HDFS NameNode重启优化

作者 新美大离线存储团队 发布于 2017年3月7日. 估计阅读时间: 29 分钟

一、背景

在Hadoop集群整个生命周期里，由于调整参数、Patch、升级等多种场景需要频繁操作NameNode重启，不论采用何种架构，重启期间集群整体存在可用性和可靠性的风险，所以优化NameNode重启非常关键。

本文基于Hadoop-2.x和HA with QJM社区架构和系统设计（如图1所示），通过梳理NameNode重启流程，并在此基础上，阐述对NameNode重启优化实践。

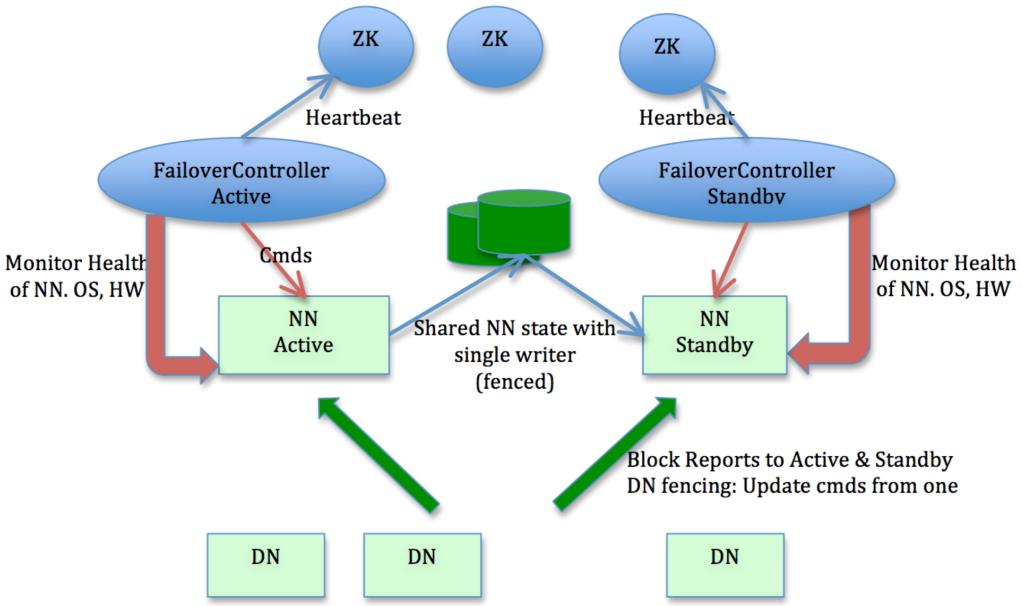

图1 HDFS HA with QJM架构图示

二、NameNode重启流程

在HDFS的整个运行期里，所有元数据均在NameNode的内存集中管理，但是由于内存易失特性，一旦出现进程退出、宕机等异常情况，所有元数据都会丢失，给整个系统的数据安全会造成不可恢复的灾难。为了更好的容错能力，NameNode会周期进行Checkpoint，将其中的一部分元数据（文件系统的目录树Namespace）刷到持久化设备上，即二进制文件FSImage，这样的话即使NameNode出现异常也能从持久化设备上恢复元数据，保证了数据的安全可靠。

但是仅周期进行Checkpoint仍然无法保证所有数据的可靠，如前次Checkpoint之后写入的数据依然存在丢失的问题，所以将两次Checkpoint之间对Namespace写操作实时写入EditLog文件，通过这种方式可以保证HDFS元数据的绝对安全可靠。

事实上，除Namespace外，NameNode还管理非常重要的元数据BlocksMap，描述数据块Block与DataNode节点之间的对应关系。NameNode并没有对这部分元数据同样操作持久化，原因是每个DataNode已经持有属于自己管理的Block集合，将所有DataNode的Block集合汇总后即可构造出完整BlocksMap。

HA with QJM架构下，NameNode的整个重启过程中始终以SBN（StandbyNameNode）角色完成。与前述流程对应，启动过程分以下几个阶段：

1. 加载FSImage；
2. 回放EditLog；
3. 执行Checkpoint；（非必须步骤，结合实际情况和参数确定，后续详述）
4. 收集所有DataNode的注册和数据块汇报；

默认情况下，NameNode会保存两个FSImage文件，与此对应，也会保存对应两次Checkpoint之后的所有EditLog文件。一般来说，NameNode重启后，通过对FSImage文件名称判断，选择加载最新的FSImage文件及回放该Checkpoint之后生成的所有EditLog，完成后根据加载的EditLog中操作条目数及距上次Checkpoint时间间隔（后续详述）确定是否需要执行Checkpoint，之后进入等待所有DataNode注册和元数据汇报阶段，当这部分数据收集完成后，NameNode的重启流程结束。

从线上NameNode历次重启时间数据看，各阶段耗时占比基本接近如图2所示。

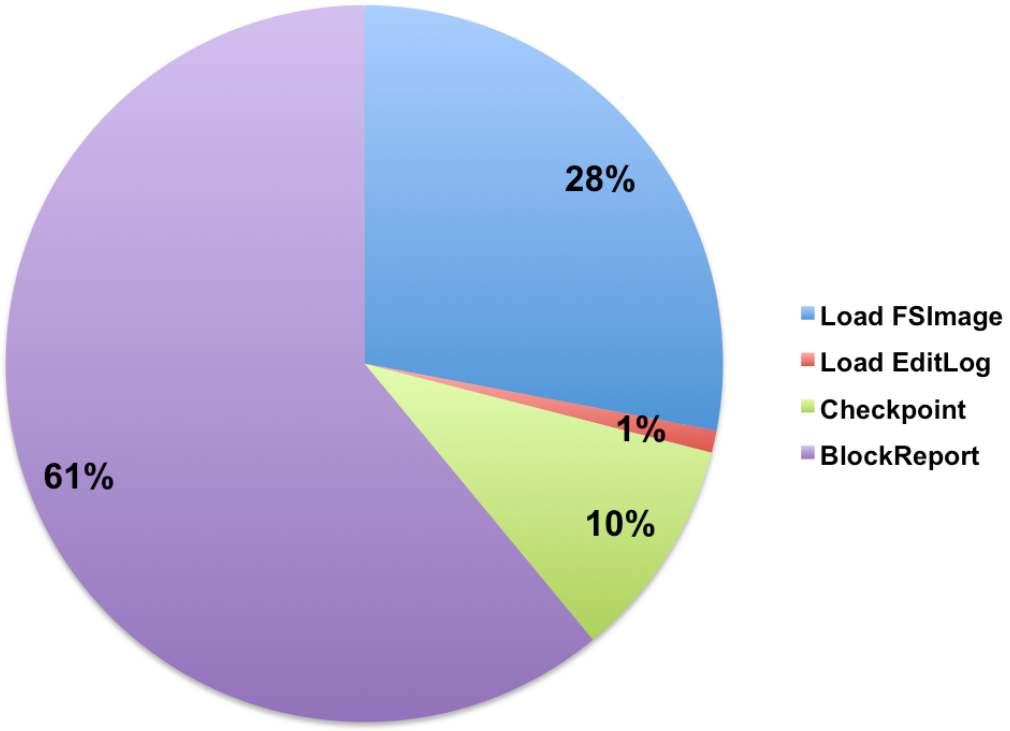

图2 NameNode重启各阶段耗时占比

经过优化，在元数据总量540M（目录树240M，数据块300M），超过4K规模的集群上重启NameNode总时间~35min，其中加载FSImage耗时~15min，秒级回放EditLog，数据块汇报耗时~20min，基本能够满足生产环境的需求。

2.1 加载FSImage

如前述，FSImage文件记录了HDFS整个目录树Namespace相关的元数据。从Hadoop-2.4.0起，FSImage开始采用Google Protobuf编码格式描述（HDFS-5698），详细描述文件见fsimage.proto。根据描述文件和实现逻辑，FSImage文件格式如图3所示。

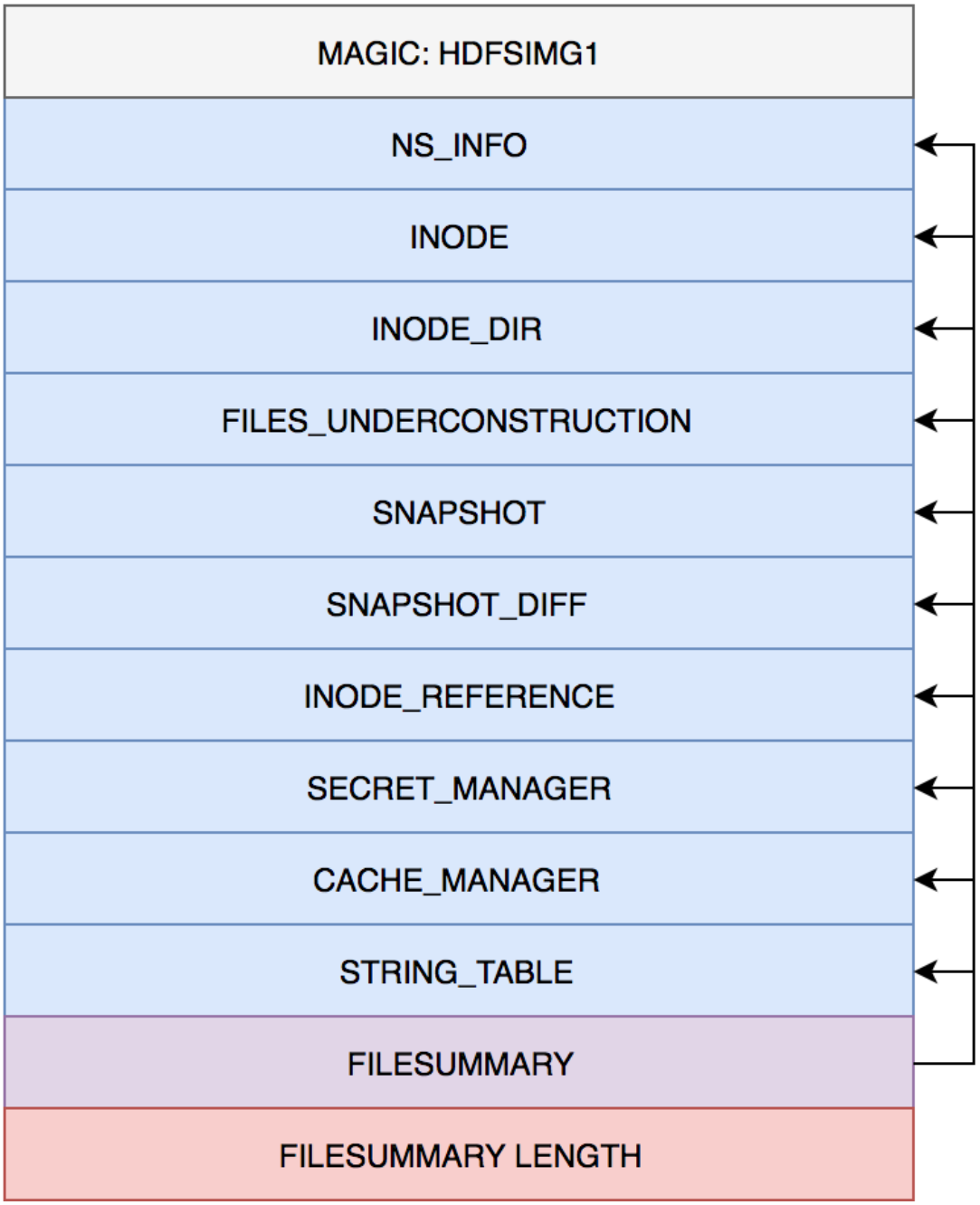

图3 FSImage文件格式

从fsimage.proto和FSImage文件存储格式容易看到，除了必要的文件头部校验（MAGIC）和尾部文件索引（FILESUMMARY）外，主要包含以下核心数据：

（0）NS_INFO（NameSystemSection）：记录HDFS文件系统的全局信息，包括NameSystem的ID，当前已经分配出去的最大BlockID以及TransactionId等信息；

（1）INODE（INodeSection）：整个目录树所有节点数据，包括INodeFile/INodeDirectory/INodeSymlink等所有类型节点的属性数据，其中记录了如节点id，节点名称，访问权限，创建和访问时间等等信息；

（2）INODE_DIR（INodeDirectorySection）：整个目录树中所有节点之间的父子关系，配合INODE可构建完整的目录树；

（3）FILES_UNDERCONSTRUCTION（FilesUnderConstructionSection）：尚未完成写入的文件集合，主要为重启时重建Lease集合；

（4）SNAPSHOT（SnapshotSection）：记录Snapshot数据，快照是Hadoop 2.1.0引入的新特性，用于数据备份、回滚，以防止因用户误操作导致集群出现数据问题；

（5）SNAPSHOT_DIFF（SnapshotDiffSection）：执行快照操作的目录/文件的Diff集合数据，与SNAPSHOT一起构建较完整的快照管理能力；

（6）INODE_REFERENCE（INodeReferenceSection）：当目录/文件被操作处于快照，且该目录/文件被重命名后，会存在多条访问路径，INodeReference就是为了解决该问题；

（7）SECRET_MANAGER（SecretManagerSection）：记录DelegationKey和DelegationToken数据，根据DelegationKey及由DelegationToken构造出的DelegationTokenIdentifier方便进一步计算密码，以上数据可以完善所有合法Token集合；

（8）CACHE_MANAGER（CacheManagerSection）：集中式缓存特性全局信息，集中式缓存特性是Hadoop-2.3.0为提升数据读性能引入的新特性；

（9）STRING_TABLE（StringTableSection）：字符串到id的映射表，维护目录/文件的Permission字符到ID的映射，节省存储空间；

NameNode执行Checkpoint时，遵循Protobuf定义及上述文件格式描述，重启加载FSImage时，同样按照Protobuf定义的格式从文件流中读出相应数据构建整个目录树Namespace及其他元数据。将FSImage文件从持久化设备加载到内存并构建出目录树结构后，实际上并没有完全恢复元数据到最新状态，因为每次Checkpoint之后还可能存在大量HDFS写操作。

2.2 回放EditLog

NameNode在响应客户端的写请求前，会首先更新内存相关元数据，然后再把这些操作记录在EditLog文件中，可以看到内存状态实际上要比EditLog数据更及时。

记录在EditLog之中的每个操作又称为一个事务，对应一个整数形式的事务编号。在当前实现中多个事务组成一个Segment，生成独立的EditLog文件，其中文件名称标记了起止的事务编号，正在写入的EditLog文件仅标记起始事务编号。EditLog文件的格式非常简单，没再通过Google Protobuf描述，文件格式如图4所示。

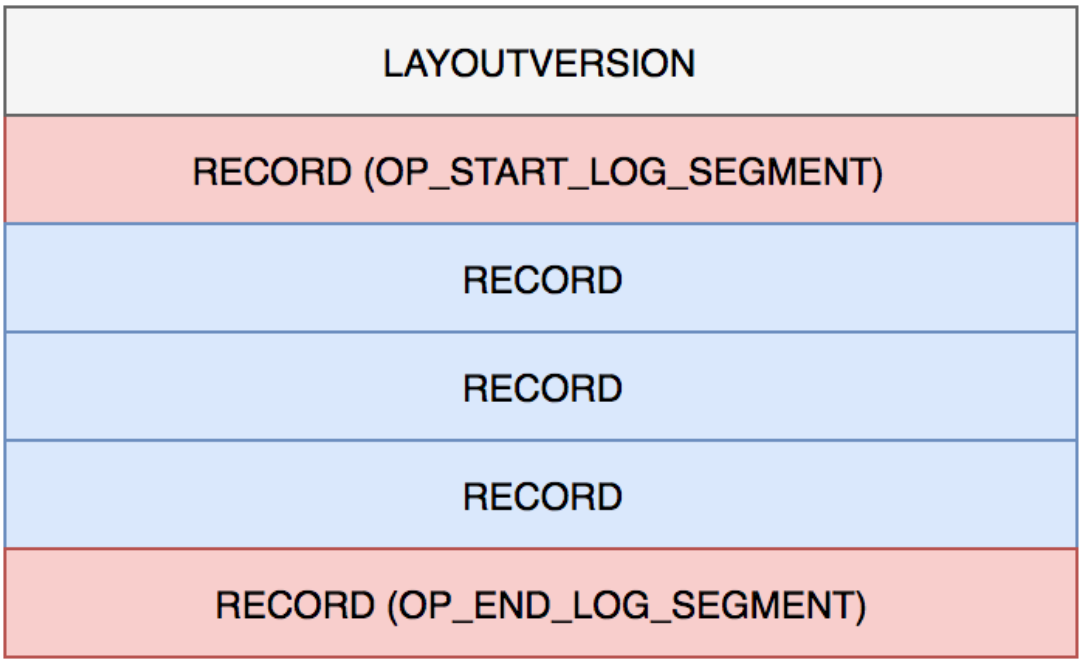

图4 EditLog文件格式

一个完整的EditLog文件包括四个部分内容，分别是：

（0）LAYOUTVERSION：版本信息；

（1）OP_START_LOG_SEGMENT：标识文件开始；

（2）RECORD：顺序逐个记录HDFS写操作的事务内容；

（3）OP_END_LOG_SEGMENT：标记文件结束；

NameNode加载FSImage完成后，即开始对该FSImage文件之后（通过比较FSImage文件名称中包含的事务编号与EditLog文件名称的起始事务编号大小确定）生成的所有EditLog严格按照事务编号从小到大逐个遵循上述的格式进行每一个HDFS写操作事务回放。

NameNode加载完所有必需的EditLog文件数据后，内存中的目录树即恢复到了最新状态。

2.3 DataNode注册汇报

经过前面两个步骤，主要的元数据被构建，HDFS的整个目录树被完整建立，但是并没有掌握从数据块Block与DataNode之间的对应关系BlocksMap，甚至对DataNode的情况都不掌握，所以需要等待DataNode注册，并完成对从DataNode汇报上来的数据块汇总。待汇总的数据量达到预设比例（dfs.namenode.safemode.threshold-pct）后退出Safemode。

NameNode重启经过加载FSImage和回放EditLog后，所有DataNode不管进程是否发生过重启，都必须经过以下两个步骤：

（0）DataNode重新注册RegisterDataNode；

（1）DataNode汇报所有数据块BlockReport；

对于节点规模较大和元数据量较大的集群，这个阶段的耗时会非常可观。主要有三点原因：

（0）处理BlockReport的逻辑比较复杂，相对其他RPC操作耗时较长。图5对比了BlockReport和AddBlock两种不同RPC的处理时间，尽管AddBlock操作也相对复杂，但是对比来看，BlockReport的处理时间显著高于AddBlock处理时间；

（1）NameNode对每一个BlockReport的RPC请求处理都需要持有全局锁，也就是说对于BlockReport类型RPC请求实际上是串行处理；

（2）NameNode重启时所有DataNode集中在同一时间段进行BlockReport请求；

(点击放大图像)

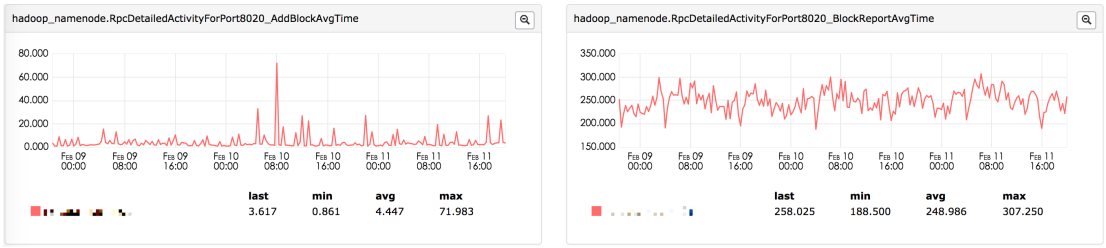

图5 BlockReport和AddBlock两个RPC处理时间对比

前文NameNode内存全景中详细描述过Block在NameNode元数据中的关键作用及与Namespace/DataNode/BlocksMap的复杂关系，从中也可以看出，每个新增Block需要维护多个关系，更何况重启过程中所有Block都需要建立同样复杂关系，所以耗时相对较高。

三、重启优化

根据前面对NameNode重启过程的简单梳理，在各个阶段可以适当的实施优化以加快NameNode重启过程。

0、HDFS-7097 解决重启过程中SBN执行Checkpoint时不能处理BlockReport请求的问题；

Fix：2.7.0

Hadoop-2.7.0版本前，SBN（StandbyNameNode）在执行Checkpoint操作前会先获得全局读写锁fsLock，在此期间，BlockReport请求由于不能获得全局写锁会持续处于等待状态，直到Checkpoint完成后释放了fsLock锁后才能继续。NameNode重启的第三个阶段，同样存在这种情况。而且对于规模较大的集群，每次Checkpoint时间在分钟级别，对整个重启过程影响非常大。实际上，Checkpoint是对目录树的持久化操作，并不涉及BlocksMap数据结构，所以Checkpoint期间是可以让BlockReport请求直接通过，这样可以节省期间BlockReport排队等待带来的时间开销，HDFS-7097正是将锁粒度放小解决了Checkpoint过程不能处理BlockReport类型RPC请求的问题。

与HDFS-7097相对，另一种思路也值得借鉴，就是重启过程尽可能避免出现Checkpoint。触发Checkpoint有两种情况：时间周期或HDFS写操作事务数，分别通过参数dfs.namenode.checkpoint.period和dfs.namenode.checkpoint.txns控制，默认值分别是3600s和1,000,000，即默认情况下一个小时或者写操作的事务数超过1,000,000触发一次Checkpoint。为了避免在重启过程中频繁执行Checkpoint，可以适当调大dfs.namenode.checkpoint.txns，建议值10,000,000 ~ 20,000,000，带来的影响是EditLog文件累计的个数会稍有增加。从实践经验上看，对一个有亿级别元数据量的NameNode，回放一个EditLog文件（默认1,000,000写操作事务）时间在秒级，但是执行一次Checkpoint时间通常在分钟级别，综合权衡减少Checkpoint次数和增加EditLog文件数收益比较明显。

1、HDFS-6763 解决SBN每间隔1min全局计算和验证Quota值导致进程Hang住数秒的问题；

Fix：2.8.0

ANN（ActiveNameNode）将HDFS写操作实时写入JN的EditLog文件，为同步数据，SBN默认间隔1min从JN拉取一次EditLog文件并进行回放，完成后执行全局Quota检查和计算，当Namespace规模变大后，全局计算和检查Quota会非常耗时，在此期间，整个SBN的Namenode进程会被Hang住，以至于包括DN心跳和BlockReport在内的所有RPC请求都不能及时处理。NameNode重启过程中这个问题影响突出。

实际上，SBN在EditLog Tailer阶段计算和检查Quota完全没有必要，HDFS-6763将这段处理逻辑后移到主从切换时进行，解决SBN进程间隔1min被Hang住的问题。

从优化效果上看，对一个拥有接近五亿元数据量，其中两亿数据块的NameNode，优化前数据块汇报阶段耗时~30min，其中触发超过20次由于计算和检查Quota导致进程Hang住~20s的情况，整个BlockReport阶段存在超过5min无效时间开销，优化后可到~25min。

2、HDFS-7980 简化首次BlockReport处理逻辑优化重启时间；

Fix：2.7.1

NameNode加载完元数据后，所有DataNode尝试开始进行数据块汇报，如果汇报的数据块相关元数据还没有加载，先暂存消息队列，当NameNode完成加载相关元数据后，再处理该消息队列。对第一次块汇报的处理比较特别（NameNode重启后，所有DataNode的BlockReport都会被标记成首次数据块汇报），为提高处理速度，仅验证块是否损坏，之后判断块状态是否为FINALIZED，若是建立数据块与DataNode的映射关系，建立与目录树中文件的关联关系，其他信息一概暂不处理。对于非初次数据块汇报，处理逻辑要复杂很多，对报告的每个数据块，不仅检查是否损坏，是否为FINALIZED状态，还会检查是否无效，是否需要删除，是否为UC状态等等；验证通过后建立数据块与DataNode的映射关系，建立与目录树中文件的关联关系。

初次数据块汇报的处理逻辑独立出来，主要原因有两方面：

（0）加快NameNode的启动时间；测试数据显示含~500M元数据的NameNode在处理800K个数据块的初次块汇报的处理时间比正常块汇报的处理时间可降低一个数量级；

（1）启动过程中，不提供正常读写服务，所以只要确保正常数据（整个Namespace和所有FINALIZED状态Blocks）无误，无效和冗余数据处理完全可以延后到IBR（IncrementalBlockReport）或下次BR（BlockReport）；

这本来是非常合理和正常的设计逻辑，但是实现时NameNode在判断是否为首次数据块块汇报的逻辑一直存在问题，导致这段非常好的改进点逻辑实际上长期并未真正执行到，直到HDFS-7980在Hadoop-2.7.1修复该问题。HDFS-7980的优化效果非常明显，测试显示，对含80K Blocks的BlockReport RPC请求的处理时间从~500ms可优化到~100ms，从重启期整个BlockReport阶段看，在超过600M元数据，其中300M数据块的NameNode显示该阶段从~50min优化到~25min。

3、HDFS-7503 解决重启前大删除操作会造成重启后锁内写日志降低处理能力；

Fix：2.7.0

若NameNode重启前产生过大删除操作，当NameNode加载完FSImage并回放了所有EditLog构建起最新目录树结构后，在处理DataNode的BlockReport时，会发现有大量Block不属于任何文件，Hadoop-2.7.0版本前，对于这类情况的输出日志逻辑在全局锁内，由于存在大量IO操作的耗时，会严重拉长处理BlockReport的处理时间，影响NameNode重启时间。HDFS-7503的解决办法非常简单，把日志输出逻辑移出全局锁外。线上效果上看对同类场景优化比较明显，不过如果重启前不触发大的删除操作影响不大。

4、防止热备节点SBN（StandbyNameNode）/冷备节点SNN（SecondaryNameNode）长时间未正常运行堆积大量Editlog拖慢NameNode重启时间；

不论选择HA热备方案SBN（StandbyNameNode）还是冷备方案SNN（SecondaryNameNode）架构，执行Checkpoint的逻辑几乎一致，如图6所示。如果SBN/SNN服务长时间未正常运行，Checkpoint不能按照预期执行，这样会积压大量EditLog。积压的EditLog文件越多，重启NameNode需要加载EditLog时间越长。所以尽可能避免出现SNN/SBN长时间未正常服务的状态。

图6 Checkpoint流程

在一个有500M元数据的NameNode上测试加载一个200K次HDFS事务操作的EditLog文件耗时~5s，按照默认2min的EditLog滚动周期，如果一周时间SBN/SNN未能正常工作，则会累积~5K个EditLog文件，此后一旦发生NameNode重启，仅加载EditLog文件的时间就需要~7h，也就是整个集群存在超过7h不可用风险，所以切记要保证SBN/SNN不能长时间故障。

5、HDFS-6425 HDFS-6772 NameNode重启后DataNode快速退出blockContentsStale状态防止PostponedMisreplicatedBlocks过大影响对其他RPC请求的处理能力；

Fix: 2.6.0, 2.7.0

当集群中大量数据块的实际存储副本个数超过副本数时（跨机房架构下这种情况比较常见），NameNode重启后会迅速填充到PostponedMisreplicatedBlocks，直到相关数据块所在的所有DataNode汇报完成且退出Stale状态后才能被清理。如果PostponedMisreplicatedBlocks数据量较大，每次全遍历需要消耗大量时间，且整个过程也要持有全局锁，严重影响处理BlockReport的性能，HDFS-6425和HDFS-6772分别将可能在BlockReport逻辑内部遍历非常大的数据结构PostponedMisreplicatedBlocks优化到异步执行，并在NameNode重启后让DataNode快速退出blockContentsStale状态避免PostponedMisreplicatedBlocks过大入手优化重启效率。

6、降低BlockReport时数据规模；

NameNode处理BlockReport的效率低主要原因还是每次BlockReport所带的Block规模过大造成，所以可以通过调整Block数量阈值，将一次BlockReport分成多盘分别汇报，以提高NameNode对BlockReport的处理效率。可参考的参数为：dfs.blockreport.split.threshold，默认值1,000,000，即当DataNode本地的Block个数超过1,000,000时才会分盘进行汇报，建议将该参数适当调小，具体数值可结合NameNode的处理BlockReport时间及集群中所有DataNode管理的Block量分布确定。

7、重启完成后对比检查数据块上报情况；

前面提到NameNode汇总DataNode上报的数据块量达到预设比例（dfs.namenode.safemode.threshold-pct）后就会退出Safemode，一般情况下，当NameNode退出Safemode后，我们认为已经具备提供正常服务的条件。但是对规模较大的集群，按照这种默认策略及时执行主从切换后，容易出现短时间丢块的问题。考虑在200M数据块的集群，默认配置项dfs.namenode.safemode.threshold-pct=0.999，也就是当NameNode收集到200M*0.999=199.8M数据块后即可退出Safemode，此时实际上还有200K数据块没有上报，如果强行执行主从切换，会出现大量的丢块问题，直到数据块汇报完成。应对的办法比较简单，尝试调大dfs.namenode.safemode.threshold-pct到1，这样只有所有数据块上报后才会退出Safemode。但是这种办法一样不能保证万无一失，如果启动过程中有DataNode汇报完数据块后进程挂掉，同样存在短时间丢失数据的问题，因为NameNode汇总上报数据块时并不检查副本数，所以更稳妥的解决办法是利用主从NameNode的JMX数据对比所有DataNode当前汇报数据块量的差异，当差异都较小后再执行主从切换可以保证不发生上述问题。

8、其他；

除了优化NameNode重启时间，实际运维中还会遇到需要滚动重启集群所有节点或者一次性重启整集群的情况，不恰当的重启方式也会严重影响服务的恢复时间，所以合理控制重启的节奏或选择合适的重启方式尤为关键，HDFS集群启动方式分析一文对集群重启方式进行了详细的阐述，这里就不再展开。

经过多次优化调整，从线上NameNode历次的重启时间监控指标上看，收益非常明显，图7截取了其中几次NameNode重启时元数据量及重启时间开销对比，图中直观显示在500M元数据量级下，重启时间从~4000s优化到~2000s。

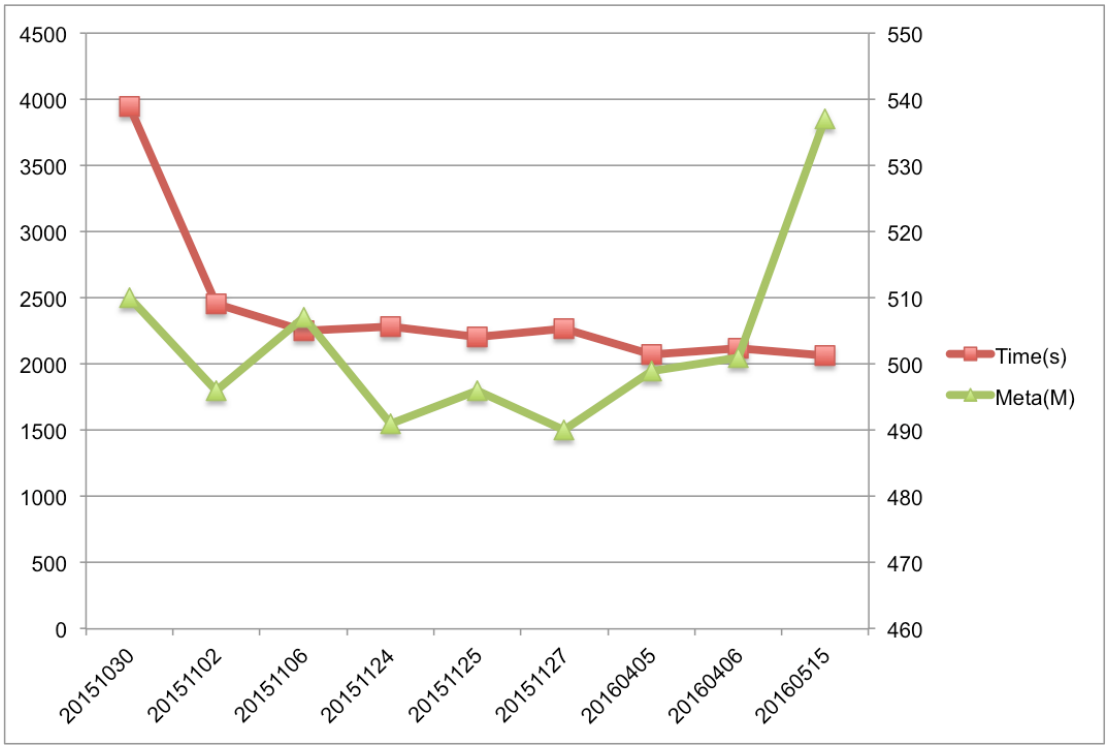

图7 NameNode重启时间对比

这里罗列了一小部分实践过程中可以有效优化重启NameNode时间或者重启全集群的点，其中包括了社区成熟Patch和相关参数优化，虽然实现逻辑都很小，但是实践收益非常明显。当然除了上述提到，NameNode重启还有很多可以优化的地方，比如优化FSImage格式，并行加载等等，社区也在持续关注和优化，部分讨论的思路也值得关注、借鉴和参考。

四、总结

NameNode重启甚至全集群重启在整个Hadoop集群的生命周期内是比较频繁的运维操作，优化重启时间可以极大提升运维效率，避免可能存在的风险。本文通过分析NameNode启动流程，并结合实践过程简单罗列了几个供参考的有效优化点，借此希望能给实践过程提供可优化的方向和思路。

五、参考

NameNode内存全景

NameNode内存详解

Apache Hadoop

Hadoop Source

HDFS Issues

Cloudera Blog

感谢丁晓昀对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（@InfoQ，@丁晓昀），微信（微信号：InfoQChina）关注我们。