



# 第十四届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA

## 数智赋能 共筑未来



北京国际会议中心 | 2023/8/17-19



# 基于TiDB的分布式 UbiSQL 在平安集团的运维实践

平安科技+DBA+熊浪

## UbiSQL

UbiSQL是平安基于开源项目TiDB进行自主研发的新一代原生分布式数据库产品。

UbiSQL具有源代码自主可控，兼容MySQL语法，采用原生分布式计算、存储分离架构，支持分布式事务、多副本强一致等特性。

UbiSQL具备跨数据中心级高可靠，实现多数据中心实时写入。



UbiStore存储引擎  
优化数据编码  
减少40%存储空间



UbiSQL 易用性  
提供统计信息管理、资源隔离等功能易用性



UbiSQL 安全加密  
提供数据加密功能对业务透明，降低应用改造成本



UbiSQL 云服务  
自动故障转移，全面提升用户体验

UbiSQL来源Ubiquitous 的缩写，寓意无处不在

# 目录

**/>:** Batch Client机制引发 TiKV gRPC 饥饿

---

**/>:** region失联，偶发SQL执行超过40s

---

**/>:** 100T Oracle迁移UbiSQL实践

---

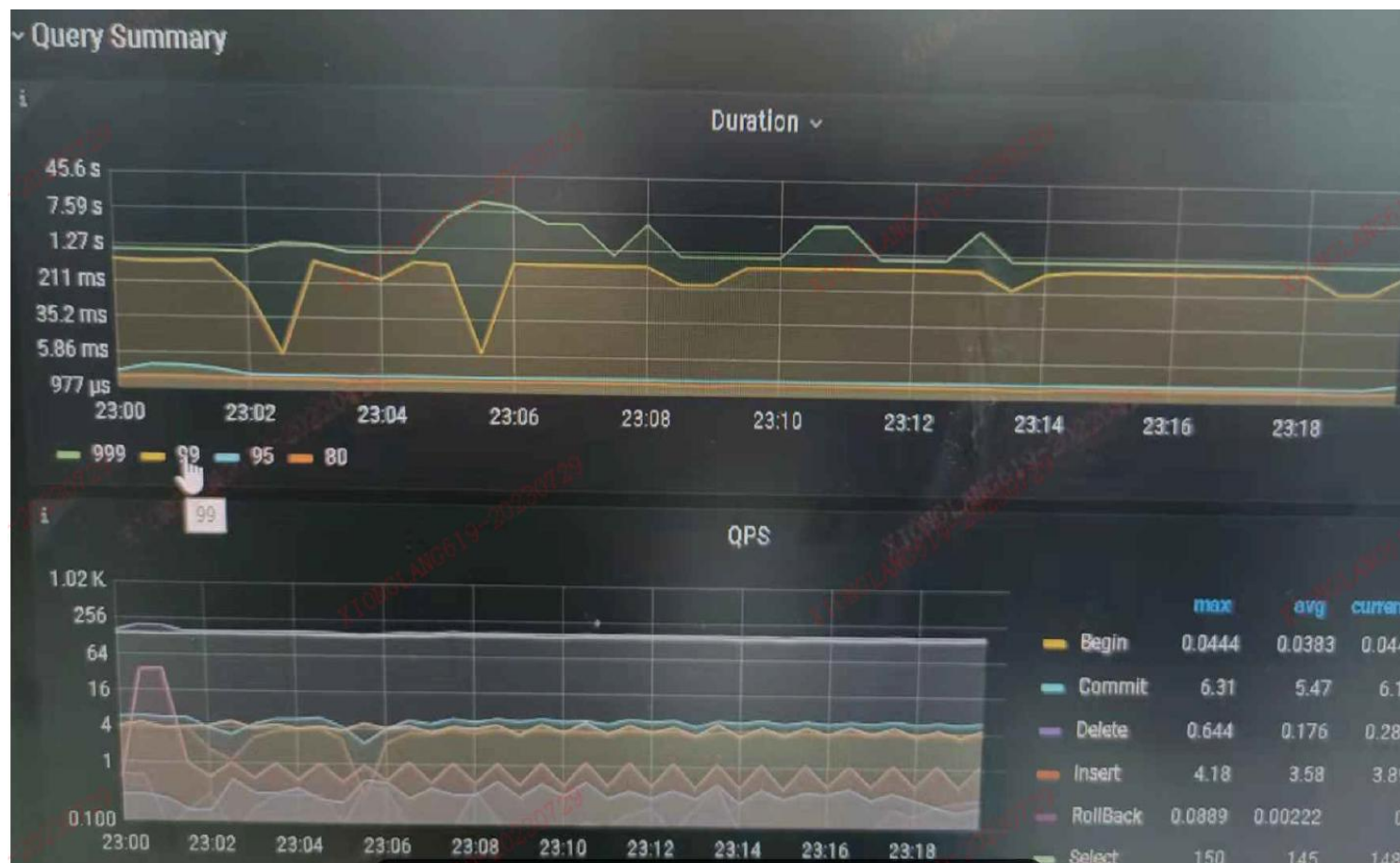
**/>:** 使用UbiSQL的心得

---



# Batch Client机制引发 TiKV gRPC 饥饿

现象：某平台灰度上线后，集群 Duration P999 偶尔出现 8s 以上尖刺，业务接口出现少量超时现象。



初步分析：

- Duration P999 尖刺出现时间无规律，白天晚上都有出现，当时集群负载及主机资源使用都较低；
- 每次出现尖刺之前，都会先有 Duration P99 降低的现象；
- 慢请求对应不同的 TiKV 实例；
- 问题时段集群 QPS 较低，主机资源使用也在正常水

# Batch Client机制引发 TiKV gRPC 饥饿

SQL	结束运行时间	总执行时间	Coprocessor 请求数	最长处理时间实例	最长等待时间
ANALYZE TABLE ... PARTITION ...	今天 05:44	1.2 hour	0		
ANALYZE TABLE ... PARTITION ...	今天 06:46	1.0 hour	0		
SELECT /*+ use_index(...) */ ...	今天 09:26	2.5 min	6.9K	30.88.130.254300	30.88.130.254300
SET SQL_SELECT_LIMIT = DEFAULT;	今天 05:16	20.1 s	1	30.88.130.274300	30.88.130.274300
SELECT VARIABLE_VALUE FROM `mysql`.`GLOBAL_VARIABLES` WHERE ...	今天 05:16	13.6 s	0		
SELECT /*+ use_index(pscp_sms_fetch, inn_pscp_sms_status...) */ ...	今天 05:16	12.9 s	1	30.88.130.274300	30.88.130.274300
SELECT VARIABLE_VALUE FROM `mysql`.`GLOBAL_VARIABLES` WHERE ...	今天 05:16	12.5 s	0		
SELECT version, table_id, modify_count, count FROM mysql...	今天 05:16	12.2 s	2	30.88.130.274300	30.88.130.274300
SELECT VARIABLE_VALUE FROM `mysql`.`GLOBAL_VARIABLES` WHERE ...	今天 05:16	11.9 s	0		
SELECT VARIABLE_VALUE FROM `mysql`.`GLOBAL_VARIABLES` WHERE ...	今天 05:16	10.7 s	0		
SELECT VARIABLE_VALUE FROM `mysql`.`GLOBAL_VARIABLES` WHERE ...	今天 05:16	9.8 s	0		
SELECT VARIABLE_VALUE FROM `mysql`.`GLOBAL_VARIABLES` WHERE ...	今天 05:16	9.3 s	0		
SET SQL_SELECT_LIMIT = DEFAULT;	今天 05:16	8.6 s	0		
SELECT VARIABLE_VALUE FROM `mysql`.`GLOBAL_VARIABLES` WHERE ...	今天 05:16	8.6 s	0		
SELECT ...	今天 05:16	8.2 s	2	30.88.130.264300	30.88.130.264300

名称	时间	描述
SQL 执行时间	11.9 s	执行 SQL 消耗的总时间
解析耗时	0 ns	解析该 SQL 查询的耗时
生成执行计划耗时	78.0 μs	生成该 SQL 的执行计划的耗时
重写执行计划耗时	83.2 μs	重写执行计划的耗时，例如常量折叠等
子查询预处理耗时	0 ns	
优化器寻找执行计划耗时	132.2 μs	优化器寻找执行计划的耗时，包括视图优化和物理优化的耗时

id	task	estRows	operator info	actRows	execution info
Projection_9	root	0.00	tsmscp.pscp_sms_0		time: 39.6s, loops: 1, Concurrency: OFF
IndexLookUp_26	root	0.00	limit embedded	0	time: 39.5s, loops: 1, table task: {total time: 2m37.8s, min: 0, concurrency: 41}
Limit_25	cop[tikv]	0.00	offset: 0, count: 0	0	time: 39.5s, loops: 1, cop task: {total time: 2m37.8s, min: 0, concurrency: 41}
IndexRangeScan_23	cop[tikv]	0.00	table: pscp_s	0	time: 39.5s, loops: 1, rpc time: 39.6s
TableRowIDScan_24	cop[tikv]	0.00	table: pscp_sms	0	tikv task {time: 0s, loops: 1}

SQL分析:

- 1、只有 1 个 COP 请求但是耗时 20s（最高 50s+），平常执行时间在毫秒级别，执行计划未发生变化，时间分布集中在 rpc\_time，即 TiDB gRPC 发起 request 到收到 response 的时间；
- 2、internal SQL：查询 GLOBAL\_VARIABLES，执行计划为 PointGet，时间消耗 10s+；

看起来和 SQL 本身无关，同时检查 node\_exporter 和 blackbox\_exporter 也未发现异常。



# Batch Client机制引发 TiKV gRPC 饥饿



线程池资源使用都不高，从 Handle duration 可以看出数据库当时有 Analyze 动作，查看 TiDB Server (stats owner) 日志，发现每次 COP 慢请求都是在某张表 analyze 之后出现。

# Batch Client机制引发 TiKV gRPC 饥饿

1, insert 语句 (来自 ETL, 每条 SQL 2000行) 报错

```
[2022/04/17 21:23:46.821 +08:00] [INFO] [conn.go:870] ["command dispatched failed"]
```

```
... [err="[executor:1390]Prepared statement contains too many placeholders"]
```

2, 1分多钟后开始收集统计信息

```
[2022/04/17 21:25:14.072 +08:00] [INFO] [update.go:784] ["[stats] auto analyze triggered"] [sql="analyze table `XXXXXXXXXXXXXXXXXXXX` partition `p20211025`"] [reason="table unanalyzed, time since last updated 1m2.272035284s"]
```

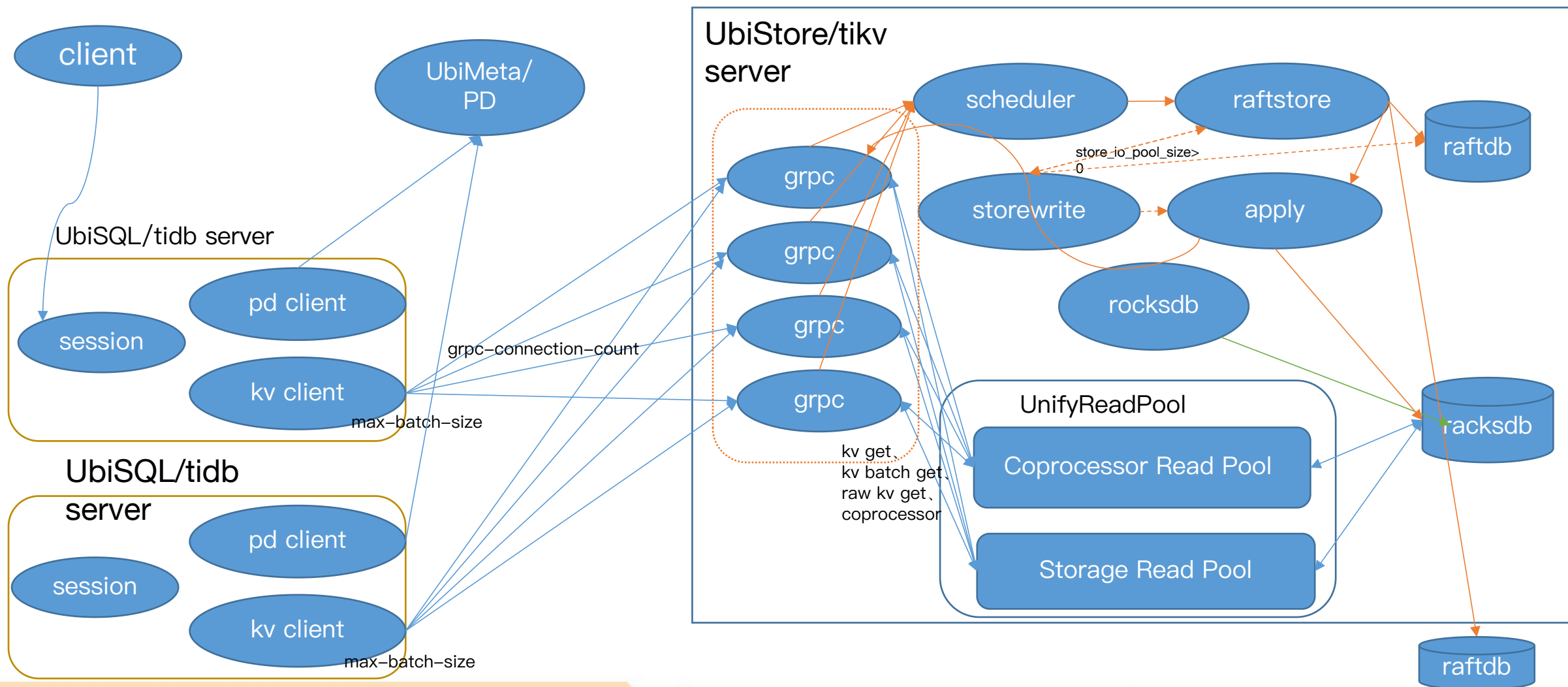
3, COP 请求升高

```
[2022/04/17 21:25:33.401 +08:00] [INFO] [coprocessor.go:1034] ["[TIME_COP_PROCESS] resp_time:7.54184014s  
txnStartTS:18446744073709551615 region_id:1908738 store_addr:30.88.130.27:6504 kv_process_ms:7527  
scan_total_write:142544 scan_proce
```

问题引发原因基本确认了, database.xxxxx 每天一个分区, 每个分区 5000万数据, 通过 ETL 从 Oracle 库同步历史数据, 每同步完一个分区, 大概 1min 后开始收集统计信息, Analyze 导致 tidb duration 升高, KV Get 和 COP 请求都受到影响, 包括业务 SQL 和 internal SQL 都会偶尔出现很慢的情况, 这也解释了每次出问题前都有 Duration P99 下降的现象, 原因是某个分区完成同步了, 之前的 P99 主要为批量 insert 语句。



## Batch Client机制引发 TiKV gRPC 饥饿



# Batch Client机制引发 TiKV gRPC 饥饿

## gRPC 线程饥饿原理

在启用 batch client 的情况下，每一条 TiDB 到 TiKV 的 TCP 连接中会建立一个 gRPC 双向流，每个 TiDB 到 TiKV 之间会建立多个双向流。而 TiKV 的 gRPC 线程在处理这些双向流时采取的是贪婪模式，即除非一个流中暂时没有更多数据了，否则会一直处理这个流，不会切换到其他流上。这样，一些流就会出现饥饿的现象。

## 解决办法

将 max-batch-size 设置为 0 即关闭 Batch Client，gRPC 采用简单模式进行数据交互，绕开了 batch client 对 gRPC 请求的这一套处理逻辑：

- 1, gRPC 连接在多个 goroutine 之间是共享的，所以在网络层中，不同 goroutine 之间的 SQL 并不会互相影响，可以同时发送，所以业务请求的发送不会受到 analyze 影响；
- 2, 不同的请求不会再进行封包，系统整体压力不大，要求更低延迟的场景改成 0 更合适；

## 结论

新分区发起 analyze 会有大量的 COP 请求，执行时间长，开启 batch client 后 tikv 可能一直在处理 analyze 请求相关的双向流，导致其他的双向流出现饥饿，使得业务 SQL 或者其它 internal SQL 出现异常慢查询现象。

## region失联，偶发SQL执行超过40s

现象：对核心集群巡检过程中，发现偶尔出现 40s 慢 SQL

名称	时间	描述
SQL 执行时间	40.5 s	执行 SQL 耗费的自然时间
解析耗时	226.9 $\mu$ s	解析该 SQL 查询的耗时
生成执行计划耗时	2.5 ms	生成该 SQL 的执行计划的耗时
重写执行计划耗时	713.9 $\mu$ s	重写执行计划的耗时，例如常
子查询预处理耗时	0 ns	
优化执行计划耗时	1.5 ms	优化器寻找执行计划的耗时，包
Coprocessor 执行耗时	59.6 ms	UbiSQL Coprocessor 算子等待
Coprocessor 累计等待耗时	145.0 ms	TiKV 准备并等待 Coprocessor
Coprocessor 累计执行耗时	88.0 ms	TiKV 执行 Coprocessor 任务的



## region失联，偶发SQL执行超过40s

分析：Dashboard中看到并没有消耗时间高的节点，故从tidb日志中查看，在Cop 请求访问 Region 31772694 时发生大量 RegionMiss backoff，直到 40s 超时后 SQL 执行报错。

```
[2022/09/27 17:21:02.594 +08:00] [INFO] [coprocessor.go:1034] ["[TIME_COP_PROCESS] resp_time:319.389497ms txnStartTS:436278506397106236 region_id:317726"]  
[2022/09/27 17:21:06.119 +08:00] [WARN] [backoff.go:329] ["regionMiss backoffer.maxSleep 40000ms is exceeded, errors:\nregion_id:31772694, region_ver:42"]  
[2022/09/27 17:21:06.119 +08:00] [ERROR] [distsql.go:1009] ["table reader fetch next chunk failed"] [conn=2535109] [error="[tikv:9005]Region is unavaila"]  
[2022/09/27 17:21:06.120 +08:00] [INFO] [conn.go:870] ["command dispatched failed"] [conn=2535109] [connInfo="id:2535109, addr:10.33.63.207:24427 status"]
```

分析：Dashboard中看到并没有消耗时间高的节点，故从tidb日志中查看，在Cop 请求访问 Region 31772694 时发生大量 RegionMiss backoff，直到 40s 超时后 SQL 执行报错。



## region失联，偶发SQL执行超过40s



16:53:49.609, PD 发起 merge-region 调度, source: 31772694, target: 29952726

```
[2022/09/27 16:53:49.609 +08:00] [INFO] [operator_controller.go:424] ["add operator"] [region-id=31772694] [operator="\merge-region (merge: region 31772694 into region 29952726)"]
[2022/09/27 16:53:49.609 +08:00] [INFO] [operator_controller.go:620] ["send schedule command"] [region-id=31772694] [step="merge region 31772694 into region 29952726"]
[2022/09/27 16:53:49.609 +08:00] [INFO] [operator_controller.go:424] ["add operator"] [region-id=29952726] [operator="\merge-region (merge: region 31772694 into region 29952726)"]
[2022/09/27 16:53:49.609 +08:00] [INFO] [operator_controller.go:620] ["send schedule command"] [region-id=29952726] [step="merge region 31772694 into region 29952726"]
[2022/09/27 16:53:54.761 +08:00] [INFO] [operator_controller.go:620] ["send schedule command"] [region-id=31772694] [step="merge region 31772694 into region 29952726"]
[2022/09/27 16:53:54.761 +08:00] [INFO] [operator_controller.go:620] ["send schedule command"] [region-id=29952726] [step="merge region 31772694 into region 29952726"]
[2022/09/27 16:53:59.762 +08:00] [INFO] [operator_controller.go:560] ["operator timeout"] [region-id=31772694] [takes=10.152879012s] [operator="\merge-region (merge: region 31772694 into region 29952726)"]
[2022/09/27 16:53:59.762 +08:00] [INFO] [operator_controller.go:560] ["operator timeout"] [region-id=29952726] [takes=10.152813837s] [operator="\merge-region (merge: region 31772694 into region 29952726)"]
```



16:53:49.612, TiKV 完成 merge 操作后, source region 31772694 被 destroy, 并由合并后的region 29952726 上报心跳信息到 PD (region version 递增至 42223)

```
[2022/09/27 16:53:49.610 +08:00] [INFO] [pd.rs:835] ["try to merge"] [merge="target { id: 29952726 start_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA end_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA region_epoch { conf_ver: 563 version: 42220 } peers { id: 29952727 store_id: 17 } peers { id: 29952728 store_id: 6157285 } peers { id: 31685638 store_id: 30965069 }"]
[2022/09/27 16:53:49.610 +08:00] [INFO] [apply.rs:1234] ["execute admin command"] [command="cmd_type: PrepareMerge prepare_merge { min_index: 8688 target_region_id: 29952726 }"]
[2022/09/27 16:53:49.611 +08:00] [INFO] [apply.rs:2086] ["asking delegate to stop"] [source_region_id=31772694] [peer_id=29952728] [region_id=29952726]
[2022/09/27 16:53:49.611 +08:00] [INFO] [apply.rs:2933] ["source logs are all applied now"] [peer_id=31772696] [region_id=31772694]
[2022/09/27 16:53:49.611 +08:00] [INFO] [apply.rs:2852] ["remove delegate from apply delegates"] [peer_id=31772696] [region_id=31772694]
[2022/09/27 16:53:49.611 +08:00] [INFO] [router.rs:265] ["[region 31772694] shutdown mailbox"]
[2022/09/27 16:53:49.611 +08:00] [INFO] [apply.rs:2108] ["execute CommitMerge"] [source_region="id: 31772694 start_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA end_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA region_epoch { conf_ver: 563 version: 42220 } peers { id: 29952727 store_id: 17 } peers { id: 29952728 store_id: 6157285 } peers { id: 31685638 store_id: 30965069 }"]
[2022/09/27 16:53:49.612 +08:00] [INFO] [peer.rs:2612] ["notify pd with merge"] [target_region="id: 29952726 start_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA end_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA region_epoch { conf_ver: 564 version: 42220 } peers { id: 29952727 store_id: 17 } peers { id: 29952728 store_id: 6157285 } peers { id: 31685638 store_id: 30965069 }"]
[2022/09/27 16:53:49.612 +08:00] [INFO] [peer.rs:2723] ["merge finished"] [target_region="Some(id: 29952726 start_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA end_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA region_epoch { conf_ver: 564 version: 42220 } peers { id: 29952727 store_id: 17 } peers { id: 29952728 store_id: 6157285 } peers { id: 31685638 store_id: 30965069 }"]
[2022/09/27 16:53:49.612 +08:00] [INFO] [peer.rs:1777] ["starts destroy"] [merged_by_target=true] [peer_id=31772696] [region_id=31772694]
[2022/09/27 16:53:49.612 +08:00] [INFO] [peer.rs:2612] ["notify pd with merge"] [target_region="id: 29952726 start_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA end_key: 7480000000000071FF4E5F728000000068FF2638E70000000000000000FA region_epoch { conf_ver: 564 version: 42220 } peers { id: 29952727 store_id: 17 } peers { id: 29952728 store_id: 6157285 } peers { id: 31685638 store_id: 30965069 }"]
```



## region失联，偶发SQL执行超过40s

16:53:59.762, PD 在发起 merge 调度 10s 后，因未接收到新 region 的心跳信息，日志显示“operator timeout”

17:33:15.364, PD 接收到 29952726 上报的心跳信息，开始更新 region 的 key range，并将 regionversion 由 42222 更新为 42223

```
[2022/09/27 16:53:59.762 +08:00] [INFO] [operator_controller.go:560] ["operator timeout"] [region-id=29952726] [takes=10.152813837s]
[operator="\merge-region {merge: region 31772694 to 29952726} (kind:merge, region:29952726(42222,563), createdAt:2022-09-27 16:53:49.608997763 +0800
CST m=+10290714.240860054, startAt:2022-09-27 16:53:49.609557219 +0800 CST m=+10290714.241419511, currentStep:0, steps:[merge region 31772694 into
region 29952726]) timeout\""]
[2022/09/27 17:33:15.364 +08:00] [INFO] [region.go:468] ["region Version changed"] [region-id=29952726] [detail="StartKey
Changed:{7480000000000071FF4E5F728000000068FF2638E70000000000FA} -> {7480000000000071FF4E5F728000000068FF2456540000000000FA},
EndKey:{7480000000000071FF4E5F728000000068FF29701E0000000000FA}"] [old-version=42222] [new-version=42223]
```

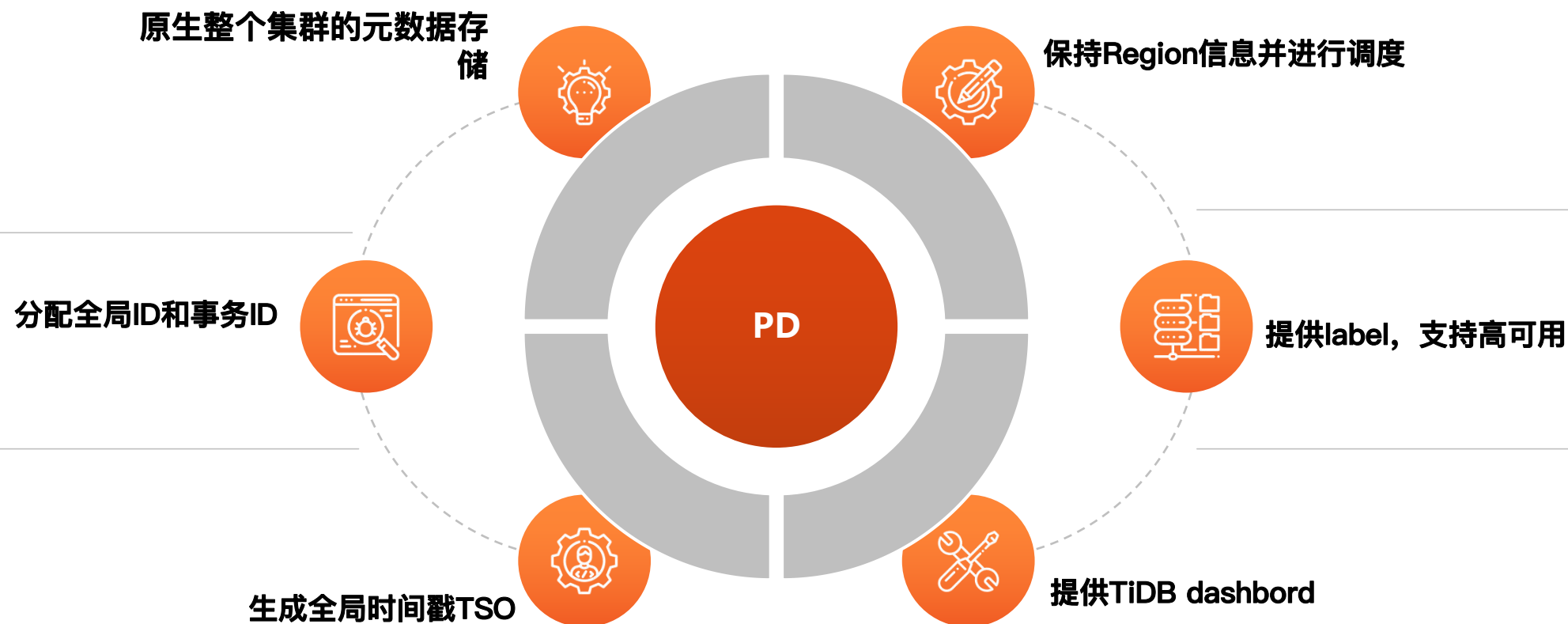
17:33:15 后，TiDB Server 日志恢复正常，未再出现 region 31772694 相关的报错信息

### 结论：

集群 region 数量较多（64万）+ 读写流量变化频繁，PD 处理 region 心跳出现堆积，未及时更新merge/split 后的 region 信息，导致 tidb 以过期的 region 信息访问 tikv 时发生 region missbackoff，最终造成 SQL 超时报错。



## region失联，偶发SQL执行超过40s



建议：PD作为核心，但PD的负重太重，需要给PD减负，才能让数据库更快，更稳。

# 100T Oracle迁移UbiSQL 实践

## Oracle当前现状

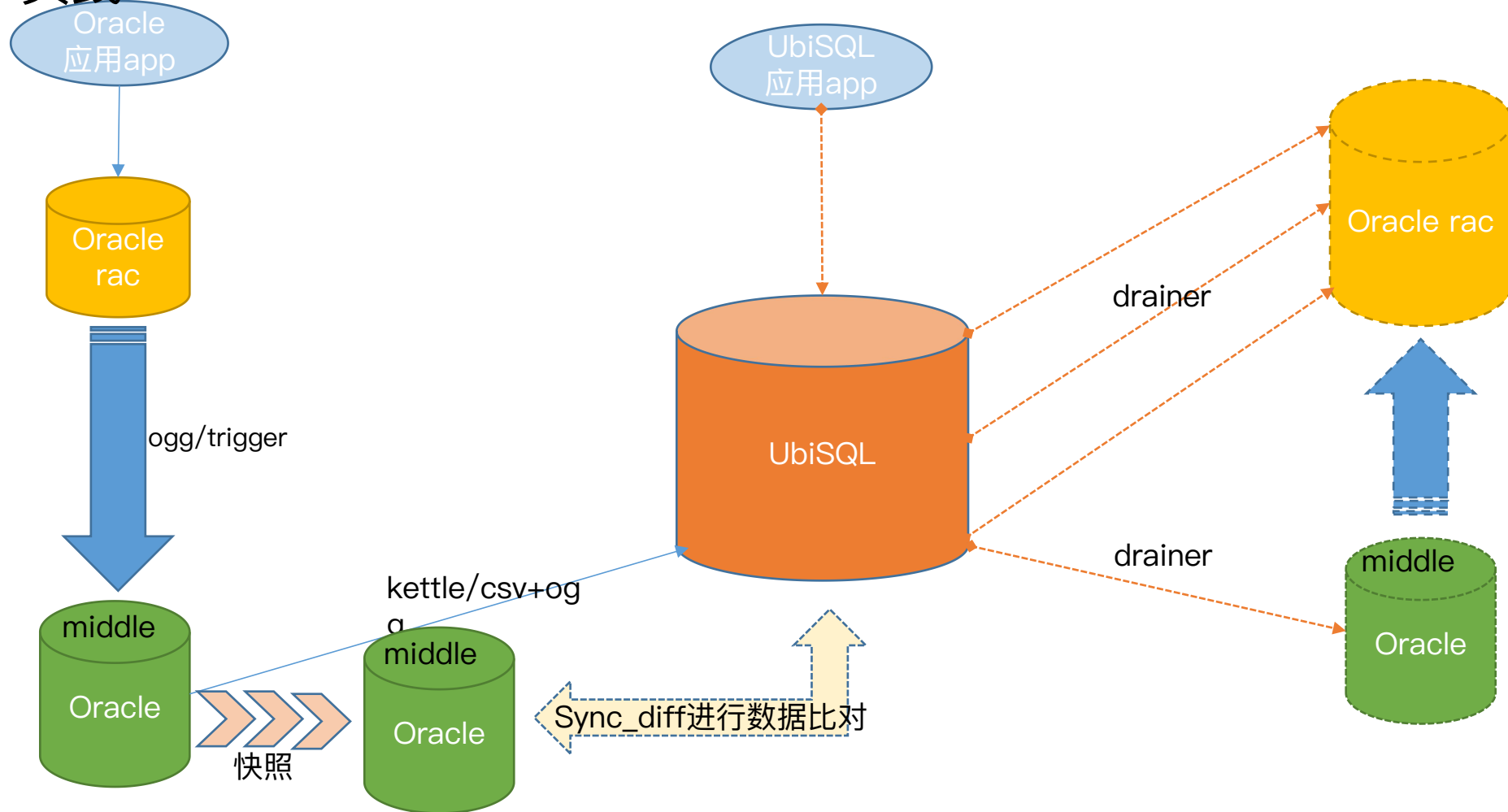
- 业务承载日调用量超1.4亿，日下发量超3000万，业务需求持续增长，当前架构难以支撑业务快速发展
- 数据库成为制约系统性能的瓶颈，异常隐患多：Oracle表空间上限32T，插入并发达到1000QPS以上时会出现索引分裂，数据库多次遇到硬解析、执行计划突变、索引分裂导致堵塞的情况
- 数据体量大（超100T），增长迅速（每月增速3~5T），Oracle数据压缩不理想，数据规模增长快，存储成本高

## Oracle迁移的难点

- 单表数据量大超过200亿行
- 多个大表业务逻辑调整，表合并与拆分
- 需保证一次性迁移的顺利，应用端需要提前模拟验证

## 100T Oracle迁移UbiSQL

## 实践



迁移:

1,有业务逻辑变更, 通过trigger催生出oracle的中间库, 中间表

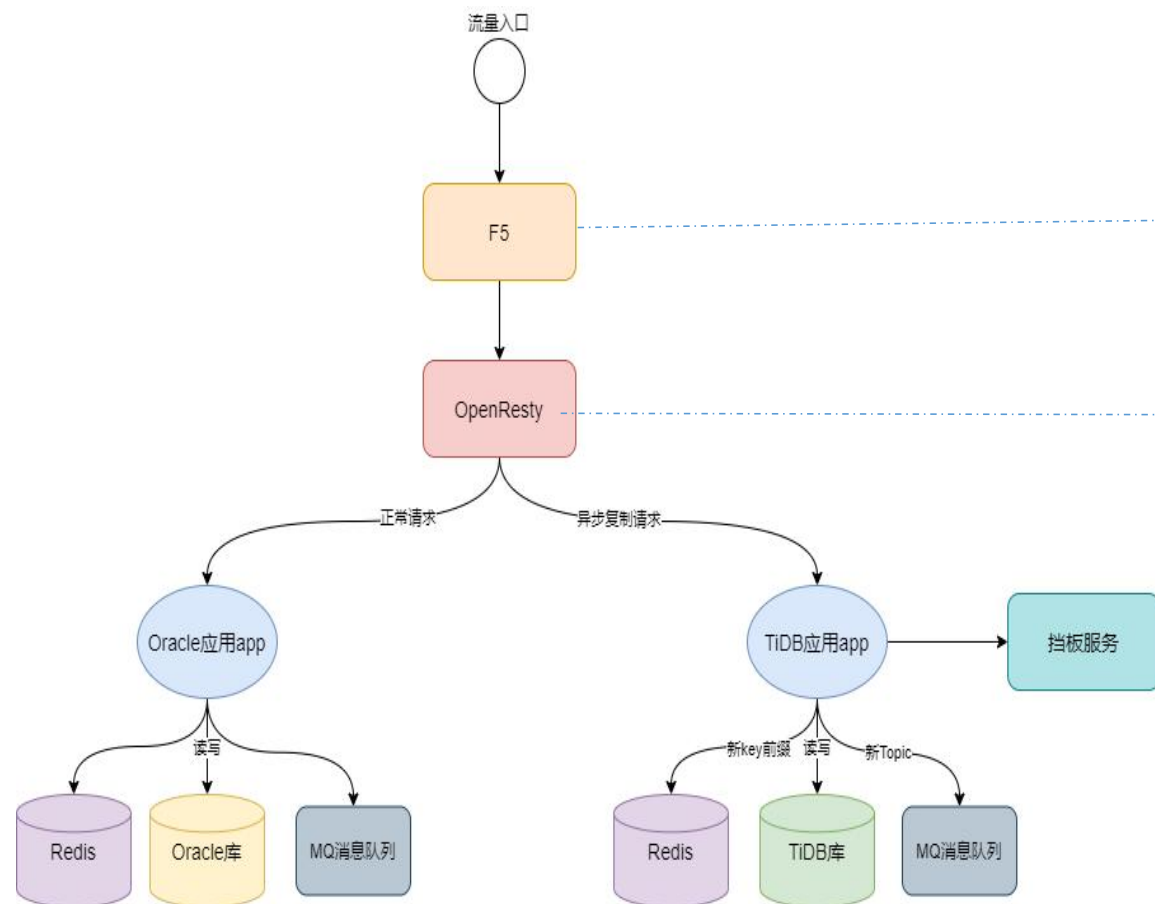
2,数据同步通过etl工具进行全量同步, ogg进行增量同步;大表采用导出csv, lightling导入的方式

3,通过生成中间库的快照,用sync\_diff进行全量数比对

回退: 拆分多个drainer将数据实时写回Oracle



# 100T Oracle迁移UbiSQL 实践



- **Nginx流量镜像**: 使用Nginx的ngx\_http\_mirror\_module模块实现流量复制, 指定复制地址, 反向代理时该模块会将http接口访问流量在Oracle应用环境执行的同时, 异步复制到TiDB应用环境上再次执行。
- **控制模块**: 利用OpenResty可执行lua脚本特性, 编写lua脚本实现开关控制是否开启流量复制、流量放大倍数的功能。
- **关联系统**: 增加挡板服务, 对部分第三方非查询类接口进行Mock。涉及到8个关联系统, 约20个接口。在调用关联系统代码中通过apollo配置动态设置是否走挡板服务, 挡板服务直接返回模拟成功报文。

## 100T Oracle迁移UbiSQL

## 实践

## 项目成果

## 成本节约

- 数据存储由100T降为**55.75T (-44.25%)**

## 性能提升

- 下发API接口性能**提升58%**，日下发量上限由3800万**提升至6000万**

## 架构升级

- 新系统架构可进行灵活扩缩容，可以支持更多业务场景及业务调用



## 亮点

## UbiSQL在平安大数据量级的系统实践之一

- 实现高并发大数据量迁移的标杆，系统顺利完成了迁移的同时性能得到了提升
- 项目的成功能对未来的迁移项目提供宝贵的经验

## 实现大规模多表关联异构数据同步沉淀一套解决方案

- 项目涉及对**超200亿数据量**的大表进行合并与拆分，没有历史方案可以支持，多次协商后使用Oracle中转库+存储过程、触发器的方案实现数据垂直合并、水平拆分的方案，最终顺利完成了数据迁移，此方案为后续类型的超大表异构同步提供了借鉴经验

## 基于流量复制的全真模拟验证方案实现应用+数据库整体架构层面的验证

- 支持生产流量按需倍数复制，实现模拟真实场景的压测；应用接口报文比对，实现应用功能的一致性验证
- 全真模拟验证使得系统在切换上线前进行了充分的测试，最终实现一次性全量切换成功

得 Select \* from table where score>500; 为什么会 OOM?





# 使用UbiSQL的心得

如何避免与解决OOM?

## 覆盖索引

将filter的条件都下推到tikv部分进行, tidb仅仅获取最终结果

## 扩大内存

增加单个节点的内存, 以达到大数据量内存要求

## TiFLASH

列式存储, 部分函数下推, 减少返回的计算的数据量

## 应用调整

对大数据量查询进行分段, 比如按照时间进行分段, 多次查询, 减少排序与聚合的数据量

## 限制内存

限制内存, 对超过内存使用的SQL设置kill阈值, 避免影响整个tidb使用

1

2

3

4

5

# 使用UbiSQL的心得

1

Tidb 5.X以前的版本应用于生产环境bug会比较多。从TiDB 5.4开始可以应用于核心数据库环境，最佳版本推荐TiDB 6.5

2

TiCDC在6.5以前bug比较多，TiDB 6以下版本使用更加稳定和快捷的binlog

3

一定要限制tidb server内存使用，达到最大后直接kill掉，避免主机的OOM

4

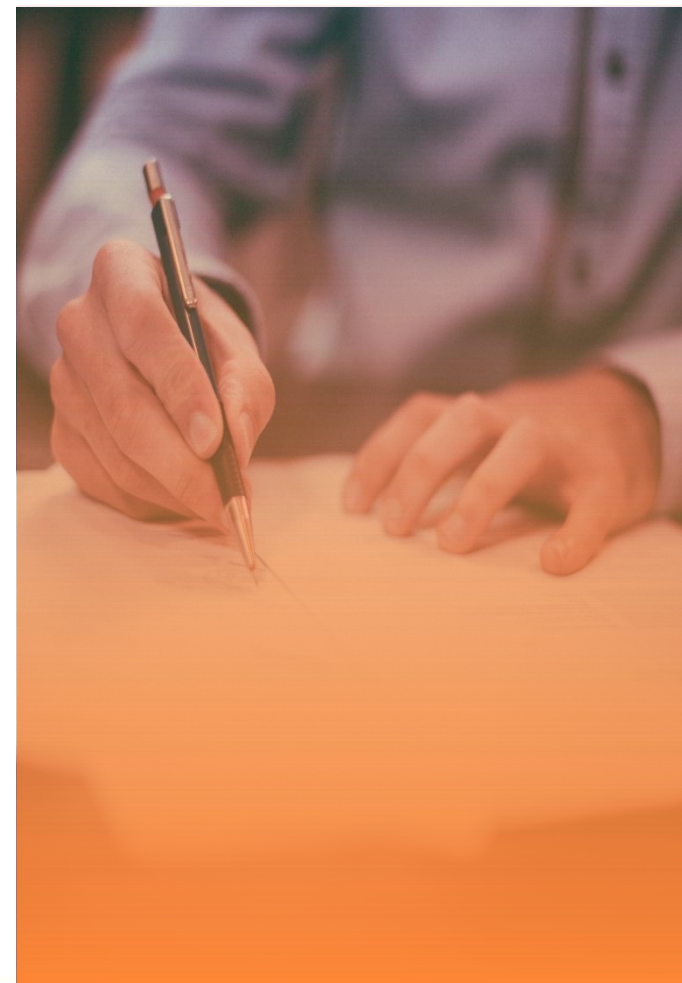
一定要设置好Label的属性，确保3副本不会在同一个区域或者同一台主机上保证高可用

5

TiKV磁盘推荐使用NVMe SSD > SATA SSD(RAID5)

6

有同城三中心机房的可以使用三中心架构省钱，省事，省心



# THANKS

TDDL

DistributedTable

DBproxy

HBase

PostgreSQL

SSD

MongoDB

GreatDB

Cassandra

Hyperbase

Hubble

DataCenter

VisualDataPlatform

Blockchain

ArgoDB

Distributed

DatabaseKernel

TemporalData

CloudnativeData

AIalgorithm