Oracle) and `datetime`. TSO is a globally unique time service, which is obtained from PD. The acceptable `datetime` format is “`2016-10-08 16:45:26.999`”. Generally, the `datetime` can be set using second precision, for example “`2016-10-08 16:45:26`”.
- When the variable is set, TiDB creates a Snapshot using its value as the timestamp, just for the data structure and there is no any overhead. After that, all the `SELECT` operations will read data from this Snapshot.

### Note:

Because the timestamp in TiDB transactions is allocated by Placement Driver (PD), the version of the stored data is also marked based on the timestamp allocated by PD. When a Snapshot is created, the version number is based on the value of the `tidb_snapshot` variable. If there is a large difference between the local time of the TiDB server and the PD server, use the time of the PD server.

After reading data from history versions, you can read data from the latest version by ending the current Session or using the `SET` statement to set the value of the `tidb_snapshot` variable to “” (empty string).

#### 9.4.2.3 How TiDB manages the data versions

TiDB implements Multi-Version Concurrency Control (MVCC) to manage data versions. The history versions of data are kept because each update/removal creates a new version of the data object instead of updating/removing the data object in-place. But not all the versions are kept. If the versions are older than a specific time, they will be removed completely to reduce the storage occupancy and the performance overhead caused by too many history versions.

In TiDB, Garbage Collection (GC) runs periodically to remove the obsolete data versions. For GC details, see [TiDB Garbage Collection \(GC\)](#)

Pay special attention to the following:

- `tidb_gc_life_time`: This system variable is used to configure the retention time of earlier modifications (default: 10m0s).
- The output of `SELECT * FROM mysql.tidb WHERE variable_name = 'tikv_gc_safe_point' ↵ '`. This is the current `safePoint` where you can read historical data up to. It is updated every time the garbage collection process is run.

#### 9.4.2.4 Example

1. At the initial stage, create a table and insert several rows of data:

```
mysql> create table t (c int);
Query OK, 0 rows affected (0.01 sec)

mysql> insert into t values (1), (2), (3);
Query OK, 3 rows affected (0.00 sec)
```

2. View the data in the table:

```
mysql> select * from t;
+---+
| c |
+---+
| 1 |
| 2 |
| 3 |
+---+
3 rows in set (0.00 sec)
```

3. View the timestamp of the table:

```
mysql> select now();
+-----+
| now() |
+-----+
| 2016-10-08 16:45:26 |
+-----+
1 row in set (0.00 sec)
```

4. Update the data in one row:

```
mysql> update t set c=22 where c=2;
Query OK, 1 row affected (0.00 sec)
```

5. Make sure the data is updated:

```
mysql> select * from t;
+---+
| c |
+---+
| 1 |
| 22 |
| 3 |
+---+
3 rows in set (0.00 sec)
```

6. Set the `tidb_snapshot` variable whose scope is Session. The variable is set so that the latest version before the value can be read.

**Note:**

In this example, the value is set to be the time before the update operation.

```
mysql> set @@tidb_snapshot="2016-10-08 16:45:26";
Query OK, 0 rows affected (0.00 sec)
```

**Note:**

You should use @@ instead of @ before `tidb_snapshot` because @@ is used to denote the system variable while @ is used to denote the user variable.

**Result:** The read from the following statement is the data before the update operation, which is the history data.

```
mysql> select * from t;
+---+
| c |
+---+
| 1 |
| 2 |
| 3 |
+---+
3 rows in set (0.00 sec)
```

- Set the `tidb_snapshot` variable to be “” (empty string) and you can read the data from the latest version:

```
mysql> set @@tidb_snapshot="";
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from t;
+---+
| c |
+---+
| 1 |
| 22 |
| 3 |
+---+
3 rows in set (0.00 sec)
```

**Note:**

You should use @@ instead of @ before `tidb_snapshot` because @@ is used to denote the system variable while @ is used to denote the user variable.

#### 9.4.2.5 How to restore historical data

Before you restore data from an older version, make sure that Garbage Collection (GC) does not clear the history data while you are working on it. This can be done by setting the `tidb_gc_life_time` variable as the following example shows. Do not forget to set the variable back to the previous value after the restore.

```
SET GLOBAL tidb_gc_life_time="60m";
```

##### Note:

Increasing the GC life time from the default 10 minutes to half an hour or more will result in additional versions of rows being retained, which might require more disk space. This might also affect the performance of certain operations such as scans when TiDB needs to skip these additional versions of the same rows during data reads.

To restore data from an older version, you can use one of the following methods:

- For simple cases, use `SELECT` after setting the `tidb_snapshot` variable and copy-paste the output, or use `SELECT ... INTO LOCAL OUTFILE` and use `LOAD DATA` to import the data later on.
- Use [Dumpling](#) to export a historical snapshot. Dumpling performs well in exporting larger sets of data.

## 9.5 Best Practices

### 9.5.1 TiDB Best Practices

This document summarizes the best practices of using TiDB, including the use of SQL and optimization tips for Online Analytical Processing (OLAP) and Online Transactional Processing (OLTP) scenarios, especially the optimization options specific for TiDB.

Before you read this document, it is recommended that you read three blog posts that introduce the technical principles of TiDB:

- [TiDB Internal \(I\) - Data Storage](#)
- [TiDB Internal \(II\) - Computing](#)
- [TiDB Internal \(III\) - Scheduling](#)

### 9.5.1.1 Preface

Database is a generic infrastructure system. It is important to consider various user scenarios during the development process and to modify the data parameters or the way to use according to actual situations in specific business scenarios.

TiDB is a distributed database compatible with the MySQL protocol and syntax. But with the internal implementation and supporting of distributed storage and transactions, the way of using TiDB is different from MySQL.

### 9.5.1.2 Basic concepts

The best practices are closely related to its implementation principles. It is recommended that you learn some of the basic mechanisms, including the Raft consensus algorithm, distributed transactions, data sharding, load balancing, the mapping solution from SQL to Key-Value (KV), the implementation method of secondary indexing, and distributed execution engines.

This section is an introduction to these concepts. For detailed information, refer to [PingCAP blog posts](#).

#### 9.5.1.2.1 Raft

Raft is a consensus algorithm that ensures data replication with strong consistency. At the bottom layer, TiDB uses Raft to replicate data. TiDB writes data to the majority of the replicas before returning the result of success. In this way, even though a few replicas might get lost, the system still has the latest data. For example, if there are three replicas, the system does not return the result of success until data has been written to two replicas. Whenever a replica is lost, at least one of the remaining two replicas have the latest data.

To store three replicas, compared with the replication of Source-Replica, Raft is more efficient. The write latency of Raft depends on the two fastest replicas, instead of the slowest one. Therefore, the implementation of geo-distributed and multiple active data centers becomes possible by using the Raft replication. In the typical scenario of three data centers distributing in two sites, to guarantee the data consistency, TiDB just needs to successfully write data into the local data center and the closer one, instead of writing to all three data centers. However, this does not mean that cross-data center deployment can be implemented in any scenario. When the amount of data to be written is large, the bandwidth and latency between data centers become the key factors. If the write speed exceeds the bandwidth or the latency is too high, the Raft replication mechanism still cannot work well.

#### 9.5.1.2.2 Distributed transactions

TiDB provides complete distributed transactions and the model has some optimizations on the basis of [Google Percolator](#). This document introduces the following features:

- Optimistic transaction model

TiDB's optimistic transaction model does not detect conflicts until the commit phase. If there are conflicts, the transaction needs retry. But this model is inefficient if the conflict is severe, because operations before retry are invalid and need to repeat.

Assume that the database is used as a counter. High access concurrency might lead to severe conflicts, resulting in multiple retries or even timeouts. Therefore, in the scenario of severe conflicts, it is recommended to use the pessimistic transaction mode or to solve problems at the system architecture level, such as placing counter in Redis. Nonetheless, the optimistic transaction model is efficient if the access conflict is not very severe.

- Pessimistic transaction mode

In TiDB, the pessimistic transaction mode has almost the same behavior as in MySQL. The transaction applies a lock during the execution phase, which avoids retries in conflict situations and ensures a higher success rate. By applying the pessimistic locking, you can also lock data in advance using `SELECT FOR UPDATE`.

However, if the application scenario has fewer conflicts, the optimistic transaction model has better performance.

- Transaction size limit

As distributed transactions need to conduct two-phase commit and the bottom layer performs Raft replication, if a transaction is very large, the commit process would be quite slow, and the following Raft replication process is thus stuck. To avoid this problem, the transaction size is limited:

- A transaction is limited to 5,000 SQL statements (by default)
- Each Key-Value entry is no more than 6 MB (by default)
- The total size of Key-Value entries is no more than 10 GB.

You can find similar limits in [Google Cloud Spanner](#).

#### 9.5.1.2.3 Data sharding

TiKV automatically shards bottom-layered data according to the range of keys. Each Region is a range of keys, which is a left-closed and right-open interval, `[StartKey, EndKey → ]`. When the amount of Key-Value pairs in a Region exceeds a certain value, the Region automatically splits into two.

#### 9.5.1.2.4 Load balancing

Placement Driver (PD) balances the load of the cluster according to the status of the entire TiKV cluster. The unit of scheduling is Region and the logic is the strategy configured by PD.

#### 9.5.1.2.5 SQL on KV

TiDB automatically maps the SQL structure into Key-Value structure. For details, see [TiDB Internal \(II\) - Computing](#).

Simply put, TiDB performs the following operations:

- A row of data is mapped to a Key-Value pair. The key is prefixed with TableID and suffixed with the row ID.
- An index is mapped as a Key-Value pair. The key is prefixed with TableID+IndexID and suffixed with the index value.

The data or indexes in the same table have the same prefix. These Key-Values are at adjacent positions in the key space of TiKV. Therefore, when the amount of data to be written is large and all is written to one table, the write hotspot is created. The situation gets worse when some index values of the continuous written data is also continuous (for example, fields that increase with time, like `update time`), which creates a few write hotspots and becomes the bottleneck of the entire system.

Similarly, if all data is read from a focused small range (for example, the continuous tens or hundreds of thousands of rows of data), an access hotspot of data is likely to occur.

#### 9.5.1.2.6 Secondary index

TiDB supports the complete secondary indexes, which are also global indexes. Many queries can be optimized by index. Thus, it is important for applications to make good use of secondary indexes.

Lots of MySQL experience is also applicable to TiDB. It is noted that TiDB has its unique features. The following are a few notes when using secondary indexes in TiDB.

- The more secondary indexes, the better?

Secondary indexes can speed up queries, but adding an index has side effects. The previous section introduces the storage model of indexes. For each additional index, there will be one more Key-Value when inserting a piece of data. Therefore, the more indexes, the slower the writing speed and the more space it takes up.

In addition, too many indexes affects the runtime of the optimizer, and inappropriate indexes mislead the optimizer. Thus, more secondary indexes does not mean better performance.

- Which columns should create indexes?

As is mentioned above, index is important but the number of indexes should be proper. You must create appropriate indexes according to the application characteristics. In principle, you need to create an index on the columns involved in the query to improve the performance. The following are situations that need to create indexes:

- For columns with a high degree of differentiation, filtered rows are remarkably reduced through indexes.
- If there are multiple query criteria, you can choose composite indexes. Note to put the columns with the equivalent condition before composite indexes.

For example, if a commonly used query is `select * from t where c1 = 10 and c2 = 100 and c3 > 10`, you can create a composite index `Index cidx (c1, c2, c3)`. In this way, you can use the query condition to create an index prefix and then scan.

- The difference between querying through indexes and directly scanning the table

TiDB has implemented global indexes, so indexes and data of the table are not necessarily on the same data sharding. When querying through indexes, it should firstly scan indexes to get the corresponding row ID and then use the row ID to get the data. Thus, this method involves two network requests and has a certain performance overhead.

If the query involves lots of rows, scanning index proceeds concurrently. When the first batch of results is returned, getting the data of the table can then proceed. Therefore, this is a parallel + pipeline model. Though the two accesses create overhead, the latency is not high.

The following two conditions do not have the problem of two accesses:

- Columns of the index have already met the query requirement. Assume that the `c` column on the `t` table has an index and the query is `select c from t where c > 10`. At this time, all needed data can be obtained if you access the index. This situation is called **Covering Index**. But if you focus more on the query performance, you can put into index a portion of columns that do not need to be filtered but need to be returned in the query result, creating composite index. Take `select c1, c2 from t where c1 > 10` as an example. You can optimize this query by creating composite index `Index c12 (c1, c2)`.
- The primary key of the table is integer. In this case, TiDB uses the value of the primary key as row ID. Thus, if the query condition is on the primary key, you can directly construct the range of the row ID, scan the table data, and get the result.
- Query concurrency

As data is distributed across many Regions, queries run in TiDB concurrently. But the concurrency by default is not high in case it consumes lots of system resources. Besides, the OLTP query usually does not involve a large amount of data and the low concurrency is enough. But for the OLAP query, the concurrency is high and TiDB modifies the query concurrency through the following system variables:

- `tidb_distsql_scan_concurrency`:

The concurrency of scanning data, including scanning the table and index data.

- `tidb_index_lookup_size`:

If it needs to access the index to get row IDs before accessing the table data, it uses a batch of row IDs as a single request to access the table data. This parameter sets the size of a batch. The larger batch increases latency, while the smaller one might lead to more queries. The proper size of this parameter is related to the amount of data that the query involves. Generally, no modification is required.

- `tidb_index_lookup_concurrency`:

If it needs to access the index to get row IDs before accessing the table data, the concurrency of getting data through row IDs every time is modified through this parameter.

- Ensure the order of results through indexes

You can use indexes to filter or sort data. Firstly, get row IDs according to the index order. Then, return the row content according to the return order of row IDs. In this way, the returned results are ordered according to the index column. It has been mentioned earlier that the model of scanning index and getting row is parallel + pipeline. If the row is returned according to the index order, a high concurrency between two queries does not reduce latency. Thus, the concurrency is low by default, but it can be modified through the `tidb_index_serial_scan_concurrency` variable.

- Reverse index scan

TiDB supports scanning an ascending index in reverse order, at a speed slower than normal scan by 20%. If the data is changed frequently and thus too many versions exist, the performance overhead might be higher. It is recommended to avoid reverse index scans as much as possible.

### 9.5.1.3 Scenarios and practices

In the last section, we discussed some basic implementation mechanisms of TiDB and their influence on usage. This section introduces specific usage scenarios and operation practices, from deployment to application usage.

#### 9.5.1.3.1 Deployment

Before deployment, read [Software and Hardware Requirements](#).

It is recommended to deploy the TiDB cluster using [TiUP](#). This tool can deploy, stop, destroy, and upgrade the whole cluster, which is quite convenient. It is not recommended to manually deploy the TiDB cluster, which might be troublesome to maintain and upgrade later.

#### 9.5.1.3.2 Data import

To improve the write performance during the import process, you can tune TiKV's parameters as stated in [Tune TiKV Memory Parameter Performance](#).

### 9.5.1.3.3 Write

As mentioned before, TiDB limits the size of a single transaction in the Key-Value layer. As for the SQL layer, a row of data is mapped to a Key-Value entry. For each additional index, one more Key-Value entry is added.

#### Note:

When you set the size limit for transactions, you need to consider the overhead of TiDB encoding and the extra transaction key. It is recommended that **the number of rows of each transaction is less than 200 and the data size of a single row is less than 100 KB**; otherwise, the performance is bad.

It is recommended to split statements into batches or add a limit to the statements, whether they are `INSERT`, `UPDATE` or `DELETE` statements.

When deleting a large amount of data, it is recommended to use `Delete from t where ↪ xx limit 5000;`. It deletes through the loop and use `Affected Rows == 0` as a condition to end the loop.

If the amount of data that needs to be deleted at a time is large, this loop method gets slower and slower because each deletion traverses backward. After deleting the previous data, lots of deleted flags remain for a short period (then all is cleared by Garbage Collection) and affect the following `DELETE` statement. If possible, it is recommended to refine the `WHERE` condition. Assume that you need to delete all data on `2017-05-26`, you can use the following statements:

```
for i from 0 to 23:
 while affected_rows > 0:
 delete from t where insert_time >= i:00:00 and insert_time < (i+1)
 ↪ :00:00 limit 5000;
 affected_rows = select affected_rows()
```

This pseudocode means to split huge chunks of data into small ones and then delete, so that the earlier `Delete` statements do not affect the later ones.

### 9.5.1.3.4 Query

For query requirements and specific statements, refer to [System Variables](#).

You can control the concurrency of SQL execution through the `SET` statement and the selection of the `Join` operator through hints.

In addition, you can also use MySQL's standard index selection, the hint syntax, or control the optimizer to select indexes through `Use Index/Ignore Index` hint.

If the application scenario has both OLTP and OLAP workloads, you can send the OLTP request and OLAP request to different TiDB servers, diminishing the impact of OLAP on OLTP. It is recommended to use machines with high-performance hardware (for example, more processor cores and larger memory) for the TiDB server that processes OLAP workloads.

To completely isolate OLTP and OLAP workloads, it is recommended to run OLAP applications on TiFlash. TiFlash is a columnar storage engine with great performance on OLAP workloads. TiFlash can achieve physical isolation on the storage layer and guarantees consistent reads.

#### 9.5.1.3.5 Monitoring and log

The monitoring metrics is the best method to learn the status of the system. It is recommended that you deploy the monitoring system along with your TiDB cluster.

TiDB uses [Grafana + Prometheus](#) to monitor the system status. The monitoring system is automatically deployed and configured if you deploy TiDB using TiUP.

There are lots of items in the monitoring system, the majority of which are for TiDB developers. You do not have to understand these items without an in-depth knowledge of the source code. Some items that are related to applications or to the state of system key components are selected and put in a separate [overview](#) panel for users.

In addition to monitoring, you can also view the system logs. The three components of TiDB, tidb-server, tikv-server, and pd-server, each has a `--log-file` parameter. If this parameter has been configured when the cluster is started, logs are stored in the file configured by the parameter and log files are automatically archived on a daily basis. If the `--log-file` parameter has not been configured, the log is output to `stderr`.

Starting from TiDB 4.0, TiDB provides [TiDB Dashboard](#) UI to improve usability. You can access TiDB Dashboard by visiting [http://\\$%7BPD\\_IP%7D:\\$%7BPD\\_PORT%7D/dashboard](http://$%7BPD_IP%7D:$%7BPD_PORT%7D/dashboard) in your browser. TiDB Dashboard provides features such as viewing cluster status, performance analysis, traffic visualization, cluster diagnostics, and log searching.

#### 9.5.1.3.6 Documentation

The best way to learn about a system or solve the problem is to read its documentation and understand its implementation principles.

TiDB has a large number of official documents both in Chinese and English. If you have met an issue, you can start from [FAQ](#) and [TiDB Cluster Troubleshooting Guide](#). You can also search the issue list or create an issue in [TiDB repository on GitHub](#).

TiDB also has many useful ecosystem tools. See [Ecosystem Tool Overview](#) for details.

For more articles on the technical details of TiDB, see the [PingCAP official blog site](#).

#### 9.5.1.4 Best scenarios for TiDB

TiDB is suitable for the following scenarios:

- The data volume is too large for a standalone database
- You do not want to do sharding
- The access mode has no obvious hotspot
- Transactions, strong consistency, and disaster recovery are required
- You hope to have real-time Hybrid Transaction/Analytical Processing (HTAP) analytics and reduce storage links

### 9.5.2 Best Practices for Developing Java Applications with TiDB

This document introduces the best practice for developing Java applications to better use TiDB. Based on some common Java application components that interact with the backend TiDB database, this document also provides the solutions to commonly encountered issues during development.

#### 9.5.2.1 Database-related components in Java applications

Common components that interact with the TiDB database in Java applications include:

- Network protocol: A client interacts with a TiDB server via the standard [MySQL protocol](#).
- JDBC API and JDBC drivers: Java applications usually use the standard [JDBC \(Java Database Connectivity\)](#) API to access a database. To connect to TiDB, you can use a JDBC driver that implements the MySQL protocol via the JDBC API. Such common JDBC drivers for MySQL include [MySQL Connector/J](#) and [MariaDB Connector/J](#).
- Database connection pool: To reduce the overhead of creating a connection each time it is requested, applications usually use a connection pool to cache and reuse connections. JDBC [DataSource](#) defines a connection pool API. You can choose from different open-source connection pool implementations as needed.
- Data access framework: Applications usually use a data access framework such as [MyBatis](#) and [Hibernate](#) to further simplify and manage the database access operations.
- Application implementation: The application logic controls when to send what commands to the database. Some applications use [Spring Transaction](#) aspects to manage transactions' start and commit logics.

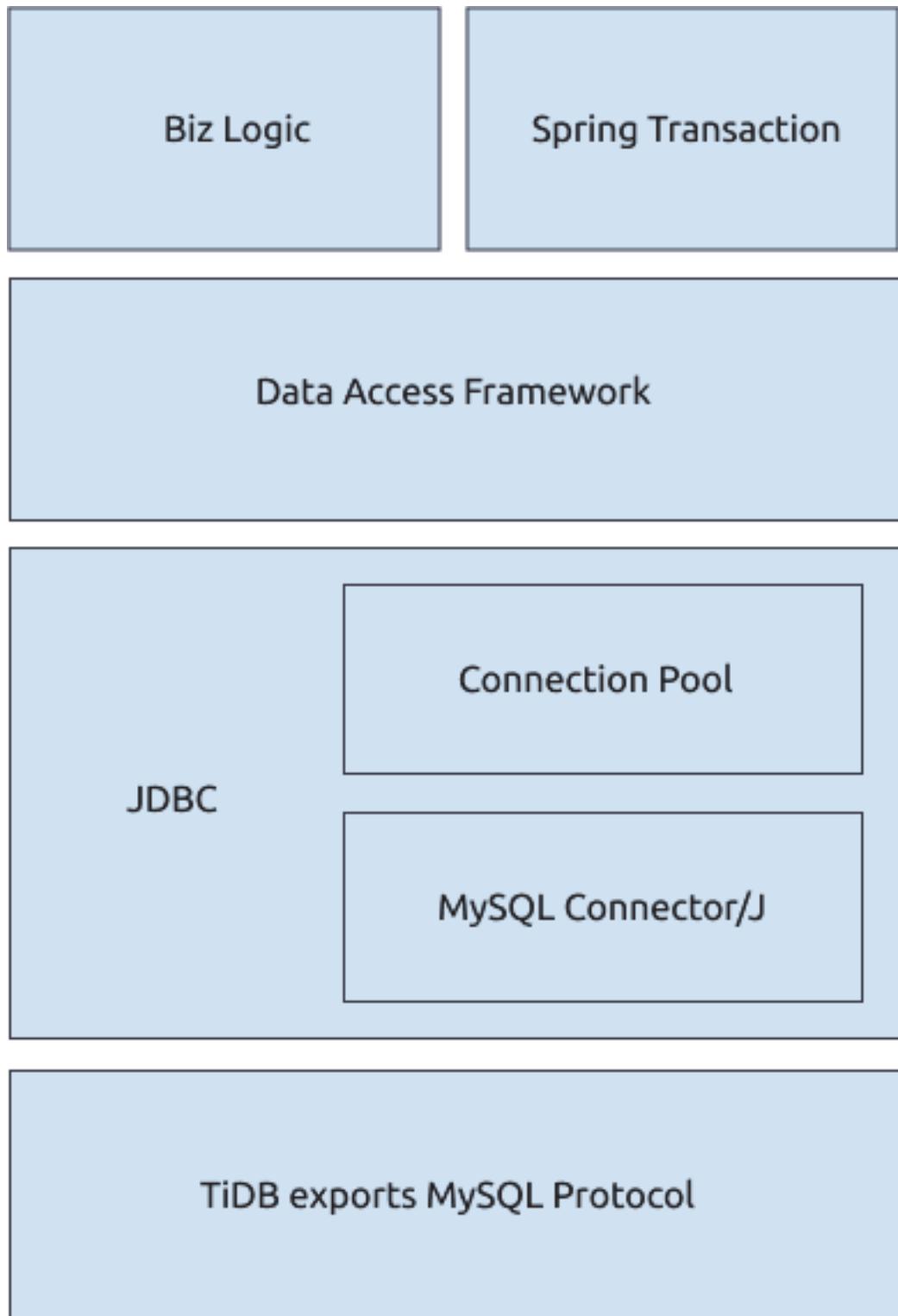


Figure 79: Java application components

From the above diagram, you can see that a Java application might do the following things:

- Implement the MySQL protocol via the JDBC API to interact with TiDB.
- Get a persistent connection from the connection pool.
- Use a data access framework such as MyBatis to generate and execute SQL statements.
- Use Spring Transaction to automatically start or stop a transaction.

The rest of this document describes the issues and their solutions when you develop a Java application using the above components.

### 9.5.2.2 JDBC

Java applications can be encapsulated with various frameworks. In most of the frameworks, JDBC API is called on the bottommost level to interact with the database server. For JDBC, it is recommended that you focus on the following things:

- JDBC API usage choice
- API Implementer's parameter configuration

#### 9.5.2.2.1 JDBC API

For JDBC API usage, see [JDBC official tutorial](#). This section covers the usage of several important APIs.

##### Use Prepare API

For OLTP (Online Transactional Processing) scenarios, the SQL statements sent by the program to the database are several types that can be exhausted after removing parameter changes. Therefore, it is recommended to use [Prepared Statements](#) instead of regular [execution from a text file](#) and reuse Prepared Statements to execute directly. This avoids the overhead of repeatedly parsing and generating SQL execution plans in TiDB.

At present, most upper-level frameworks call the Prepare API for SQL execution. If you use the JDBC API directly for development, pay attention to choosing the Prepare API.

In addition, with the default implementation of MySQL Connector/J, only client-side statements are preprocessed, and the statements are sent to the server in a text file after `? →` is replaced on the client. Therefore, in addition to using the Prepare API, you also need to configure `useServerPrepStmts = true` in JDBC connection parameters before you perform statement preprocessing on the TiDB server. For detailed parameter configuration, see [MySQL JDBC parameters](#).

##### Use Batch API

For batch inserts, you can use the [addBatch/executeBatch API](#). The `addBatch() →` method is used to cache multiple SQL statements first on the client, and then send them to the database server together when calling the `executeBatch` method.

**Note:**

In the default MySQL Connector/J implementation, the sending time of the SQL statements that are added to batch with `addBatch()` is delayed to the time when `executeBatch()` is called, but the statements will still be sent one by one during the actual network transfer. Therefore, this method usually does not reduce the amount of communication overhead.

If you want to batch network transfer, you need to configure `rewriteBatchedStatements = true` in the JDBC connection parameters. For the detailed parameter configuration, see [Batch-related parameters](#).

Use `StreamingResult` to get the execution result

In most scenarios, to improve execution efficiency, JDBC obtains query results in advance and save them in client memory by default. But when the query returns a super large result set, the client often wants the database server to reduce the number of records returned at a time, and waits until the client's memory is ready and it requests for the next batch.

Usually, there are two kinds of processing methods in JDBC:

- Set `FetchSize` to `Integer.MIN_VALUE` to ensure that the client does not cache. The client will read the execution result from the network connection through `StreamingResult`.
- To use Cursor Fetch, first set `FetchSize` as a positive integer and configure `useCursorFetch=true` in the JDBC URL.

TiDB supports both methods, but it is preferred that you use the first method, because it is a simpler implementation and has a better execution efficiency.

### 9.5.2.2.2 MySQL JDBC parameters

JDBC usually provides implementation-related configurations in the form of JDBC URL parameters. This section introduces [MySQL Connector/J's parameter configurations](#) (If you use MariaDB, see [MariaDB's parameter configurations](#)). Because this document cannot cover all configuration items, it mainly focuses on several parameters that might affect performance.

Prepare-related parameters

This section introduces parameters related to Prepare.

`useServerPrepStmts`

`useServerPrepStmts` is set to `false` by default, that is, even if you use the Prepare API, the “prepare” operation will be done only on the client. To avoid the parsing overhead of the

server, if the same SQL statement uses the Prepare API multiple times, it is recommended to set this configuration to `true`.

To verify that this setting already takes effect, you can do:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > QPS By Instance**.
- If `COM_QUERY` is replaced by `COM_STMT_EXECUTE` or `COM_STMT_PREPARE` in the request, it means this setting already takes effect.

#### `cachePrepStmts`

Although `useServerPrepStmts=true` allows the server to execute Prepared Statements, by default, the client closes the Prepared Statements after each execution and does not reuse them. This means that the “prepare” operation is not even as efficient as text file execution. To solve this, it is recommended that after setting `useServerPrepStmts=true`, you should also configure `cachePrepStmts=true`. This allows the client to cache Prepared Statements.

To verify that this setting already takes effect, you can do:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > QPS By Instance**.
- If the number of `COM_STMT_EXECUTE` in the request is far more than the number of `COM_STMT_PREPARE`, it means this setting already takes effect.



Figure 80: QPS By Instance

In addition, configuring `useConfigs=maxPerformance` will configure multiple parameters at the same time, including `cachePrepStmts=true`.

#### `prepStmtCacheSqlLimit`

After configuring `cachePreStmts`, also pay attention to the `prepStmtCacheSqlLimit` configuration (the default value is 256). This configuration controls the maximum length of the Prepared Statements cached on the client.

The Prepared Statements that exceed this maximum length will not be cached, so they cannot be reused. In this case, you may consider increasing the value of this configuration depending on the actual SQL length of the application.

You need to check whether this setting is too small if you:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > QPS By Instance**.
- And find that `cachePreStmts=true` has been configured, but `COM_STMT_PREPARE` is still mostly equal to `COM_STMT_EXECUTE` and `COM_STMT_CLOSE` exists.

#### `prepStmtCacheSize`

`prepStmtCacheSize` controls the number of cached Prepared Statements (the default value is 25). If your application requires “preparing” many types of SQL statements and wants to reuse Prepared Statements, you can increase this value.

To verify that this setting already takes effect, you can do:

- Go to TiDB monitoring dashboard and view the request command type through **Query Summary > QPS By Instance**.
- If the number of `COM_STMT_EXECUTE` in the request is far more than the number of `COM_STMT_PREPARE`, it means this setting already takes effect.

#### Batch-related parameters

While processing batch writes, it is recommended to configure `rewriteBatchedStatements` → `true`. After using `addBatch()` or `executeBatch()`, JDBC still sends SQL one by one by default, for example:

```
pstmt = prepare("insert into t (a) values(?)");
pstmt.setInt(1, 10);
pstmt.addBatch();
pstmt.setInt(1, 11);
pstmt.addBatch();
pstmt.setInt(1, 12);
pstmt.executeBatch();
```

Although `Batch` methods are used, the SQL statements sent to TiDB are still individual `INSERT` statements:

```
insert into t(a) values(10);
insert into t(a) values(11);
insert into t(a) values(12);
```

But if you set `rewriteBatchedStatements=true`, the SQL statements sent to TiDB will be a single `INSERT` statement:

```
insert into t(a) values(10),(11),(12);
```

Note that the rewrite of the `INSERT` statements is to concatenate the values after multiple “values” keywords into a whole SQL statement. If the `INSERT` statements have other differences, they cannot be rewritten, for example:

```
insert into t (a) values (10) on duplicate key update a = 10;
insert into t (a) values (11) on duplicate key update a = 11;
insert into t (a) values (12) on duplicate key update a = 12;
```

The above `INSERT` statements cannot be rewritten into one statement. But if you change the three statements into the following ones:

```
insert into t (a) values (10) on duplicate key update a = values(a);
insert into t (a) values (11) on duplicate key update a = values(a);
insert into t (a) values (12) on duplicate key update a = values(a);
```

Then they meet the rewrite requirement. The above `INSERT` statements will be rewritten into the following one statement:

```
insert into t (a) values (10), (11), (12) on duplicate key update a =
 ↪ values(a);
```

If there are three or more updates during the batch update, the SQL statements will be rewritten and sent as multiple queries. This effectively reduces the client-to-server request overhead, but the side effect is that a larger SQL statement is generated. For example:

```
update t set a = 10 where id = 1; update t set a = 11 where id = 2; update
 ↪ t set a = 12 where id = 3;
```

In addition, because of a [client bug](#), if you want to configure `rewriteBatchedStatements` → `=true` and `useServerPrepStmts=true` during batch update, it is recommended that you also configure the `allowMultiQueries=true` parameter to avoid this bug.

### Integrate parameters

Through monitoring, you might notice that although the application only performs `INSERT` operations to the TiDB cluster, there are a lot of redundant `SELECT` statements. Usually this happens because JDBC sends some SQL statements to query the settings, for example, `select @@session.transaction_read_only`. These SQL statements are useless for TiDB, so it is recommended that you configure `useConfigs=maxPerformance` to avoid extra overhead.

`useConfigs=maxPerformance` configuration includes a group of configurations:

```
cacheServerConfiguration=true
useLocalSessionState=true
```

```
elideSetAutoCommits=true
alwaysSendSetIsolation=false
enableQueryTimeouts=false
```

After it is configured, you can check the monitoring to see a decreased number of SELECT statements.

#### Timeout-related parameters

TiDB provides two MySQL-compatible parameters that controls the timeout: `wait_timeout` and `max_execution_time`. These two parameters respectively control the connection idle timeout with the Java application and the timeout of the SQL execution in the connection; that is to say, these parameters control the longest idle time and the longest busy time for the connection between TiDB and the Java application. The default value of both parameters is 0, which by default allows the connection to be infinitely idle and infinitely busy (an infinite duration for one SQL statement to execute).

However, in an actual production environment, idle connections and SQL statements with excessively long execution time negatively affect databases and applications. To avoid idle connections and SQL statements that are executed for too long, you can configure these two parameters in your application's connection string. For example, set `sessionVariables → =wait_timeout=3600` (1 hour) and `sessionVariables=max_execution_time=300000` (5 minutes).

### 9.5.2.3 Connection pool

Building TiDB (MySQL) connections is relatively expensive (for OLTP scenarios at least), because in addition to building a TCP connection, connection authentication is also required. Therefore, the client usually saves the TiDB (MySQL) connections to the connection pool for reuse.

Java has many connection pool implementations such as [HikariCP](#), [tomcat-jdbc](#), [durid](#), [c3p0](#), and [dbcp](#). TiDB does not limit which connection pool you use, so you can choose whichever you like for your application.

#### 9.5.2.3.1 Configure the number of connections

It is a common practice that the connection pool size is well adjusted according to the application's own needs. Take HikariCP as an example:

- `maximumPoolSize`: The maximum number of connections in the connection pool. If this value is too large, TiDB consumes resources to maintain useless connections. If this value is too small, the application gets slow connections. So configure this value for your own good. For details, see [About Pool Sizing](#).
- `minimumIdle`: The minimum number of idle connections in the connection pool. It is mainly used to reserve some connections to respond to sudden requests when the application is idle. You can also configure it according to your application needs.

The application needs to return the connection after finishing using it. It is also recommended that the application use the corresponding connection pool monitoring (such as `metricRegistry`) to locate the connection pool issue in time.

### 9.5.2.3.2 Probe configuration

The connection pool maintains persistent connections to TiDB. TiDB does not proactively close client connections by default (unless an error is reported), but generally there will be network proxies such as LVS or HAProxy between the client and TiDB. Usually, these proxies will proactively clean up connections that are idle for a certain period of time. In addition to paying attention to the idle configuration of the proxies, the connection pool also needs to keep alive or probe connections.

If you often see the following error in your Java application:

```
The last packet sent successfully to the server was 3600000 milliseconds ago
 ↵ . The driver has not received any packets from the server. com.mysql.
 ↵ jdbc.exceptions.jdbc4.CommunicationsException: Communications link
 ↵ failure
```

If `n` in `n milliseconds ago` is 0 or a very small value, it is usually because the executed SQL operation causes TiDB to exit abnormally. To find the cause, it is recommended to check the TiDB stderr log.

If `n` is a very large value (such as 3600000 in the above example), it is likely that this connection was idle for a long time and then closed by the intermediate proxy. The usual solution is to increase the value of the proxy's idle configuration and allow the connection pool to:

- Check whether the connection is available before using the connection every time
- Regularly check whether the connection is available using a separate thread.
- Send a test query regularly to keep alive connections

Different connection pool implementations might support one or more of the above methods. You can check your connection pool documentation to find the corresponding configuration.

### 9.5.2.4 Data access framework

Applications often use some kind of data access framework to simplify database access.

#### 9.5.2.4.1 MyBatis

[MyBatis](#) is a popular Java data access framework. It is mainly used to manage SQL queries and complete the mapping between result sets and Java objects. MyBatis is highly compatible with TiDB. MyBatis rarely has problems based on its historical issues.

Here this document mainly focuses on the following configurations.

### Mapper parameters

MyBatis Mapper supports two parameters:

- `select 1 from t where id = #{param1}` will be converted to `select 1 from t ↵ where id =?` as a Prepared Statement and be “prepared”, and the actual parameter will be used for reuse. You can get the best performance when using this parameter with the previously mentioned Prepare connection parameters.
- `select 1 from t where id = ${param2}` will be replaced with `select 1 from t ↵ where id = 1` as a text file and be executed. If this statement is replaced with different parameters and is executed, MyBatis will send different requests for “preparing” the statements to TiDB. This might cause TiDB to cache a large number of Prepared Statements, and executing SQL operations this way has injection security risks.

### Dynamic SQL Batch

#### [Dynamic SQL - foreach](#)

To support the automatic rewriting of multiple `INSERT` statements into the form of `insert ... values(...), (...), ...`, in addition to configuring `rewriteBatchedStatements =true` in JDBC as mentioned before, MyBatis can also use dynamic SQL to semi-automatically generate batch inserts. Take the following mapper as an example:

```
<insert id="insertTestBatch" parameterType="java.util.List" fetchSize="1">
 insert into test
 (id, v1, v2)
 values
 <foreach item="item" index="index" collection="list" separator=",">
 (
 #{item.id}, #{item.v1}, #{item.v2}
)
 </foreach>
 on duplicate key update v2 = v1 + values(v1)
</insert>
```

This mapper generates an `insert on duplicate key update` statement. The number of `(?, ?, ?)` following “values” is determined by the number of passed lists. Its final effect is similar to using `rewriteBatchStatements=true`, which also effectively reduces communication overhead between the client and TiDB.

As mentioned before, you also need to note that the Prepared Statements will not be cached after their maximum length exceeds the value of `prepStmtCacheSqlLimit`.

### Streaming result

A previous section introduces how to stream read execution results in JDBC. In addition to the corresponding configurations of JDBC, if you want to read a super large result set in MyBatis, you also need to note that:

- You can set `fetchSize` for a single SQL statement in the mapper configuration (see the previous code block). Its effect is equivalent to calling `setFetchSize` in JDBC.
- You can use the query interface with `ResultHandler` to avoid getting the entire result set at once.
- You can use the `Cursor` class for stream reading.

If you configure mappings using XML, you can stream read results by configuring `fetchSize="-2147483648"(Integer.MIN_VALUE)` in the mapping's `<select>` section.

```
<select id="getAll" resultMap="postResultMap" fetchSize="-2147483648">
 select * from post;
</select>
```

If you configure mappings using code, you can add the `@Options(fetchSize = Integer.MIN_VALUE)` annotation and keep the type of results as `Cursor` so that the SQL results can be read in streaming.

```
@Select("select * from post")
@Options(fetchSize = Integer.MIN_VALUE)
Cursor<Post> queryAllPost();
```

#### 9.5.2.4.2 ExecutorType

You can choose `ExecutorType` during `openSession`. MyBatis supports three types of executors:

- Simple: The Prepared Statements are called to JDBC for each execution (if the JDBC configuration item `cachePrepStmts` is enabled, repeated Prepared Statements will be reused)
- Reuse: The Prepared Statements are cached in `executor`, so that you can reduce duplicate calls for Prepared Statements without using the JDBC `cachePrepStmts`
- Batch: Each update operation (`INSERT/DELETE/UPDATE`) will first be added to the batch, and will be executed until the transaction commits or a `SELECT` query is performed. If `rewriteBatchStatements` is enabled in the JDBC layer, it will try to rewrite the statements. If not, the statements will be sent one by one.

Usually, the default value of `ExecutorType` is `Simple`. You need to change `ExecutorType` when calling `openSession`. If it is the batch execution, you might find that in a transaction the `UPDATE` or `INSERT` statements are executed pretty fast, but it is slower when reading data or committing the transaction. This is actually normal, so you need to note this when troubleshooting slow SQL queries.

### 9.5.2.5 Spring Transaction

In the real world, applications might use [Spring Transaction](#) and AOP aspects to start and stop transactions.

By adding the `@Transactional` annotation to the method definition, AOP starts the transaction before the method is called, and commits the transaction before the method returns the result. If your application has a similar need, you can find `@Transactional` in code to determine when the transaction is started and closed.

Pay attention to a special case of embedding. If it occurs, Spring will behave differently based on the [Propagation](#) configuration. Because TiDB does not support savepoint, nested transactions are not supported yet.

### 9.5.2.6 Misc

This section introduces some useful tools for Java to help you troubleshoot issues.

#### 9.5.2.6.1 Troubleshooting tools

Using the powerful troubleshooting tools of JVM is recommended when an issue occurs in your Java application and you do not know the application logic. Here are a few common tools:

`jstack`

`jstack` is similar to `pprof/goroutine` in Go, which can easily troubleshoot the process stuck issue.

By executing `jstack pid`, you can output the IDs and stack information of all threads in the target process. By default, only the Java stack is output. If you want to output the C++ stack in the JVM at the same time, add the `-m` option.

By using `jstack` multiple times, you can easily locate the stuck issue (for example, a slow query from application's view due to using Batch ExecutorType in Mybatis) or the application deadlock issue (for example, the application does not send any SQL statement because it is preempting a lock before sending it).

In addition, `top -p $ PID -H` or Java swiss knife are common methods to view the thread ID. Also, to locate the issue of “a thread occupies a lot of CPU resources and I don't know what it is executing”, do the following steps:

- Use `printf "%x\n" pid` to convert the thread ID to hexadecimal.
- Go to the `jstack` output to find the stack information of the corresponding thread.

`jmap & mat`

Unlike `pprof/heap` in Go, `jmap` dumps the memory snapshot of the entire process (in Go, it is the sampling of the distributor), and then the snapshot can be analyzed by another tool `mat`.

Through mat, you can see the associated information and attributes of all objects in the process, and you can also observe the running status of the thread. For example, you can use mat to find out how many MySQL connection objects exist in the current application, and what is the address and status information of each connection object.

Note that mat only handles reachable objects by default. If you want to troubleshoot young GC issues, you can adjust mat configuration to view unreachable objects. In addition, for investigating the memory allocation of young GC issues (or a large number of short-lived objects), using Java Flight Recorder is more convenient.

#### trace

Online applications usually do not support modifying the code, but it is often desired that dynamic instrumentation is performed in Java to locate issues. Therefore, using btrace or arthas trace is a good option. They can dynamically insert trace code without restarting the application process.

#### Flame graph

Obtaining flame graphs in Java applications is tedious. For details, see [Java Flame Graphs Introduction: Fire For Everyone!](#).

### 9.5.2.7 Conclusion

Based on commonly used Java components that interact with databases, this document describes the common problems and solutions for developing Java applications with TiDB. TiDB is highly compatible with the MySQL protocol, so most of the best practices for MySQL-based Java applications also apply to TiDB.

Join us at [TiDB Community slack channel](#), and share with broad TiDB user group about your experience or problems when you develop Java applications with TiDB.

### 9.5.3 Best Practices for Using HAProxy in TiDB

This document describes best practices for configuration and usage of [HAProxy](#) in TiDB. HAProxy provides load balancing for TCP-based applications. From TiDB clients, you can manipulate data just by connecting to the floating virtual IP address provided by HAProxy, which helps to achieve load balance in the TiDB server layer.

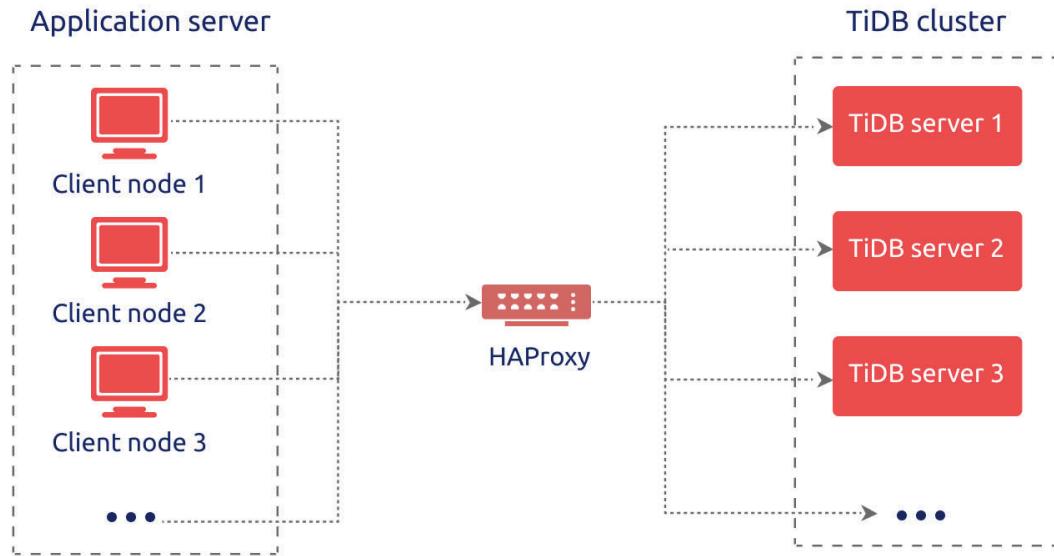


Figure 81: HAProxy Best Practices in TiDB

#### 9.5.3.1 HAProxy overview

HAProxy is free, open-source software written in C language that provides a high availability load balancer and proxy server for TCP and HTTP-based applications. Because of its fast and efficient use of CPU and memory, HAProxy is now widely used by many well-known websites such as GitHub, Bitbucket, Stack Overflow, Reddit, Tumblr, Twitter, Tuenti, and AWS (Amazon Web Services).

HAProxy is written in the year 2000 by Willy Tarreau, the core contributor to the Linux kernel, who is still responsible for the maintenance of the project and provides free software updates in the open-source community. In this guide, HAProxy 2.5.0 is used. It is recommended to use the latest stable version. See [the released version of HAProxy](#) for details.

#### 9.5.3.2 Basic features

- **High Availability:** HAProxy provides high availability with support for a graceful shutdown and a seamless switchover;
- **Load Balancing:** Two major proxy modes are supported: TCP, also known as layer 4, and HTTP, also known as layer 7. No less than 9 load balancing algorithms are supported, such as roundrobin, leastconn and random;
- **Health Check:** HAProxy periodically checks the status of HTTP or TCP mode of the server;

- **Sticky Session:** HAProxy can stick a client to a specific server for the duration when the application does not support sticky sessions;
- **SSL:** HTTPS communication and resolution are supported;
- **Monitoring and Statistics:** Through the web page, you can monitor the service state and traffic flow in real time.

### 9.5.3.3 Before you begin

Before you deploy HAProxy, make sure that you meet the hardware and software requirements.

#### 9.5.3.3.1 Hardware requirements

For your server, it is recommended to meet the following hardware requirements. You can also improve server specifications according to the load balancing environment.

Hardware resource	Minimum specification
CPU	2 cores, 3.5 GHz
Memory	16 GB
Storage	50 GB (SATA)
Network Interface Card	10G Network Card

#### 9.5.3.3.2 Software requirements

You can use the following operating systems and make sure the required dependencies are installed. If you use yum to install HAProxy, the dependencies are installed along with it and you do not need to separately install them again.

##### Operating systems

Operating system version	Architecture
Linux 2.4	x86, x86_64, Alpha, SPARC, MIPS, and PA-RISC
Linux 2.6 or 3.x	x86, x86_64, ARM, SPARC, and PPC64
Solaris 8 or 9	UltraSPARC II and III
Solaris 10	Opteron and UltraSPARC
FreeBSD 4.10 ~ 10	x86
OpenBSD 3.1 or later versions	i386, AMD64, macppc, Alpha, and SPARC64
AIX 5.1 ~ 5.3	Power™

##### Dependencies

- epel-release
- gcc
- systemd-devel

To install the dependencies above, run the following command:

```
yum -y install epel-release gcc systemd-devel
```

#### 9.5.3.4 Deploy HAProxy

You can easily use HAProxy to configure and set up a load-balanced database environment. This section shows general deployment operations. You can customize the [configuration file](#) based on your actual scenario.

##### 9.5.3.4.1 Install HAProxy

1. Download the package of the HAProxy 2.5.0 source code:

```
wget https://github.com/haproxy/haproxy/archive/refs/tags/v2.5.0.zip
```

2. Unzip the package:

```
unzip v2.5.0.zip
```

3. Compile the application from the source code:

```
cd haproxy-2.5.0
make clean
make -j 8 TARGET=linux-glibc USE_THREAD=1
make PREFIX=${/app/haproxy} SBINDIR=${/app/haproxy/bin} install #
 ↪ Replace `${/app/haproxy}` and `${/app/haproxy/bin}` with your
 ↪ custom directories.
```

4. Reconfigure the profile:

```
echo 'export PATH=/app/haproxy/bin:$PATH' >> /etc/profile
```

5. Check whether the installation is successful:

```
which haproxy
```

HAProxy commands

Execute the following command to print a list of keywords and their basic usage:

```
haproxy --help
```

Option	Description
-v	Reports the version and build date.
-vv	Displays the version, build options, libraries versions and usable pollers.
-d	Enables debug mode.
-db	Disables background mode and multi-process mode.
-dM [< → byte → >]	Forces memory poisoning, which means that each and every memory region allocated with malloc() or pool_alloc2() will be filled with <byte> before being passed to the caller.

Option	Description
-V	Enables verbose mode (disables quiet mode).
-D	Starts as a daemon.
-C <dir>	Changes to directory <dir> before loading configuration files.
-W	Master-worker mode.
-q	Sets “quiet” mode: This disables some messages during the configuration parsing and during startup.
-c	Only performs a check of the configuration files and exits before trying to bind.
-n ↗ limit ↘ >	Limits the per-process connection limit to <limit>.

Option	Description
<code>-m &lt;   ↳ limit   ↳ &gt;</code>	Limits the total allocatable memory to <limit> megabytes across all processes.
<code>-N &lt;   ↳ limit   ↳ &gt;</code>	Sets the default per-proxy maxconn to <limit> instead of the builtin default value (usually 2000).
<code>-L &lt;name   ↳ &gt;</code>	Changes the local peer name to <name>, which defaults to the local hostname.
<code>-p &lt;file   ↳ &gt;</code>	Writes all processes' PIDs into <file> during startup.

Option	Description
-de	Disables the use of epoll(7). epoll(7) is available only on Linux 2.6 and some custom Linux 2.4 systems.
-dp	Disables the use of poll(2). select(2) might be used instead.
-dS	Disables the use of splice(2), which is broken on older kernels.
-dR	Disables SO_REUSEPORT usage.
-dr	Ignores server address resolution failures.
-dV	Disables SSL verify on the server side.

Option	Description
<code>-sf &lt;→ pidlist&gt;</code>	Sends the “finish” signal to the PIDs in pidlist after startup. The processes which receive this signal wait for all sessions to finish before exiting. This option must be specified last, followed by any number of PIDs. Technically speaking, SIGTTOU and SIGUSR1 are sent.

Option	Description
<code>-st &lt;→ pidlist&gt;</code>	Sends the “terminate” signal to the PIDs in pidlist after startup. The processes which receive this signal terminate immediately, closing all active sessions. This option must be specified last, followed by any number of PIDs. Technically speaking, SIGTTOU and SIGTERM are sent.

Option	Description
-x < → unix_socket → >	Connects the specified socket and retrieves all the listening sockets from the old process. Then, these sockets are used instead of binding new ones.
-S <bind → >[,< → bind_options, → >...]	In master-worker CLI. This CLI enables access to the CLI of every worker. Useful for debugging, it's a convenient way of accessing a leaving process.

For more details on HAProxy command line options, refer to [Management Guide of HAProxy](#) and [General Commands Manual of HAProxy](#).

#### 9.5.3.4.2 Configure HAProxy

A configuration template is generated when you use yum to install HAProxy. You can also customize the following configuration items according to your scenario.

```

global # Global configuration.
 log 127.0.0.1 local2 # Global syslog servers (up to two).
 chroot /var/lib/haproxy # Changes the current directory and
 ↪ sets superuser privileges for the startup process to improve
 ↪ security.
 pidfile /var/run/haproxy.pid # Writes the PIDs of HAProxy processes
 ↪ into this file.
 maxconn 4096 # The maximum number of concurrent
 ↪ connections for a single HAProxy process. It is equivalent to the
 ↪ command-line argument "-n".
 nbthread 48 # The maximum number of threads. (The
 ↪ upper limit is equal to the number of CPUs)
 user haproxy # Same with the UID parameter.
 group haproxy # Same with the GID parameter. A
 ↪ dedicated user group is recommended.
 daemon # Makes the process fork into
 ↪ background. It is equivalent to the command line "-D" argument. It
 ↪ can be disabled by the command line "-db" argument.
 stats socket /var/lib/haproxy/stats # The directory where statistics
 ↪ output is saved.

defaults # Default configuration.
 log global # Inherits the settings of the global
 ↪ configuration.
 retries 2 # The maximum number of retries to
 ↪ connect to an upstream server. If the number of connection attempts
 ↪ exceeds the value, the backend server is considered unavailable.
 timeout connect 2s # The maximum time to wait for a
 ↪ connection attempt to a backend server to succeed. It should be set
 ↪ to a shorter time if the server is located on the same LAN as
 ↪ HAProxy.
 timeout client 30000s # The maximum inactivity time on the
 ↪ client side.
 timeout server 30000s # The maximum inactivity time on the
 ↪ server side.

listen admin_stats # The name of the Stats page reporting
 ↪ information from frontend and backend. You can customize the name
 ↪ according to your needs.
bind 0.0.0.0:8080 # The listening port.
mode http # The monitoring mode.
option httplog # Enables HTTP logging.
maxconn 10 # The maximum number of concurrent
 ↪ connections.

```

```

stats refresh 30s # Automatically refreshes the Stats
 ↪ page every 30 seconds.
stats uri /haproxy # The URL of the Stats page.
stats realm HAProxy # The authentication realm of the
 ↪ Stats page.
stats auth admin:pingcap123 # User name and password in the Stats
 ↪ page. You can have multiple user names.
stats hide-version # Hides the version information of
 ↪ HAProxy on the Stats page.
stats admin if TRUE # Manually enables or disables the
 ↪ backend server (supported in HAProxy 1.4.9 or later versions).

listen tidb-cluster # Database load balancing.
 bind 0.0.0.0:3390 # The Floating IP address and
 ↪ listening port.
mode tcp # HAProxy uses layer 4, the transport
 ↪ layer.
balance leastconn # The server with the smallest number
 ↪ of connections receives the connection. "leastconn" is recommended
 ↪ where long sessions are expected, such as LDAP, SQL and TSE, rather
 ↪ than protocols using short sessions, such as HTTP. The algorithm
 ↪ is dynamic, which means that server weights might be adjusted on
 ↪ the fly for slow starts for instance.
server tidb-1 10.9.18.229:4000 check inter 2000 rise 2 fall 3 # Detects
 ↪ port 4000 at a frequency of once every 2000 milliseconds. If it is
 ↪ detected as successful twice, the server is considered available;
 ↪ if it is detected as failed three times, the server is considered
 ↪ unavailable.
server tidb-2 10.9.39.208:4000 check inter 2000 rise 2 fall 3
server tidb-3 10.9.64.166:4000 check inter 2000 rise 2 fall 3

```

#### 9.5.3.4.3 Start HAProxy

To start HAProxy, run `haproxy`. `/etc/haproxy/haproxy.cfg` is read by default (recommended).

```
haproxy -f /etc/haproxy/haproxy.cfg
```

#### 9.5.3.4.4 Stop HAProxy

To stop HAProxy, use the `kill -9` command.

1. Run the following command:

```
ps -ef | grep haproxy
```

2. Terminate the process of HAProxy:

```
kill -9 ${haproxy.pid}
```

#### 9.5.4 Highly Concurrent Write Best Practices

This document describes best practices for handling highly-concurrent write-heavy workloads in TiDB, which can help to facilitate your application development.

##### 9.5.4.1 Target audience

This document assumes that you have a basic understanding of TiDB. It is recommended that you first read the following three blog articles that explain TiDB fundamentals, and [TiDB Best Practices](#):

- [Data Storage](#)
- [Computing](#)
- [Scheduling](#)

##### 9.5.4.2 Highly-concurrent write-intensive scenario

The highly concurrent write scenario often occurs when you perform batch tasks in applications, such as clearing, settlement and so on. This scenario has the following features:

- A huge volume of data
- The need to import historical data into database in a short time
- The need to read a huge volume of data from database in a short time

These features pose these challenges to TiDB:

- The write or read capacity must be linearly scalable.
- Database performance is stable and does not decrease as a huge volume of data is written concurrently.

For a distributed database, it is important to make full use of the capacity of all nodes and to prevent a single node from becoming the bottleneck.

##### 9.5.4.3 Data distribution principles in TiDB

To address the above challenges, it is necessary to start with the data segmentation and scheduling principle of TiDB. Refer to [Scheduling](#) for more details.

TiDB splits data into Regions, each representing a range of data with a size limit of 96M by default. Each Region has multiple replicas, and each group of replicas is called a Raft Group. In a Raft Group, the Region Leader executes the read and write tasks

(TiDB supports **Follower-Read**) within the data range. The Region Leader is automatically scheduled by the Placement Driver (PD) component to different physical nodes evenly to distribute the read and write pressure.

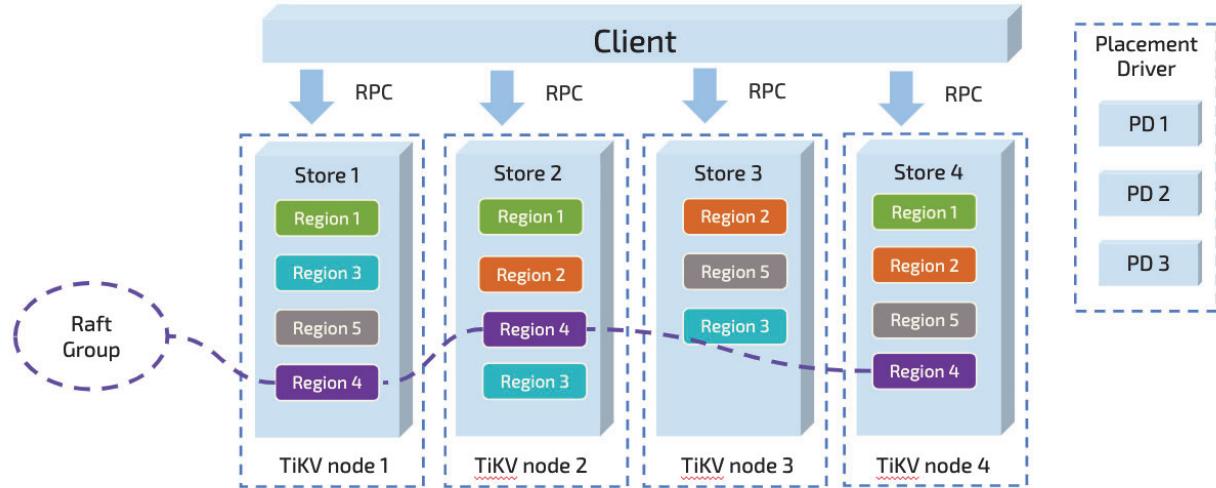


Figure 82: TiDB Data Overview

In theory, by the virtue of this architecture, TiDB can linearly scale its read and write capacities and make full use of the distributed resources so long as there is no `AUTO_INCREMENT` primary key in the write scenario, or there is no monotonically increasing index. From this point of view, TiDB is especially suitable for the highly-concurrent and write-intensive scenario. However, the actual situation often differs from the theoretical assumption.

#### Note:

No `AUTO_INCREMENT` primary key in the write scenario or no monotonically increasing index means no write hotspot in the application.

#### 9.5.4.4 Hotspot case

The following case explains how a hotspot is generated. Take the table below as an example:

```
CREATE TABLE IF NOT EXISTS TEST_HOTSPOT(
 id BIGINT PRIMARY KEY,
 age INT,
 user_name VARCHAR(32),
```

```
email VARCHAR(128)
```

```
)
```

This table is simple in structure. In addition to `id` as the primary key, no secondary index exists. Execute the following statement to write data into this table. `id` is discretely generated as a random number.

```
INSERT INTO TEST_HOTSPOT(id, age, user_name, email) values(%v, %v, '%v', '%
↪ v');
```

The load comes from executing the above statement intensively in a short time.

In theory, the above operation seems to comply with the TiDB best practices, and no hotspot is caused in the application. The distributed capacity of TiDB can be fully used with adequate machines. To verify whether it is truly in line with the best practices, a test is conducted in the experimental environment, which is described as follows:

For the cluster topology, 2 TiDB nodes, 3 PD nodes and 6 TiKV nodes are deployed. Ignore the QPS performance, because this test is to clarify the principle rather than for benchmark.

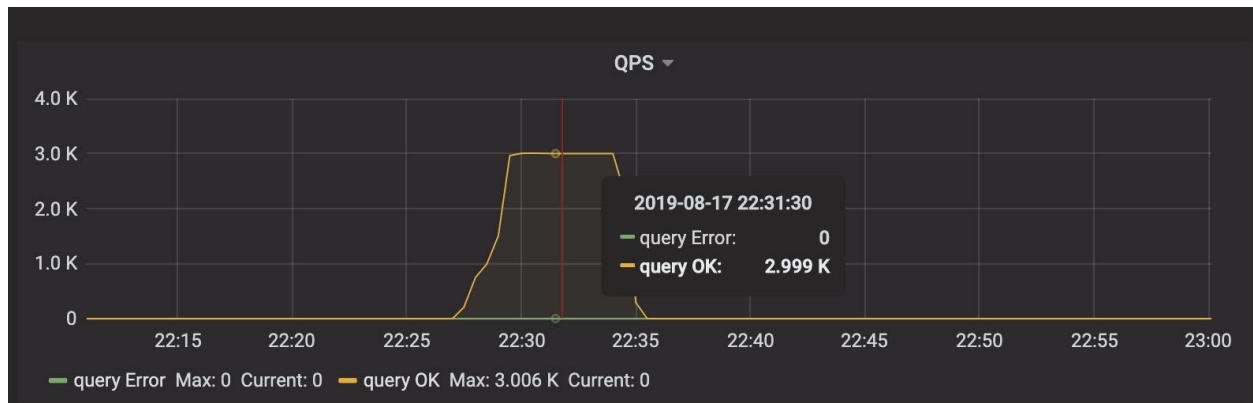


Figure 83: QPS1

The client starts “intensive” write requests in a short time, which is 3K QPS received by TiDB. In theory, the load pressure should be evenly distributed to 6 TiKV nodes. However, from the CPU usage of each TiKV node, the load distribution is uneven. The `tikv-3` node is the write hotspot.

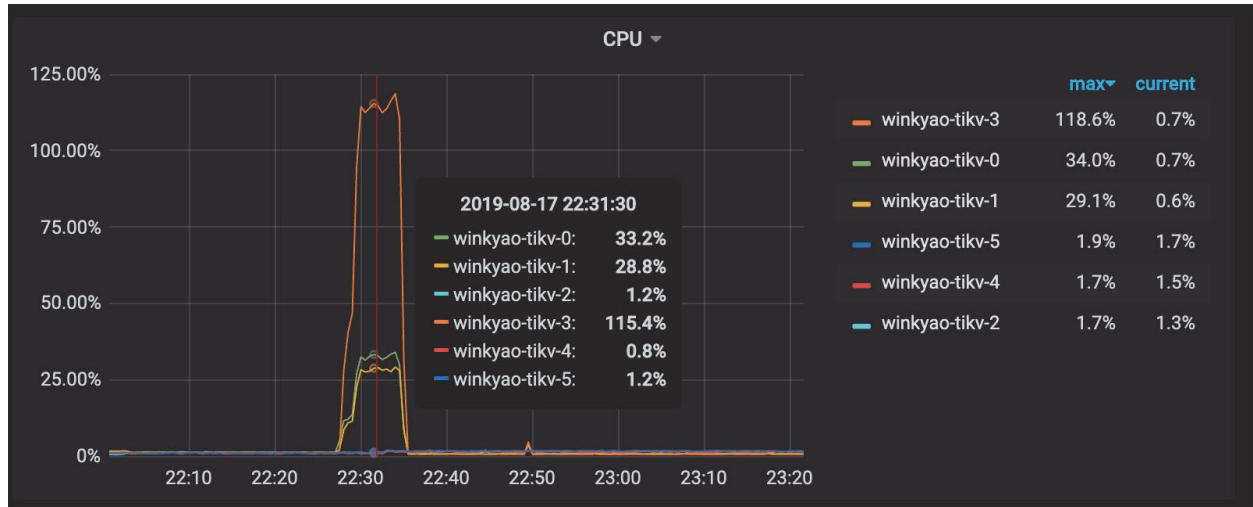


Figure 84: QPS2

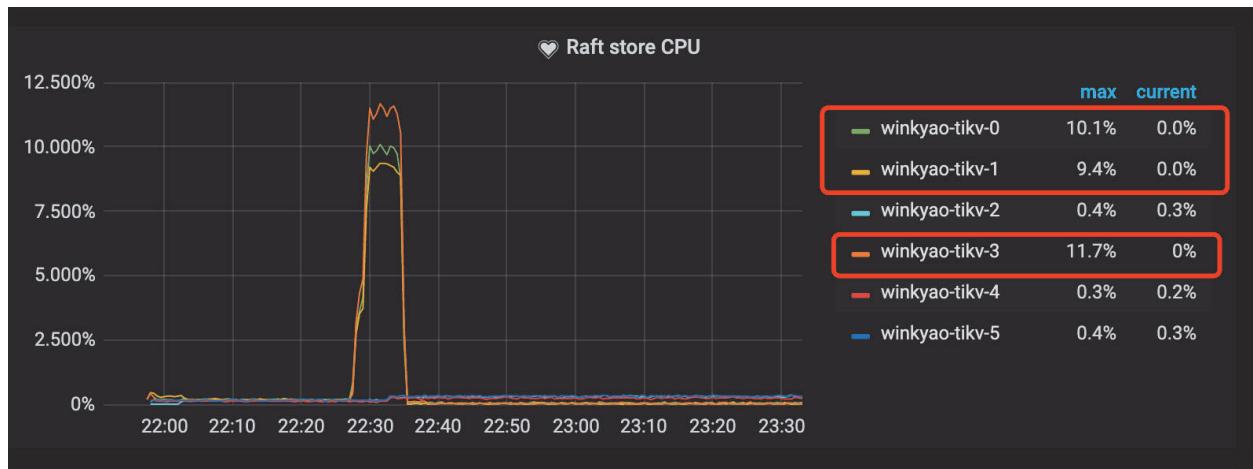


Figure 85: QPS3

**Raft store CPU** is the CPU usage rate for the `raftstore` thread, usually representing the write load. In this scenario, `tikv-3` is the Leader of this Raft Group; `tikv-0` and `tikv-1` are the followers. The loads of other nodes are almost empty.

The monitoring metrics of PD also confirms that hotspot has been caused.



Figure 86: QPS4

#### 9.5.4.5 Hotspot causes

In the above test, the operation does not reach the ideal performance expected in the best practices. This is because only one Region is split by default to store the data of each newly created table in TiDB, with the following data range:

[CommonPrefix + TableID, CommonPrefix + TableID + 1)

In a short period of time, a huge volume of data is continuously written to the same Region.

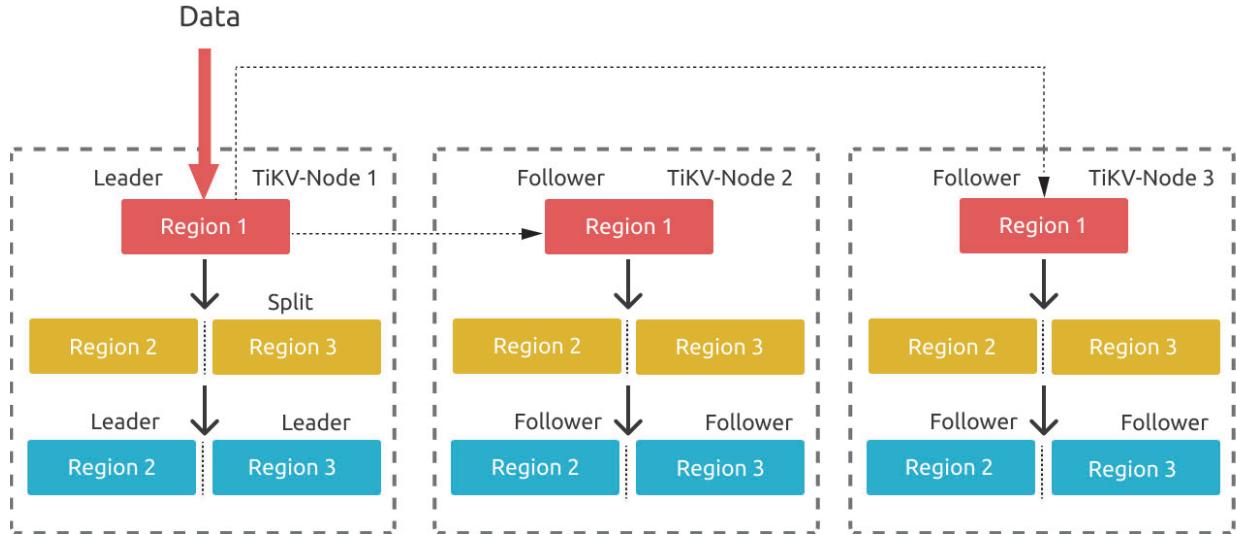


Figure 87: TiKV Region Split

The above diagram illustrates the Region splitting process. As data is continuously written into TiKV, TiKV splits a Region into multiple Regions. Because the leader election is started on the original store where the Region Leader to be split is located, the leaders of the two newly split Regions might be still on the same store. This splitting process might also happen on the newly split Region 2 and Region 3. In this way, write pressure is concentrated on TiKV-Node 1.

During the continuous write process, after finding that hotspot is caused on Node 1, PD evenly distributes the concentrated Leaders to other nodes. If the number of TiKV nodes is more than the number of Region replicas, TiKV will try to migrate these Regions to idle nodes. These two operations during the write process are also reflected in the PD's monitoring metrics:

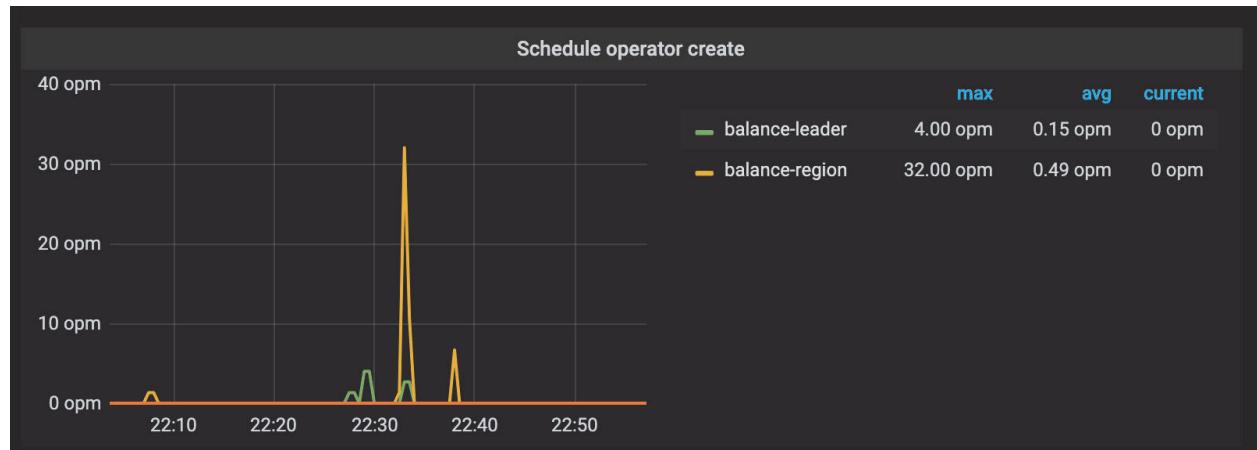


Figure 88: QPS5

After a period of continuous writes, PD automatically schedules the entire TiKV cluster to a state where pressure is evenly distributed. By that time, the capacity of the whole cluster can be fully used.

In most cases, the above process of causing a hotspot is normal, which is the Region warm-up phase of database. However, you need to avoid this phase in highly-concurrent write-intensive scenarios.

#### 9.5.4.6 Hotspot solution

To achieve the ideal performance expected in theory, you can skip the warm-up phase by directly splitting a Region into the desired number of Regions and scheduling these Regions in advance to other nodes in the cluster.

In v3.0.x, v2.1.13 and later versions, TiDB supports a new feature called [Split Region](#). This new feature provides the following new syntaxes:

```
SPLIT TABLE table_name [INDEX index_name] BETWEEN (lower_value) AND (
 ↪ upper_value) REGIONS region_num
```

```
SPLIT TABLE table name [INDEX index name] BY (value list) [, (value list)]
```

However, TiDB does not automatically perform this pre-split operation. The reason is related to the data distribution in TiDB.

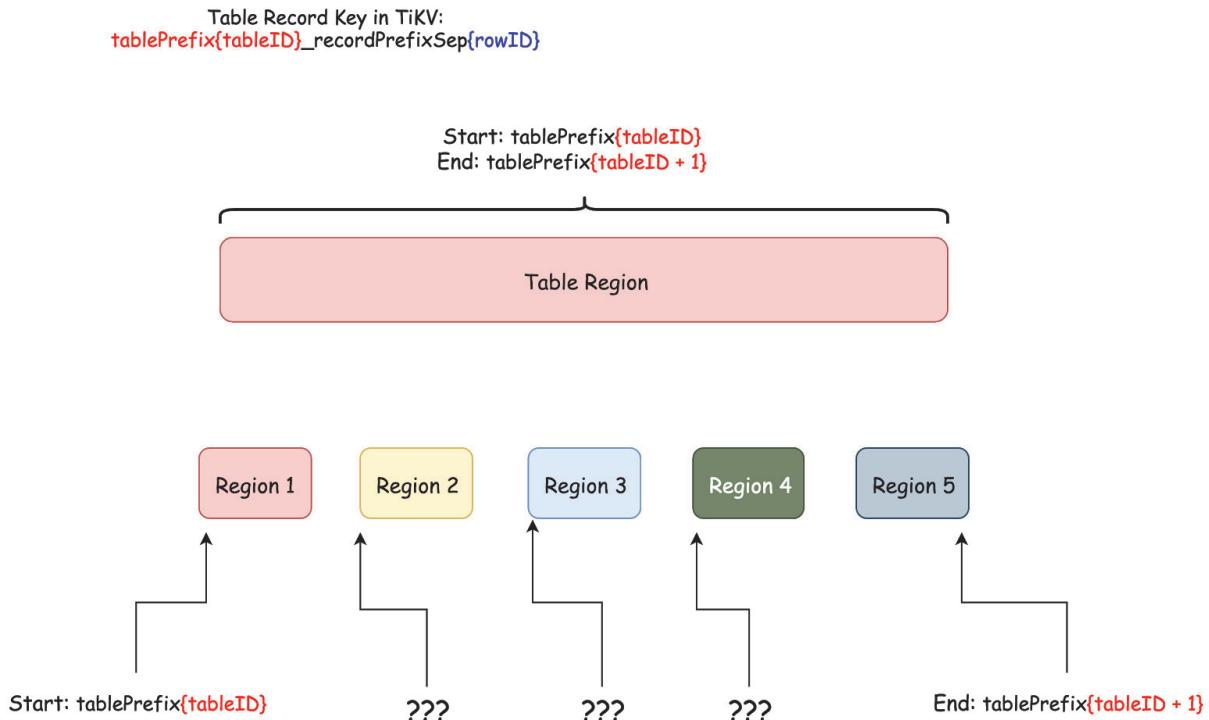


Figure 89: Table Region Range

From the diagram above, according to the encoding rule of a row's key, the `rowID` is the only variable part. In TiDB, `rowID` is an `Int64` integer. However, you might not need to evenly split the `Int64` integer range to the desired number of ranges and then to distribute these ranges to different nodes, because Region split must also be based on the actual situation.

If the write of `rowID` is completely discrete, the above method will not cause hotspots. If the row ID or index has a fixed range or prefix (for example, discretely insert data into the range of [2000w, 5000w]), no hotspot will be caused either. However, if you split a Region using the above method, data might still be written to the same Region at the beginning.

TiDB is a database for general usage and does not make assumptions about the data distribution. So it uses only one Region at the beginning to store the data of a table and automatically splits the Region according to the data distribution after real data is inserted.

Given this situation and the need to avoid the hotspot problem, TiDB offers the `SPLIT → Region` syntax to optimize performance for the highly-concurrent write-heavy scenario. Based on the above case, now scatter Regions using the `SPLIT Region` syntax and observe the load distribution.

Because the data to be written in the test is entirely discrete within the positive range, you can use the following statement to pre-split the table into 128 Regions within the range of `minInt64` and `maxInt64`:

```
SPLIT TABLE TEST_HOTSPOT BETWEEN (0) AND (9223372036854775807) REGIONS 128;
```

After the pre-split operation, execute the `SHOW TABLE test_hotspot REGIONS;` statement to check the status of Region scattering. If the values of the `SCATTERING` column are all 0, the scheduling is successful.

You can also check the Region distribution using the `table-regions.py` script. Currently, the Region distribution is relatively even:

```
[root@172.16.4.4 scripts]# python table-regions.py --host 172.16.4.3 --port
→ 31453 test test_hotspot
[RECORD - test.test_hotspot] - Leaders Distribution:
total leader count: 127
store: 1, num_leaders: 21, percentage: 16.54%
store: 4, num_leaders: 20, percentage: 15.75%
store: 6, num_leaders: 21, percentage: 16.54%
store: 46, num_leaders: 21, percentage: 16.54%
store: 82, num_leaders: 23, percentage: 18.11%
store: 62, num_leaders: 21, percentage: 16.54%
```

Then operate the write load again:

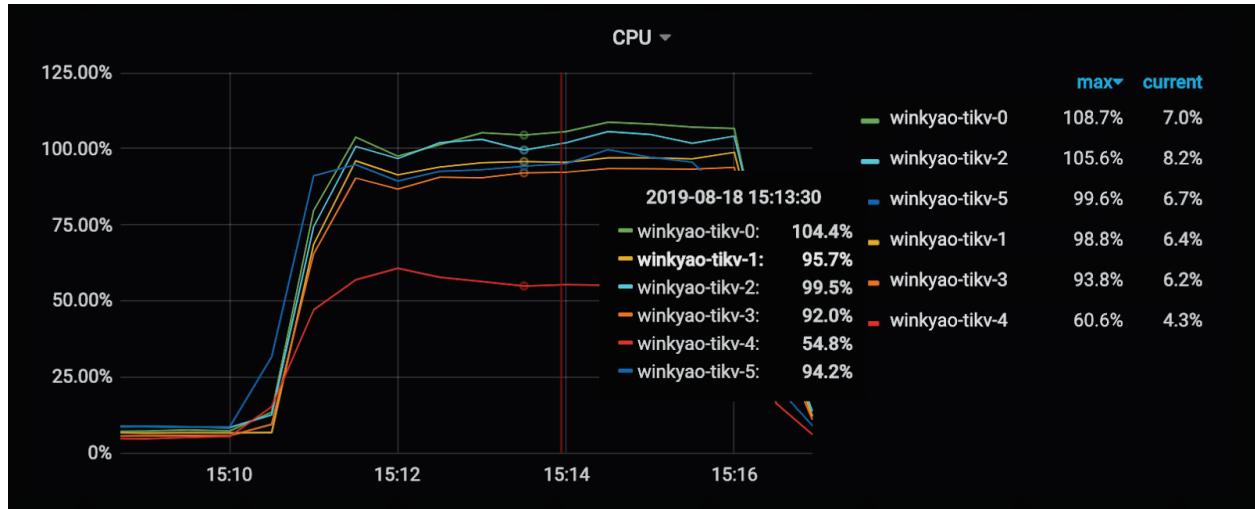


Figure 90: QPS6

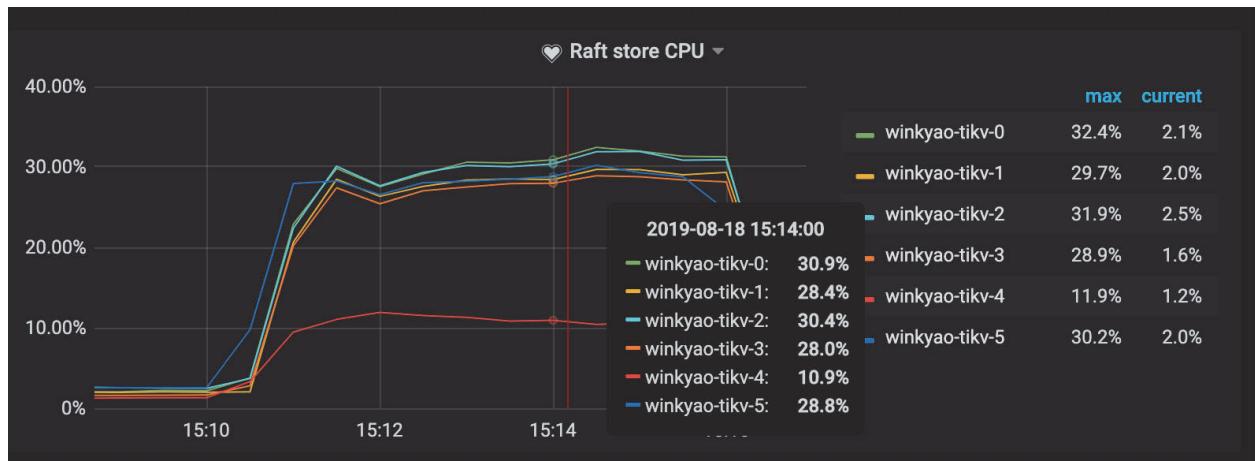


Figure 91: QPS7

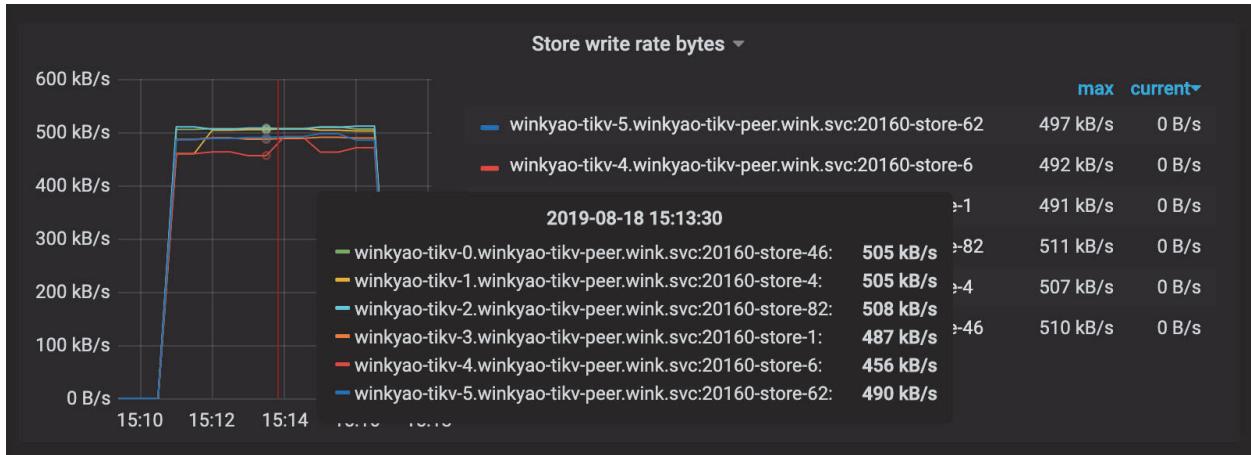


Figure 92: QPS8

You can see that the apparent hotspot problem has been resolved now.

In this case, the table is simple. In other cases, you might also need to consider the hotspot problem of index. For more details on how to pre-split the index Region, refer to [Split Region](#).

#### 9.5.4.7 Complex hotspot problems

##### Problem one:

If a table does not have a primary key, or the primary key is not the `Int` type and you do not want to generate a randomly distributed primary key ID, TiDB provides an implicit `_tidb_rowid` column as the row ID. Generally, when you do not use the `SHARD_ROW_ID_BITS` parameter, the values of the `_tidb_rowid` column are also monotonically increasing, which might causes hotspots too. Refer to [SHARD\\_ROW\\_ID\\_BITS](#) for more details.

To avoid the hotspot problem in this situation, you can use `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` when creating a table. For more details about `PRE_SPLIT_REGIONS`, refer to [Pre-split Regions](#).

`SHARD_ROW_ID_BITS` is used to randomly scatter the row ID generated in the `_tidb_rowid` column. `PRE_SPLIT_REGIONS` is used to pre-split the Region after a table is created.

##### Note:

The value of `PRE_SPLIT_REGIONS` must be smaller than or equal to that of `SHARD_ROW_ID_BITS`.

Example:

```
create table t (a int, b int) SHARD_ROW_ID_BITS = 4 PRE_SPLIT_REGIONS=3;
```

- SHARD\_ROW\_ID\_BITS = 4 means that the values of `tidb_rowid` will be randomly distributed into 16 ( $16=2^4$ ) ranges.
- PRE\_SPLIT\_REGIONS=3 means that the table will be pre-split into 8 ( $2^3$ ) Regions after it is created.

When data starts to be written into table `t`, the data is written into the pre-split 8 Regions, which avoids the hotspot problem that might be caused if only one Region exists after table creation.

#### Note:

The `tidb_scatter_region` global variable affects the behavior of `PRE_SPLIT_REGIONS`.

This variable controls whether to wait for Regions to be pre-split and scattered before returning results after the table creation. If there are intensive writes after creating the table, you need to set the value of this variable to 1, then TiDB will not return the results to the client until all the Regions are split and scattered. Otherwise, TiDB writes data before the scattering is completed, which will have a significant impact on write performance.

#### Problem two:

If a table's primary key is an integer type, and if the table uses `AUTO_INCREMENT` to ensure the uniqueness of the primary key (not necessarily continuous or incremental), you cannot use `SHARD_ROW_ID_BITS` to scatter the hotspot on this table because TiDB directly uses the row values of the primary key as `_tidb_rowid`.

To address the problem in this scenario, you can replace `AUTO_INCREMENT` with `AUTO_RANDOM` (a column attribute) when inserting data. Then TiDB automatically assigns values to the integer primary key column, which eliminates the continuity of the row ID and scatters the hotspot.

##### 9.5.4.8 Parameter configuration

In v2.1, the `latch mechanism` is introduced in TiDB to identify transaction conflicts in advance in scenarios where write conflicts frequently appear. The aim is to reduce the retry of transaction commits in TiDB and TiKV caused by write conflicts. Generally, batch tasks use the data already stored in TiDB, so the write conflicts of transaction do not exist. In this situation, you can disable the latch in TiDB to reduce memory allocation for small objects:

```
[txn-local-latches]
enabled = false
```

### 9.5.5 Best Practices for Monitoring TiDB Using Grafana

When you [deploy a TiDB cluster using TiUP](#) and have added Grafana and Prometheus in the topology configuration, a set of [Grafana + Prometheus monitoring platform](#) is deployed simultaneously to collect and display metrics for various components and machines in the TiDB cluster. This document describes best practices for monitoring TiDB using Grafana. It aims to help you use metrics to analyze the status of the TiDB cluster and diagnose problems.

#### 9.5.5.1 Monitoring architecture

[Prometheus](#) is a time series database with a multi-dimensional data model and a flexible query language. [Grafana](#) is an open source monitoring system for analyzing and visualizing metrics.

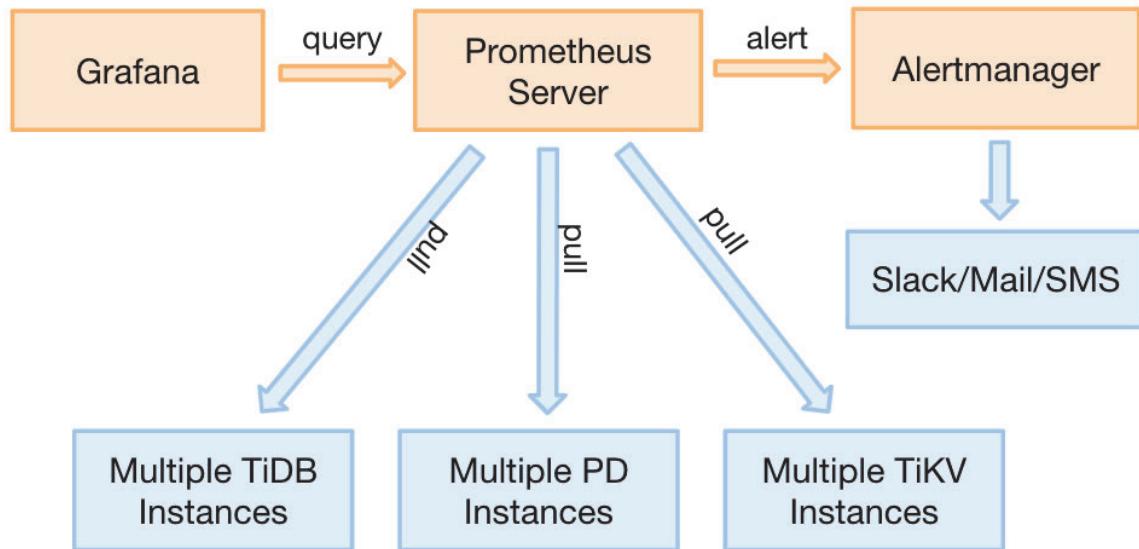


Figure 93: The monitoring architecture in the TiDB cluster

For TiDB 2.1.3 or later versions, TiDB monitoring supports the pull method. It is a good adjustment with the following benefits:

- There is no need to restart the entire TiDB cluster if you need to migrate Prometheus. Before adjustment, migrating Prometheus requires restarting the entire cluster because the target address needs to be updated.

- You can deploy 2 separate sets of Grafana + Prometheus monitoring platforms (not highly available) to prevent a single point of monitoring.
- The Pushgateway which might become a single point of failure is removed.

#### 9.5.5.2 Source and display of monitoring data

The three core components of TiDB (TiDB server, TiKV server and PD server) obtain metrics through the HTTP interface. These metrics are collected from the program code, and the ports are as follows:

Component	Port
TiDB server	10080
TiKV server	20181
PD server	2379

Execute the following command to check the QPS of a SQL statement through the HTTP interface. Take the TiDB server as an example:

```
curl http://__tidb_ip__:10080/metrics |grep tidb_executor_statement_total
```

```
Check the real-time QPS of different types of SQL statements. The
→ numbers below are the cumulative values of counter type (scientific
→ notation).
tidb_executor_statement_total{type="Delete"} 520197
tidb_executor_statement_total{type="Explain"} 1
tidb_executor_statement_total{type="Insert"} 7.20799402e+08
tidb_executor_statement_total{type="Select"} 2.64983586e+08
tidb_executor_statement_total{type="Set"} 2.399075e+06
tidb_executor_statement_total{type="Show"} 500531
tidb_executor_statement_total{type="Use"} 466016
```

The data above is stored in Prometheus and displayed on Grafana. Right-click the panel and then click the **Edit** button (or directly press the E key) shown in the following figure:

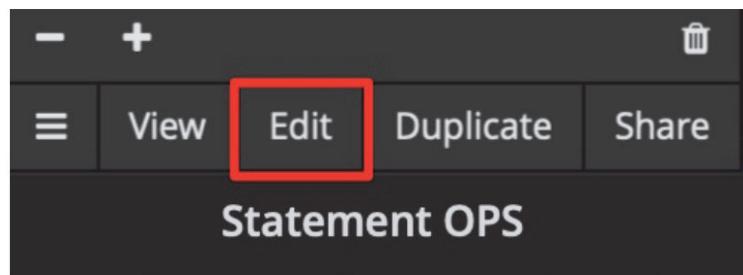


Figure 94: The Edit entry for the Metrics tab

After clicking the **Edit** button, you can see the query expression with the `tidb_executor_statement_total` metric name on the Metrics tab. The meanings of some items on the panel are as follows:

- `rate[1m]`: The growth rate in one minute. It can only be used for the data of counter type.
- `sum`: The sum of values.
- `by type`: The summed data is grouped by type in the original metric value.
- `Legend format`: The format of the metric name.
- `Resolution`: The step width defaults to 15 seconds. Resolution means whether to generate one data point for multiple pixels.

The query expression on the **Metrics** tab is as follows:



Figure 95: The query expression on the Metrics tab

Prometheus supports many query expressions and functions. For more details, refer to [Prometheus official website](#).

### 9.5.5.3 Grafana tips

This section introduces seven tips for efficiently using Grafana to monitor and analyze the metrics of TiDB.

#### 9.5.5.3.1 Tip 1: Check all dimensions and edit the query expression

In the example shown in the [source and display of monitoring data](#) section, the data is grouped by type. If you want to know whether you can group by other dimensions and quickly check which dimensions are available, you can use the following method: **Only keep**

the metric name on the query expression, no calculation, and leave the `Legend format` field blank. In this way, the original metrics are displayed. For example, the following figure shows that there are three dimensions (`instance`, `job` and `type`):



Figure 96: Edit query expression and check all dimensions

Then you can modify the query expression by adding the `instance` dimension after `type`, and adding `{{instance}}` to the `Legend format` field. In this way, you can check the QPS of different types of SQL statements that are executed on each TiDB server:



Figure 97: Add an instance dimension to the query expression

#### 9.5.5.3.2 Tip 2: Switch the scale of the Y-axis

Take Query Duration as an example, the Y-axis defaults to be on a binary logarithmic scale (log2n), which narrows the gap in display. To amplify changes, you can switch it to a linear scale. Comparing the following two figures, you can easily notice the difference in display, and locate the time when an SQL statement runs slowly.

Of course, a linear scale is not suitable for all situations. For example, if you observe the performance trend for the duration of a month, there might be noises with a linear scale, making it hard to observe.

The Y-axis uses a binary logarithmic scale by default:

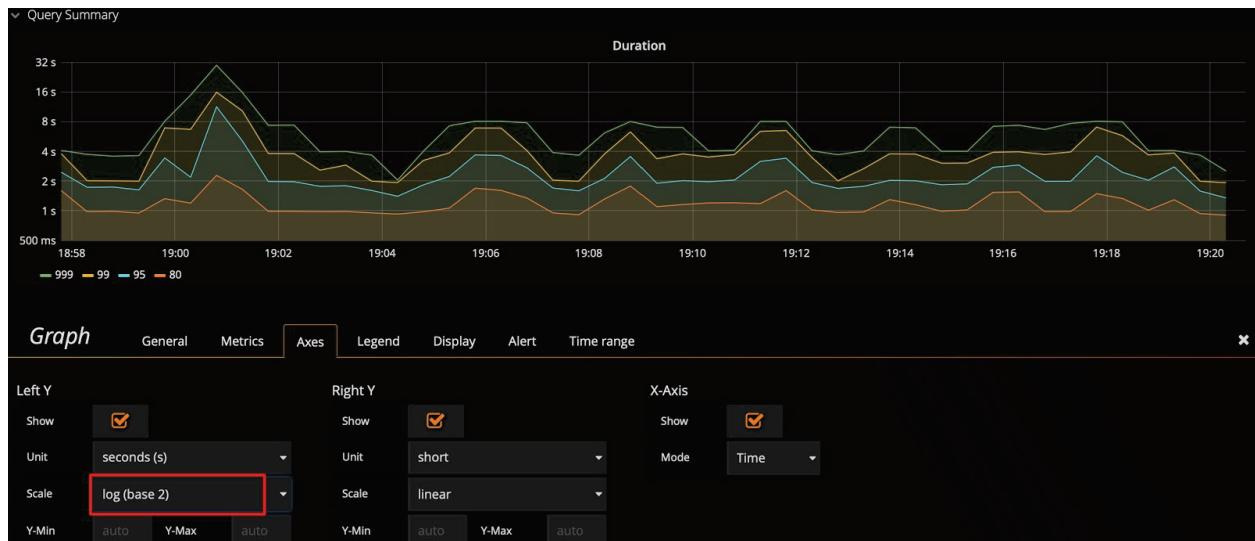


Figure 98: The Y-axis uses a binary logarithmic scale

Switch the Y-axis to a linear scale:



Figure 99: Switch to a linear scale

#### Tip:

Combining tip 2 with tip 1, you can find a `sql_type` dimension to help you immediately analyze whether the `SELECT` statement or the `UPDATE` statement is slow; you can even locate the instance with slow SQL statements.

#### 9.5.5.3.3 Tip 3: Modify the baseline of the Y-axis to amplify changes

You might still cannot see the trend after switching to the linear scale. For example, in the following figure, you want to observe the real-time change of `Store size` after scaling the cluster, but due to the large baseline, small changes are not visible. In this situation, you can modify the baseline of the Y-axis from 0 to `auto` to zoom in the upper part. Check the two figures below, you can see data migration begins.

The baseline defaults to 0:

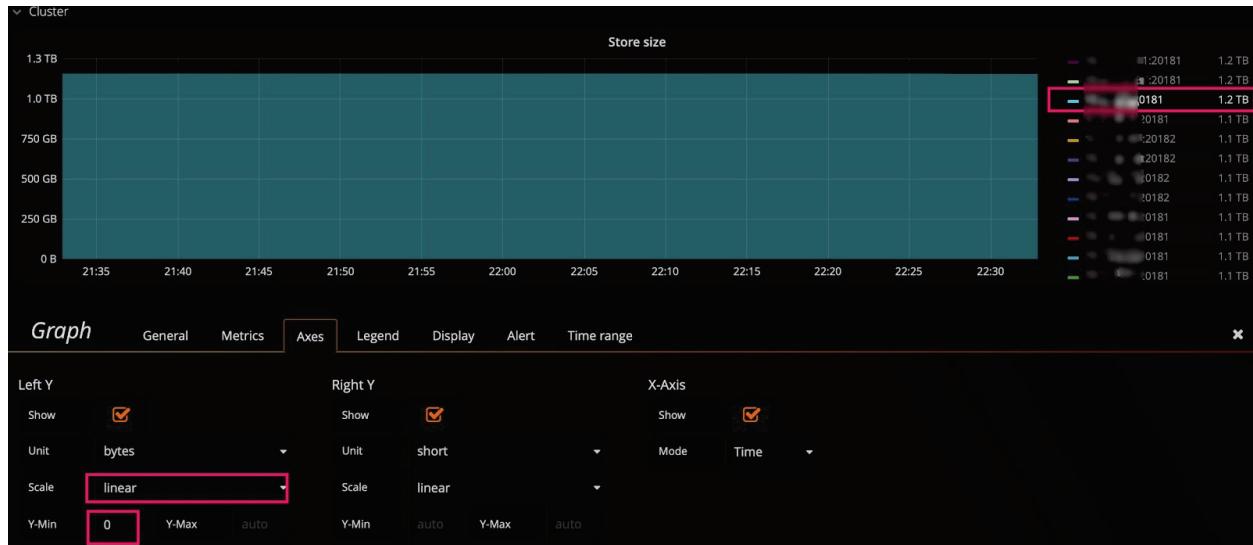


Figure 100: Baseline defaults to 0

Change the baseline to auto:



Figure 101: Change the baseline to auto

#### 9.5.5.3.4 Tip 4: Use Shared crosshair or Tooltip

In the **Settings** panel, there is a **Graph Tooltip** panel option which defaults to **Default**.

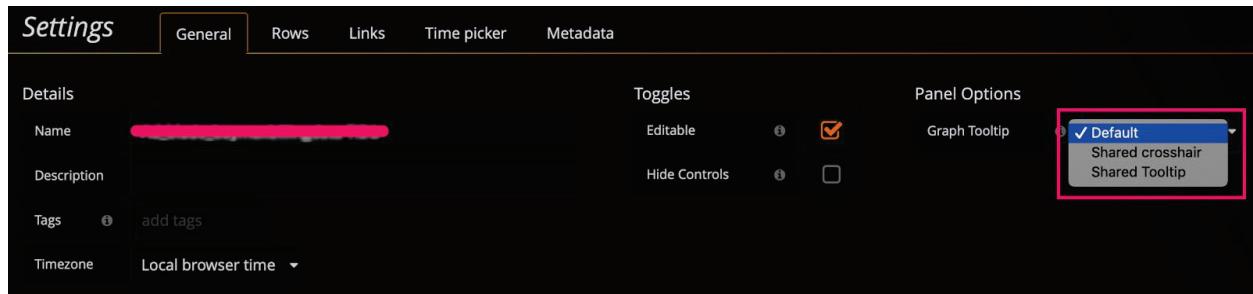


Figure 102: Graphic presentation tools

You can use **Shared crosshair** and **Shared Tooltip** respectively to test the effect as shown in the following figures. Then, the scales are displayed in linkage, which is convenient to confirm the correlation of two metrics when diagnosing problems.

Set the graphic presentation tool to **Shared crosshair**:

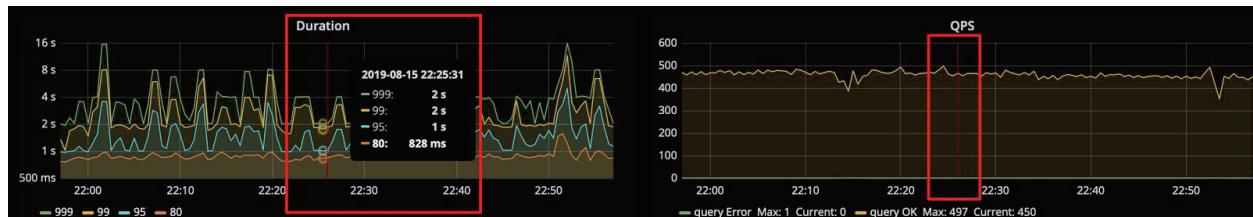


Figure 103: Set the graphical presentation tool to Shared crosshair

Set the graphical presentation tool to **Shared Tooltip**:

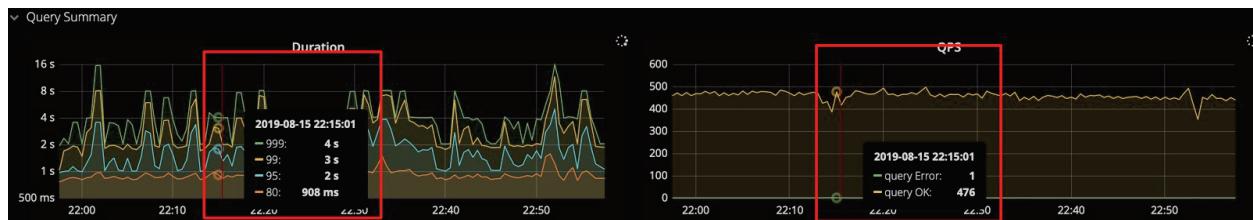


Figure 104: Set the graphic presentation tool to Shared Tooltip

#### 9.5.5.3.5 Tip 5: Enter IP address:port number to check the metrics in history

PD's dashboard only shows the metrics of the current leader. If you want to check the status of a PD leader in history and it no longer exists in the drop-down list of the `instance` field, you can manually enter `IP address:2379` to check the data of the leader.

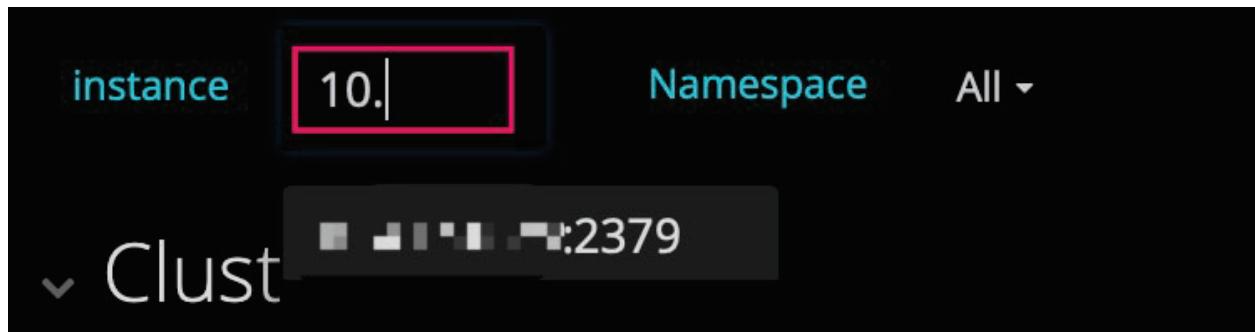


Figure 105: Check the metrics in history

#### 9.5.5.3.6 Tip 6: Use the Avg function

Generally, only Max and Current functions are available in the legend by default. When the metrics fluctuate greatly, you can add other summary functions such as the Avg function to the legend to check the overall trend for the duration of time.

Add summary functions such as the Avg function:

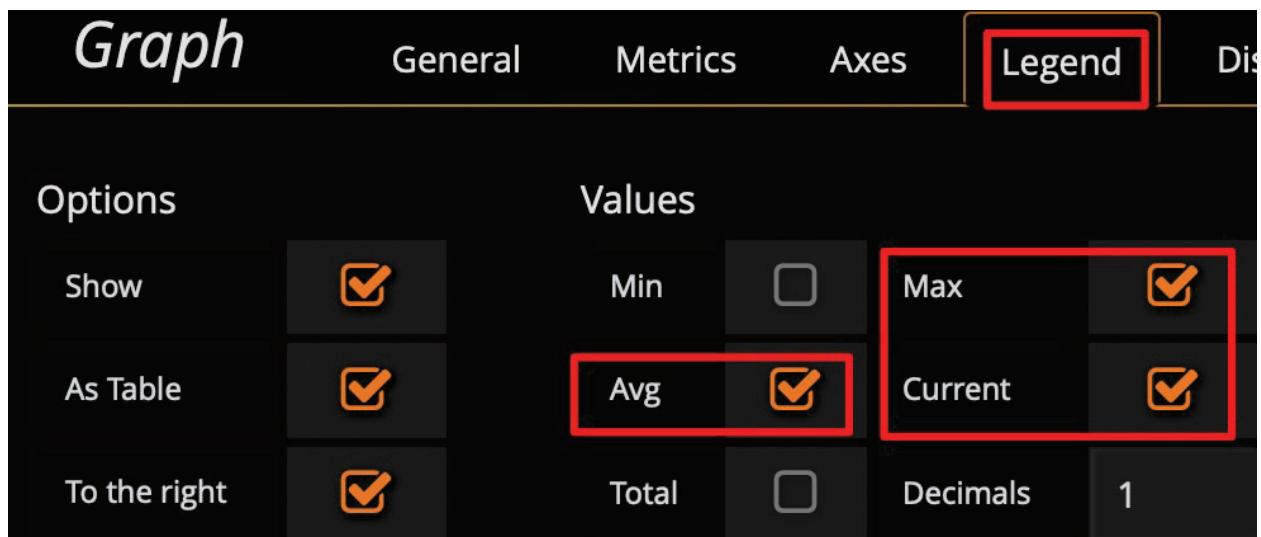


Figure 106: Add summary functions such as Avg

Then check the overall trend:

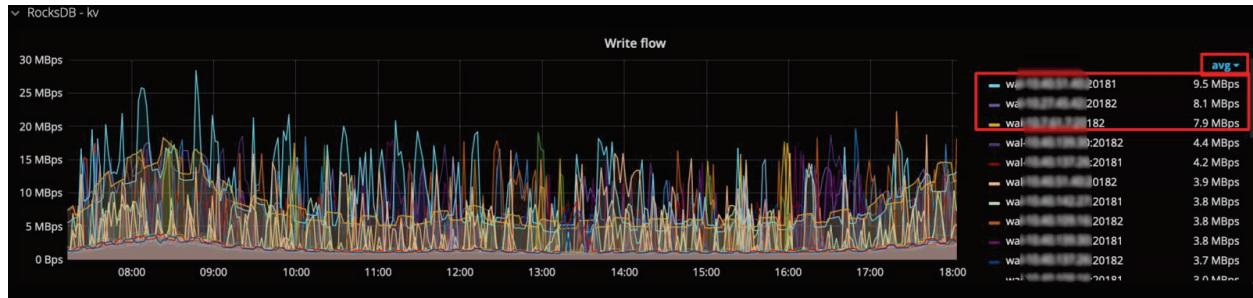


Figure 107: Add Avg function to check the overall trend

#### 9.5.5.3.7 Tip 7: Use the API of Prometheus to obtain the result of query expressions

Grafana obtains data through the API of Prometheus and you can use this API to obtain information as well. In addition, it also has the following usages:

- Automatically obtains information such as the cluster size and status.
- Makes minor changes to the expression to provide information for the report, such as counting the total amount of QPS per day, the peak value of QPS per day, and the response time per day.
- Performs regular health inspection on the important metrics.

The API of Prometheus is shown as follows:



Figure 108: The API of Prometheus

```

curl -u user:pass 'http://__grafana_ip__:3000/api/datasources/proxy/1/api/v1
 ↪ /query_range?query=sum(tikv_engine_size_bytes%7
 ↪ Binstancexxxxxxxxx20181%22%7D)%20by%20(instance)&start=1565879269&end
 ↪ =1565882869&step=30' |python -m json.tool

```

```
{
 "data": {
```

```

"result": [
 {
 "metric": {
 "instance": "xxxxxxxxxx:20181"
 },
 "values": [
 [
 1565879269,
 "1006046235280"
],
 [
 1565879299,
 "1006057877794"
],
 [
 1565879329,
 "1006021550039"
],
 [
 1565879359,
 "1006021550039"
],
 [
 1565882869,
 "1006132630123"
]
]
 }
],
"resultType": "matrix"
},
"status": "success"
}

```

#### 9.5.5.4 Summary

The Grafana + Prometheus monitoring platform is a very powerful tool. Making good use of it can improve efficiency, saving you a lot of time on analyzing the status of the TiDB cluster. More importantly, it can help you diagnose problems. This tool is very useful in the operation and maintenance of TiDB clusters, especially when there is a large amount of data.

## 9.5.6 PD Scheduling Best Practices

This document details the principles and strategies of PD scheduling through common scenarios to facilitate your application. This document assumes that you have a basic understanding of TiDB, TiKV and PD with the following core concepts:

- leader/follower/learner
- operator
- operator step
- pending/down
- region/peer/Raft group
- region split
- scheduler
- store

### Note:

This document initially targets TiDB 3.0. Although some features are not supported in earlier versions (2.x), the underlying mechanisms are similar and this document can still be used as a reference.

### 9.5.6.1 PD scheduling policies

This section introduces the principles and processes involved in the scheduling system.

#### 9.5.6.1.1 Scheduling process

The scheduling process generally has three steps:

1. Collect information

Each TiKV node periodically reports two types of heartbeats to PD:

- **StoreHeartbeat**: Contains the overall information of stores, including disk capacity, available storage, and read/write traffic
- **RegionHeartbeat**: Contains the overall information of regions, including the range of each region, peer distribution, peer status, data volume, and read/write traffic

PD collects and restores this information for scheduling decisions.

2. Generate operators

Different schedulers generate the operators based on their own logic and requirements, with the following considerations:

- Do not add peers to a store in abnormal states (disconnected, down, busy, low space)
- Do not balance regions in abnormal states
- Do not transfer a leader to a pending peer
- Do not remove a leader directly
- Do not break the physical isolation of various region peers
- Do not violate constraints such as label property

### 3. Execute operators

To execute the operators, the general procedure is:

1. The generated operator first joins a queue managed by `OperatorController`.
2. `OperatorController` takes the operator out of the queue and executes it with a certain amount of concurrency based on the configuration. This step is to assign each operator step to the corresponding region leader.
3. The operator is marked as “finish” or “timeout” and removed from the queue.

#### 9.5.6.1.2 Load balancing

Region primarily relies on `balance-leader` and `balance-region` schedulers to achieve load balance. Both schedulers target distributing regions evenly across all stores in the cluster but with separate focuses: `balance-leader` deals with region leader to balance incoming client requests, whereas `balance-region` concerns itself with each region peer to redistribute the pressure of storage and avoid exceptions like out of storage space.

`balance-leader` and `balance-region` share a similar scheduling process:

1. Rate stores according to their resource availability.
2. `balance-leader` or `balance-region` constantly transfer leaders or peers from stores with high scores to those with low scores.

However, their rating methods are different. `balance-leader` uses the sum of all region sizes corresponding to leaders in a store, whereas the way of `balance-region` is relatively complicated. Depending on the specific storage capacity of each node, the rating method of `balance-region` might:

- based on the amount of data when there is sufficient storage (to balance data distribution among nodes).
- based on the available storage when there is insufficient storage (to balance the storage availability on different nodes).
- based on the weighted sum of the two factors above when neither of the situations applies.

Because different nodes might differ in performance, you can also set the weight of load balancing for different stores. `leader-weight` and `region-weight` are used to control the leader weight and region weight respectively (“1” by default for both). For example, when the `leader-weight` of a store is set to “2”, the number of leaders on the node is about twice as many as that of other nodes after the scheduling stabilizes. Similarly, when the `leader-weight` of a store is set to “0.5”, the number of leaders on the node is about half as many as that of other nodes.

#### 9.5.6.1.3 Hot regions scheduling

For hot regions scheduling, use `hot-region-scheduler`. Since TiDB v3.0, the process is performed as follows:

1. Count hot regions by determining read/write traffic that exceeds a certain threshold for a certain period based on the information reported by stores.
2. Redistribute these regions in a similar way to load balancing.

For hot write regions, `hot-region-scheduler` attempts to redistribute both region peers and leaders; for hot read regions, `hot-region-scheduler` only redistributes region leaders.

#### 9.5.6.1.4 Cluster topology awareness

Cluster topology awareness enables PD to distribute replicas of a region as much as possible. This is how TiKV ensures high availability and disaster recovery capability. PD continuously scans all regions in the background. When PD finds that the distribution of regions is not optimal, it generates an operator to replace peers and redistribute regions.

The component to check region distribution is `replicaChecker`, which is similar to a scheduler except that it cannot be disabled. `replicaChecker` schedules based on the configuration of `location-labels`. For example, `[zone,rack,host]` defines a three-tier topology for a cluster. PD attempts to schedule region peers to different zones first, or to different racks when zones are insufficient (for example, 2 zones for 3 replicas), or to different hosts when racks are insufficient, and so on.

#### 9.5.6.1.5 Scale-down and failure recovery

Scale-down refers to the process when you take a store offline and mark it as “offline” using a command. PD replicates the regions on the offline node to other nodes by scheduling. Failure recovery applies when stores failed and cannot be recovered. In this case, regions with peers distributed on the corresponding store might lose replicas, which requires PD to replenish on other nodes.

The processes of scale-down and failure recovery are basically the same. `replicaChecker` → finds a region peer in abnormal states, and then generates an operator to replace the abnormal peer with a new one on a healthy store.

### 9.5.6.1.6 Region merge

Region merge refers to the process of merging adjacent small regions. It serves to avoid unnecessary resource consumption by a large number of small or even empty regions after data deletion. Region merge is performed by `mergeChecker`, which processes in a similar way to `replicaChecker`: PD continuously scans all regions in the background, and generates an operator when contiguous small regions are found.

Specifically, when a newly split Region exists for more than the value of `split-merge-interval` (1h by default), if any of the following conditions occurs, this Region triggers the Region merge scheduling:

- The size of this Region is smaller than the value of the `max-merge-region-size` (20 MiB by default)
- The number of keys in this Region is smaller than the value of `max-merge-region-keys` (200,000 by default).

### 9.5.6.2 Query scheduling status

You can check the status of scheduling system through metrics, pd-ctl and logs. This section briefly introduces the methods of metrics and pd-ctl. Refer to [PD Monitoring Metrics](#) and [PD Control](#) for details.

#### 9.5.6.2.1 Operator status

The [Grafana PD/Operator](#) page shows the metrics about operators, among which:

- Schedule operator create: Operator creating information
- Operator finish duration: Execution time consumed by each operator
- Operator step duration: Execution time consumed by the operator step

You can query operators using pd-ctl with the following commands:

- `operator show`: Queries all operators generated in the current scheduling task
- `operator show [admin | leader | region]`: Queries operators by type

#### 9.5.6.2.2 Balance status

The [Grafana PD/Statistics - Balance](#) page shows the metrics about load balancing, among which:

- Store leader/region score: Score of each store
- Store leader/region count: The number of leaders/regions in each store
- Store available: Available storage on each store

You can use store commands of pd-ctl to query balance status of each store.

#### 9.5.6.2.3 Hot Region status

The **Grafana PD/Statistics - hotspot** page shows the metrics about hot regions, among which:

- Hot write region's leader/peer distribution: the leader/peer distribution in hot write regions
- Hot read region's leader distribution: the leader distribution in hot read regions

You can also query the status of hot regions using pd-ctl with the following commands:

- `hot read`: Queries hot read regions
- `hot write`: Queries hot write regions
- `hot store`: Queries the distribution of hot regions by store
- `region topread [limit]`: Queries the region with top read traffic
- `region topwrite [limit]`: Queries the region with top write traffic

#### 9.5.6.2.4 Region health

The **Grafana PD/Cluster/Region health** panel shows the metrics about regions in abnormal states.

You can query the list of regions in abnormal states using pd-ctl with region check commands:

- `region check miss-peer`: Queries regions without enough peers
- `region check extra-peer`: Queries regions with extra peers
- `region check down-peer`: Queries regions with down peers
- `region check pending-peer`: Queries regions with pending peers

#### 9.5.6.3 Control scheduling strategy

You can use pd-ctl to adjust the scheduling strategy from the following three aspects. Refer to [PD Control](#) for more details.

##### 9.5.6.3.1 Add/delete scheduler manually

PD supports dynamically adding and removing schedulers directly through pd-ctl. For example:

- `scheduler show`: Shows currently running schedulers in the system
- `scheduler remove balance-leader-scheduler`: Removes (disable) balance-leader-scheduler
- `scheduler add evict-leader-scheduler 1`: Adds a scheduler to remove all leaders in Store 1

### 9.5.6.3.2 Add/delete Operators manually

PD also supports adding or removing operators directly through pd-ctl. For example:

- `operator add add-peer 2 5`: Adds peers to Region 2 in Store 5
- `operator add transfer-leader 2 5`: Migrates the leader of Region 2 to Store 5
- `operator add split-region 2`: Splits Region 2 into two regions evenly in size
- `operator remove 2`: Removes currently pending operator in Region 2

### 9.5.6.3.3 Adjust scheduling parameter

You can check the scheduling configuration using the `config show` command in pd-ctl, and adjust the values using `config set {key} {value}`. Common adjustments include:

- `leader-schedule-limit`: Controls the concurrency of transferring leader scheduling
- `region-schedule-limit`: Controls the concurrency of adding/deleting peer scheduling
- `disable-replace-offline-replica`: Determines whether to disable the scheduling to take nodes offline
- `disable-location-replacement`: Determines whether to disable the scheduling that handles the isolation level of regions
- `max-snapshot-count`: Controls the maximum concurrency of sending/receiving snapshots for each store

### 9.5.6.4 PD scheduling in common scenarios

This section illustrates the best practices of PD scheduling strategies through several typical scenarios.

#### 9.5.6.4.1 Leaders/regions are not evenly distributed

The rating mechanism of PD determines that leader count and region count of different stores cannot fully reflect the load balancing status. Therefore, it is necessary to confirm whether there is load imbalancing from the actual load of TiKV or storage usage.

Once you have confirmed that leaders/region are not evenly distributed, you need to check the rating of different stores.

If the scores of different stores are close, it means PD mistakenly believes that leaders/regions are evenly distributed. Possible reasons are:

- There are hot regions that cause load imbalancing. In this case, you need to analyze further based on [hot regions scheduling](#).
- There are a large number of empty regions or small regions, which leads to a great difference in the number of leaders in different stores and high pressure on Raft store. This is the time for a [region merge](#) scheduling.

- Hardware and software environment varies among stores. You can adjust the values of `leader-weight` and `region-weight` accordingly to control the distribution of leader/region.
- Other unknown reasons. Still you can adjust the values of `leader-weight` and `region-weight` to control the distribution of leader/region.

If there is a big difference in the rating of different stores, you need to examine the operator-related metrics, with special focus on the generation and execution of operators. There are two main situations:

- When operators are generated normally but the scheduling process is slow, it is possible that:
  - The scheduling speed is limited by default for load balancing purpose. You can adjust `leader-schedule-limit` or `region-schedule-limit` to larger values without significantly impacting regular services. In addition, you can also properly ease the restrictions specified by `max-pending-peer-count` and `max-snapshot-count`.
  - Other scheduling tasks are running concurrently, which slows down the balancing. In this case, if the balancing takes precedence over other scheduling tasks, you can stop other tasks or limit their speeds. For example, if you take some nodes offline when balancing is in progress, both operations consume the quota of `region-schedule-limit`. In this case, you can limit the speed of scheduler to remove nodes, or simply set `disable-replace-offline-replica = true` to temporarily disable it.
  - The scheduling process is too slow. You can check the **Operator step duration** metric to confirm the cause. Generally, steps that do not involve sending and receiving snapshots (such as `TransferLeader`, `RemovePeer`, `PromoteLearner`) should be completed in milliseconds, while steps that involve snapshots (such as `AddLearner` and `AddPeer`) are expected to be completed in tens of seconds. If the duration is obviously too long, it could be caused by high pressure on TiKV or bottleneck in network, etc., which needs specific analysis.
- PD fails to generate the corresponding balancing scheduler. Possible reasons include:
  - The scheduler is not activated. For example, the corresponding scheduler is deleted, or its limit is set to “0”.
  - Other constraints. For example, `evict-leader-scheduler` in the system prevents leaders from being migrating to the corresponding store. Or label property is set, which makes some stores reject leaders.
  - Restrictions from the cluster topology. For example, in a cluster of 3 replicas across 3 data centers, 3 replicas of each region are distributed in different data centers due to replica isolation. If the number of stores is different among these data centers, the scheduling can only reach a balanced state within each data center, but not balanced globally.

#### 9.5.6.4.2 Taking nodes offline is slow

This scenario requires examining the generation and execution of operators through related metrics.

If operators are successfully generated but the scheduling process is slow, possible reasons are:

- The scheduling speed is limited by default. You can adjust `leader-schedule-limit` or `replica-schedule-limit` to larger values. Similarly, you can consider loosening the limits on `max-pending-peer-count` and `max-snapshot-count`.
- Other scheduling tasks are running concurrently and racing for resources in the system. You can refer to the solution in [Leaders/regions are not evenly distributed](#).
- When you take a single node offline, a number of region leaders to be processed (around 1/3 under the configuration of 3 replicas) are distributed on the node to remove. Therefore, the speed is limited by the speed at which snapshots are generated by this single node. You can speed it up by manually adding an `evict-leader-scheduler` to migrate leaders.

If the corresponding operator fails to generate, possible reasons are:

- The operator is stopped, or `replica-schedule-limit` is set to “0”.
- There is no proper node for region migration. For example, if the available capacity size of the replacing node (of the same label) is less than 20%, PD will stop scheduling to avoid running out of storage on that node. In such case, you need to add more nodes or delete some data to free the space.

#### 9.5.6.4.3 Bringing nodes online is slow

Currently, bringing nodes online is scheduled through the balance region mechanism. You can refer to [Leaders/regions are not evenly distributed](#) for troubleshooting.

#### 9.5.6.4.4 Hot regions are not evenly distributed

Hot regions scheduling issues generally fall into the following categories:

- Hot regions can be observed via PD metrics, but the scheduling speed cannot keep up to redistribute hot regions in time.

**Solution:** adjust `hot-region-schedule-limit` to a larger value, and reduce the limit quota of other schedulers to speed up hot regions scheduling. Or you can adjust `hot-→ region-cache-hits-threshold` to a smaller value to make PD more sensitive to traffic changes.

- Hotspot formed on a single region. For example, a small table is intensively scanned by a massive amount of requests. This can also be detected from PD metrics. Because you cannot actually distribute a single hotspot, you need to manually add a `split-region` operator to split such a region.

- The load of some nodes is significantly higher than that of other nodes from TiKV-related metrics, which becomes the bottleneck of the whole system. Currently, PD counts hotspots through traffic analysis only, so it is possible that PD fails to identify hotspots in certain scenarios. For example, when there are intensive point lookup requests for some regions, it might not be obvious to detect in traffic, but still the high QPS might lead to bottlenecks in key modules.

**Solutions:** Firstly, locate the table where hot regions are formed based on the specific business. Then add a `scatter-range-scheduler` scheduler to make all regions of this table evenly distributed. TiDB also provides an interface in its HTTP API to simplify this operation. Refer to [TiDB HTTP API](#) for more details.

#### 9.5.6.4.5 Region merge is slow

Similar to slow scheduling, the speed of region merge is most likely limited by the configurations of `merge-schedule-limit` and `region-schedule-limit`, or the region merge scheduler is competing with other schedulers. Specifically, the solutions are:

- If it is known from metrics that there are a large number of empty regions in the system, you can adjust `max-merge-region-size` and `max-merge-region-keys` to smaller values to speed up the merge. This is because the merge process involves replica migration, so the smaller the region to be merged, the faster the merge is. If the merge operators are already generated rapidly, to further speed up the process, you can set `patrol-region-interval` to `10ms` (the default value of this configuration item is `10ms` in v5.3.0 and later versions of TiDB). This makes region scanning faster at the cost of more CPU consumption.
- A lot of tables have been created and then emptied (including truncated tables). These empty Regions cannot be merged if the split table attribute is enabled. You can disable this attribute by adjusting the following parameters:
  - TiKV: Set `split-region-on-table` to `false`. You cannot modify the parameter dynamically.
  - PD
    - \* Set `key-type` to `"txn"` or `"raw"`. You can modify the parameter dynamically.
    - \* Or keep `key-type` as `table` and set `enable-cross-table-merge` to `true`. You can modify the parameter dynamically.

**Note:**

After placement rules are enabled, properly switch the value of `key-type` between `txn` and `raw` to avoid the failure of decoding.

For v3.0.4 and v2.1.16 or earlier, the `approximate_keys` of regions are inaccurate in specific circumstances (most of which occur after dropping tables), which makes the number of keys break the constraints of `max-merge-region-keys`. To avoid this problem, you can adjust `max-merge-region-keys` to a larger value.

#### 9.5.6.4.6 Troubleshoot TiKV node

If a TiKV node fails, PD defaults to setting the corresponding node to the **down** state after 30 minutes (customizable by configuration item `max-store-down-time`), and rebalancing replicas for regions involved.

Practically, if a node failure is considered unrecoverable, you can immediately take it offline. This makes PD replenish replicas soon in another node and reduces the risk of data loss. In contrast, if a node is considered recoverable, but the recovery cannot be done in 30 minutes, you can temporarily adjust `max-store-down-time` to a larger value to avoid unnecessary replenishment of the replicas and resources waste after the timeout.

In TiDB v5.2.0, TiKV introduces the mechanism of slow TiKV node detection. By sampling the requests in TiKV, it calculates a score ranging from 1 to 100. A TiKV node with a score greater than or equal to 80 is marked as slow. You can add `evict-slow-→ store-scheduler` to detect and schedule slow nodes. When one and only one slow node appears, and the slow score reaches the upper limit (100 by default), all leaders in the node will be evicted.

### 9.5.7 Best Practices for TiKV Performance Tuning with Massive Regions

In TiDB, data is split into Regions, each storing data for a specific key range. These Regions are distributed among multiple TiKV instances. As data is written into a cluster, millions of or even tens of millions of Regions are created. Too many Regions on a single TiKV instance can bring a heavy burden to the cluster and affect its performance.

This document introduces the workflow of Raftstore (a core module of TiKV), explains why a massive amount of Regions affect the performance, and offers methods for tuning TiKV performance.

#### 9.5.7.1 Raftstore workflow

A TiKV instance has multiple Regions on it. The Raftstore module drives the Raft state machine to process Region messages. These messages include processing read or write requests on Regions, persisting or replicating Raft logs, and processing Raft heartbeats. However, an increasing number of Regions can affect performance of the whole cluster. To understand this, it is necessary to learn the workflow of Raftstore shown as follows:

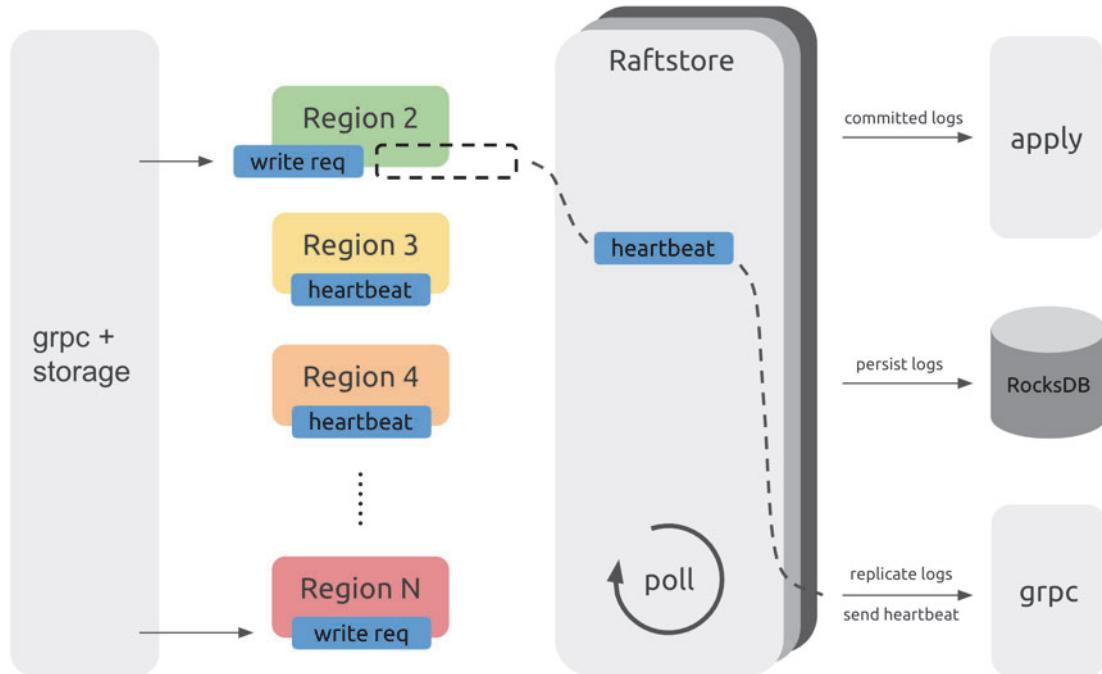


Figure 109: Raftstore Workflow

**Note:**

This diagram only illustrates the workflow of Raftstore and does not represent the actual code structure.

From the above diagram, you can see that requests from the TiDB servers, after passing through the gRPC and storage modules, become read and write messages of KV (key-value), and are sent to the corresponding Regions. These messages are not immediately processed but are temporarily stored. Raftstore polls to check whether each Region has messages to process. If a Region has messages to process, Raftstore drives the Raft state machine of this Region to process these messages and perform subsequent operations according to the state changes of these messages. For example, when write requests come in, the Raft state machine stores logs into disk and sends logs to other Region replicas; when the heartbeat interval is reached, the Raft state machine sends heartbeat information to other Region replicas.

#### 9.5.7.2 Performance problem

From the Raftstore workflow diagram, messages in each Region are processed one by one. When a large number of Regions exist, it takes Raftstore some time to process the heartbeats of these Regions, which can cause some delay. As a result, some read and write requests are not processed in time. If read and write pressure is high, the CPU usage of the

Raftstore thread might easily become the bottleneck, which further increases the delay and affects the performance.

Generally, if the CPU usage of the loaded Raftstore reaches 85% or higher, Raftstore goes into a busy state and becomes the bottleneck. At the same time, `propose wait duration` can be as high as hundreds of milliseconds.

#### Note:

- For the CPU usage of Raftstore as mentioned above, Raftstore is single-threaded. If Raftstore is multi-threaded, you can increase the CPU usage threshold (85%) proportionally.
- Because I/O operations exist in the Raftstore thread, CPU usage cannot reach 100%.

##### 9.5.7.2.1 Performance monitoring

You can check the following monitoring metrics in Grafana's **TiKV Dashboard**:

- Raft store CPU in the **Thread-CPU** panel

Reference value: lower than `raftstore.store-pool-size * 85%`.

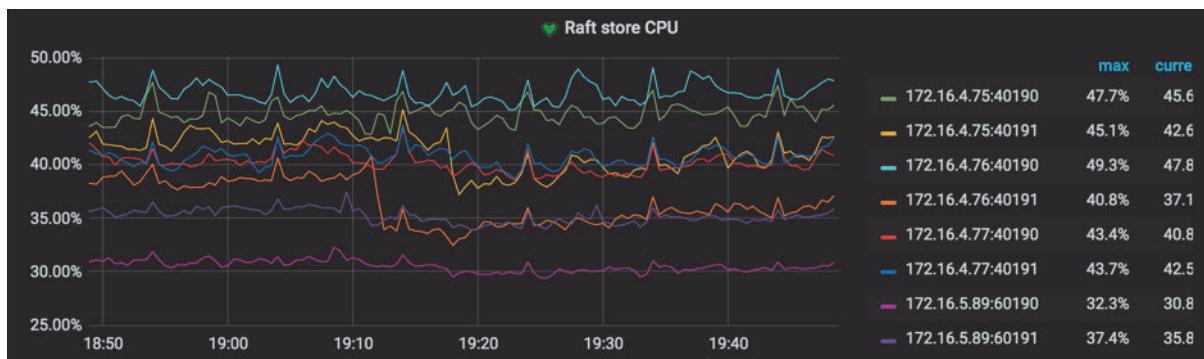


Figure 110: Check Raftstore CPU

- Propose wait duration in the **Raft Propose** panel

`Propose wait duration` is the delay between the time a request is sent to Raftstore and the time Raftstore actually starts processing the request. Long delay means that Raftstore is busy, or that processing the append log is time-consuming, making Raftstore unable to process the request in time.

Reference value: lower than 50~100 ms according to the cluster size

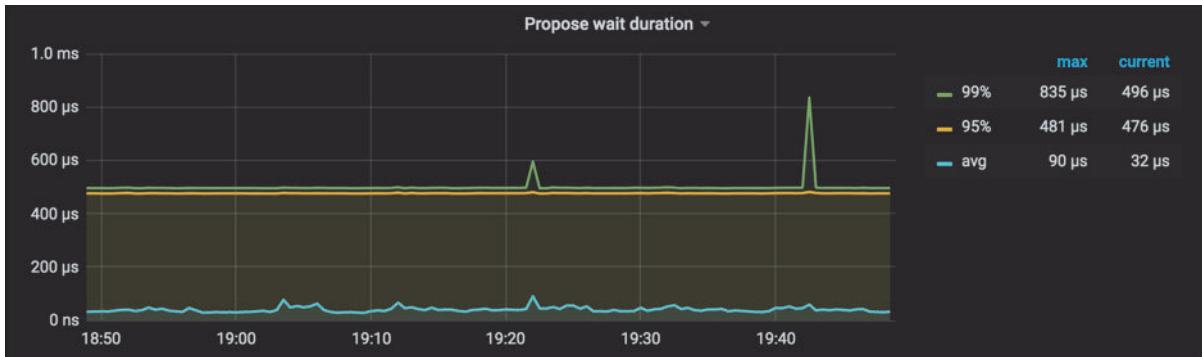


Figure 111: Check Propose wait duration

#### 9.5.7.3 Performance tuning methods

After finding out the cause of a performance problem, try to solve it from the following two aspects:

- Reduce the number of Regions on a single TiKV instance
- Reduce the number of messages for a single Region

##### 9.5.7.3.1 Method 1: Increase Raftstore concurrency

Raftstore has been upgraded to a multi-threaded module since TiDB v3.0, which greatly reduces the possibility that a Raftstore thread becomes the bottleneck.

By default, `raftstore.store-pool-size` is configured to 2 in TiKV. If a bottleneck occurs in Raftstore, you can properly increase the value of this configuration item according to the actual situation. But to avoid introducing unnecessary thread switching overhead, it is recommended that you do not set this value too high.

##### 9.5.7.3.2 Method 2: Enable Hibernate Region

In the actual situation, read and write requests are not evenly distributed on every Region. Instead, they are concentrated on a few Regions. Then you can minimize the number of messages between the Raft leader and the followers for the temporarily idle Regions, which is the feature of Hibernate Region. In this feature, Raftstore does not send tick messages to the Raft state machines of idle Regions if not necessary. Then these Raft state machines will not be triggered to generate heartbeat messages, which can greatly reduce the workload of Raftstore.

Hibernate Region is enabled by default in [TiKV master](#). You can configure this feature according to your needs. For details, refer to [Configure Hibernate Region](#).

##### 9.5.7.3.3 Method 3: Enable Region Merge

**Note:**

Region Merge is enabled by default since TiDB v3.0.

You can also reduce the number of Regions by enabling Region Merge. Contrary to Region Split, Region Merge is the process of merging adjacent small Regions through scheduling. After dropping data or executing the Drop Table or Truncate Table statement, you can merge small Regions or even empty Regions to reduce resource consumption.

Enable Region Merge by configuring the following parameters:

```
>> pd-ctl config set max-merge-region-size 20
>> pd-ctl config set max-merge-region-keys 200000
>> pd-ctl config set merge-schedule-limit 8
```

Refer to [Region Merge](#) and the following three configuration parameters in the [PD configuration file](#) for more details:

- [max-merge-region-size](#)
- [max-merge-region-keys](#)
- [merge-schedule-limit](#)

The default configuration of the Region Merge parameters is rather conservative. You can speed up the Region Merge process by referring to the method provided in [PD Scheduling Best Practices](#).

#### 9.5.7.3.4 Method 4: Increase the number of TiKV instances

If I/O resources and CPU resources are sufficient, you can deploy multiple TiKV instances on a single machine to reduce the number of Regions on a single TiKV instance; or you can increase the number of machines in the TiKV cluster.

#### 9.5.7.3.5 Method 5: Adjust `raft-base-tick-interval`

In addition to reducing the number of Regions, you can also reduce pressure on Raftstore by reducing the number of messages for each Region within a unit of time. For example, you can properly increase the value of the `raft-base-tick-interval` configuration item:

```
[raftstore]
raft-base-tick-interval = "2s"
```

In the above configuration, `raft-base-tick-interval` is the time interval at which Raftstore drives the Raft state machine of each Region, which means at this time interval, Raftstore sends a tick message to the Raft state machine. Increasing this interval can effectively reduce the number of messages from Raftstore.

Note that this interval between tick messages also determines the intervals between `election timeout` and `heartbeat`. See the following example:

```
raft-election-timeout = raft-base-tick-interval * raft-election-timeout-
 ↪ ticks
raft-heartbeat-interval = raft-base-tick-interval * raft-heartbeat-ticks
```

If Region followers have not received the heartbeat from the leader within the `raft → -election-timeout` interval, these followers determine that the leader has failed and start a new election. `raft-heartbeat-interval` is the interval at which a leader sends a heartbeat to followers. Therefore, increasing the value of `raft-base-tick-interval` can reduce the number of network messages sent from Raft state machines but also makes it longer for Raft state machines to detect the leader failure.

#### 9.5.7.4 Other problems and solutions

This section describes some other problems and solutions.

##### 9.5.7.4.1 Switching PD Leader is slow

PD needs to persist Region Meta information on etcd to ensure that PD can quickly resume to provide Region routing services after switching the PD Leader node. As the number of Regions increases, the performance problem of etcd appears, making it slower for PD to get Region Meta information from etcd when PD is switching the Leader. With millions of Regions, it might take more than ten seconds or even tens of seconds to get the meta information from etcd.

To address this problem, `use-region-storage` is enabled by default in PD since TiDB v3.0. With this feature enabled, PD stores Region Meta information on local LevelDB and synchronizes the information among PD nodes through other mechanisms.

##### 9.5.7.4.2 PD routing information is not updated in time

In TiKV, pd-worker regularly reports Region Meta information to PD. When TiKV is restarted or switches the Region leader, PD needs to recalculate Region's `approximate → size / keys` through statistics. Therefore, with a large number of Regions, the single-threaded pd-worker might become the bottleneck, causing tasks to be piled up and not processed in time. In this situation, PD cannot obtain certain Region Meta information in time so that the routing information is not updated in time. This problem does not affect the actual reads and writes, but might cause inaccurate PD scheduling and require several round trips when TiDB updates Region cache.

You can check **Worker pending tasks** under **Task** in the **TiKV Grafana** panel to determine whether pd-worker has tasks piled up. Generally, pending tasks should be kept at a relatively low value.

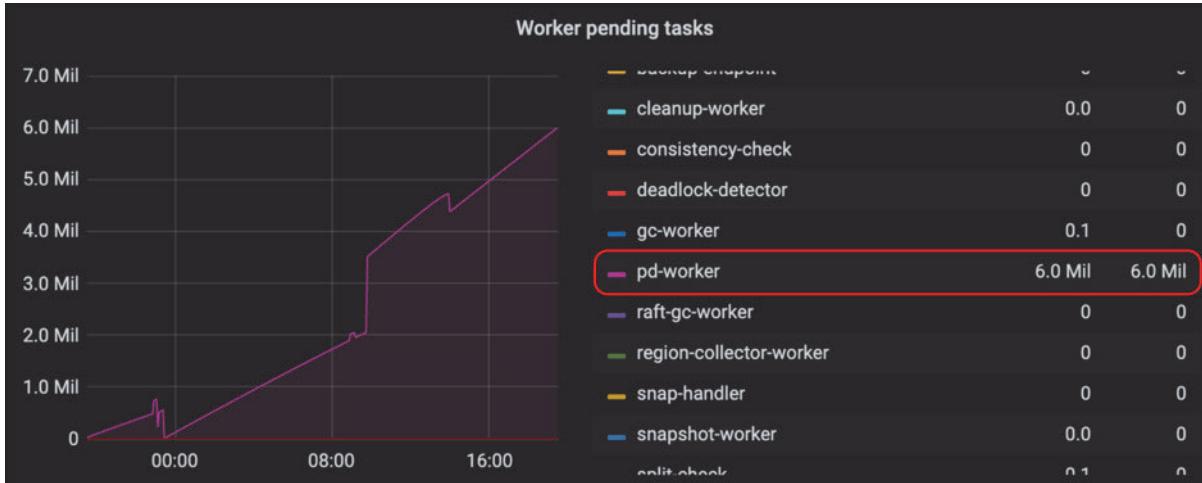


Figure 112: Check pd-worker

pd-worker has been optimized for better performance since [v3.0.5](#). If you encounter a similar problem, it is recommended to upgrade to the latest version.

#### 9.5.7.4.3 Prometheus is slow to query metrics

In a large-scale cluster, as the number of TiKV instances increases, Prometheus has greater pressure to query metrics, making it slower for Grafana to display these metrics. To ease this problem, metrics pre-calculation is configured since v3.0.

### 9.5.8 Best Practices for Three-Node Hybrid Deployment

For a TiDB cluster, if you have no requirements on high performance but need to control the cost, you can deploy the TiDB, TiKV, and PD components on three machines in a hybrid way.

This document offers an example of three-node hybrid deployment and a TPC-C test against the deployed cluster. Based on this example, this document offers best practices for the deployment scenario and its parameter adjustment.

#### 9.5.8.1 Prerequisites for deployment and the test method

In this example, three physical machines are used for deployment, each with 16 CPU cores and 32 GB of memory. On each machine (node), one TiDB instance, one TiKV instance, and one PD instance are deployed in a hybrid way.

Because PD and TiKV both store information on the disk, the read and write latency of disk directly affects the latency of the PD and TiKV services. To avoid the situation that

PD and TiKV compete for disk resources and affect each other, it is recommended to use different disks for PD and TiKV.

In this example, the TPC-C 5000 Warehouse data is used in TiUP bench and the test lasts 12 hours with the `terminals` parameter set to 128 (concurrency). Close attention is paid to metrics related to performance stability of the cluster.

The image below shows the QPS monitor of the cluster within 12 hours with the default parameter configuration. From the image, you can see an obvious performance jitter.



Figure 113: QPS with default config

After the parameter adjustment, the performance is improved.



Figure 114: QPS with modified config

### 9.5.8.2 Parameter adjustment

Performance jitter occurs in the image above, because the default thread pool configuration and the resource allocation to background tasks are for machines with sufficient resources. In the hybrid deployment scenario, the resources are shared among multiple components, so you need to limit the resource consumption via configuration parameters.

The final cluster configuration for this test is as follows:

```
tikv:
 readpool.unified.max-thread-count: 6
 server.grpc-concurrency: 2
 storage.scheduler-worker-pool-size: 2
 gc.max-write-bytes-per-sec: 300K
 rocksdb.max-background-jobs: 3
```

```

rocksdb.max-sub-compactions: 1
rocksdb.rate-bytes-per-sec: "200M"

tidb:
 performance.committer-concurrency: 4
 performance.max-procs: 8

```

The following sections introduce the meanings and the adjustment methods of these parameters.

#### 9.5.8.2.1 Configuration of TiKV thread pool size

This section offers best practices for adjusting parameters that relate to the resource allocation of thread pools for foreground applications. Reducing these thread pool sizes will compromise performance, but in the hybrid deployment scenario with limited resources, the cluster itself is hard to achieve high performance. In this scenario, the overall stability of the cluster is preferred over performance.

If you conduct an actual load test, you can first use the default configuration and observe the actual resource usage of each thread pool. Then you can adjust the corresponding configuration items and reduce the sizes of the thread pools that have lower usage.

`readpool.unified.max-thread-count`

The default value of this parameter is 80% of the number of machine threads. In a hybrid deployment scenario, you need to manually calculate and specify this value. You can first set it to 80% of the expected number of CPU threads used by TiKV.

`server.grpc-concurrency`

This parameter defaults to 4. Because in the existing deployment plan, the CPU resources are limited and the actual requests are few. You can observe the monitoring panel, lower the value of this parameter, and keep the usage rate below 80%.

In this test, the value of this parameter is set to 2. Observe the **gRPC poll CPU** panel and you can see that the usage rate is just around 80%.

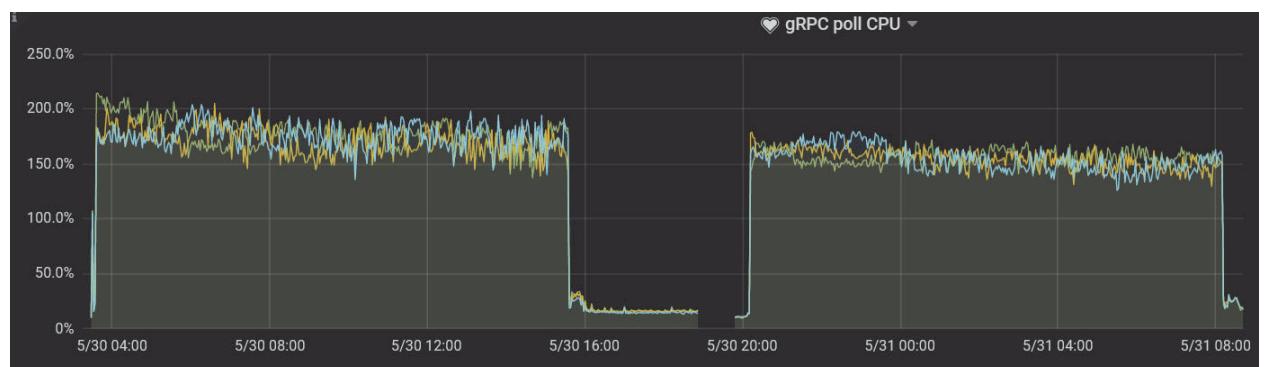


Figure 115: gRPC Pool CPU

### `storage.scheduler-worker-pool-size`

When TiKV detects that the CPU core number of the machine is greater than or equal to 16, this parameter value defaults to 8. When the CPU core number is smaller than 16, the parameter value defaults to 4. This parameter is used when TiKV converts complex transaction requests to simple key-value reads or writes, but the scheduler thread pool does not perform any writes.

Ideally, the usage rate of the scheduler thread pool is kept between 50% and 75%. Similar to the gRPC thread pool, the `storage.scheduler-worker-pool-size` parameter defaults to a larger value during the hybrid deployment, which makes resource usage insufficient. In this test, the value of this parameter is set to 2, which is in line with the best practices, a conclusion drawn by observing the corresponding metrics in the **Scheduler worker CPU** panel.

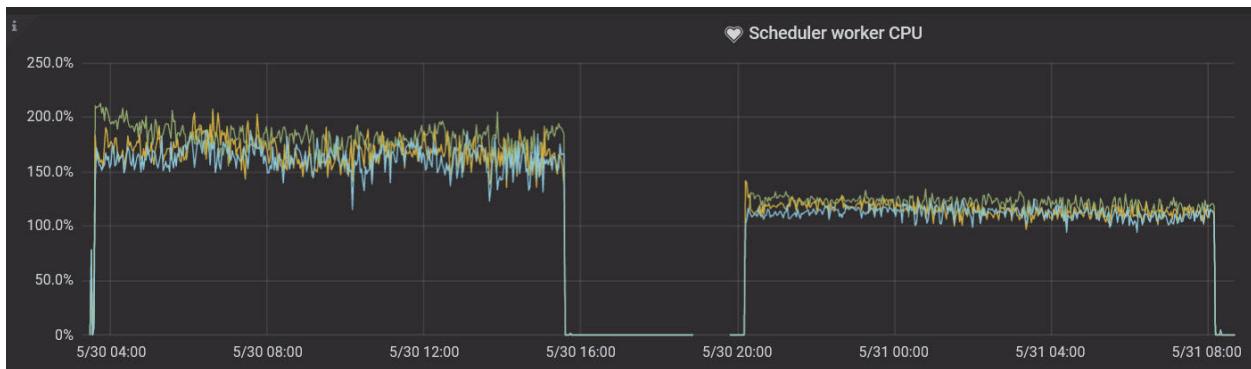


Figure 116: Scheduler Worker CPU

#### 9.5.8.2.2 Resource configuration for TiKV background tasks

In addition to foreground tasks, TiKV regularly sorts data and cleans outdated data in background tasks. The default configuration allocates sufficient resources to these tasks for the scenario of high-traffic writes.

However, in the hybrid deployment scenario, this default configuration is not in line with the best practices. You need to limit the resource usage of background tasks by adjusting the following parameters.

`rocksdb.max-background-jobs` and `rocksdb.max-sub-compactions`

The RocksDB thread pool is used to perform compaction and flush jobs. The default value of `rocksdb.max-background-jobs` is 8, which obviously exceeds the resource that is in need. Therefore, the value should be adjusted to limit the resource usage.

`rocksdb.max-sub-compactions` indicates the number of concurrent sub-tasks allowed for a single compaction job, which defaults to 3. You can lower this value when the write traffic is not high.

In the test, the `rocksdb.max-background-jobs` value is set to 3 and the `rocksdb.max-sub-compactions` value is set to 1. No write stall occurs during the 12-hour test with the TPC-C load. When optimizing the two parameter values according to the actual load, you can lower the values gradually based on monitoring metrics:

- If write stall occurs, increase the value of `rocksdb.max-background-jobs`.
- If the write stall persists, set the value of `rocksdb.max-sub-compactions` to 2 or 3.

#### `rocksdb.rate-bytes-per-sec`

This parameter is used to limit the disk traffic for the background compaction jobs. The default configuration has no limit for this parameter. To avoid the situation that compaction jobs occupy the resources of foreground services, you can adjust this parameter value according to the sequential read and write speed of the disk, which reserves enough disk bandwidth for foreground services.

The method of optimizing the RocksDB thread pool is similar to that of optimizing the compaction thread pool. You can determine whether the value you have adjusted is suitable according to whether write stall occurs.

#### `gc.max_write_bytes_per_sec`

Because TiDB uses the multi-version concurrency control (MVCC) model, TiKV periodically cleans old version data in the background. When the available resources are limited, this operation causes periodical performance jitter. You can use the `gc.max_write_bytes_per_sec` parameter to limit the resource usage of such an operation.



Figure 117: GC Impact

In addition to setting this parameter value in the configuration file, you can also dynamically adjust this value in tikv-ctl.

```
tiup ctl tikv --host=${ip:port} modify-tikv-config -n gc.
 ↪ max_write_bytes_per_sec -v ${limit}
```

#### Note:

In application scenarios with frequent updates, limiting GC traffic might cause the MVCC versions to pile up and affect read performance. Currently, to achieve a balance between performance jitter and performance decrease, you might need to try multiple times to adjust the value of this parameter.

#### 9.5.8.2.3 TiDB parameter adjustment

Generally, you can adjust the TiDB parameters of execution operators using system variables such as `tidb_hash_join_concurrency` and `tidb_index_lookup_join_concurrency`.

In this test, these parameters are not adjusted. In the load test of your actual application, if the execution operators consume an excessive amount of CPU resources, you can limit the resource usage of specific operators according to your application scenario. For more details, see [TiDB system variables](#).

##### `performance.max-procs`

This parameter is used to control how many CPU cores an entire Go process can use. By default, the value is equal to the number of CPU cores of the current machine or cgroups.

When Go is running, a proportion of threads is used for background tasks such as GC. If you do not limit the value of the `performance.max-procs` parameter, these background tasks will consume too much CPU.

#### 9.5.9 Local Read under Three Data Centers Deployment

In the model of three data centers, a Region has three replicas which are isolated in each data center. However, due to the requirement of strongly consistent read, TiDB must access the Leader replica of the corresponding data for every query. If the query is generated in a data center different from that of the Leader replica, TiDB needs to read data from another data center, thus causing the access latency to increase.

This document describes how to use the [Stale Read](#) feature to avoid cross-center access and reduce the access latency at the expense of real-time data availability.