

Tiky -

持久化存储 = Key-Value，有序遍历

按照二进制顺序排序

保存到 RocksDB (单机)

数据复制： raft.

数据分区： region

split size.

一个 range 64 MB out.

① 保证每个 node & range 对应关系不变

- 一个 region 在数据只在一个节点上

② 存储容量全局水平扩展

③ 读取均衡。

用一个组件记录 region 在节点上的分布。

②. 在 region 为单位对 raft 进行复制和管理

每个 region 有多个 replica.

replica 分布到不同 node. 构成 raft group  
raft 保证一致性.

MVCC: 多版本并发修改. 使用版本控制. 在 key from  
Version

key - version  $\rightarrow$  value

Transaction: transaction 执行过程中检测写写冲突.  
Submit 时检测,  
写入冲突严重时拒绝.

计数.

Relational  $\rightarrow$  key-value in 哪里:

表元信息

table 中 in how  
index.

~~A~~ 表的元信息 : — .

~~A~~ table 中的 row: OLTP 风格. 行存 .

二级索引 .

由 table -> table ID . 由 index -> index ID .

由 row -> row ID (如有唯一索引 primary key 则用 primary key) .

table ID 前缀部分 -> index ID . row ID 后缀部分 -> int 64 .

① key-value pair :

key : table prefix { table ID } -> second prefix sep { row ID }

value : { col1, col2 ... }

② index :

key : table prefix { table ID } -> index prefix sep { index ID } ->  
index Column's Value .

value : row ID

Unique Index

key : table prefix { table ID } -> index prefix sep { index ID } ->  
index Column's Value  
— row ID .

value : null .

Unique index .

\* 其中 xx prefix 的定义：

Var {

table prefix = [ ] bytes { }

second prefix sep = [ ] bytes { ; - ; }

index prefix sep = [ ] bytes { ; ; }

保证编码前缀编码后比较关系不变（前缀相同）

Memcomparable

\* 元信息管理：

为每个 Database base & Table 分配唯一一个 ID (UUID)

编码为 key-value

调度

副本数量、副本分布、新加节点、节点下线。

↳ 3个 basic operation:

Push replica Add replica. leader + transfer -

Add Replica Remove Replica Transfer Leader

\* 信息收集

① TiKV 节点与 PD 之间有心跳包，PD 检测 store 存活以及是否有新加入。

心跳包信息：总磁盘容量、可用容量、region 数量、数据写入速度  
发送 / 接收 snapshot 条数、是否过期。  
标签信息。

② 每个 raft group 与 leader 与 PD 之间的心跳包：  
汇报 region 状态 → 每次发心跳包判断一下是否满足  
Leader 位置、follower 位置、样线 replica 个数 - 副本位置  
数据写入/读取速度

## ★ 调度策略

① 一个 region 在 replica 数量正确：

通过 region leader 的心跳包发现 replica 数量不满足。

原因：① 某 node 样线，上面的数据全部丢失。

② 某样线节点恢复服务，之前剩余的 replica 数量过多。

③ 管理员修改 max-replicas 配置。

② 一个raft group 有多个 replica 不在同一位置 不是同一节点  
多节点在同一台物理机器上。?

Tikv 节点分布在多个机器上。

Tikv 节点分布在多个 JDC 中。

本质上都是共同位置属性

③ replica 在 store 之间分布均匀。

④ Leader 数量在 store 之间均匀分配 (这里有变动?)

⑤ hot spots 在 store 之间分布均匀。

⑥ 各个 store 存储空间大致相同。

⑦ 调度速度控制。

⑧ 手动干预。

PD. (placement driver)

PD 是 TiDB 里面全局中心总控节点。

支持 auto-failover。

\* 初始化：至少 3-replicas

PD 动作：集群启动方式、initial-cluster 和静态方式  
join 的动态方式

etcd 端口

2379 - 处理外部请求.

2380 : etcd peer 之间相互通信

静态

→ 互斥

etcd 自身启动只能使用一种方式初始化.  
初始化方式只需要用  
内部端口.

动态 join

因为这里调用 etcd  
的命令，需要发送到  
etcd 进行，故使用外部  
端口。

## ☆ 选举

PD 从 leader 和 etcd 自己的 leader 不一样.

选举流程:

- ① 检查当前集群是否有 leader - 有 leader, PD watch 这个 leader.  
leader 掉了就重新开始！
- ② 没 leader 开始 campaign - 创建一个 lessor, 通过 etcd  
事务机制写入相关信息.
- ③ 成为 leader 后 - 定期保活处理.

PD 脱离时, 原先写入的 leader key 会因 lease 到期而自动删除.  
其他 PD watch 到, 重新选举.

- ④ 初始化 raft cluster, 主要从 etcd 里重新载入集群元信息.  
拿到最新 TSO 信息.
- ⑤ 定期更新 TSO - 监听 lessor 生效过期以及外部变更通知.

\* TSO：一个全局时间戳  $\rightarrow$  TiDB分布式事务的基础。  
需要保证PD可以快速分配TSO，且TSO一定是单调递增的。

TSO：64位整型，physical time + logical time。

当前 Unix time (ms)       $1 << 18$  bits  $\approx 2^{18}$  ms

过程：

- ① PD成为Leader后，从etcd上获取上次保存的时间。  
如果比本地时间大，则等待直到当前时间大于这个值。  
② PD可以分配TSO后，先向etcd申请一个最大时间。

当前时间  $t_1 + t_{max}$ ，之后PD可以在内存中使用这一段时间窗口。

当前时间大于  $t_1 + t_{max}$  后，更新为  $t_2 + t_{max}$

③ 外部请求TSO时，PD可直接计算并返回（在内存中）。

④ Client会批量请求TSO，提高性能。

# PD 调度

在 PD 主要对两种资源进行调度。storage 和 leader

\* 关键 interface 和 structure

Scheduler - 调度资源的接口

```
// Scheduler is an interface to schedule resources.  
type Scheduler interface {  
    ① GetName() string  
    ② GetResourceKind() ResourceKind  
    ③ Schedule(cluster *clusterInfo) Operator  
}
```

- ① 不同 scheduler 不能重名。
- ② 返回 scheduler 处理的资源类型。(Leader, storage)
- ③ Schedule 中生成实际调度 operator。

Operator 对一个 region 进行调度

```
// Operator is an interface to schedule region.  
type Operator interface {  
    ① GetRegionID() uint64  
    ② GetResourceKind() ResourceKind  
    ③ Do(region *regionInfo) (*pdpb.RegionHeartbeatResponse, bool)  
}
```

- ① 得到需要调度的 region ID

- ② DO 执行实际操作 - 返回一个 heart beat。

多个 operator 可以组成更上层的 operator，但是有相同 two resource kind  
即 不能在同一组 operator 中 操控不同 resource

## Selector / Filter . 选择合适的 source 和 target .

```
// Selector is an interface to select source and target store to schedule.  
type Selector interface {  
    SelectSource(stores []*storeInfo, filters ...Filter) *storeInfo  
    SelectTarget(stores []*storeInfo, filters ...Filter) *storeInfo  
}
```

```
// Filter is an interface to filter source and target store.  
type Filter interface {  
    // Return true if the store should not be used as a source store.  
    FilterSource(store *storeInfo) bool  
    // Return true if the store should not be used as a target store.  
    FilterTarget(store *storeInfo) bool  
}
```

filter 返回 true . 则不能选择这个 store .

## Controller . 控制调度速度 .

```
// Controller is an interface to control the speed of different schedulers.  
type Controller interface {  
    Ctx() context.Context  
    Stop()  
    ① GetInterval() time.Duration  
    ② AllowSchedule() bool
```

① 返回调度间隔时间

② 是否允许调度 .

## Coordinator . 管理所有 Scheduler 和 controller .

```
// ScheduleController combines Scheduler with Controller.  
type ScheduleController struct {  
    Scheduler  
    Controller  
}
```

在 region heart beat 时 . 会看这个 region 生否需要被调度 .

还有一个 replica Check Controller 定期检查 region 生否需要调度

# PD 源码.

LR coordinator.go 为 run 打始调度

## Core

region.go :

regioninfo :

Learners ?

Voters. 选举时的选举者 voter 的 peer .

Leader . Leader mode

Down Peers : 下线的 peer

Pending Peers : 待挂的 peer

Written Bytes ; 已经写入的 bytes

ReadBytes ; 读的 bytes

Approximate Size : 大小估计 .

- 这里 size 的单位用 MB ( < 20 )

healthregion : 检查 down peers .

pending peers . learners & tips

0. 如果是则返回 true .

region 是健康的 .

Get off Followers . 返回不在同一个 store 的  
followers .

RegionMap :

→ RegionEntry set map .

ids . id in 数组 . (regionID) .

totalSize .

regionEntry .

regionInfo + 位置

regionMap 删掉一个条目 . 是把最后一个移到当前被删  
条目位置上 .

regionsInfo :

regionTree .

regionMap .

leaders .

followers .

learners .

PendingPeers .

带星号 uint64 in regionMapForMap .

storeID → regionID

→ regionInfo .

region stats - Observe.

一个 region statistics 涵盖多个 region stats .

diff Region Peers Info :

从两个 peers 的 regionInfo 对比分析 .

Coordinator.go : 管理所有 controller for scheduler

line 64, 65. operator, scheduler

operator is key. uint64.

scheduler is key. string (不能重名.)

\* With cancel 通常 返回继承的 context 对象. 可以比它们的 context 更早取消.

当返回的 cancel 被调用 或从 context 的 done channel 被关闭时这个 context 的 done channel 也关闭.

\* Dispatch - 检查 operator .

\* patrol regions .

为什么 scanRegion 可以将它的 key 放进去.

scanRegion (Cache.go) & scanRange (region.go).

→ scanRange (region-tree.go)

没有看到处理空 key 的地方. . . . .

调度间隔等信息存在 schedule Cfg 中。

Schedule-by-namespace 生成一个调度同  
namespace 中 stores 的 operators.

将 operator 放到 region 是否存在， region  
中 operator 在 target 是否匹配，需要将 this operator  
是否比 region 已有 operator 有更高优先级，将为零 replace

## balance - leader.go

balance - leader 调度时 source 和 target 分别到  
分数最高和分数最低的 leader.

Schedule 为什么 transfer Leader In 和  
transfer Leader out 分别进圆 ?

两种操作. transfer Leader Out 是把 Leader 移给  
随便一个 target. In 在 target 找一个 follower 为 Leader 并  
在 in store.

并不是 source 和 target 之间传递 Leader.

## store.go

为什么计算 score 为 leader 的 resource score ?

用  $(\text{leadersize} + \text{delta}) / \max(\text{leader weight}, \text{min weight})$ ,  
而 regionScore 等于  $\text{available} / \text{used capacity}$  ?

region Score :

$\text{avail} - \frac{\text{delta}}{\text{amp}} > (\text{high space}) \cdot \text{capa} : \text{region size} + \text{delta}$

$\text{avail} - \frac{\text{delta}}{\text{amp}} < (\text{low space}) \cdot \text{capa} : \text{max score} - (\text{avail} - \frac{\text{delta}}{\text{amp}})$

为什么先平衡再判断是否可操作？不浪费吗？  
(create operator 是)

is Registration. should balance  
这个是 ok 的.



balance region 中就先进行了判断.

## BalanceRegion.go

① Schedule 里面为什么要当 stone fix peer. 会话一起吗

一起吗 ✓.

balance - leader.  $\Rightarrow$  调度该 region .

balance - region 通过不放 in region

Replicabase score len(labels) - index - 1 .

## hot-region.go

hot-region - hot-write-region - hot-read-region -

hot-write-region 和 hot-read-region 互斥

stores/statistics:

readStatAsLeader

writeStatAsPeer

writeStatAsLeader

为什么 balance hot Read Region 不平衡资源，而

balance hot WriteRegion 才有资源，可能是读写性能差异？

balance Hot Read Region

没必要为平衡读分配太多空闲

先 balance by leader 然后才 balance by peer - 完成

都先完成 skip 之后 balance

# Operator.go

StoneInfluence :

RegionSize, RegionCount, LeaderSize, LeaderCount  
(*在 basic-cluster.go*)

每一种调度操作都要计算 influence (针对后端)  
和是否完成 (isFinish)

pears -> stores

OperatorStep

Basic scheduling steps that can't be subdivided

{  
fmt.Stringer  
IsFinish  
Influence

Check 检查当前 step 是否已完成，返回下一个要执行  
bad step.

leader.go

Concurrent 的用法还是不清楚。

在 execLeaderLoop 及 leaderLoop 中关于  
etcd Leader 部分不懂。

A schedule is

balance-region .

balance-leader .

erect-leader ,

grant-leader .

table .

random-region

shuffle-leader

shuffle-region

# Pipeline

Struct:

transport.

dialtimeout, maximumduration - ,

TLS info

ID.

url.

clusterID.

SnapShutter -

ServerStat.

LeaderStat

ErrorC

walpicker { max  
wals  
picked

status

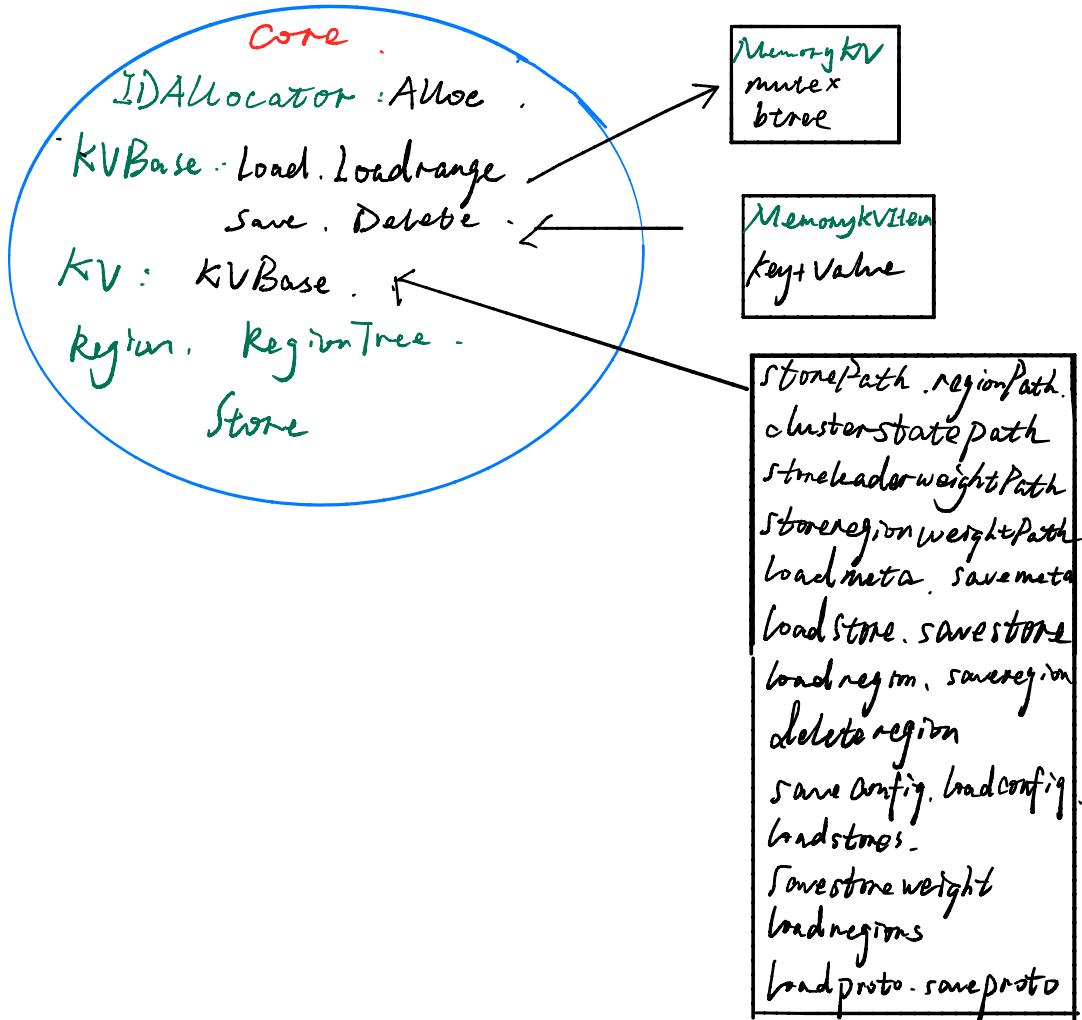
Raft { Process  
IsIDRemoved.  
ReportUnreachable.  
ReportSnapshot -

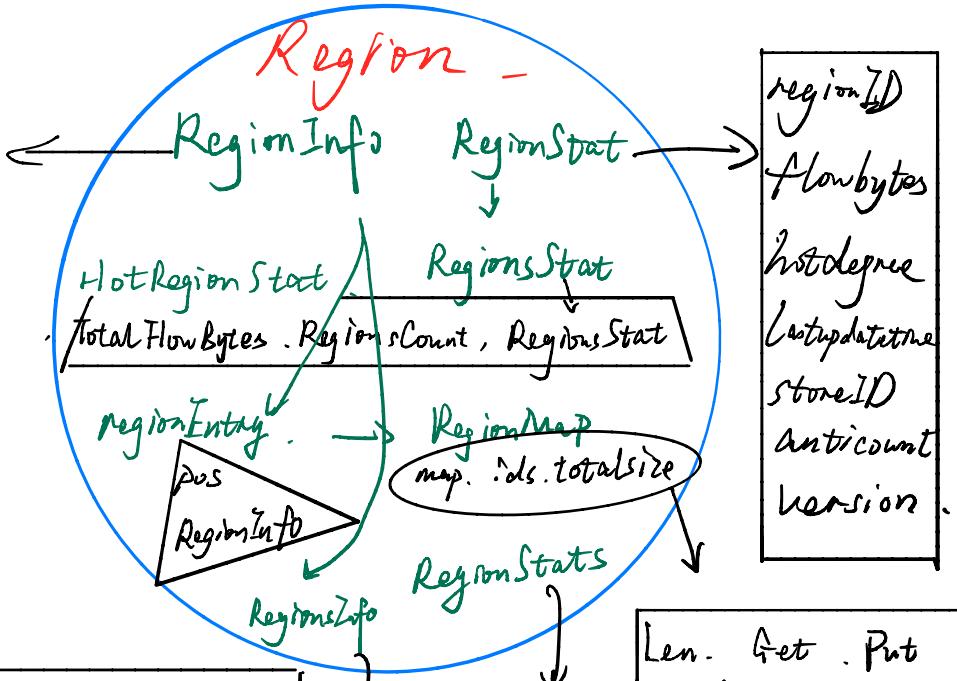
start .

stop

handle

post



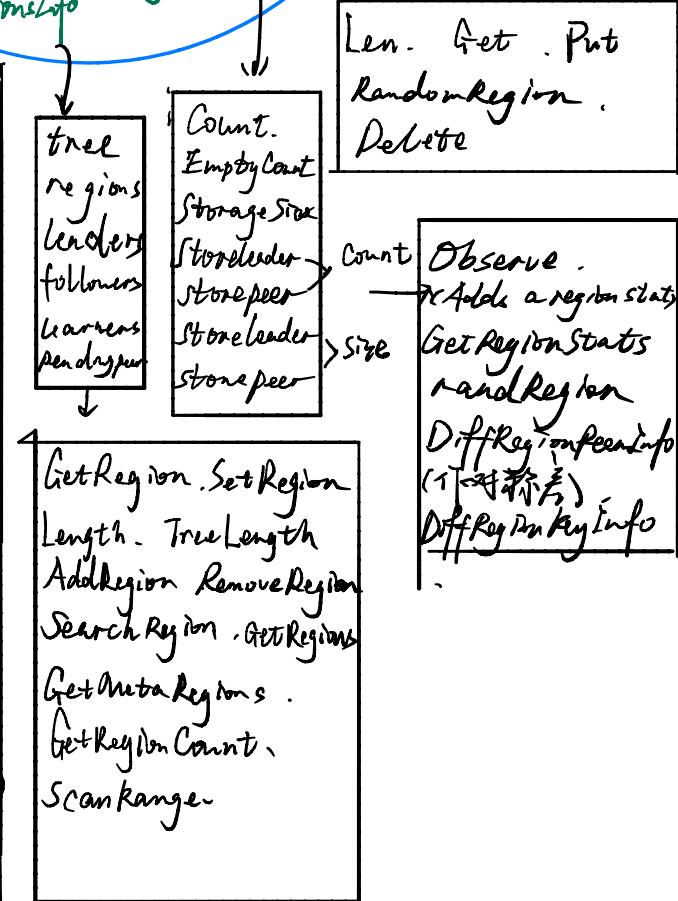


learners  
 voters.  
 Leader  
 Down Peers  
 Pending Peers  
 Written bytes  
 Read Bytes  
 Approximate size

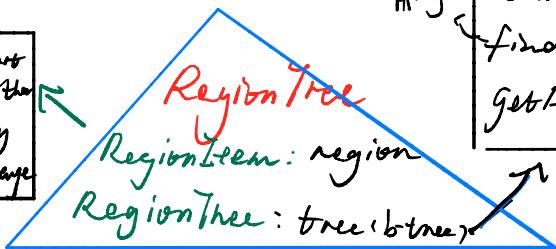
regionID  
 flowbytes  
 indegree  
 lastupdateime  
 storeID  
 anticount  
 version.

NewRegionInfo  
 Classify learner and voter  
 RegionFromHeartbeat  
 Clone (copy of current ri)  
 Get learners/voters/Peer/  
 DownPeer/DownVoter/  
 DownLearner/Pending.../  
 storepeer...

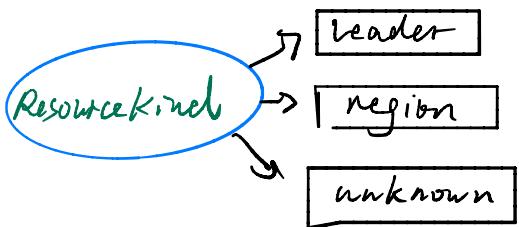
Remove storepeer.  
 Add peer. Get storeID(peer)  
 GetFollowers, / Follower (random)  
 GetDiffFollowers (different from  
 all the followers of another region)



<sup>4</sup>  
Less if the start key is less than other  
Contains: if key contained in the range



Region  
length, update,  
remove, search -> A key  
find, ScanRange,  
GetAdjacentRegions



# stone

metadb.store  
stats  
blocked  
leadercount  
RegionCount  
LeaderSize  
RegionSize  
PendingPeerCount  
LasthbTS  
LeaderWeight  
KeyBrokerage

StoneHotRegionInfos

stoneInfo

stonesInfo

Close . IsBlocked / Up / Offline / OnStone  
Block / Unblock (stop / continue balances)  
Downtime . LeaderScore . RegionScore  
StorageSize . AvailableRatio  
IsLowSpace . ResourceCount ResourceSize  
ResourceScore . ResourceWeight .  
GetStartTS . GetUpTime  
IsDisconnected / Unhealth -  
GetLableValue . CompareLcation  
MergeLabels

GetStone . SetStone

Block / Unblock Stone

GetStones . GetMetaStone

GetStoneCount . SetLeaderCount

SetRegionCount .

SetLeader / Region Size

TotalWrittenBytes .

TotalReadBytes .

GetStonesWriteStat .

GetStonesReadStat .

## Namespace .

Classifier :

GetAllNamespace.

GetStoreNamespace.

GetRegionNamespace .

IsNamespaceExist

AllowMerge .

RegisterClassifier -

CreateClassifier .

init

## NamespaceChecker .

cluster .

filters .

classifier -



Check : Verifies a regions namespace, return an operator .

SelectBestPeerToRelocate : Return a new peer to move region .

SelectBestStoreToRelocate : Randomly return the store to relocate  
isExists . . filter -

GetNamespaceStores : return the namespaces all stores .

BasicCluster

OpInfluence .

StoreInfluence

BasicCluster -

Stores. Regions Hot Cache

StoresInfluence

RegionsInfluence

GetStoresInfluence

GetRegionsInfluence

RegionSize

RegionCount .

LeaderSize

LeaderCount -

NewBasicCluster .

GetStores . GetStore . GetRegion

GetRegionStores . GetFollowersStores .

GetLeaderStore . GetAdjacentRegion -

BlockStore . UnblockStore RandFollowRegion -

RandLeaderRegion . IsRegionHot . RegionWriteStats

RegionReadStats . PutStore PutRegion

CheckWrite/Read Status

ResourceSize (Kind )

**Filter**. filter (interface) < FilterSource  
Filter Target

excludedfilter sources filters all specified stones  
targets

blockfilter filters all stones that are blocked from balance

stateFilter . not up.

healthFilter Busy or down.

disconnectFilter disconnected

PendingPeerCountFilter currently handling too many Pending Peers

snapshotCountFilter handly too many snapshots

cacheFilter . Cache all stones that are in the cache

storageThresholdFilter almost full

distinctScoreFilter labels stones ensures distinct score not decrease

nameSpaceFilter classifier namespace not belong to a specified namespace

rejectLeaderFilter . marked as reject leader from being  
the target of leader transfer



## Operator

- `operatorStep (interface)`

`transferLeader: fromStore, toStore`

`AddPeer: ToStore, PeerID`

`AddLearner: ToStore, PeerID`

`PromoteLearner: ToStore, PeerID`

`RemovePeer: FromStore`

`MergeRegion: FromRegion, ToRegion, IsPassive`

`SplitRegion, startKey, endKey`

- `OPERATOR` . `operatorHistory`

`desc, regionID,  
regionEpoch, kind, currentStep,  
steps, createTime, stepTime, Level`

`fmt, stringer,  
IsFinish  
Influence`

`FinishTime  
From  
To  
Kind`

`String, Desc, setDesc, AttachKind, RegionID, SetPriorityLevel,  
RegionEpoch, Kind, ElapsedTime, Len, Step, GetPriorityLevel,  
Check (check if current step is finished, returns next step to take action),  
IsFinish, IsTimeOut, History, CreateRemovePeerOperation -  
Influence (calculate the store difference which unfinished steps make),  
CreateNonePeerOperator - getRegionFolloworIDs, removePeerSteps,  
CreateMergeRegionOperator - matchPeerSteps  
GetIntersectionStores`

## Rangecluster :

cluster . regions , tolerantSizeRatio .

GenRangeCluster . updateStoneInfo . GetStone .

GetStones . SetTolerantSizeRatio , RandFollowerRegion

RandLeaderRegion .

GetRegionStones ( All stones that contain the region's peers )

GetFollowerStones . GetLeaderStone

## ReplicaChecker :

cluster . classifier . filters .

Check . verifies a region's replicas , creates an operator if need ,

SelectBestReplacementStone . SelectBestPeerToAddReplica .

SelectBestStoneToAddReplica . selectWorstPeer .

checkDownPeer . checkOfflinePeer . checkBestReplacement .

RegionScatter :-

Selected Stores : mutex . stores .

put . reset . newFiber .

(All stores) .

RegionScatteren : cluster . classifier - filter . Selected

Scatter . scatterRegion . selectPeerToReplace .

Collect Available Stores .

raft (3-replicas 方案)

对 store 在性能进行排序。

分  $\frac{2}{3}$  good nodes.  $\frac{1}{3}$  bad nodes.

replica 部署时 2个在 good 1个在 bad.

leader 在 good

leader 部署不均匀  
并发展？

Add Replica 只能访问

性能 to store ?

good 下载  $\rightarrow$  good.

bad 下载  $\rightarrow$  bad.

和均衡会创建副本时直连了。

Adjacent region for leader 不能在同一个

stone 也不能在两个 region in public stone.

对于 Leader - ' 初始化后 (随机数) , 把随机分  
布从 leader 移动到  $\frac{2}{3}$  的好节点上,  
通过调用 leader-weight ?

通过分 namespace 实现自动调度

把好、坏节点分到 2 个 namespace .

(先不考虑其他 namespace 间隔)

每个节点 计算成性能

初始化直接排序 - 写 store 的 info

接到 heartbeat - 调度时 都查一下 peers .  
满足 ~~而~~ 的一差 .

- ① 动态节点.
- ② 高负载下性能  $\rightarrow$  模型、
- ③ 迁移优先级 .
- ④ 增删节点 - 副本 .

