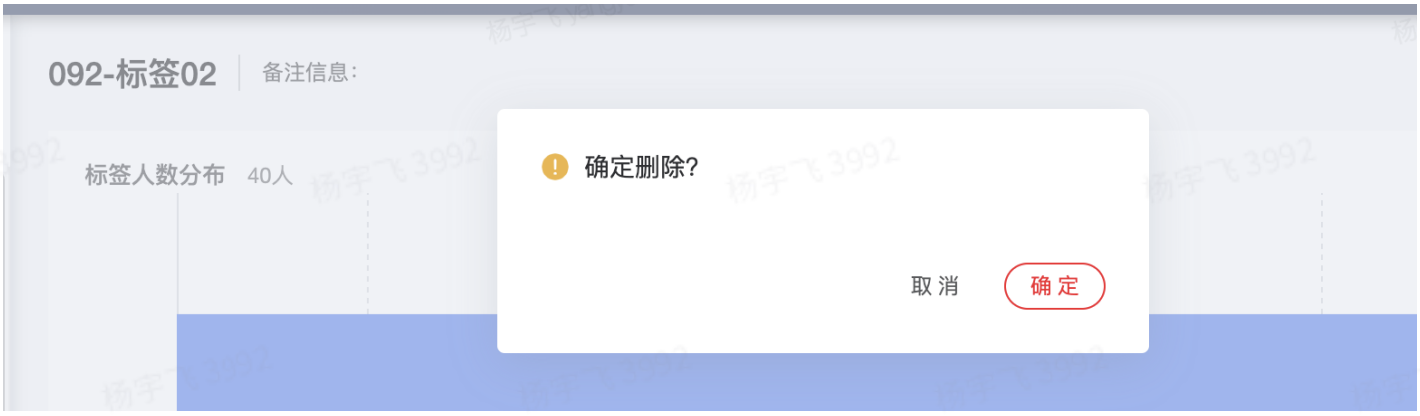
一场由删除标签引发的悲剧

背景介绍



v1.0 仅删除mysql记录

方法实现:

- 删除列：仅删除mysql记录，将标签状态由0改为1表示删除
- 创建标签，分配列:
  - 首先，获取clickhouse表结构，然后获取mysql中没有被删除的标签
  - (1) 如果clickhouse中的列都被分配，则重新创建列分配标签
  - (2) 如果clickhouse中有列未分配，则从未分配列中选择一个分配给新创建的标签

导致问题

- 历史数据遗留
  - 由于删除标签时没有删除clickhouse中的数据，当下次创建标签还可以分配到这一列
  - 此时，上一个标签的数据仍然存在，如果用该标签创建群体就可以看到历史标签值

	Mysql标签	新增标签前Clickhouse	新增标签	新增标签后Clickhouse
正常场景	1 label_1 0 2 label_2 0 3 label_3 0	label_1 label_2 label_3 标签值1 标签值2 标签值3	4 label_4 0	label_1 label_2 label_3 label_4 标签值1 标签值2 标签值3 标签值4
问题场景	1 label_1 0 2 label_2 1 3 label_3 0	label_1 label_2 label_3 标签值1 标签值2 标签值3	4 label_2 0	label_1 label_2 label_3 标签值1 标签值2 标签值3 标签值4

解决方案

- 删除标签时，将数据列一并删除

v2.0 删除标签同时删除clickhouse列

方法实现：

- 调用clickhouse的drop_column接口进行删除

导致问题

- 列删除仍可能被分配，但是最终标签计算失败
  - 由于Clickhouse删除列有延迟，删除标签后马上创建新的标签时，clickhouse中的列还没有被删除
  - 此时，创建标签获取clickhouse表结构中仍然包含刚被删除的列
  - 但是数据库中的记录已经被删除，所以利用上面的分配列策略，仍然可能分配到这一列
  - 从而，导致虽然分配列成功了，但是过一段时间这一列就会从clickhouse中删除，最终导致导入标签数据失败

	Mysql标签	新增标签前Clickhouse	新增标签	新增标签后Clickhouse
问题场景	1 label_1 0 2 label_2 1	label_1 label_2 label_3 标签值1 标签值2 标签值3	4 label_2 0	label_1 label_3 标签值1 标签值3

	3 label_3 0			

解决方案

- 设置正在删除标志位
- 异步检查clickhouse中是否删除成功，删除成功后才将标签状态修改为删除

v3.0 设置正在删除标志位

方法实现：

- 删除标签时，首先将状态改为2，表示正在删除
- 发送clickhouse删除列请求后，开启一个线程异步检测列是否删除成功，如果删除成功再将标签的状态改为1
- 同时，创建标签时，获取mysql中没有被删除的标签会将正常标签和删除中标签，从而与clickhouse的表结构保持一致

	Mysql标签	Clickhouse	新增标签	
				异步线程检测clickhouse中标签未删除
删除标签后，分配列时	1 label_1 0 2 label_2 2 3 label_3 0	label_1 label_2 label_3 标签值1 标签值2 标签值3	4 label_4 0	
				异步线程检测clickhouse中标签列已删除
分配列后，导入数据时	1 label_1 0 2 label_2 1 3 label_3 0 4 label_4 0	label_1 label_3 label_4 标签值1 标签值3 标签值4		

导致问题：

- 列删除仍可能被分配
  - 多线程同步问题：

	Mysql标签	Clickhouse	新增标签	
				异步线程检测clickhouse中标签未删除
删除标签后，分配列时获取clickhouse列	1 label_1 0 2 label_2 2 3 label_3 0	label_1 label_2 label_3 标签值1 标签值2 标签值3		
				异步线程检测clickhouse中标签已删除
删除标签后，分配列时获取mysql标签并分配列	1 label_1 0 2 label_2 1 3 label_3 0	label_1 label_3 标签值1 标签值3	4 label_2 0	
分配列后，导入数据时	1 label_1 0 2 label_2 1 3 label_3 0 4 label_2 0	label_1 label_3 标签值1 标签值3		

- 新问题：两个标签同时创建，请求分配到不同的服务器，可能分配到同一列

解决方案：

- 修改分配列逻辑
- 设置分布式锁

v4.0 调整删除策略并添加分布式锁

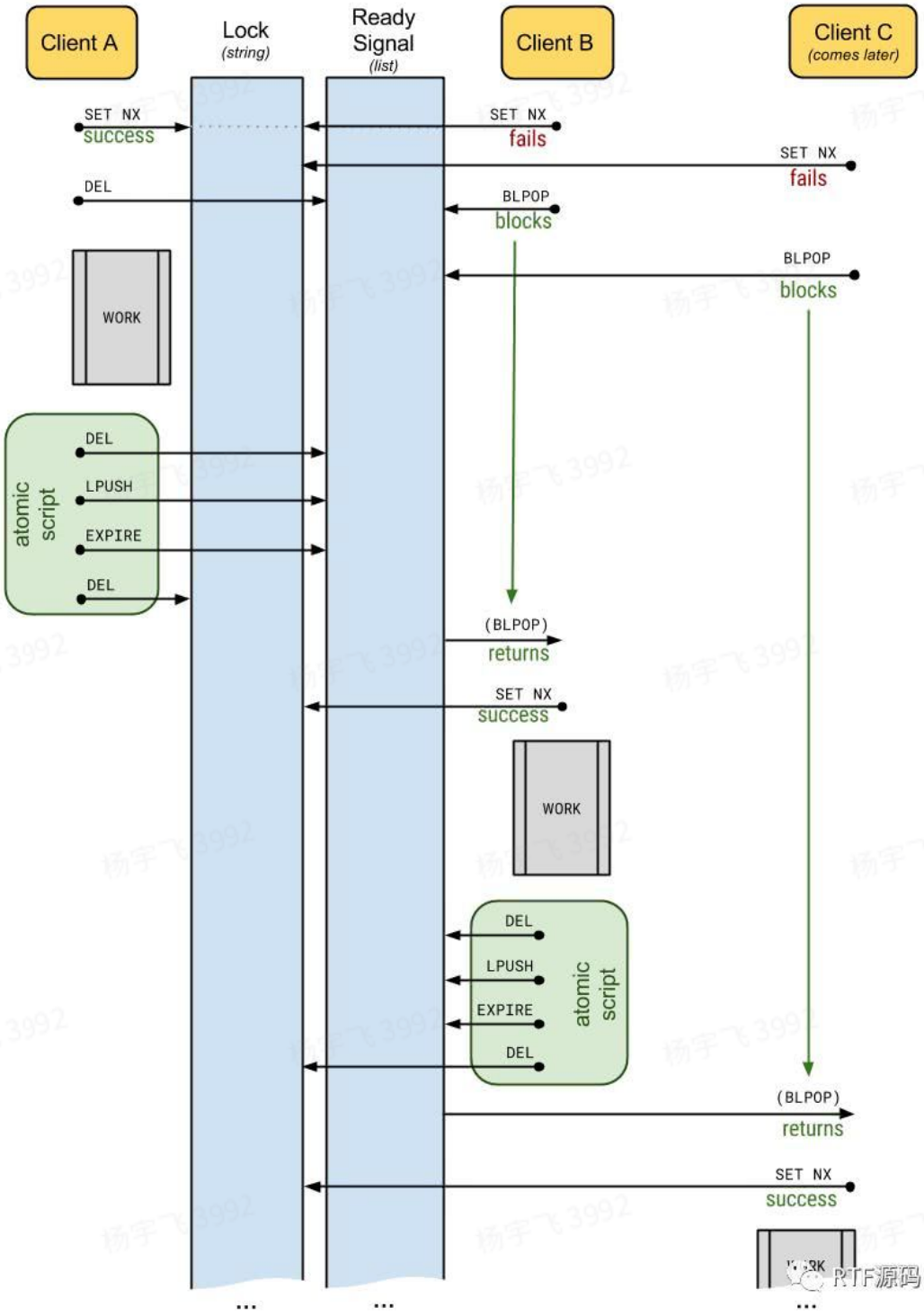
方法实现：

- 修改分配列逻辑：由于删除标签不会再次分配，所以获取mysql中App下所有的标签（包括正常标签和已删除标签），只分配clickhouse中空闲，未在mysql中有记录的列

	Mysql标签	新增标签前Clickhouse	新增标签	新增标签后Clickhouse
正常场景	1 label_1 0 2 label_2 0 3 label_3 0	label_1 label_2 label_3 标签值1 标签值2 标签值3	4 label_4 0	label_1 label_2 label_3 label_4 标签值1 标签值2 标签值3 标签值4

删除场景	1 label_1 0 2 label_2 1 3 label_3 0	label_1 label_2 label_3 标签值1 标签值2 标签值3	4 label_4 0	label_1 label_3 label4 标签值1 标签值3 标签值4
删除场景	1 label_1 0 2 label_2 1 3 label_3 0	label_1 label_3 标签值1 标签值3	4 label_4 0	label_1 label_3 label4 标签值1 标签值3 标签值4

- 设置分布式锁：为了避免分布式场景下，分配到同一个列，使用redis实现分布式锁进行解决



- acquire_lock

```
def acquire_lock(cls, cluster_name, db_name=None, table_name=None,
  data_type_name=None, lock_timeout=60, acquire_timeout=60):
    k = cls.gen_key(cluster_name=cluster_name, db_name=db_name,
      table_name=table_name,
                    data_type_name=data_type_name)
    identifier = str(uuid.uuid4())

    start = time.time()
    end = time.time() + acquire_timeout
```

```
    while time.time() < end:
        if cls.client.setnx(k, identifier, _raise=True):
            cls.client.expire(k, lock_timeout)
            logger.info('locking time: {}'.format(time.time() - start))
            logger.debug('acquire lock_lock: {}'.format(time.time()))
            return identifier
        elif not cls.client.ttl(k):
            cls.client.expire(k, lock_timeout)

        time.sleep(0.1)
```

- release_lock

```
def release_lock(cls, identifier, cluster_name, db_name=None, table_name=None, data_type_name=None):
    while True:
        try:
            iden = cls.get(cluster_name=cluster_name, db_name=db_name, table_name=table_name, data_type_name=data_type_name, _raise=True)
            if iden and iden.decode('utf-8') == identifier:
                cls.del_one(cluster_name=cluster_name, db_name=db_name, table_name=table_name, data_type_name=data_type_name)
                return True
            break
        except Exception as e:
            logger.info('release_lock exception: '.format(e))
            raise e
    return False
```

存在问题和下一步改进：

- 释放锁并非原子操作：
  - 事务方法：公司pyredis不支持事务，因为redis的事务处理并不好
  - 执行LUA脚本方法：目前风神的redis集群不支持，需要升级
- Redis的Master节点宕机：分布式锁proposal
  - RedLock算法
  - 使用etcd分布式kv存储：采用`raft`协议作为一致性算法
  - 使用zookeeper框架

参考：

Redis 新人学习相关资料汇总

https://blog.halukshan.com/blog/2020/03/31/redis-distributed-lock-python/

https://zhuanlan.zhihu.com/p/112016634
https://www.cnblogs.com/angelyan/p/11523846.html