链滴

HDFS 的 HA 机制深入理解

作者：wpf375516041

原文链接：https://ld246.com/article/1569588240681

来源网站：链滴

<h2 id="背景">背景</h2>
最近修改了 zookeeper 集群的 znode 的最大数据量，需要更新 hdfs zkfc 的配置并重启，涉及 hdfs 高可用相关的知识，所以做了一些整理。
在 Hadoop2.X 之前，Namenode 是 HDFS 集群中可能发生单点故障的节点，每个 HDFS 集群只有个 namenode，一旦这个节点不可用，则整个 HDFS 集群将处于不可用状态，HDFS 高可用（HA）案就是为了解决上述问题而产生的。
<h2 id="架构">架构</h2>
在 HA HDFS 集群中会同时运行两个 Namenode，一个作为活动的 Namenode（Active），一作为备份的 Namenode（Standby）。备份的 Namenode 的命名空间与活动的 Namenode 是实同步的，所以当活动的 Namenode 发生故障而停止服务时，备份 Namenode 可以立即切换为活动态，而不影响 HDFS 集群服务

<h2 id="HA-逻辑">HA 逻辑</h2>
在一个 HA 集群中，会配置两个独立的 Namenode。在任意时刻，只有一个节点作为活动的节，另一个节点则处于备份状态。活动的 Namenode 负责执行所有修改命名空间以及删除备份数据块操作，而备份的 Namenode 则执行同步操作，以保持与活动节点命名空间的一致性。
为了使备份节点与活动节点的状态能够同步一致，两个节点都需要同一组独立运行的节点（JournalNdes，JNS）通信。当 Active Namenode 执行了修改命名空间的操作时，它会定期将执行的操作记在 editlog 中，并写入 JNS 的多数节点中。而 Standby Namenode 会一直监听 JNS 上 editlog 的化，如果发现 editlog 有改动，Standby Namenode 就会读取 editlog 并与当前的命名空间合并。发生了错误切换时，Standby 节点会保证已经从 JNS 上读取了所有 editlog 并与命名空间合并，然才会从 Standby 状态切换为 Active 状态。通过这种机制，保证了 Active Namenode 与 Standby Nmenode 之间命名空间状态的一致性，也就是第一关系链的一致性。
为了使错误切换能够很快的执行完毕，就要保证 Standby 节点也保存了实时的数据快的存储信息，就是第二关系链。这样发生错误切换时，Standby 节点就不需要等待所有的数据节点进行全量数据块报，而直接可以切换到 Active 状态。为了实现这个机制，Datanode 会同时向这两个 Namenode 送心跳以及块汇报信息。这样就实现了 Active Namenode 和 standby Namenode 的元数据就完全致，一旦发生故障，就可以马上切换，也就是热备。
这里需要注意的是 Standby Namenode 只会更新数据块的存储信息，并不会向 namenode 发送复或者删除数据块的指令，这些指令只能由 Active namenode 发送。
<h2 id="脑裂问题处理">脑裂问题处理</h2>
在 HA 架构中有一个非常重要的问题，就是需要保证同一时刻只有一个处于 Active 状态的 Namnode，否则就会出现两个 Namenode 同时修改命名空间的问提，也就是脑裂（Split-brain）。脑的 HDFS 集群很可能造成数据块的丢失，以及向 Datanode 下发错误的指令等异常情况。为了预防裂的情况，HDFS 提供了三个级别的隔离机制（fencing）:
1.共享存储隔离：同一时间只允许一个 Namenode 向 JournalNodes 写入 editlog 数据。
2.客户端隔离：同一时间只允许一个 Namenode 响应客户端的请求。
3.Datanode 隔离：同一时间只允许一个 Namenode 向 Datanode 下发名字节点指令，例如删除、制数据块指令等等。
<h2 id="共享-editlog-日志文件">共享 editlog 日志文件</h2>
在 HA 实现中还有一个非常重要的部分就是 Active Namenode 和 Standby Namenode 之间何共享 editlog 日志文件。Active Namenode 会将日志文件写到共享存储上。Standby Namenode 会实时的从共享存储读取 editlog 文件，然后合并到 Standby Namenode 的命名空间中。这样一旦ctive Namenode 发生错误，Standby Namenode 可以立即切换到 Active 状态。在 Hadoop2.6，提供了 QJM（Quorum Journal Manager）方案来解决 HA 共享存储问题。
所有的 HA 实现方案都依赖于一个保存 editlog 的共享存储，这个存储必须是高可用的，并且能被集群中所有的 Namenode 同时访问。Quorum Journa 是一个基于 paxos 算法的 HA 设计方案。
Quorum Journal 方案中有两个重要的组件。
1.JournalNoe（JN）：运行在 N 台独立的物理机器上，它将 editlog 文件保存在 JournalNode 的地磁盘上，同时 JournalNode 还对外提供 RPC 接口 QJournalProtocol 以执行远程读写 editlog 文的功能。
2。QuorumJournalManager(QJM):运行在 NameNode 上，（目前 HA 集群只有两个 Namenode

，通过调用 RPC 接口 QJournalProtocol 中的方法向 JournalNode 发送写入、排斥、同步 editlog

Quorum Journal 方案依赖于这样一个概念：HDFS 集群中有 2N+1 个 JN 存储 editlog 文件，
些 editlog 文件是保存在 JN 的本地磁盘上的。每个 JN 对 QJM 暴露 QJM 接口 QJournalProtocol
允许 Namenode 读写 editlog 文件。当 Namenode 向共享存储写入 editlog 文件时，它会通过 Q
M 向集群中所有的 J N 发送写 editlog 文件请求，当有一半以上的 J N 返回写操作成功时，即认
写成功。这个原理是基于 Paxos 算法的。
使用 Quorum Journal 实现的 HA 方案有一下优点：
1.JN 进程可以运行在普通的 PC 上，而无需配置专业的共享存储硬件。
2.不需要单独实现 fencing 机制，Quorum Journal 模式中内置了 fencing 功能。
3. Quorum Journa 不存在单点故障，集群中有 2N+1 个 Journal，可以允许有 N 个 Journal Node
死亡。
4. JN 不会因为其中一个机器的延迟而影响整体的延迟，而且也不会因为 JN 数量的增多而影响性能
因为 Namenode 向 JournalNode 发送日志是并行的）
当 HA 集群中发生 Namenode 异常切换时，需要在共享存储上 fencing 上一个活动的节点以保
该节点不能再向共享存储写入 editlog。基于 Quorum Journal 模式的 H A 提供了 epoch number
解决互斥问题，这个概念可以在分布式文件系统中找到。epoch number 具有以下几个性质。
1.当一个 Namenode 变为活动状态时，会分配给他一个 epoch number。
2.每个 epoch number 都是唯一的，没有任意两个 Namenode 有相同的 epoch number。
3.epoch number 定义了 Namenode 写 editlog 文件的顺序。对于任意两个 namenode ,拥有更大
poch number 的 Namenode 被认为是活动节点。
当一个 Namenode 切换为活动状态时，它的 QJM 会向所有的 JN 发送命令，以获取该 JN 的
后一个 promise epoch 变量值。当 Q J M 接受到了集群中多于一半的 J N 回复后，它会将所接收
的最大值加一，并保存到 myepoch 中，之后 Q J M 会将该值发送给所有的 JN 并提出更新请求。
个 JN 会将该值与自身的 epoch 值相互比较，如果新的 myepoch 比较大，则 JN 更新，并返回更新
功；如果小，则返回更新失败。如果 QJM 接收到超过一半的 JN 返回成功，则设置它的 epoch num
er 为 myepoch；，否则它终止尝试为一个活动的 Namenode，并抛出异常。
当活动的 NameNode 成功获取并更新了 epoch number 后，调用任何修改 editlog 的 RPC 请
都必须携带 epoch number。当 RPC 请求到达 JN 后，JN 会将请求者的 epoch 与自身保存的 epoch
相互对比，若请求者的 epoch 更大，JN 就会更新自己的 epoch，并执行相应的操作，如果请求者的
poch 小，就会拒绝相应的请求。当集群中大多数的 JN 拒绝了请求时，这次操作就失败了。
当 HDFS 集群发生 Namenode 错误切换后，原来的 standby Namenode 将集群的 epoch number
加一后更新。这样原来的 Active namenode 的 epoch number 肯定小于这个值，当这个节点执行写
editlog 操作时，由于 JN 节点不接收 epoch number 小于自身的 promise epoch 的写请求，所以
次写请求会失败，也就达到了 fencing 的目的。
<h2 id="恢复流程">恢复流程</h2>
当 Namenode 发生主从切换时，原来的 Standby namenode 会接管共享存储并执行写 editlog
的操作。在切换之前，对于共享存储会执行以下操作：
1.fencing 原来的 Active Namenode。
2.恢复正在处理的 editlog。由于 Namenode 发生了主从切换，集群中 JN 上正在执行写入操作的 edi
log 数据可能不一致。例如，可能出现某些 JN 上的 editlog 正在写入，但是当前 Active Namenode
发生错误，这时该 JN 上的 editlog 文件就与已完成写入的 JN 不一致。在这种情况下，需要对 JN
所有状态不一致的 editlog 文件执行恢复操作，将他们的数据同步一致，并且将 editlog 文件转化为 F
NALIZED 状态。
3.当不一致的 editlog 文件完成恢复之后，这时原来的 Standby Namenode 就可以切换为 Active N
menode 并执行写 editlog 的操作。
4.写 editlog。
日志恢复操作可以分为以下几个阶段：
1.确定需要执行恢复操作的 editlog 段落：在执行恢复操作之前，QJM 会执行 newEpoch（）调用
产生新的 epoch number，JN 接收到这个请求后除了执行更新 epoch number 外，还会将该 JN 上
存的最新的 editlog 段落的 txid 返回。当集群中的大多数 JN 都发回了这个响应后，QJM 就可以确
出集群中最新的一个正在处理 editlog 段落的 txid，然后 QJM 就会对这个 txid 对应的 editlog 段落
行恢复操作了。
2.准备恢复：QJM 向集群中的所有 JN 发送 RPC 请求，查询执行恢复操作的 editlog 段落文件

所有 JN 上的状态，这里的状态包括 editlog 文件是 in-propress 还是 FINALIZED 状态，以及 editlo 文件的长度。
3.接受恢复：QJM 接收到 JN 发回的 JN 发回的响应后，会根据恢复算法选择执行恢复操作的源点。然后 QJM 会发送 RPC 请求给每一个 J N，这个请求会包含两部分信息：源 editlog 段落文件息，以及供 JN 下载这个源 editlog 段落的 url。
接收到这个 RPC 请求之后，JN 会执行以下操作：
1）同步 editlog 段落文件，如果 JN 磁盘上的 editlog 段落文件与请求中的段落文件状态不同，则 JN 会从当前请求中的 url 上下载段落文件，并替换磁盘上的 editlog 段落文件。
2）持久化恢复元数据，JN 会将执行恢复操作的 editlog 段落文件的状态、触发恢复操作的 QJM 的 e och number 等信息（恢复的元数据信息）持久化到磁盘上。
3）当这些操作都执行成功后，JN 会返回成功响应给 QJM，如果集群中的大多数 JN 都返回了成功则此次恢复操作执行成功。
4.完成 editlog 段落文件：到这步操作时，QJM 就能确定集群中大多数的 JN 保存的 editlog 文的状态已经一致了，并且 JN 持久化了恢复信息。QJM 就会向 JN 发送指令，将这个 editlog 段落文的状态转化为 FINALIZED 状态，，并且 JN 会删除持久化的恢复元数据，因为磁盘上保存的 editlog 文件信息已经是正确的了，不需要保存恢复的元数据。
<h2 id="ZKFC-对-NameNode-状态的控制">ZKFC 对 NameNode 状态的控制</h2>
FC 是要和 NN ——对应的，两个 NN 就要部署两个 FC。它负责监控 NN 的状态，并及时的把态信息写入 ZK。它通过一个独立线程周期性的调用 NN 上的一个特定接口来获取 NN 的健康状态。F 也有选择谁作为 Active NN 的权利，因为最多只有两个节点，目前选择策略还比较简单（先到先得轮换）。
<h2 id="ZKFC-功能">ZKFC 功能</h2>
1.Health monitoring
zkfc 定期对本地的 NN 发起 health-check 的命令，如果 NN 正确返回，那么这个 NN 被认为是 OK 的。否则被认为是失效节点。
2.ZooKeeper Session Management
当本地 NN 是健康的时候，zkfc 将会在 zk 中持有一个 session。如果本地 NN 又正好是 active 的那么 zkfc 还有持有一个”ephemeral”的节点作为锁，一旦本地 NN 失效了，那么这个节点将会被动删除。
3.ZooKeeper-based election
如果本地 NN 是健康的，并且 zkfc 发现没有其他的 NN 持有那个独占锁。那么他将试图去获取该锁一旦成功，那么它就需要执行 Failover，然后成为 active 的 NN 节点。Failover 的过程是：第一步对之前的 NN 执行 fence，如果需要的话。第二步，将本地 NN 转换到 active 状态。
另外：
如果一个 Active 因 HealthMonitor 监控到状态异常，这里会作出判断，先通过 Fencing 功能关闭（确保关闭或者不能提供服务），然后在 ZK 上删除它对应 ZNode。
发送上述事件后，在另外一台机器上的 ZKFC 中的 ActiveStandbyElector 会收到事件，并重新行选举（尝试创建特定 ZNode），它将获得成功并更改 NN 中状态，从而实现 Active 节点的变更。/p>
<h2 id="ZKFC-实现">ZKFC 实现</h2>

基本原理
zk 的基本特性：
(1) 可靠存储小量数据且提供强一致性
(2) ephemeral node（创建的锁节点），在创建它的客户端关闭后，可以自动删除
(3) 对于 node 状态的变化，可以提供异步的通知(watcher)
zk 在 zkfc 中可以提供的功能：
(1) Failure detector（通过 watcher 监听机制实现）：及时发现出故障的 NN，并通知 zkfc
(2) Active node locator: 帮助客户端定位哪个是 Active 的 NN
(3) Mutual exclusion of active state（通过加锁）：保证某一时刻只有一个 Active 的 NN
模块
(1) ZKFailoverController(DFSZKFailoverController): 驱动整个 ZKFC 的运转，通过向 HealthMonit r 和 ActiveStandbyElector 注册回调函数的方式，subscribe HealthMonitor 和 ActiveStandbyElec or 的事件，并做相应的处理

(2) HealthMonitor: 定期 check NN 的健康状况，在 NN 健康状况发生变化时，通过回调函数把变通知给 ZKFailoverController
(3) ActiveStandbyElector: 管理 NN 在 zookeeper 上的状态，zookeeper 上对应 node 的结点发变化时，通过回调函数把变化通知给 ZKFailoverController
(4) FailoverController: 提供做 graceful failover 的相关功能(dfs admin 可以通过命令行工具手工发 failover)

系统架构
如上图所示，通常情况下 Namenode 和 ZKFC 同布署在同一台物理机器上, HealthMonitor, FailoveController, ActiveStandbyElector 在同一个 JVM 进程中(即 ZKFC), Namenode 是一个单独的 JVM 进程。如上图所示，ZKFC 在整个系统中有几个重要的作用：
(1) Monitor and try to take active lock: 向 zookeeper 抢锁，抢锁成功的 zkfc，指导对应的 NN 为 active 的 NN; watch 锁对应的 znode，当前 active NN 的状态发生变化导致失锁时，及时抢锁努力成为 active NN
(2) Monitor NN liveness and health: 定期检查对应 NN 的状态, 当 NN 状态发生变化时，及时通过 KFC 做相应的处理
(3) Fences other NN when needed: 当前 NN 要成为 active NN 时，需要 fence 其它的 NN，不同时有多个 active NN
线程模型
ZKFC 的线程模型总体上来讲比较简单的，它主要包括三类线程，一是主线程；一是 HealthMonitor 线程; 一是 zookeeper 客户端的线程。它们的主要工作方式是：
(1) 主线程在启动所有的服务后就开始循环等待
(2) HealthMonitor 是一个单独的线程，它定期向 NN 发包，检查 NN 的健康状况
(3) 当 NN 的状态发生变化时，HealthMonitor 线程会回调 ZKFailoverController 注册进来的回调数，通知 ZKFailoverController NN 的状态发生了变化
(4) ZKFailoverController 收到通知后，会调用 ActiveStandbyElector 的 API，来管理在 zookeeper 上的结点的状态
(5) ActiveStandbyElector 会调用 zookeeper 客户端 API 监控 zookeeper 上结点的状态，发生变时，回调 ZKFailoverController 的回调函数，通知 ZKFailoverController，做出相应的变化
