PingCAP

TiDB 培训 - Placement Driver

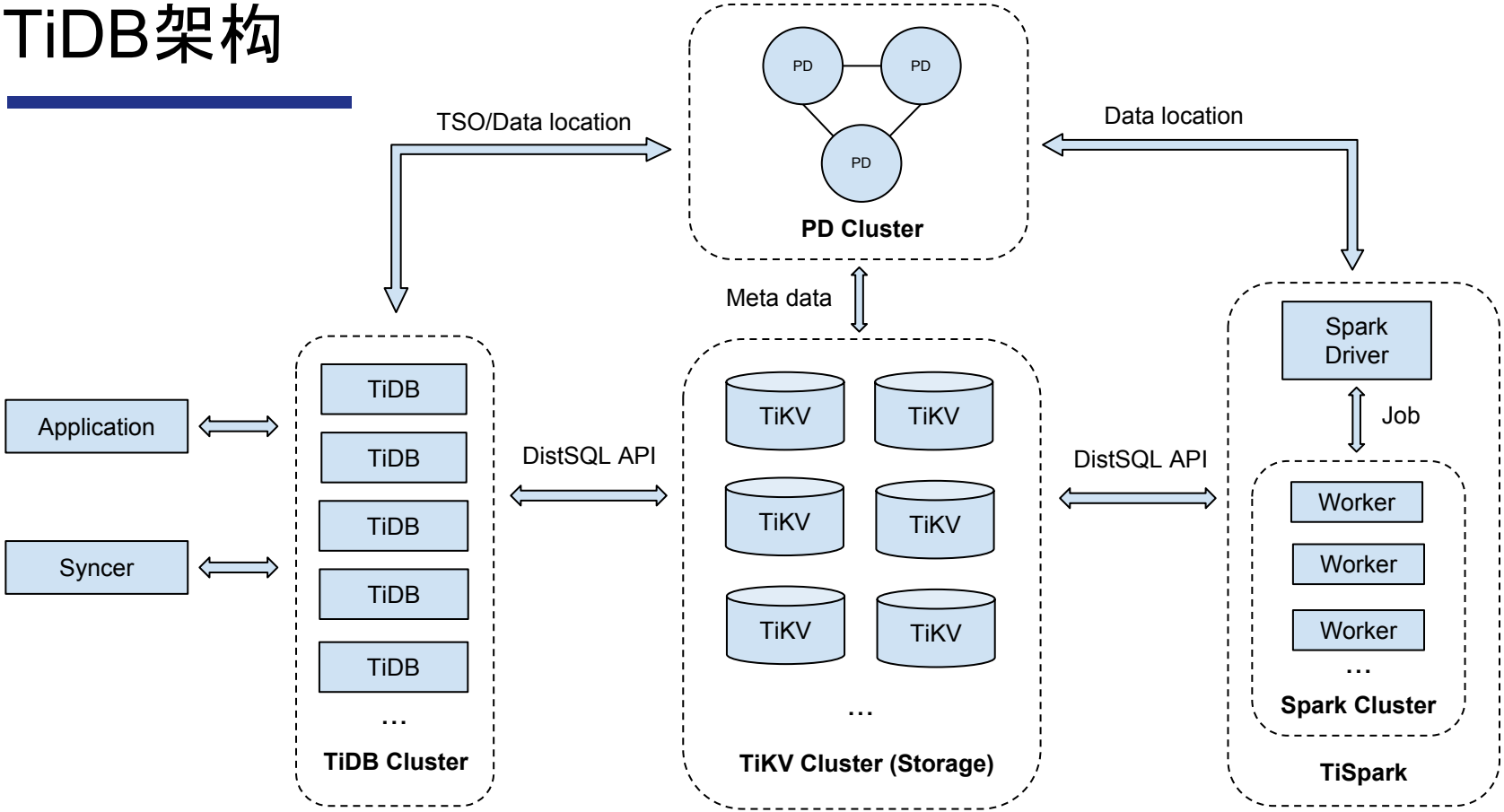
PingCAP

PingCAP

大纲

- TiDB 架构
- PD Server 介绍
- PD Server 架构
- PD Server 功能
- PD Server 运维
- PD Server 监控

TiDB架构


PD
PD
PD
PD Cluster
TSO/Data location
Data location
Meta data
Application
Syncer
TiDB
TiDB
TiDB
TiDB
TiDB
...
TiDB Cluster
DistSQL API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
...
TiKV Cluster (Storage)
DistSQL API
Spark Driver
Job
Worker
Worker
Worker
...
Spark Cluster
TiSpark

TiDB 组件 -- 基础服务

TiDB Server

TiDB Server 负责接收 SQL 请求，处理 SQL 相关的逻辑，并通过 PD 找到存储计算所需数据的 TiKV 地址，与 TiKV 交互获取数据，最终返回结果。

TiDB Server 是无状态的，其本身并不存储数据，只负责计算，可以无限水平 扩展，可以通过负载均衡组件（如LVS、HAProxy 或 F5）对外提供统一的接入地址。

PD Server

Placement Driver (简称 PD) 是整个集群的管理模块，其主要工作有三个：一是存储集群的元信息（某个 Key 存储在哪个 TiKV 节点）；二是对 TiKV 集群进行调度和负载均衡（如数据的迁移、Raft group leader 的迁移等）；三是分配全局唯一且递增的事务 ID。

PD 是一个集群，需要部署奇数个 节点，一般线上推荐至少部署 3 个节点。

TiKV Server

TiKV Server 负责存储数据，从外部看 TiKV 是一个分布式的提供事务的 Key-Value 存储引擎。存储数据的基本单位是 Region，每个 Region 负责存储一个 Key Range 的数据，每个 TiKV 节点会负责多个 Region 。TiKV 使用 Raft 协议做复制，保持数据的一致性和容灾。副本以 Region 为单位进行管理，不同节点上的多个 Region 构成一个 Raft Group，互为副本。

TiKV 至少部署 3 个以上的节点实例。

PingCAP

TiDB 组件 -- 分析服务

TiSpark

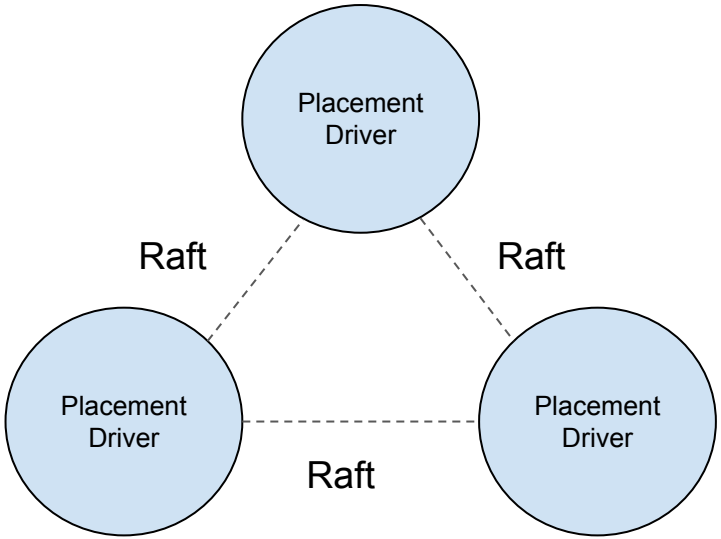
TiSpark 是基于 TiDB 数据库为解决用户复杂 OLAP 需求而推出的产品，TiSpark 是将 Spark SQL 直接运行在分布式存储引擎 TiKV 上的 OLAP 解决方案。借助 Spark 平台，同时融合 TiKV 分布式集群的优势，和 TiDB 一起为用户一站式解决 HTAP（Hybrid Transactional/Analytical Processing）需求。

TiSpark 基于原生的 Spark 进行了二次开发，利用 Spark 集群的分布式和计算优势，集成并依赖于 TiKV 集群和 Placement Driver(PD)，使计算下推并直接访问 TiKV 存储的数据，从而完成准实时的 OLAP 场景。

Placement Driver 介绍

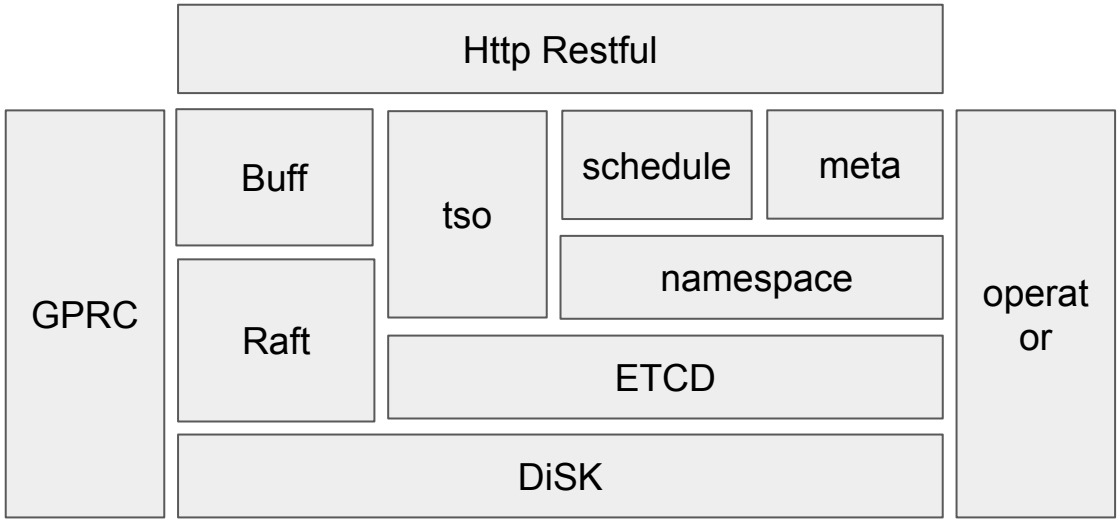
Placement Driver (后续以 PD 简称) 是 TiDB 里面全局中心总控节点，它负责整个集群的调度，负责全局 ID 的生成，以及全局时间戳 TSO 的生成等。PD 还保存着整个集群 TiKV 的元信息，负责给 client 提供路由功能。

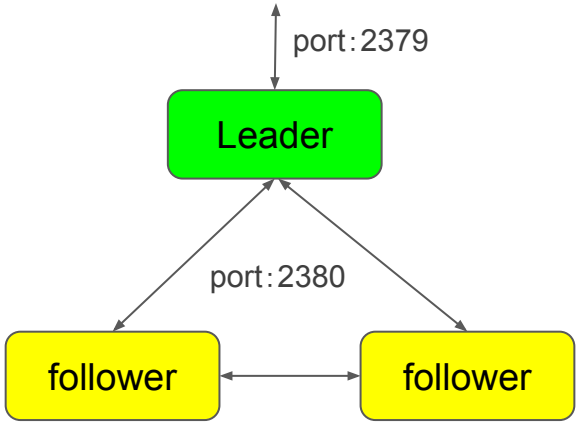
- PD 概念来自于 Google Spanner
- PD 作为大脑，提供了整个集群的全局视图
- 存储集群元数据信息
  - Clients have cache of placement information.
- 维护集群的高可用数据副本
  - 默认 3 副本
- 根据负载进行 Balance 的数据调度
- PD 本身也是高可用集群 .
  - Thanks to Raft.



PD 架构

PD 内部集成 etcd，使用 Raft 协议保证系统高可用；对外提供 grpc 接口与 restful 接口；通过接收 heartbeat 获取 tikv 信息并通过 operator 完成对region 的管理和调度。





PD 架构设计

在架构上面，PD 所有的数据都是通过 TiKV 主动上报获知的。同时，PD 对整个 TiKV 集群的调度等操作，也只会在 TiKV 发送 heartbeat 命令的结果里面返回相关的命令，让 TiKV 自行去处理。这样设计上面就非常简单，我们完全可以认为 PD 是一个无状态的服务（当然，PD 仍然会将一些信息持久化到 etcd），所有的操作都是被动触发，即使 PD 挂掉，新选出的 PD leader 也能立刻对外服务，无需考虑任何之前的中间状态。

作为中心总控节点，PD 通过集成 etcd，自动的支持 auto failover，无需担心单点故障问题。同时，PD 也通过 etcd 的 raft，保证了数据的强一致性，不用担心数据丢失的问题。

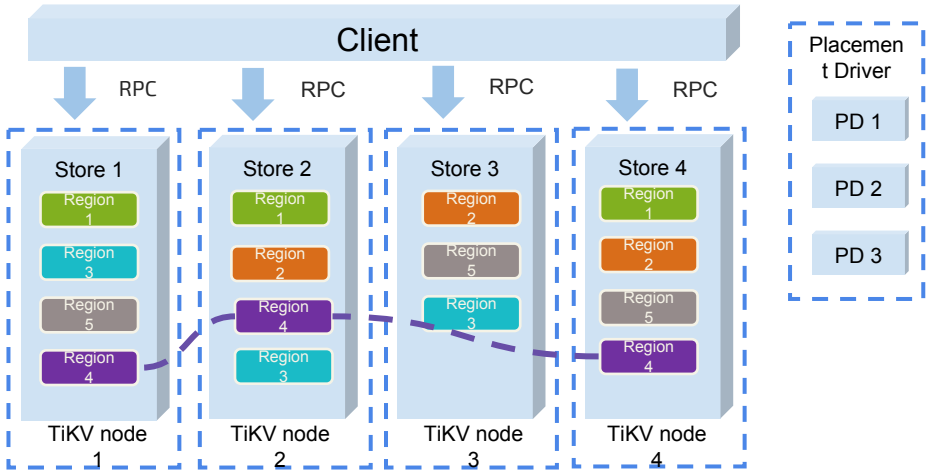
PD 功能

PD 是 TiDB 的大脑，是整个 TiDB 集群的管理与调度中心，其主要完成以下三部分功能：

1. **分配全局唯一且递增的事务 ID**
    a. **TSO**
2. **存储集群的元信息**
    a. **store、region**
    b. **心跳**
3. **对 TiKV 集群进行调度和负载均衡**
    a. **调度 与 执行**
    b. **路由**

PD 调度的几个概念

- TiKV Node : Node 可以认为是一个实际的物理机器，每个 Node 负责一个或者多个 Store。
- Store : Store 使用 RocksDB 进行实际的数据存储，通常一个 Store 对应一块硬盘。
- Region : Region 是数据移动的最小单元，对应的是 Store 里面一块实际的数据区间。每个 Region 会有多个副本（replica），每个副本位于不同的 Store，而这些副本组成了一个 Raft group。



PingCAP

TSO

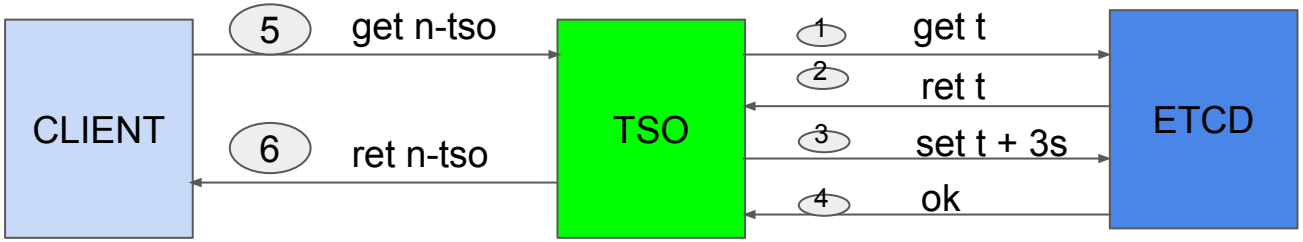
TSO (Timestamp Oracle)是一个全局的时间戳，它是 TiDB 实现分布式事务的基石。所以对于 PD 来说，我们首先要保证它能快速大量的为事务分配 TSO，同时也需要保证分配的 TSO 一定是单调递增的，不可能出现回退的情况。

TSO 是一个 int64 的整形，它由 physical time + logical time 两个部分组成。Physical time 是当前 unix time 的毫秒时间，而 logical time 则是一个最大 1 << 18 的计数器。也就是说 1ms，PD 最多可以分配 262144 个 TSO，这个能满足绝大多数情况了。

TSO 分配

对于 TSO 的保存于分配，PD 会做如下处理：

1. 当 PD 初始化或切换成为 leader 之后，PD 会从 ETCD 上面获取上一次保存的时间，如果发现本地的时间比这个大，则会继续等待直到当前的时间大于这个值；
2. 当 PD 能分配 TSO 之后，首先会向 ETCD 申请一个最大的时间，例如，假设当前时间是 t1，每次最多能申请 3s 的时间窗口，PD 会向 etcd 保存 t1 + 3s 的时间值，然后 PD 就能在内存里面直接使用这一段时间窗口。当当前的时间 t2 大于 t1 + 3s 之后，PD 就会在向 etcd 继续更新为 t2 + 3s。这么处理的好处在于，即使 PD 当掉，新启动的 PD 也会从上一次保存的最大的时间之后开始分配 TSO，也就是 1 处理的情况；
3. PD 在内存里面保存 t+3 的可分配的时间窗口，在 Client 请求 TSO 的时候，PD 能直接在内存里面计算 TSO 并返回。
4. Client 批量的向 PD Leader 获取 TSO，Client 会首先收集一批事务的 TSO 请求，譬如 n 个，然后直接向 PD 发送命令，参数就是 n，PD 收到命令之后，会生成 n 个 TSO 返回给客户端；
5. PD leader 收到客户端请求后，在内存中根据 t+3s 的时间戳，计算出 TSO 返回给Client。



PD 元信息

- 生成 Cluster ID
  - 防止用户多个集群配错了，或者重新部署时数据没清干净
- 生成唯一 ID
  - RegionID，StoreID，PeerID，etc.
- Bootstrap
  - 集群从无到有的过程，确定第一个 Store 和第一个 Region

心跳（Heartbeat）

PD 所有关于集群的数据都是由 TiKV 主动心跳上报的，PD 对 TiKV 的调度也是在心跳的时候完成的。通常 PD 会处理两种心跳，一个是 TiKV 自身 store 的心跳，而另一个则是 store 里面 region 的 leader peer 上报的心跳。

- Store 的心跳：Store 的心跳主要包括了当前 store 的状态信息，包括该 store 有多少个 region，有多少个 region 的 leader peer 在该 store 上面，也包含 address、存储空间、snapshot 状况、流量等等。PD 在接收的到 store 心跳后，就是将心跳里面当前的 store 的一些状态缓存到 cache 里面，做个后续调度的依据。
- Region 的心跳：Region 心跳包括当前 Region 的信息和流量信息，例如最大 值，最小值，Region 大小等。只有 Region 的 leader peer 才会去上报所属 region 的信息，follower peer 是不会上报的。PD 在收到 region 的心跳之后，首先也会将其放入 cache 里面，如果 PD 发现 region 的 epoch 有变化，就会将这个 region 的信息也保存到 etcd 里面。

信息收集（Heartbeat）

调度依赖于整个集群信息的收集，简单来说，我们需要知道每个 TiKV 节点的状态以及每个 Region 的状态。TiKV 集群会向 PD 汇报两类消息：

每个 TiKV 节点会定期向 PD 汇报节点的整体信息

TiKV 节点（Store）与 PD 之间存在心跳包，一方面PD 通过心跳包检测每个 Store 是否存活，以及是否有新加入的Store；另一方面，心跳包中也会携带这个 Store 的状态信息，主要包括：

- 总磁盘容量
- 可用磁盘容量
- 承载的 Region 数量
- 数据写入速度
- 发送/接受的 Snapshot 数量（Replica 之间可能会通过 Snapshot 同步数据）
- 是否过载
- 标签信息（标签是具备层级关系的一系列 Tag）

每个 Raft Group 的 Leader 会定期向 PD 汇报信息

每个 Raft Group 的 Leader 和 PD 之间存在心跳包，用于汇报这个 Region 的状态，主要包括下面几点信息：

- Leader 的位置
- Followers 的位置
- 掉线 Replica 的个数
- 数据写入/读取的速度

调度 (Operator)

在 PD，主要有两类调度，首先是对资源的调度，包括存储 storage 以及计算 leader；其次是管理类调度，根据 store 状态进行监控和调度。由此，主要调度内容有：

- Region Balance 多副本调度；
- Region Split / Merge 操作；
- Region Leader 调度；
- Region Replica 调度；
- Hot Region 调度
- Namespace 调度

- New Store
- Delete Store
- Store Disconnted(30s)
- Store Down
- Store Offline
- Store Tombstone

注：Operator 只是 PD 提供给 tikv 的建议，具体是否被执行以 tikv 为准。

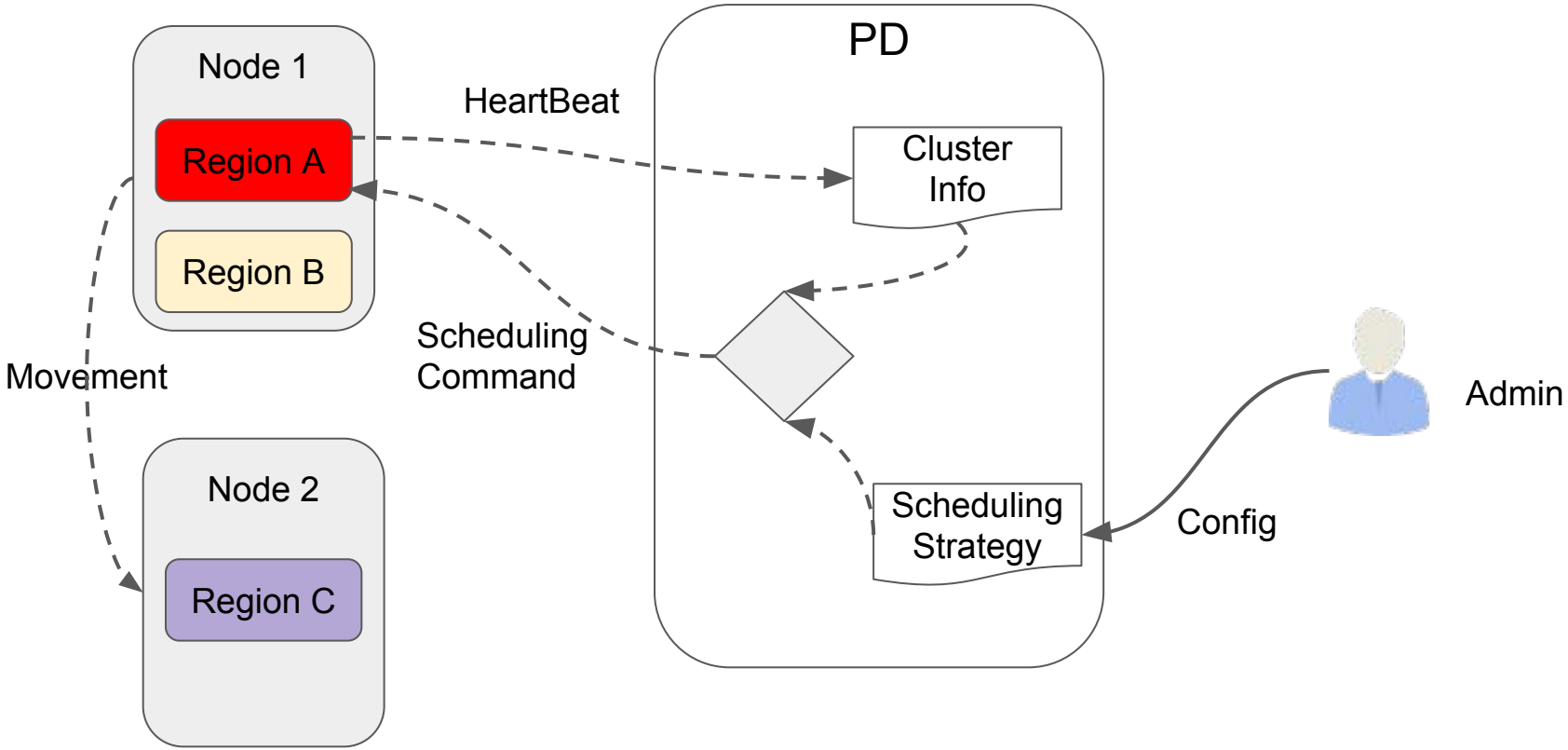
调度 (Balance)

- LeaderBalance
  - 统计不同 Store 上的 Leader 数量
    - 从 Leader 最多的 Store 上找出一个 Leader Peer，将 leadership 移走
    - 从 Leader 最少的 Store 上找出一个 Follower Peer，将 leadership 移入
- RegionBalance
  - 统计不同 Store 上的 Peer 数量
    - 从 Peer 最多（磁盘空间最紧张）的 Store 上找出一个 Region
    - 找到 Peer 最少（磁盘空间最富余）的 Store
    - 生成 [AddPeer, RemovePeer] 或 [AddPeer, TransferLeader, RemovePeer]
- HotRegionBalance
  - 统计一段时间内的 Region 流量排行榜
  - 统计排行榜 TopN 在 Store 的分布情况
  - 生成 Operator 使之均衡

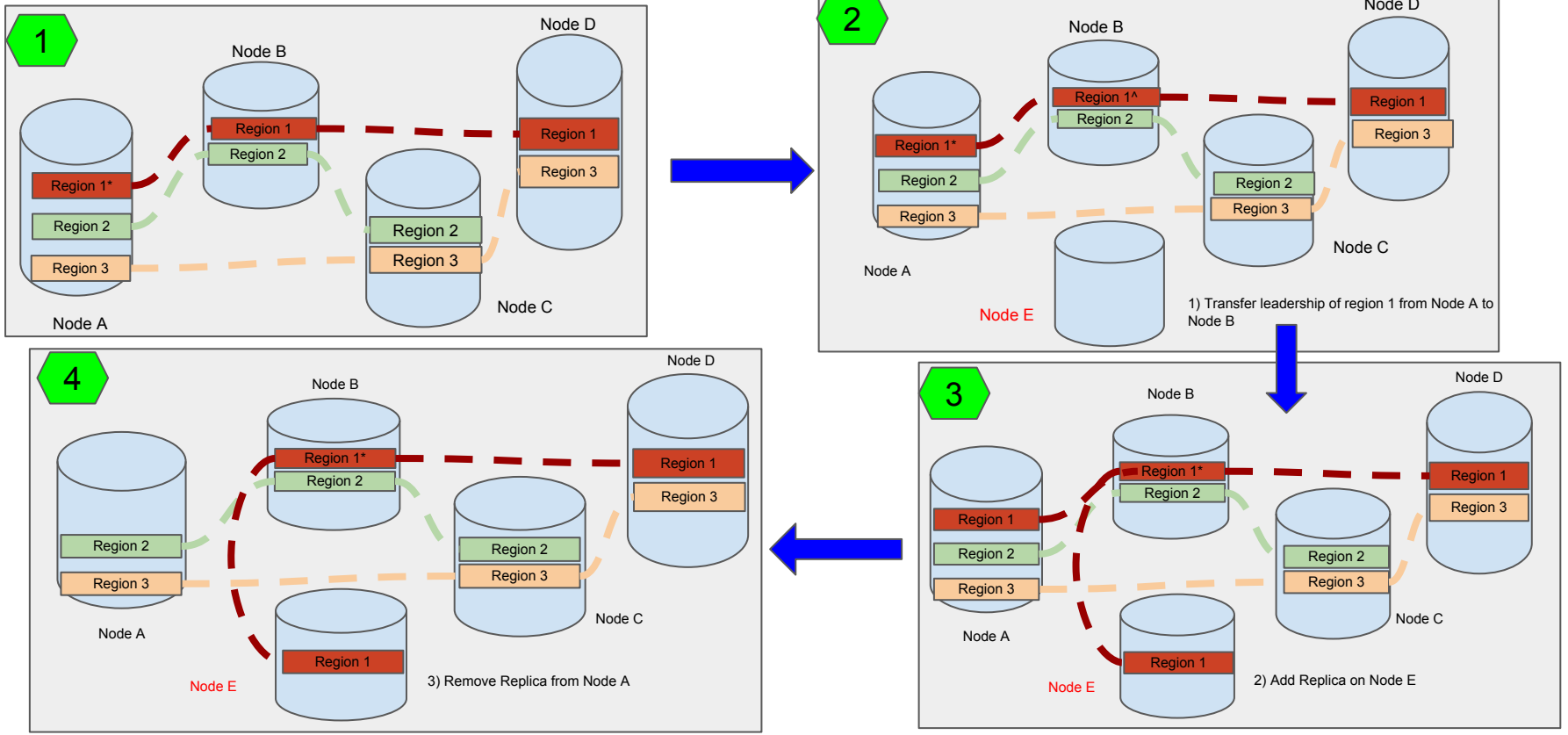
PD 调度 -- 多副本管理

- 使用多副本保证数据安全（Data safety）
- 维持数据副本数
  - 副本数不足：AddPeer
  - 副本数过多：RemovePeer
- 优化数据的地理位置分布
  - tikv-server 按照拓扑结构打上多级 labels
  - PD 根据拓扑结构移动数据（AddPeer+RemovePeer），使得多个副本尽可能隔离

PD 调度 -- Region 调度

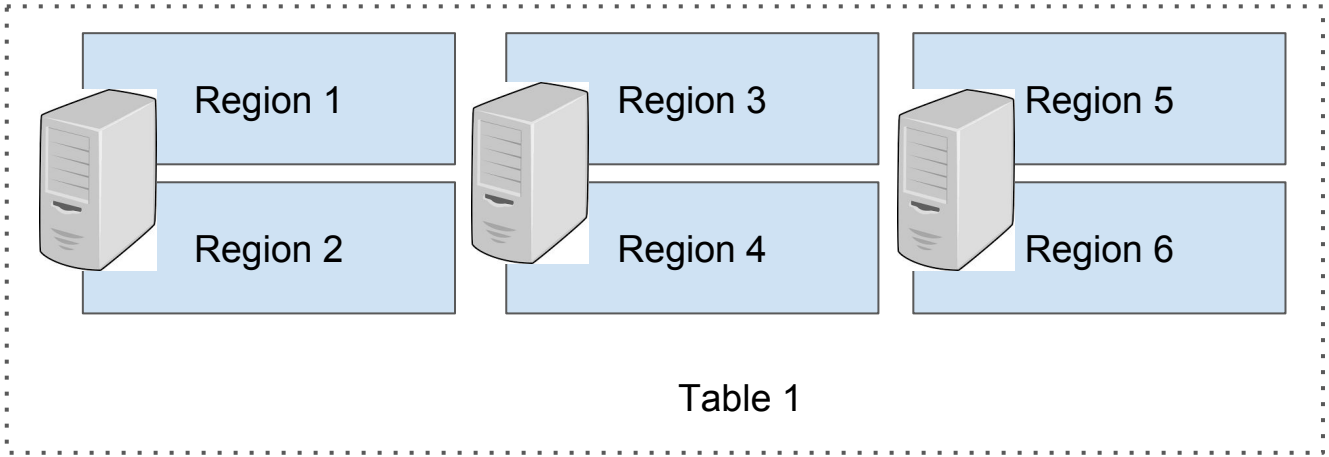


PD 调度 -- TiKV扩容


1
Node A
Node B
Node C
Node D
Region 1*
Region 2
Region 3
Region 1
Region 2
Region 3
Region 1
Region 3
2
Node A
Node B
Node C
Node D
Node E
Region 1*
Region 2
Region 3
Region 1^
Region 2
Region 2
Region 3
Region 1
Region 3
1) Transfer leadership of region 1 from Node A to Node B
3
Node A
Node B
Node C
Node D
Node E
Region 1
Region 2
Region 3
Region 1*
Region 2
Region 2
Region 3
Region 1
Region 3
Region 1
2) Add Replica on Node E
4
Node A
Node B
Node C
Node D
Node E
Region 2
Region 3
Region 1*
Region 2
Region 2
Region 3
Region 1
Region 3
Region 1
3) Remove Replica from Node A

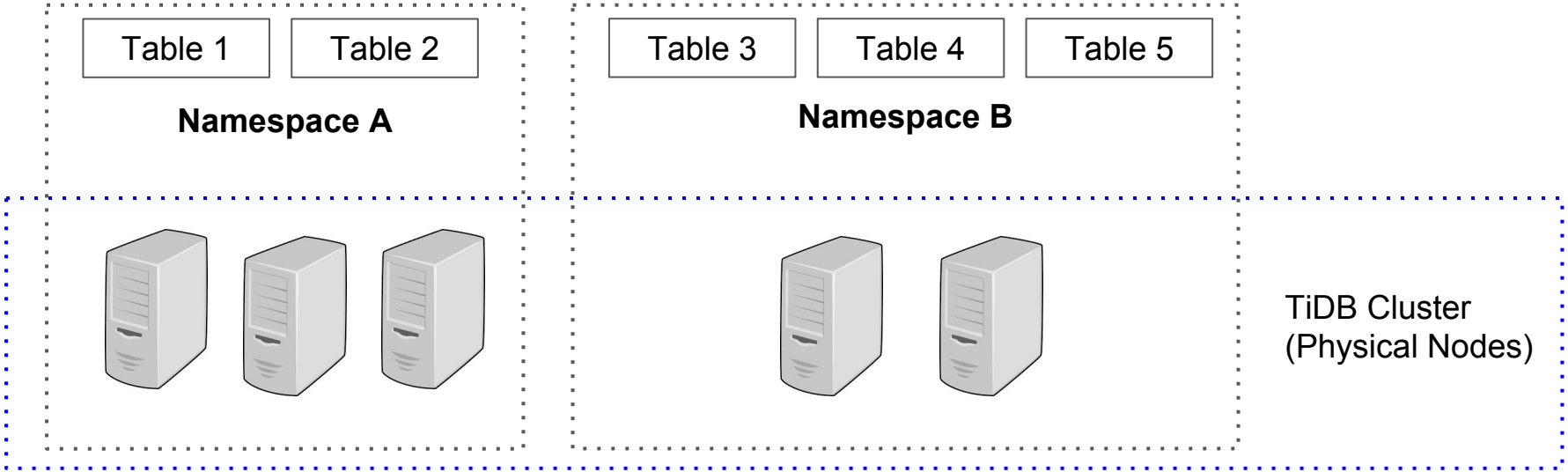
PD 调度 -- Region Hotspot 调度

- Pre-scheduling
- 动态迁移 Hot region



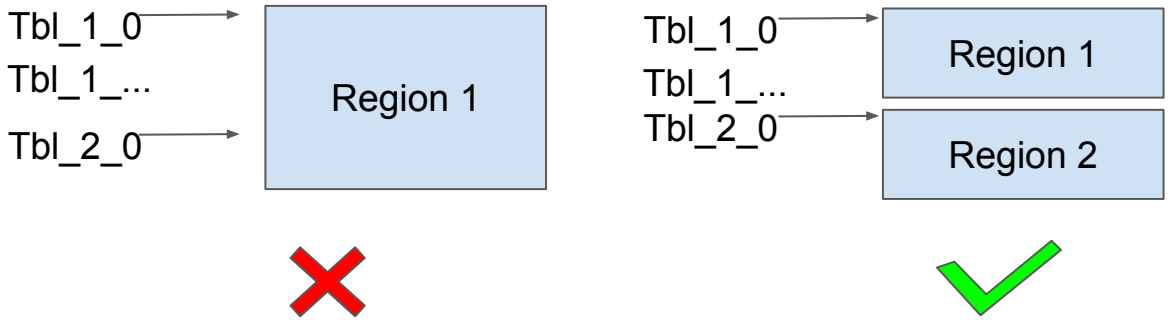
PD 调度 -- Namespace 管理与调度

在 PD 进行资源调度的过程中，引入 Namespace 作为管理对象，将不同的 store 和 region 划分到不同的 namespace 下，在调度时分别针对不同的 namespace 依次来做调度。



PD Namespace 设计要点

- 1 region 所存储的数据只能来源于 1 个表
  - 小于一个 region 的大小时使用一个 region；
  - 大于一个 region 的大小时进行分裂；
- 不同的物理节点分配不同的 workload
- 资源隔离
- 实现多租户的基础



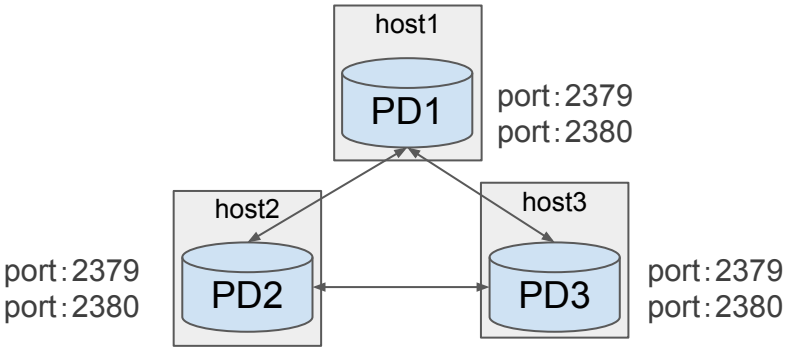
路由

因为 PD 保存了所有 TiKV 的集群信息，自然对 client 提供了路由的功能。假设 client 要对 key 写入一个值。

1. client 先从 PD 获取 key 属于哪一个 region，PD 将这个 region 相关的元信息返回。
2. client 自己 cache，这样就不需要每次都从 PD 获取。然后直接给 region 的 leader peer 发送命令。
3. 有可能 region 的 leader 已经漂移到其他 peer，TiKV 会返回 NotLeader 错误，并带上新的 leader 的地址，client 在 cache 里面更新，并重新向新的 leader 发送请求。
4. 也有可能 region 的 version 已经变化，譬如 split 了，这时候，key 可能已经落入了新的 region 上面，client 会收到 StaleCommand 的错误，于是重新从 PD 获取，进入状态 1。

PD 初始化与扩容

PD 通过集成了 ETCD，使用 Raft 实现了PD 集群的高可用。因此，PD 实例通常是单数个进行部署。通常情况下，我们需要启动至少三个PD 实例，数据有三份备份，才能保证数据的安全。现阶段 PD 有集群启动方式，initial-cluster 的静态方式以及 join 的动态方式。



1. 对于静态初始化，我们直接在三个 PD 启动的时候，启动参数中配置 initial-cluster 设置为 :pd1=http://host1:2380,pd2=http://host2:2380,pd3=http://host3:2380。

2. 对于动态初始化，假设我们先启动 pd1，成功后，然后扩容启动 pd2，加入到 pd1 的集群里面，启动参数 join 设置为 pd1=http://host1:2379；然后启动 pd3，加入到 pd1，pd2 形成的集群里面，join 设置为 pd1=http://host1:2379。

注意：要求三个 PD 实例分别部署在三台不同机柜的主机设备中，并保证三台主机之间的时间一致。

PD Lead 选举

当 PD 启动之后，我们就需要选出一个PD leader 对外提供服务。虽然 etcd 自身也有 Raft leader，但我们还是觉得使用自己的 leader，也就是 PD 的 leader 跟 etcd 自己的 leader 是不一样的。当 PD 启动之后，Leader 的选举如下：

1. 检查当前集群是不是有leader，如果有 leader，就 watch 这个 leader，只要发现 leader 掉了，就重新开始1。
2. 如果没有 leader，开始 campaign 活动。首先创建一个 Lessor，并且通过 etcd 的事务机制写入相关信息，如果leader key 的 CreateRevision 为 0，表明其他 PD 还没有写入，那么我就可以将我自己的leader 相关信息写入，同时会带上一个 Lease。如果事务执行失败，表明其他的 PD 已经成为了 leader，那么就重新回到1。
3. 写入成功成为 leader 之后，我们对定期进行保活处理。当 PD 崩溃，原先写入的 leader key 会因为 lease 到期而自动删除，这样其他的 PD 就能 watch 到，重新开始选举。
4. 成为 leader 之后，立即初始化 Raft cluster，主要是从 etcd 里面重新载入集群的元信息，拿到最新的TSO 信息。
5. 所有做完之后，开始定期更新TSO，监听 lessor 是否过期，并在过期之前进行续签 lessor。

PD 安装与升级

PD 的安装与升级建议使用 ansible 方式进行，详细请参考 [TiDB Ansible 部署方案](#) 指南。

PD 设备要求

生产环境

组件	CPU	内存	硬盘类型	硬盘数量	单块硬盘大小	网络	实例数量(最低要求)
TiDB	32核+	128 GB+	SSD	最低2块	500 GB+	2块+ 万兆网卡	2
PD	16核+	32 GB+	SSD	最低2块	200 GB+	2块+ 万兆网卡	3
TiKV	32核+	128 GB+	SSD	最低2块	200~500 GB	2块+ 万兆网卡	3
监控	16核+	32 GB+	SAS	最低4块	200 GB+	2块+ 千兆网卡	1
						服务器总计	9

注意：

1. PD 部署至少三个实例；
2. 要求 三个 PD 实例分别部署在三台不同机柜的主机设备中；
3. 保证安装 PD 的三台主机之间的时间一致。

PD 迁移

在 PD 运维过程中，由于 PD 是整个集群的心脏和大脑，在迁移过程中需要遵循一定的规则，保障 PD 可以在线进行迁移，使用 ansible 迁移步骤有：

1. 先修改 inventory，新增 PD 节点实例 IP；
2. 使用-l ips 方法单独 bootstrap 和 deploy 这些新的 PD；
3. 单独手工修改新增 pd 的启动参数，将 initial-cluster 模式修改为 join 模式，选择 pd leader 作为 join 对象并启动 pd；
4. 所有新增加 PD 启动成功后，将 PD leader 手工切换到新节点上；切换成功后，可以逐一删除原来的 PD member；
5. 修改 inventory，rolling update tikv 和 tidb，将 tikv 和 tidb 使用的 PD 对象切换到新加的 PD 上；
6. 检查 pd 状态，检查 pd member 与 leader 正常。

PD 运维 -- pd-ctl

PD Control 是 PD 的命令行工具，用于获取集群状态信息和调整集群参数，store 管理和进行 Region 调度。

pd-ctl 支持单命令模式和交互模式两种进行操作和管理 pd 集群。例如：`./pd-ctl -u http://127.0.0.1:2379`

pd-ctl 操作命令有：

- store [delete] <store_id>：用于显示 store 信息或者删除指定 store。
- region <region_id>：用于显示 region 信息。
- region key [--format=raw|pb|proto|protobuf] <key>：用于查询某个 key 在哪个 region 上，支持 raw 和 protobuf 格式。
- member [leader | delete]：用于显示 PD 成员信息或删除指定成员。
- config [show | set <option> <value>]：用于显示或调整配置信息。
- operator [show | add | remove]：用于显示和控制调度操作。
- scheduler [show | add | remove]：用于显示和控制调度策略。
- hot [read | write | store]：用于显示集群热点信息。

PD 运维 -- curl 查询 pd 状态

PD 对外提供了 HTTP json 访问接口，管理员可以通过网页方式或者 curl 方式直接像 pd-ctl 一样获取到 pd 的信息，方便用户查看 pd 状态和数据。

- 查看 stores 信息：curl http://pdip:2379/pd/api/v1/stores
- 查看 members 信息：curl http://pdip:2379/pd/api/v1/members
- 查看 pd leader 信息：curl http://pdip:2379/pd/api/v1/leader
- 查看 config 信息：curl http://pdip:2379/pd/api/v1/config
- 查看 regions 信息：curl http://pdip:2379/pd/api/v1/regions（注意：由于生产环境 region 比较多，输出数据大，慎用！！！）

高级功能

- 在线设置 Store label
  - store label <store_id> <label_key> <label_value>
- 改写 Store 状态，可以用于恢复错误状态，取消下线等
  - curl -X POST http://127.0.0.1:2379/pd/api/v1/store/{store_id}/state?state=Offline
- 设置 Store weight
  - store weight <leader_weight> <region_weight>
  - 默认权重都是1，0.5 表示数量为其他 store 的一半，2 表示数量为其他 store 的两倍

PD 监控

- pd role: 当前 pd 的状态
- store capacity: 集群总存储空间大小
- current storage size: 当前使用的存储空间大小
- number of region: 集群总 region 数量
- store status: 集群中 tikv 的状态信息
- schedule operators count with state: 单位时间内不同调度类型的数量
- hot region’s leader distribution: 热点 region leader 的分布情况
- hot region’s leader write bytes: 单位时间内热点 region leader 的写入量
- balance leader schedule: 单位时间内调度 region leader 的数量
- completed commands rate: 单位时间内，pd 完成命令的数量
- average completed_cmds_duration_seconds: pd 完成命令所需的时间
- etcd disk wal fsync rate: etcd 预写入磁盘的数量
- handle_txns_count: 单位时间内，pd 与 client 的通讯量
- average handle_txns_duration_seconds: pd 与 client 通讯的平均使用时间
- average wal_fsync_duration_seconds: etcd 预写入磁盘的平均使用时间
- handle_requests_count: 单位时间内，pd 向 tidb 返回请求结果的数量
- handle_requests_duration_seconds: pd 向 tidb 返回请求结果的平均时间
- region heartbeat: 单位时间内，tikv 向 pd 发送心跳的数量

PD监控中各种 Store 状态

- Up（正常）
  - 运行中，没有下线操作，并且 lastHeartbeat < 1min
- Disconnect Store（失去联系）
  - lastHeartbeat > 1min 后，pd-ctl 显示为 Disconnect 状态
- LowSpace Store（空间不足）
  - 运行中，store 汇报的使用量占用capacity 的80%以上，显示为 LowSpace store
- Down（短期故障）
  - lastHeartbeat > 30分钟（可配置）后，PD 开始为此 store 上的数据增加副本
- Offline（判了死刑）
  - 执行 store delete 下线操作后的状态
  - 此时 PD 开始搬数据，但 tikv 依然提供服务
- Tombstone（处决）
  - 搬完所有数据后，PD 自动将 store 置为此状态
  - address 被释放，可以在同样的地址新启 tikv

PD 默认参数列表

编号	定义内容	定义值	描述	是否可以调
1	lease	3秒	租约时间，如果 pd leader 不卡或者网络不卡，leader 在这个时间内会续约	不建议
2	tso-save-interval	3秒	ts预分配的时间范围。比如现在是12:00:00，ts的分配方式是先入12:00:03，成功后就可以自由地分配12:00:03之前的ts了。如果 leader 切换之后，新的leader只能分配12:00:03之后的ts。这个如果配太长的话，leader切换后新节点会有一段时间不能分配ts反之如果配太短，leader卡一下就可能影响ts分配了	不建议
3	max-size	300	日志最大大小，单位 MB	可以
4	max-days	28	日志存放时间，单位 天	可以
5	interval	15秒	向 pushgateway 汇报的时间间隔	可以
6	max-snapshot-count	3	region 副本数量	可以
7	max-store-down-time	1h	store 在断开disconnted 1个小时后，状态变为 down	可以
8	leader-schedule-limit	64	leader 调度相关，保证 leader 均匀分布	可以
9	region-schedule-limit	16	region 调度相关，保证数据均匀分布	可以
10	replica-schedule-limit	24	replica 调度相关，越大生成副本速度越快	可以
11	max-replicas	3	副本个数	可以
12	conf_ver	1	新增或者删除了 peer，conf_ver 会加 1	不可以
13	version	1	如果 region 发生了 split 或者 merge，则 version 加 1	不可以
14	Store Disconnted	60秒	store 在lastheartbeat大于1分钟后	不可以
15	LowSpace Store	20% * cap	store 剩余空间小于 capacity 的20%	

PD 调度重要参数

通过调整 leader-schedule-limit 可以控制同时进行 leader 调度的任务个数。这个值主要影响 *leader balance* 的速度，值越大调度得越快，设置为 0 则关闭调度。Leader 调度的开销较小，需要的时候可以适当调大。

```
>> config set leader-schedule-limit 4      // 最多同时进行 4 个 leader 调度
```

通过调整 region-schedule-limit 可以控制同时进行 region 调度的任务个数。这个值主要影响 *region balance* 的速度，值越大调度得越快，设置为 0 则关闭调度。Region 调度的开销较大，所以这个值不宜调得太大。

```
>> config set region-schedule-limit 2      // 最多同时进行 2 个 region 调度
```

通过调整 replica-schedule-limit 可以控制同时进行 replica 调度的任务个数。这个值主要影响节点挂掉或者下线的时候进行调度的速度，值越大调度得越快，设置为 0 则关闭调度。Replica 调度的开销较大，所以这个值不宜调得太大。

```
>> config set replica-schedule-limit 4     // 最多同时进行 4 个 replica 调度
```

```
>> config show
{
  "max-snapshot-count": 3,
  "max-store-down-time": "1h",
  "leader-schedule-limit": 8,
  "region-schedule-limit": 4,
  "replica-schedule-limit": 8,
}
```

PD 常见问题：leader 频繁切换或者选不出来 leader

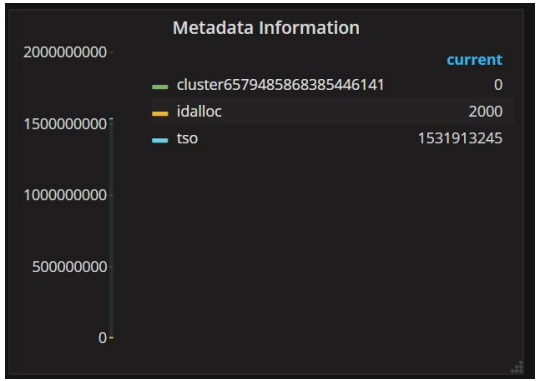
- 检查网络问题
- 检查 etcd metrics
  - Txn duration
  - wal sync duration
- PD 负载高
  - 和 tikv 或其他应用混部了
  - region heartbeat 过多

PD 常见问题：get timestamp too slow

- PD 卡了
  - PD metrics 页面 handle_request_duraion
- TiDB -> PD 的网络卡了
  - TiDB metrics 页面 pd_request_duration
- TiDB 卡了
  - TiDB metrics 页面 pd_command_duration

Recover 工具

- 用于在整个 PD 集群完全不可用了以后恢复数据
- 停掉所有 tikv，启动新的 PD
- 运行 pd-recover，需要提供如下参数：
  - clusterID：从 tikv 日志里能找到
  - allocID：从 pd 日志里能找到分配 ID 记录，如果没有 PD 日志了，那么可以检查从 tikv 的日志中检查所有 ID，然后估计一个合理的值
- 这两个参数在 grafana 上也能找到



PingCAP

Thanks

Q&A

https://github.com/pingcap/tidb

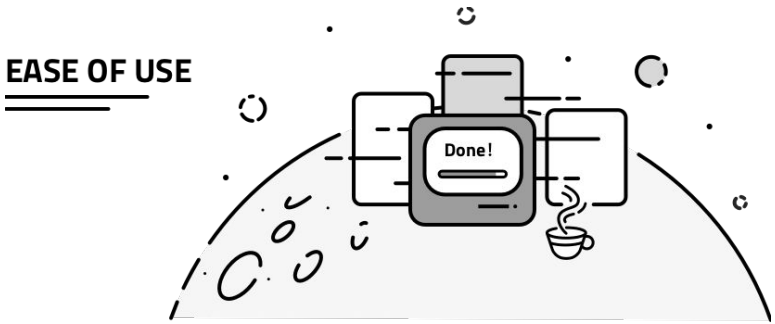
https://github.com/pingcap/tikv

https://github.com/pingcap/pd

https://github.com/pingcap/tispark

https://github.com/pingcap/docs

https://github.com/pingcap/docs-cn



PingCAP

Thanks!

Contact me:

info@pingcap.com
www.pingcap.com