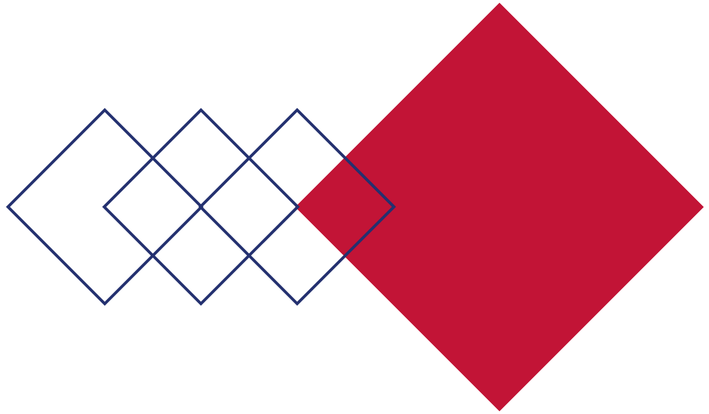
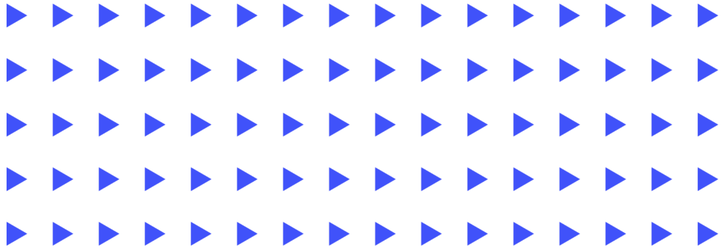
TiDB | COMMUNITY



使用内存引擎强化 TiKV

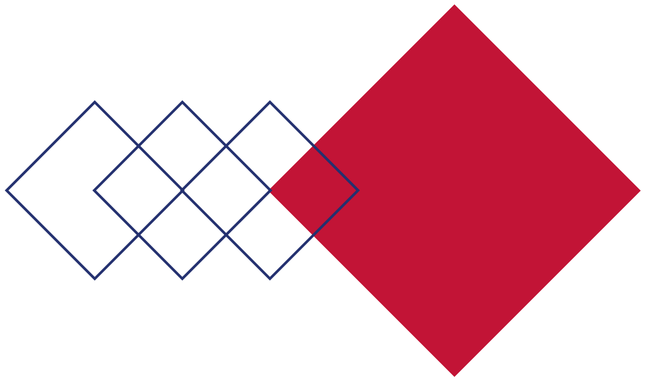


{ 唐晨杰 PingCAP 分布式存储工程师 }

TiDB | COMMUNITY

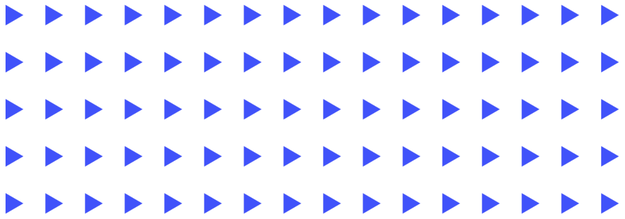
目录

TiDB | COMMUNITY



Part 1

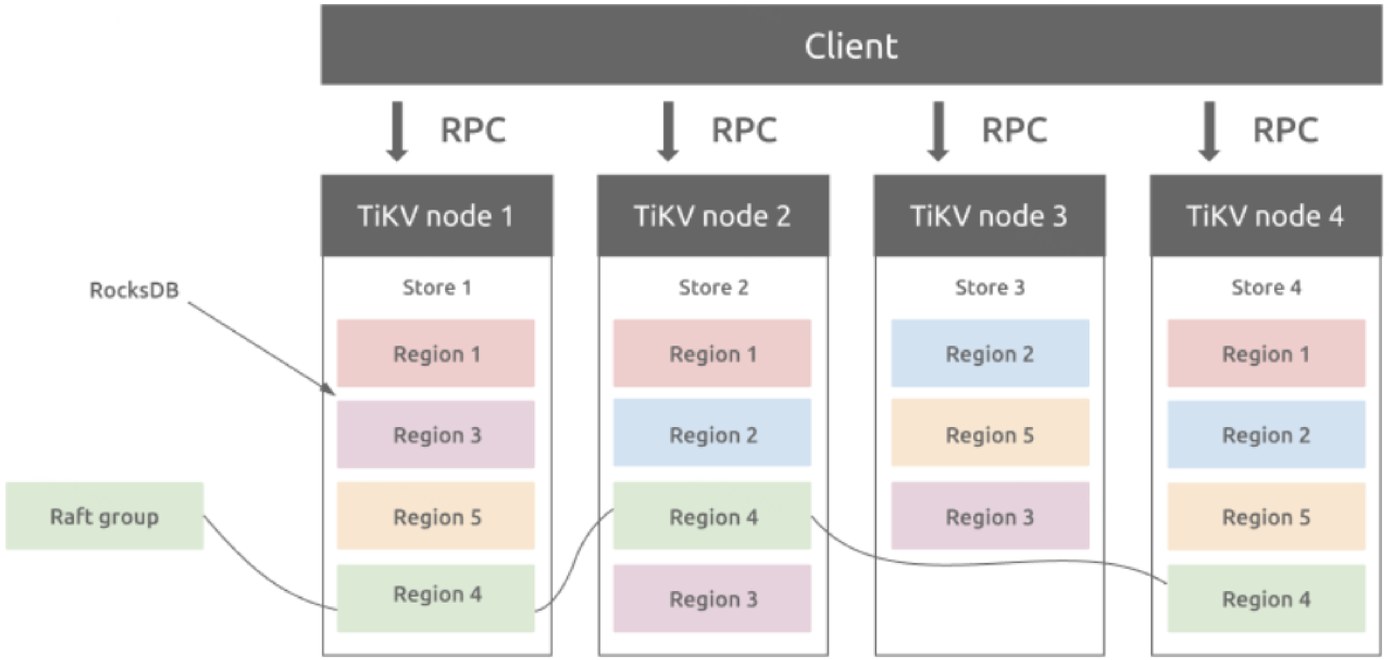
TiKV 简介



TiKV 简介

TiDB | COMMUNITY

- TiDB 的存储层
- 分布式键值数据库
- 提供满足 ACID 约束的分布式事务（percolator）接口
- Raft 协议保证多副本一致性及高可用



GitHub: https://github.com/tikv/tikv

TiKV 的数据

TiDB | COMMUNITY

TiKV 下的多版本控制（MVCC）：

```
key1-version4 -> value14
key1-version2 -> value12
key1-version1 -> value11
key2-version7 -> value27
key2-version3 -> value23
key2-version2 -> value22
key2-version1 -> value21
……
keyN-version2 -> valueN2
keyN-version1 -> valueN1
……
```

TiKV 的数据

TiDB | COMMUNITY

使用 version 4 进行遍历：

```
key1-version4 -> value14
key1-version2 -> value12
key1-version1 -> value11
key2-version7 -> value27
key2-version3 -> value23
key2-version2 -> value22
key2-version1 -> value21
……
keyN-version2 -> valueN2
keyN-version1 -> valueN1
……
```

TiDB | COMMUNITY

Part 2

TiKV 读场景痛点

冗余 MVCC 版本影响读性能

TiDB | COMMUNITY

在下面数据集中读两行数据：

key1-version100,
key1-version99,
..,
key1-version1,
key2-version100-delete,
key3-version100-delete,
..,
key10000-version100-delete,
key10001-version100

冗余 MVCC 版本影响读性能

TiDB | COMMUNITY

真实集群监控（TPCC 负载）：

当前的缓解办法以及缺陷

TiDB | COMMUNITY

垃圾回收(Garbage Collection)

- Compaction Filter
- Periodic GC

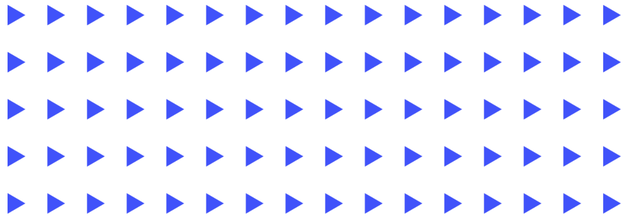
问题

- Safe point 阻塞 GC
- Compaction 需要时间

TiDB | COMMUNITY

Part 3

内存引擎概述



内存引擎是什么

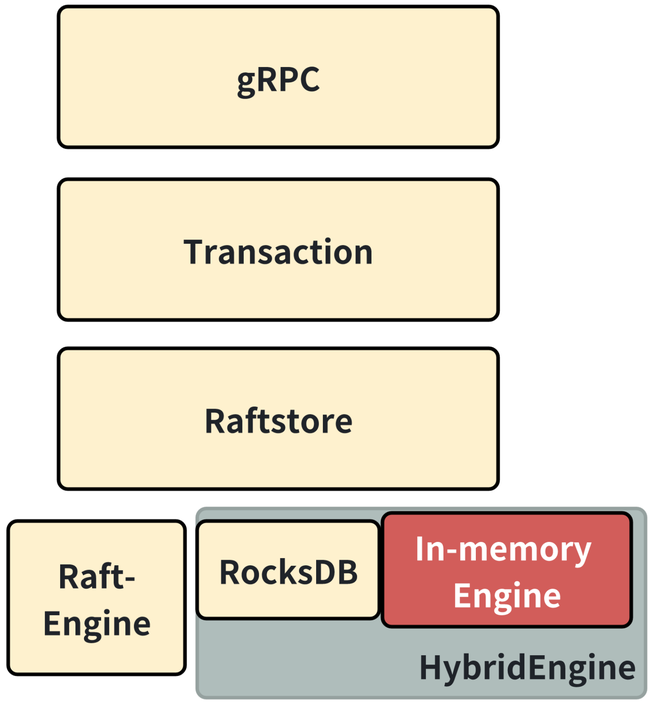
TiDB | COMMUNITY

- 内存缓存
- 缓存读热点 region
- 被缓存的 region 拥有“全量”数据
- 提供快照读
- 垃圾回收（GC）

内存引擎架构

TiDB | COMMUNITY

内存引擎在 TiKV 中的位置



gRPC
- 接收和处理上层（如 TiDB）的请求

Transaction
- 分布式事务的处理
- 下发 KV 请求给 Raftstore

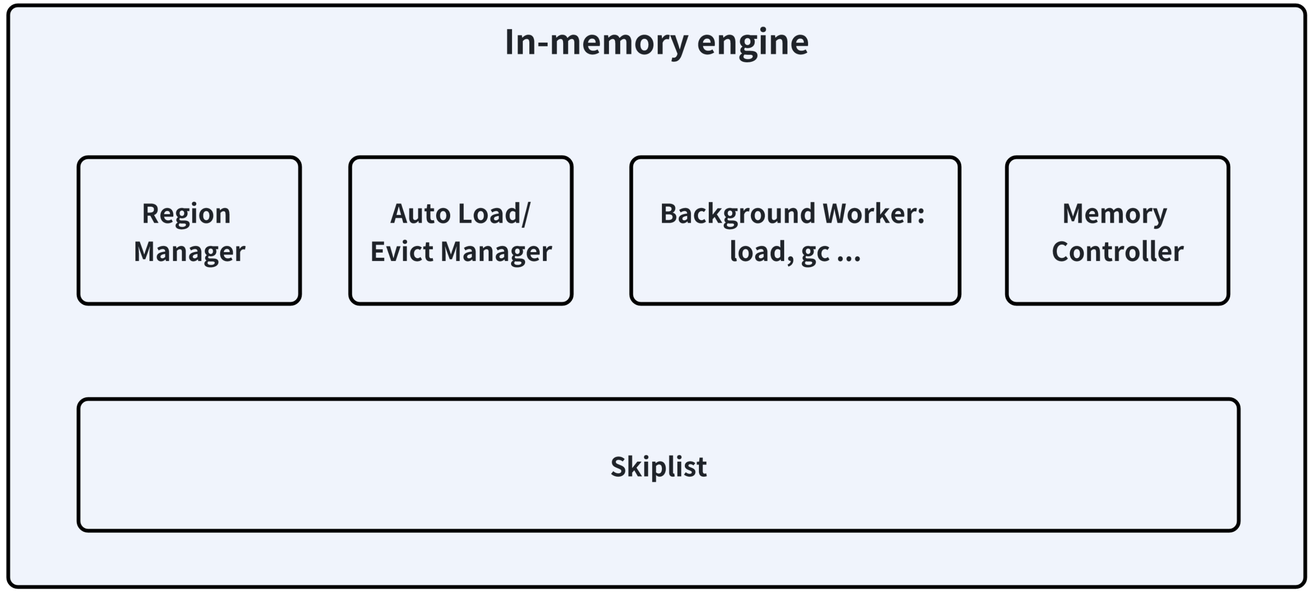
Raftstore
- Propose, replicate, commit, apply raft commands
- Region 分裂和合并
- Region 迁移
- …

Raft-Engine
- 持久化 raft commands

HybridEngine
- 持久化 KV

内存引擎的重要模块

TiDB | COMMUNITY



Skiplist
- 存储 KV

Region Manager
- 管理 region 的不同状态
- 检验读请求是否应该被处理
- ...

Auto Load/Evict Manager
- 载入哪些 region
- 驱逐哪些 region

Background worker
- 执行数据载入
- 垃圾回收
- ...

Memory Controller
- 内存管理

如何解决 MVCC 冗余版本问题?

TiDB | COMMUNITY

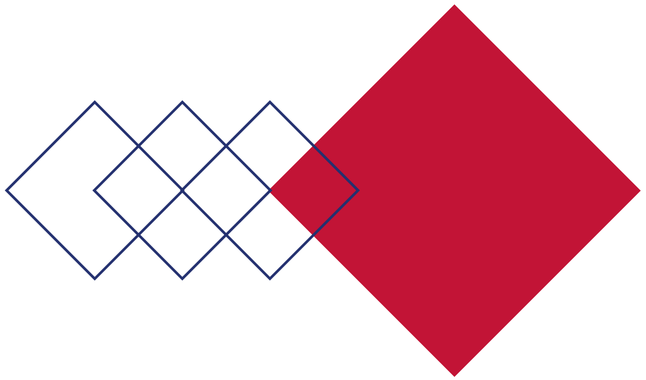
独立的 safe-point

- 可以设置更短的更新周期
- 不会受到诸如 flashback 等特性的影响

GC 的时候就地删除

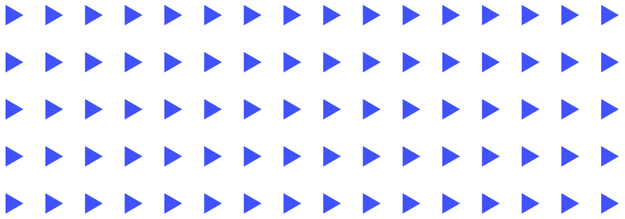
- 跳表可就地删除

TiDB | COMMUNITY



Part 4

内存引擎的实现细节



内存引擎设计细节

TiDB | COMMUNITY

RFC: https://github.com/tikv/rfcs/pull/111

Tracking Issue: https://github.com/tikv/tikv/issues/16141

读流程

TiDB | COMMUNITY

判断读请求能否访问内存引擎：

- Region 是否被缓存
- 读请求的 read ts 是否大于 region 的 safe point

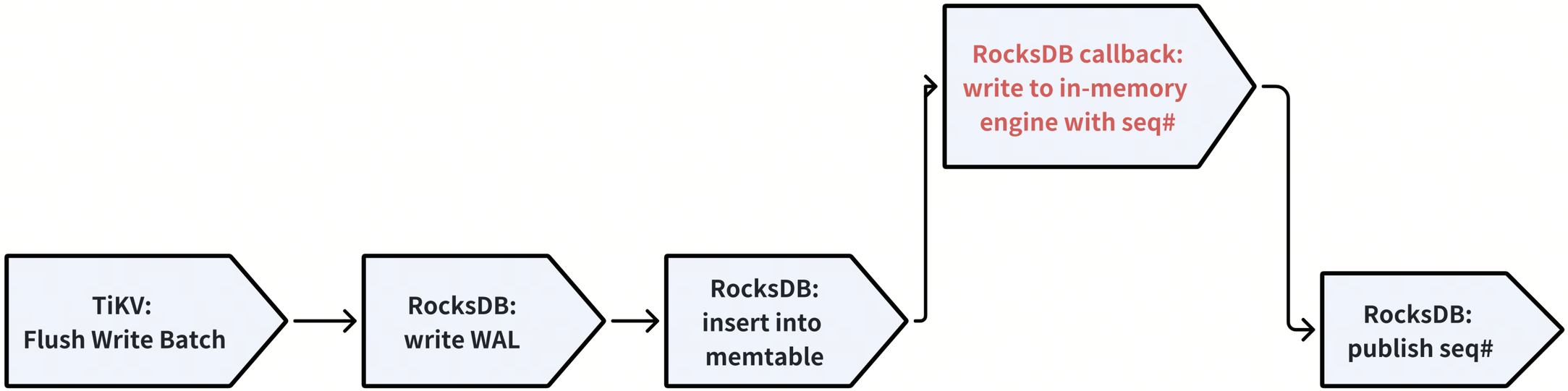
写流程

TiDB | COMMUNITY

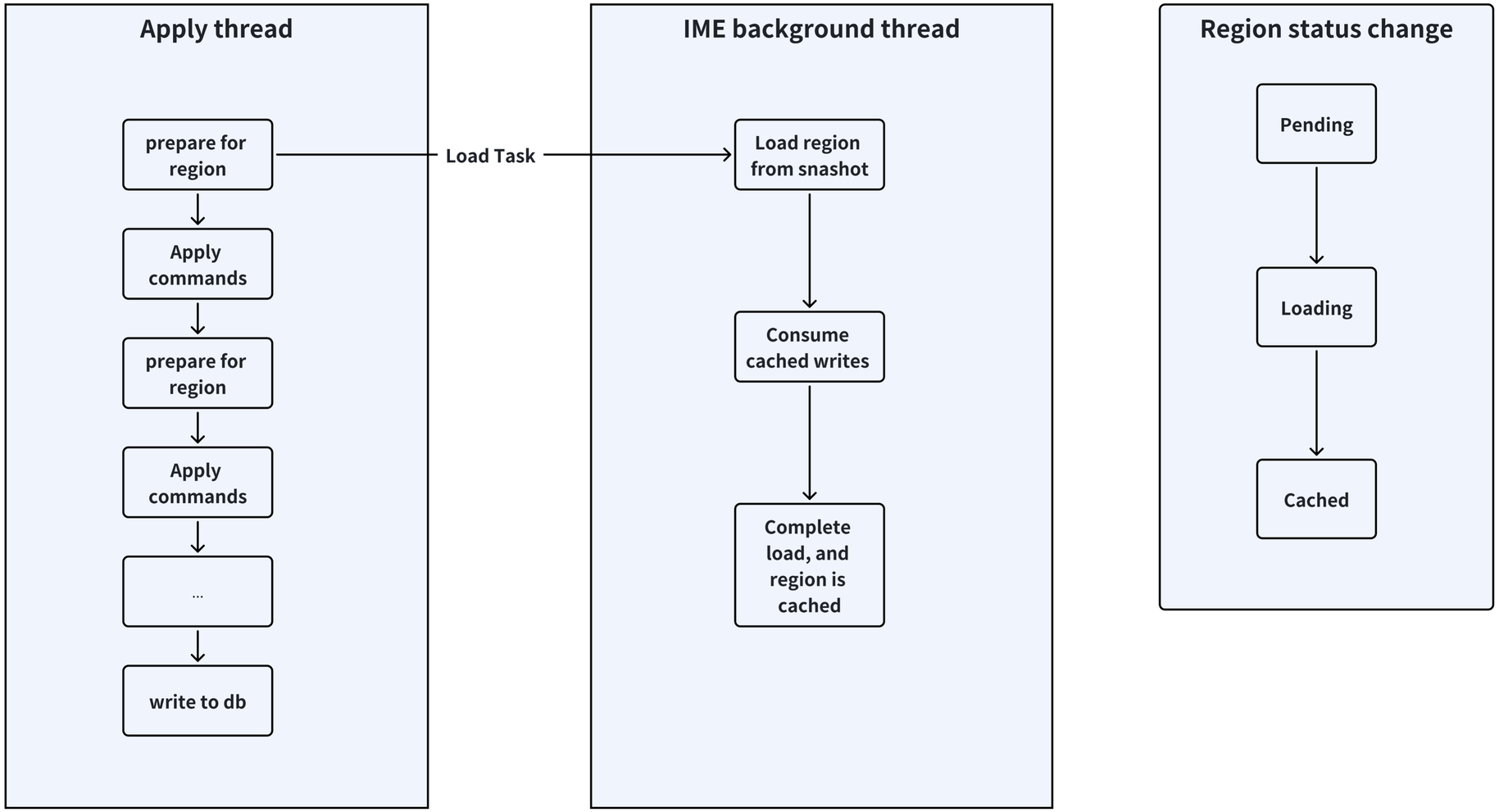
全量写

原子写

- RocksDB write batch 原子写
- Memtable callback



载入



Apply thread 是 TiKV 执行 raft commands 的线程

驱逐

TiDB | COMMUNITY

- 驱逐后的 region 不再提供度服务
- 所有 snapshots 释放才能清理数据

TiDB | COMMUNITY

Part 5

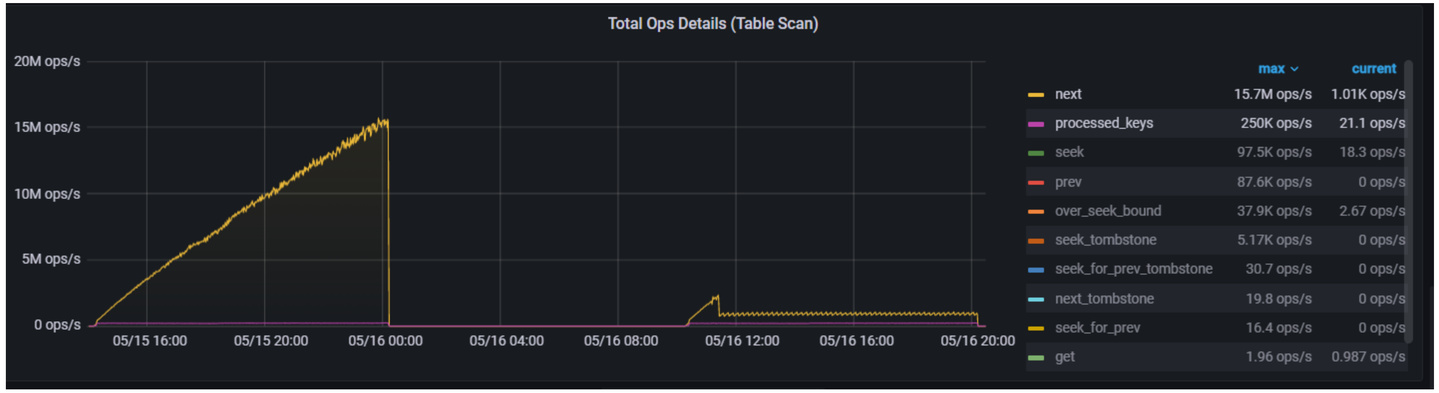
内存引擎性能测试结果

TPCC 测试结果

TiDB | COMMUNITY

3 TiKV 节点, 16C CPU, 32GB RAM.
测试时间 10 小时
Safe point 更新周期 1 day

Tests	Unified readpool CPU (per node)	Next /procseed_keys	Avg (coprocessor latency)	p80 (coprocessor latency)	p99 (coprocessor latency)	p999 (coprocessor latency)
TPCC (without IME) 50 warehouses,	267%	15.5M ops/s / 250K ops/s	4ms	8.05ms	30.9ms	50ms
TPCC (with IME) 50 warehouses,	86%	900K ops/s / 249K ops/s	1.18ms	1.4ms	18ms	47.3ms



TPCC 测试结果

TiDB | COMMUNITY

3 TiKV 节点, 16C CPU, 48GB RAM.
Safe point 更新周期 10 min （默认）

Tests	Unified readpool CPU (per node)	Next /procseed_keys	Avg (coprocessor latency)	p80 (coprocessor latency)	p99 (coprocessor latency)	p999 (coprocessor latency)
TPCC (without IME) Small dataset, 50 warehouses, 11 GiB per tikv node	280%	8.35M ops/s / 350K ops/s	1.8ms	2.9ms	11.7ms	24.3ms
TPCC (with IME) Small dataset, 50 warehouses, 11 GiB per tikv node	230%	2M ops/s / 346K ops/s	1.3ms	1.5ms	11.2ms	24.7ms
TPCC (without IME) Large dataset, 1000 warehouses, 75 GiB per tikv node	430%	2M ops/s / 803K ops/s	4.5ms	3.0ms	50ms	150ms
TPCC (with IME) Large dataset, 1000 warehouses, 75 GiB per tikv node	400%	1.09M ops/s / 800K opsp/s	4.5ms	2.6ms	50ms	150ms

注意： 本页测试使用了前面测试不同的集群和客户端参数

THANKS

PingCAP: https://www.pingcap.com/