云HBase小组成功抢救某公司自建HBase集群，挽救30+T数据

原创

阿里云云栖号 于 2018-04-18 14:37:32 发布 301 收藏 1
文章标签： 数据 修复 hbase 信息 启动

本文链接：https://blog.csdn.net/yunqiinsight/article/details/80134511

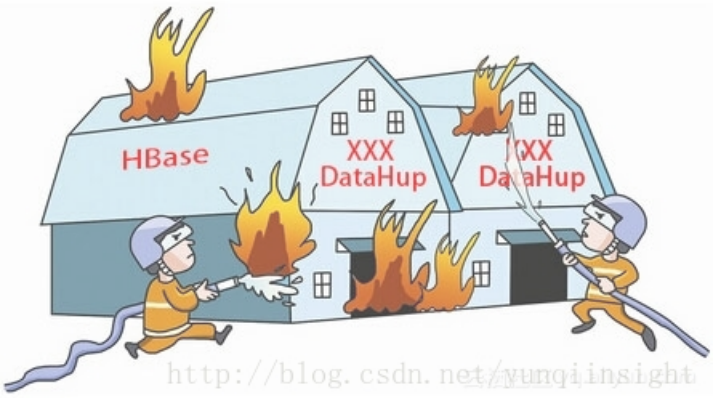
摘要： 使用过开源HBase的人都知道，运维HBase是多么复杂的事情，集群大的时候，读写压力大，配置稍微不合理一点，就可能会出现集群状态不一致的情况，糟糕一点的直接导致入库、查询某个业务表不可用, 甚至集群运行不了。

概述

使用过开源HBase的人都知道，运维HBase是多么复杂的事情，集群大的时候，读写压力大，配置稍微不合理一点，就可能会出现集群状态不一致的情况，糟糕一点的直接导致入库、查询某个业务表不可用， 甚至集群运行不了。在早期0.9x版本的时候，HBase的修复工具还有一下bug，使得即使你懂得如何修复的情况下，依然需要多次重复运行命令，绕过那些不合理的修复逻辑，甚至有时候需要自己写代码预先修复某个步骤。

背景

上周五，某公司使用的某DataHup 大数据产品自建一个HBase集群挂了！整个集群有30+T 业务数据，是公司的数据中心，集群直接启动不了。他们也是经历了熬战一天一夜的情况下，依旧没有解决恢复，还曾有过重装集群重导数据念头。最后，通过钉钉HBase技术交流群找到群主——阿里云HBase的封神。随后其立即下达命令，临时成立 HBase抢救小分队，尽力最大的努力，使用最低风险的方式，抢救最完整的集群。
蹭蹭蹭，几个抢救队员集齐，开始救火。



救火开始

虽然紧急，但是抢救工作不能乱，我们把救火过程主要分为几步：

1. 定位现象问题所在

首先与用户沟通现场环境情况，以及客户在出问题之前做过哪些重大操作，特别是一些特殊操作，平时没做过的。据用户描述已经远程观察了解到，用户使用开源的某DataHup自建了一个HBase集群， 存储公司的大量的业务，是公司的数据中心。集群有7个RegionServer、2个Master，32核256G的机器配置，业务数据量有30+T。HBase的master已经都挂了，两个RegionServer也挂了，用户使用过“重启大法”，依旧无法正常运行。
寥寥几句没有更多信息，我们只能上集群开日志，打jstack，观察HBase运行流程为什么中断或者挂掉。

首先我们先检查HDFS文件系统，fsck发现没有什么异常。其次开始检查HBase，把Debug日志打开，全部关闭HBase集群，为了便于观察现象，只启动一个Master和一个RegionServer。启动后，发现Master 因为fullscan meta表（master启动的一个流程）timeout Abort 终止了。观察meta region分配到的RegionServer也挂了，查看日志并没有异常，貌似是这个开源的DataHup当RegionServer scan数据操作超时 会被manager kill掉的样子。打jstack发现，Master确实在等待fullscan meta完成，而接管meta region的RegionServer确实一直在忙着scan meta数据，确实有忙不过来超时了。按理说，扫描meta表应该很快的才对。

检查发现HDFS的HBase meta表有1T多数据！！！进一步查看现象HFile的内容，发现了大量的Delete famly 的cell存在，而且很多是重复的，序列号（没有截图，想象一下）。问题现象定位了，用户使用这个系列的DataHup 的HBase生态时，有组件存在bug往hbase meta表写了大量的这些冗余的delete数据，导致hbase 启动时full scan meta卡着，最终导致整个集群无法正常启动运行服务。

2. 提出解决方案，评估风险

我们很快生成了两个相对较优的方案。第一个是使用离线compaction，把hbase meta表进行一次major compaction把多余的delete family删除，然后再重启即可。第二个方案是，直接移除meta 表的无用hfile, 逆向生成meta 表数据进行修复meta表即可。

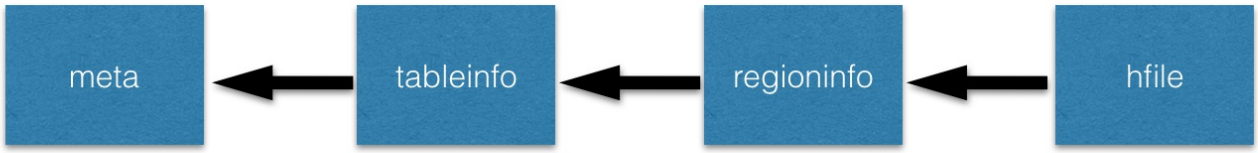
第一个方案做离线compaction对集群来说没有什么风险，缺点是离线compaction并不快，因为meta表region只有一个，执行离线meta表compaction时只有一个task，非常的缓慢耗时。

第二个方案是逆向修复meta表信息。看似风险很大，其实实际操作起来，很多风险可以降低。我们可以备份好核心的元数据，只有就可以在恢复失败的时候，还原到原来修复手术的前状态。这样一来，这个修复过程也就风险极大降低了。

3. 开始实施

秉着更安全风险更低的情况下，我们还是先选择了方案一，给meta表做离线major compaction的方案。但最终因为MapReduce和本地模式的compaction都太慢了，开始会oom，调整后，最终因meta的hfile checksum校验失败中断了。meta表的hfile都存在问题，那么这个compaction过程就不能正常进行了。

我们开始选择方案二，和用户沟通风险后，开始制定操作步骤， 把这个方案的实施带来的风险尽可能降到最低。规避这个方案存在的风险，前提是懂得这个方案会有什么风险。下面我们来分析一下，如图：



可以看到，开源HBase的meta表，是可以逆向生成回来的，但是有可能不同的DataHup生产商可能会有一些额外的信息hack进meta表里，对于这部分信息，在开源的逆向生成过程中是不包含的，存在这个关系数据丢失。但是这些核心的业务数据都存在，只是hack的第三方关联信息不存在了。有人可能会问，会有哪些数据可能hack到这里呢？曾看到过某厂商会在meta表了多加一些额外的字段用来保存virtual hostname信息，还有一些将二级索引相关的信息会保存在tableinfo 文件那里。HBase的开发商越多，什么姿势都可能存在，这个就是可能的风险点。

接下来我们开始实施，这个问题比较典型，用户的meta表里，有1T多的hfile 数据，检查hfile 发现几乎99%的hfile是delete famly相关的内容，我们就移除这些delete famly的hfile到备份目录，只留下一个正常数据的hfile，而这个hfile也仅仅有30M左右的数据。重启HBase后，正常运行。HBase一致性检查发现很幸运，没有坏文件，也没有丢失的tableinfo、regioninfo、hfile相关的block等。如果发现有文件丢失，corrupt hfile等等问题，逆向生成元数据的修复过程就可能会带来风险，但HBase集群核心业务数据依然可以完整挽救。

4. 用户再自己验证一下是否正常

通知用户验证集群运行，业务运行情况。

原文链接

阅读更多干货好文，请关注扫描以下二维码：