# 冷热分离详细设计

## 目标

支持所有doris功能，只是把部分数据放到对象存储上，以节省成本，不牺牲功能。

## 数据冷却

冷却数据进入S3的整个过程可以分为四个部分：策略、决策、执行、结果；

## 策略

可以依据数据进入之后的过期时间，比如数据进入几小时后。

#### 方式1：继续使用现有模式

目前可以在创建表时指定数据冷却时间：

SQL

```sql
CREATE TABLE example_db.table_hash
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT SUM DEFAULT "10"
)
UNIQUE KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2) BUCKETS 32
PROPERTIES(
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00"
);
```

也可以使用FE全局配置，这个参数对所有存储在SSD上的表生效

Nginx

```
storage_cooldown_second
```

通过alter table更改partition和表级别的storage_cooldown_time

Plain Text

```
alter table SET("storage_cooldown_time" = "2022-06-10 00:00:00")
alter table modify partition SET("storage_cooldown_time" = "2022-06-10 00:00:0
0")
```

Dynamic partition通过hot partition num来计算出storage cooldown time

冷却远程存储property可以应用到表和partition级别。

Nginx

```
remote_storage_cooldown_time
```

需要支持绝对时间和cooldown_second，remote和hdd使用不同的配置，也给用户带来了比较繁琐的使用界面。cooldown_second目前不支持不同的表有不同的配置，比如某些表不cooldown。

目前的远程存储创建为一种资源，

SQL

```
    CREATE RESOURCE "remote_s3"
    PROPERTIES
    (
    "type" = "s3",
    "s3_endpoint" = "http://bj.s3.com",
    "s3_region" = "bj",
    "s3_root_path" = "/path/to/root",
    "s3_access_key" = "bbb",
    "s3_secret_key" = "aaaa",
    "s3_max_connections" = "50",
    "s3_request_timeout_ms" = "3000",
    "s3_connection_timeout_ms" = "1000"
    );
```

Assembly language

```
// 表和partition增加2个属性:
remote_storage_resource: {resource_name}
remote_storage_time : {}

    CREATE TABLE example_db.table_hash
    (
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT SUM DEFAULT "10"
    )
    ENGINE=olap
    AGGREGATE KEY(k1, k2)
    DISTRIBUTED BY HASH (k1, k2) BUCKETS 32
    PROPERTIES(
    "storage_medium" = "SSD",
    "storage_cooldown_time" = "2015-06-04 00:00:00",
    "remote_storage_resource" = "remote_s3",
    "remote_storage_cooldown_time" = "2015-12-04 00:00:00"
    );
```

两个属性表示存储介质，storage_medium用来指定本地存储，remote_storage_resource用来指定远程存储。

**方式二：使用"storage policy的概念**

首先创建一个storage_policy，可以指定数据进入几小时后冷却，也可以指定绝对冷却时间，同时包含S3的属性，比如endpoint，bucket等。

SQL

```
CREATE RESOURCE "storage_policy_name"
PROPERTIES(
     "type"="storage_policy",
     "cooldown_datetime" = "2022-06-01", // time when data is transfter to mediu
m
     "cooldown_ttl" = 1h,  // data is transfter to medium after 1 hour
     "s3_*"
);
```

在表或者partition级别使用storage_policy

Plain Text

```
CREATE TABLE example_db.table_hash
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT SUM DEFAULT "10"
)
UNIQUE KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2) BUCKETS 32
PROPERTIES(
    "storage_medium" = "SSD",
    "storage_policy" = "storage_policy_name"
);
```

云上产品使用方式（社区兼容云上使用方式）：

1. 支持默认storage policy，云上产品控制面会先创建一个默认的storage policy，比如名字叫default_storage_policy，名字做成fe的conf配置，默认storage policy里边有所有S3的信息以及冷却时间。
2. 创建storage policy可以不带对象信息，只带cooldown_datetime和cooldown_ttl信息，不带对象信息的时候继承default_storage_policy里的对象信息。实现上不要直接copy，而是每次动态的通过查找default_storage_policy的方式更新，这样可以实现更改default_storage_policy的aksk之后，所有集成的都生效。
3. 展示ak，不展示sk，show的时候。

BE获取和更新策略，与获取和更新S3存储信息类似，第一期做成be定时同步，这样简单一些。

Plain Text

```
TabletMeta {
+   StoragePolicyName: storage_policy_name // can not alter
}

StoragePolicy {
    CooldownDateTime: //
    CooldownHourse: //
    S3_*:
}

// be向FE发起rpc
refreshStoragePolicy()
```

具体代码参考pr https://github.com/apache/incubator-doris/pull/8808/。

be侧，TabletMeta类增加storage_policy_name

```
Nginx
TabletMeta {
+    std::string storage_policy_name;
}
```

**社区实现的创建策略pr测试**

Pr: https://github.com/apache/incubator-doris/pull/9554/files

结论：

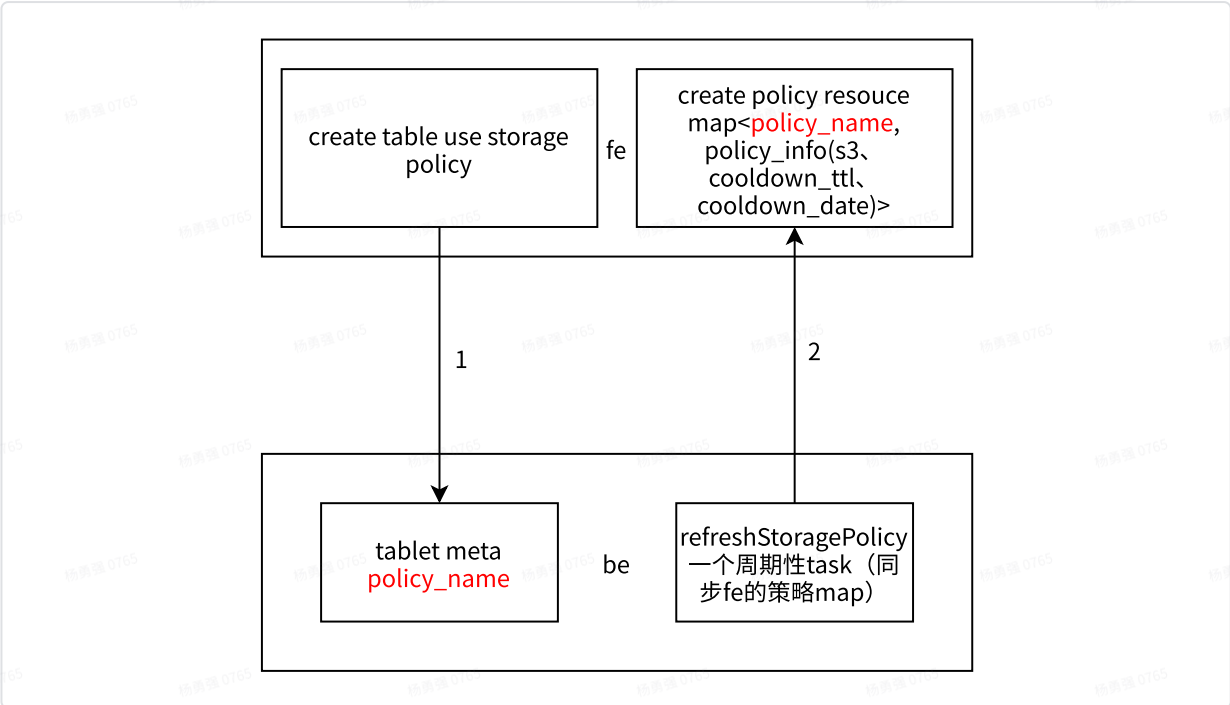
1. 社区实现的创建策略，存储策略storage_policy和存储资源storage_resource是通过名字关联的。需要分别create resource。
2. 具体的storage_policy和storage_resource功能（be部分）怎么使用，社区上的目前还没实现

```
SQL
CREATE RESOURCE "test"
PROPERTIES(
    "type"="storage_policy",
    "cooldown_datetime" = "2022-06-01",
    "cooldown_ttl" = "1h",
    "storage_resource" = "my_s3"
);

CREATE RESOURCE "my_s3"
PROPERTIES(
    "type" = "s3",
    "s3_endpoint" = "http://bj.s3.com",
    "s3_region" = "bj",
    "s3_root_path" = "/path/to/root",
    "s3_access_key" = "bbb",
    "s3_secret_key" = "aaaa",
    "s3_max_connections" = "50",
    "s3_request_timeout_ms" = "3000",
    "s3_connection_timeout_ms" = "1000"
);

```

```
mysql> SHOW RESOURCES;
+-------+--------------+---------------------------+-------------------+
| Name  | ResourceType | Item                      | Value             |
+-------+--------------+---------------------------+-------------------+
| my_s3 | s3           | s3_secret_key             | aaaa              |
```

```
| my_s3 | s3              | s3_region                | bj                |
| my_s3 | s3              | s3_access_key            |                   |
| my_s3 | s3              | s3_max_connections       | 50                |
| my_s3 | s3              | s3_connection_timeout_ms | 1000              |
| my_s3 | s3              | type                     | s3                |
| my_s3 | s3              | s3_root_path             | /path/to/root     |
| my_s3 | s3              | s3_endpoint              | http://bj.s3.com  |
| my_s3 | s3              | s3_request_timeout_ms    | 3000              |
| test  | storage_policy  | cooldown_ttl             | 1h                |
| test  | storage_policy  | storage_resource         | my_s3             |
| test  | storage_policy  | cooldown_datetime        | 2022-06-01        |
| test  | storage_policy  | type                     | storage_policy    |
+-------+-----------------+--------------------------+-------------------+
13 rows in set (0.00 sec)
```

be策略名storage_policy_name的获取方式：




## 决策

目前的cooldown是partition级别，FE来决策并发起数据从SSD到HDD迁移的。如果数据要冷却到S3，在多副本场景下，就需要确保S3只有一份数据，冷却的基本单位是rowset，目前FE不感知rowset，可以是BE发起，FE不参与。

理想来说一份数据只进入对象一次是最优的，常规做法主要包括两类：1. 协调出一个副本来做数据的上传，其他副本使用结果，分布式系统下这种方法也做不到100%只有一个副本上传数据到S3；2. 各个副本各自做，做之前询问其他副本情况，各副本通过一定的时间随机以及询问配合，做到大多数情况下只有一个副本上传数据到S3. 方案2更为简单，因此可以选用方案2.

方案2细化：

1. BE询问其它副本相同版本的数据是否已经上传到S3
   a. 如果有BE已经开始任务，则等待10min再询问
   b. 如果全部BE没有开始任务，则自己开始上传任务
   c. 每个be上的rowset id都不一样，因此并发做会产生垃圾，不会产生数据正确性问题
   d. 这里各个副本通过一个30min内的时间随机减少任务冲突，避免复杂的任务协调

同时这个方案也可以用于优化compaction，即compaction前首先尝试从其他副本直接下载compaction的结果。

JavaScript

```
StorageEngine::_cooldown_tasks_producer_callback {
    // generate cooldown task for each tablet
    // 目前一个tablet不允许并发的compaction运行，
    // 这里可以做到compaction和cooldown都是严格按序单并发
    // 可以大幅简化rowset更改并发带来的问题
    submit_task()
}

class CoolDownTask { //这里会考虑把compaction和cooldown做成tabletTask，然后逻辑上就会很顺利的做串行，他们做工作是替换rowset
    TabletSharedPtr _tablet;
    execute() {
        // 查询其他副本是否在做cooldown或者是否有结果
        // 对比返回的version和rowset结果
        // 如果有结果就使用相关的rowset
        // 如果没有自己发起cooldown
    }
}

```

## 执行

单机IO栈：IO 支持简单存算分离　IO读路径

1. 数据上传到S3
2. 替换本地rowset，这样结合决策部分，多副本没法保障一个副本只有一份有效数据在S3，可以考虑BE之间P2P协商，引入随机因素做到99%的情况下只有一个副本上传冷数据。

C++

```
class Tablet {
    cooldown() {
        // 支持传入已经在S3的rowsetid
        // 持有schema_change_lock
        // pick oldest rowset and put it into s3
        // 复用migration生成rowset的逻辑，link改为copy到s3.
        // modify_rowsets // 外围做好compaction和cooldown的单并发
    }
}
```

为了清理单个be自身形成的垃圾，每次上传一个文件前向rocksdb写开始日志，完成后写完成日志，失败后写失败日志，并且清理残留在S3上的数据。

## 结果

### 数据组织

按照现在本地的结构来组织

### 版本 / compaction

目前各个BE独自做compaction，因此每个rowset的版本号是不一致的，不一致就没法替换。有两种方案，因为上传的数据是冷数据，各个节点使用的是相同的comaction策略，所以上传S3的数据compaction结果应该是一致的，也可以使用“决策”部分的方案对compaction简单优化。

目前compaction完成后，老的数据会被删掉，在多副本的环境下，老数据什么时候可以删？理想的来说meta可以集中同步，云版本考虑单副本暂时不考虑，多副本情况下，compaction需要多机做简单的协调，以及查询数据不在的时候FE能够容错处理。

JavaScript

```
void StorageEngine::start_delete_unused_rowset() {
}
```

### 获取

从上传成功的副本直接获取rowset元信息即可，然后将本地的rowset数据删除即可。

这部分代码应该是一个不上传数据的cooldown任务。

### 副本Clone

SnapshotManager::_create_snapshot_files使用了link，因为doris的数据不可变更，因此数据部分直接使用版本即可，可以不做快照，如果相关版本被compaction处理，下载文件失败处理即可？目前的snapshot包括两部分信息：1. hdr文件，它主要是tablet的meta信息；2. rowset文件，目前是通过link来实现。目前的clone分为三步：1. 请求上游make snapshot；2. 下载snapshot，包括hdr和rowset数据文件；3. 将数据加载到tablet，这里又使用了link。

C++

```
// 目前:
AgentService.TAgentResult make_snapshot(1:AgentService.TSnapshotRequest snapshot_request);

// Types.thrift
const i32 TSNAPSHOT_REQ_VERSION3 = 5; // corresponding to rowset in S3 version

SnapshotManager::_create_snapshot_files
    // 生成一个文件描述文件，比如FILE_LIST，里边记录文件的位置，比如s3://  local:// 文件大小信息

EngineCloneTask::_make_and_download_snapshots
    // 目前是通过http获取目录文件列表下载的方式来进行
    // 修改为如果存在FILE_LIST文件，则通过里边记录的文件位置的方式来下载，并且处理S3里边的文件
```

需要做好兼容性。

**副本删除**

目前不管是FE全局删除table或者删除BE上某个tablet，都是一样的rpc一样的参数，我们需要加上不同的参数来让be删除s3中的数据。

Java

```
// FE
DropReplicaTask {
+ boolean isEraseTableOrPartition;
}

struct TDropTabletReq {
    1: required Types.TTabletId tablet_id
    2: optional Types.TSchemaHash schema_hash
    3: optional bool isEraseTableOrPartition
}

//这儿为什么要主动发起drop任务，而不是等report处理?
Catalog::onEraseOlapTable
    new DropReplicaTask with isEraseTableOrPartition = true

ReportHandler::deleteFromBackend
    // 增加判断是否是table和partition的逻辑，对于不存在的设置isEraseTableOrPartition = true

// BE
void TaskWorkerPool::_drop_tablet_worker_thread_callback()
    // 处理isEraseTableOrPartition透传到TabletManager::drop_tablet

TabletManager::start_trash_sweep()
    // 目前只有_shutdown_tablets记录要删除的tablet，需要增加一个新的区分是否删除S3里边的数据。
    // S3的trash可以通过记录删除清单实现 TODO 细节

```

**垃圾**

不可避免会产生垃圾，我们要尽量的避免垃圾回收，从而正向的将垃圾回收掉。因为每个be会通过两阶段日志来正向清理垃圾，因此每个be不会需要额外的垃圾回收逻辑。如果be永久故障，垃圾怎么清理？可以定期触发每个tablet级别的垃圾回收，比如对过期一天时间但是没有被引用的数据可以清理掉？

工作拆解

| | FE | BE |
|---|---|---|
| 策略（1人周） | 1. 策略定义 | 1. 策略同步 |

| | | |
|---|---|---|
| | a. Create resource "type" = "life_cycle"<br>b. Create table / alter table property "life_cycle" = ""<br>2. 策略同步<br>3. 策略修改，只修改cooldown，已经cooldown的数据不受影响 | |
| 决策（2人周） | | 1. 周期性任务<br>2. 随机策略以及其他副本交互 |
| 执行（3人周） | | 1. rowset存储S3<br>a. 写s3 @程宇轩<br>b. 读s3 @芦伟<br>2. Schema change冲突，使用schema_change_lock同步即可<br>3. Compaction冲突，这里需要串行来简化tablet modify_rowset逻辑。 |
| 结果（3人周） | 1. FE处理汇总信息，支持S3空间<br>2. FE负载均衡相关逻辑更改 | 1. 向其他副本发布结果 以及 其他副本来获取结果。@芦伟 这里细节上主要是实现rpc获取信息，@程宇轩 cooldown方法需要支持已经有了s3数据的支持，就是传入一个rowset meta就行。<br>2. clone副本相关改动，主要是S3上的rowset不需要下载数据。<br>3. 向FE汇报空间信息<br>4. BE不删除S3上的rowset |
| | | |
| 效率 | 1. | 1. Schema Change优化<br>2. 对象Cache |

第一阶段做到数据能进入S3以及读取，不考虑补副本以及删除表 （3人周）

- rowset存储到S3
  - 部分IOstack调整 @程宇轩 @芦伟
  - 写S3 @程宇轩
  - 读S3 @芦伟
- 周期性任务，冷却时间可以是一个配置的值 @芦伟

第二阶段做到补副本以及删除表正常工作 （2人周）

- 扩缩容/负载均衡（clone task） @邓鑫
- compaction串行 @芦伟 和周期性任务一并考虑下
- BE不删除S3上的rowset @程宇轩

第三阶段策略 （2人周）

- 策略定义
- 策略同步

第四阶段 （2人周）

- 空间汇报
- 负载均衡

第五阶段

- Schema change优化
- 对象cache
- 多副本
  - 上传结果同步
  - s3上数据compaction删除
  - 从其他副本直接使用s3的结果

## 待确认

1. ~~rowset创建时间，比如compaction，clone的rowset的创建时间是什么规则，设计到数据冷却。~~
2. 支持表级别clone，可能设计S3上的数据组织
3. FE 看起来无法直接 restore BE trash，drop partition 暂时考虑直接删除 remote rowset，需要确认下 be trash/restore 是否真的有需求（感觉主要是应对 bug 导致的 drop replica）(P2)
4. ~~Rowset 移到 S3 后 Tablet 负载均衡策略~~
5. ~~FE dropPartition 没看到给 BE 发 drop task 的逻辑~~
6. BE page cache 的 key 可以改成 tablet_id + version + offset（现在是 filepath + offset）(P3)
7. Delete from sql 如何处理？Base compaction 在遇到 delete rowset 时放开对远程 rowset base compaction 的限制？(P2)
8. 如果 Cooldown 和 drop table/partition 并发会发生什么？(P0)

a. Drop tablet 调用 remove_all_remote_rowsets 发生在 cooldown 调用 modify_rowsets 前，upload 的数据变成垃圾数据
b. Drop tablet 调用 remove_all_remote_rowsets 发生在 cooldown 调用 modify_rowsets 后，upload 的数据会被清理
c. Drop tablet 完成后 tablet 会被放到 _shutdown_tablets 中，之后所有文件 move 到 trash 下；sweep trash 和 cooldown 并发时 cooldown upload 发现 local rowset 路径不存在会返回 error，不会影响正确性（部分已经上传的 segment 变成垃圾数据）

以上场景垃圾数据都可以在 cooldown 中被识别，a 场景需要在 modify_rowsets 前判断 TABLET_SHUTDOWN 来决定是否需要清理刚 upload 的数据

9. Clone 和 Upload 并发会发生什么？(P0)
   a. Clone dst：
      i. tablet != nullptr，前半段 calc_missed_versions, _make_and_download_snapshots 没影响，后半段 _finish_clone 有 compaction_lock 互斥不会有并发问题。但是 _finish_full_clone 可能会出现已经 upload 的 rowset 被放到 unused_rowset 中并且无法知道之后该不该删除 remote 的数据（可能被别的 BE 使用）；目前只会删除 unused_rowset 中的 local rowset
      ii. tablet != nullptr clone 后可能会导致 local rowset 版本不连续，对 compaction 的影响：
         1. Base compaction 发现 version 不连续不会进行，这里不去管也行
         2. Cumu compaction 会找到最长连续 version 进行 compaction
      iii. tablet == nullptr 没影响
   b. Clone src：
      i. capture_consistent_rowsets 拿到的 local rowset 可能已经 upload 了，虽然不会出现 link 失败的问题，但是 remote 数据会冗余，并且让 cooldown 和 make_snapshot 串行也无法避免 remote 数据冗余（A make_snapshot, B download from A, A upload, A drop）
10. Remote rowset 对 migration 的影响，可能需要更新 migration 的代码(P0)
    a. Cooldown 理论上可以不用和 migration 串行执行，cooldown 和 migration 并发时，rowset 上传后之前的 local rowset 因为被 migration 的 consistent_rowsets 引用所以不会被删除，_copy_index_and_data_files 可以正常进行，但是这样 migration 后的数据和 remote 就会出现冗余，个人倾向串行执行（cooldown try migration_lock）
11. BeRebalancer 做 replica 调度的时候会根据 tablet data size 判断调度后是否有收益，但是 tablet data size 目前算上了 remote rowset 的，会导致计算不准确(P0)【done】
12. ~~Cooldown 策略拍一下(P0)~~