Hadoop 2.0 NameNode HA和Federation实践

**一、背景**

天云趋势在2012年下半年开始为某大型国有银行的历史交易数据备份及查询提供基于Hadoop的技术解决方案，由于行业的特殊性，客户对服务的可用性有着非常高的要求，而HDFS长久以来都被单点故障的问题所困扰，直到Apache Hadoop在2012年5月发布了2.0的alpha版本，其中MRv2还很不成熟，可HDFS的新功能已经基本可用，尤其是其中的的High Availability(以下简称HA)和Federation。Cloudera也于7月制作了CDH4.0.1，包含了Hadoop 2.0的诸多新功能和组件，于是我们就基于CDH4.0.1进行了HA和Federation的测试。

此工作由我和同事张军、钱兴会共同完成。

**二、为什么需要HA和Federation**

1. **单点故障**

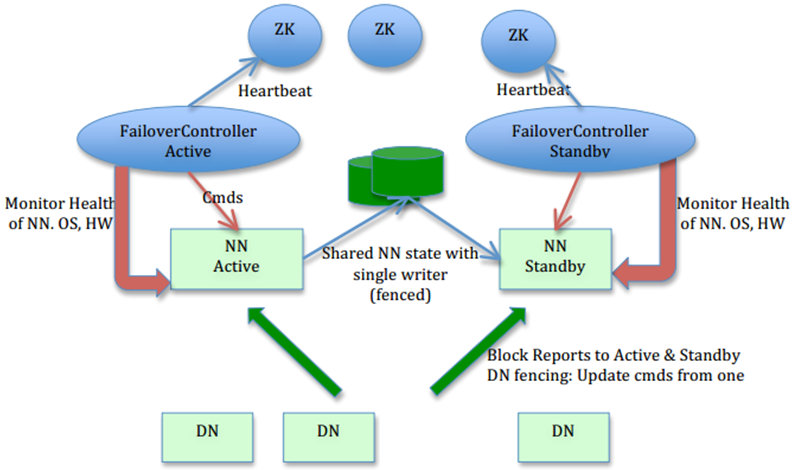
在Hadoop 2.0之前，也有若干技术试图解决单点故障的问题，我们在这里做个简短的总结

* 1. Secondary NameNode。它不是HA，它只是阶段性的合并edits和fsimage，以缩短集群启动的时间。当NameNode(以下简称NN)失效的时候，Secondary NN并无法立刻提供服务，Secondary NN甚至无法保证数据完整性：如果NN数据丢失的话，在上一次合并后的文件系统的改动会丢失。
  2. Backup NameNode ([HADOOP-4539](https://issues.apache.org/jira/browse/HADOOP-4539))。它在内存中复制了NN的当前状态，算是Warm Standby，可也就仅限于此，并没有failover等。它同样是阶段性的做checkpoint，也无法保证数据完整性。
  3. 手动把name.dir指向NFS。这是安全的Cold Standby，可以保证元数据不丢失，但集群的恢复则完全靠手动。
  4. [Facebook AvatarNode](http://hadoopblog.blogspot.tw/2010/02/hadoop-namenode-high-availability.html)。Facebook有强大的运维做后盾，所以Avatarnode只是Hot Standby，并没有自动切换，当主NN失效的时候，需要管理员确认，然后手动把对外提供服务的虚拟IP映射到Standby NN，这样做的好处是确保不会发生脑裂的场景。其某些设计思想和Hadoop 2.0里的HA非常相似，从时间上来看，Hadoop 2.0应该是借鉴了Facebook的做法。
  5. 还有若干解决方案，基本都是依赖外部的HA机制，譬如[DRBD](http://zh.wikipedia.org/zh/DRBD)，[Linux HA](http://www.linux-ha.org/wiki/Main_Page)，[VMware的FT](http://www.vmware.com/products/datacenter-virtualization/vsphere/fault-tolerance.html)等等。

1. **集群容量和集群性能**

单NN的架构使得HDFS在集群扩展性和性能上都有潜在的问题，当集群大到一定程度后，NN进程使用的内存可能会达到上百G，常用的估算公式为1G对应1百万个块，按缺省块大小计算的话，大概是64T (这个估算比例是有比较大的富裕的，其实，即使是每个文件只有一个块，所有元数据信息也不会有1KB/block)。同时，所有的元数据信息的读取和操作都需要与NN进行通信，譬如客户端的addBlock、getBlockLocations，还有DataNode的blockRecieved、sendHeartbeat、blockReport，在集群规模变大后，NN成为了性能的瓶颈。Hadoop 2.0里的HDFS Federation就是为了解决这两个问题而开发的。

**三、Hadoop 2.0里HA的实现方式**

  
图片来源： [HDFS-1623](https://issues.apache.org/jira/browse/HDFS-1623) 设计文档  
图片作者： Sanjay Radia, Suresh Srinivas

在这个图里，我们可以看出HA的大致架构，其设计上的考虑包括：

* 利用共享存储来在两个NN间同步edits信息。

以前的HDFS是share nothing but NN，现在NN又share storage，这样其实是转移了单点故障的位置，但中高端的存储设备内部都有各种RAID以及冗余硬件包括电源以及网卡等，比服务器的可靠性还是略有提高。通过NN内部每次元数据变动后的flush操作，加上NFS的close-to-open，数据的一致性得到了保证。社区现在也试图把元数据存储放到[BookKeeper](https://issues.apache.org/jira/browse/HDFS-3399)上，以去除对共享存储的依赖，Cloudera也提供了Quorum Journal Manager的实现和代码，这篇中文的blog有详尽分析：[基于QJM/Qurom Journal Manager/Paxos的HDFS HA原理及代码分析](http://yanbohappy.sinaapp.com/?p=205)

* DataNode(以下简称DN)同时向两个NN汇报块信息。

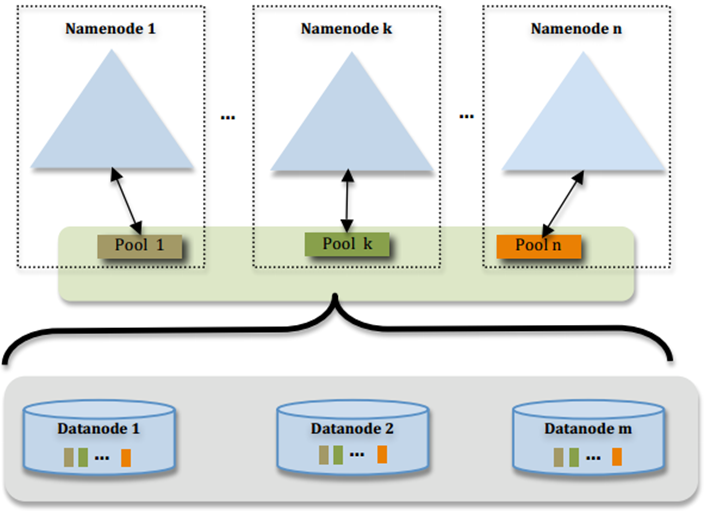
这是让Standby NN保持集群最新状态的必需步骤，不赘述。

* 用于监视和控制NN进程的FailoverController进程

显然，我们不能在NN进程内进行心跳等信息同步，最简单的原因，一次FullGC就可以让NN挂起十几分钟，所以，必须要有一个独立的短小精悍的watchdog来专门负责监控。这也是一个松耦合的设计，便于扩展或更改，目前版本里是用ZooKeeper(以下简称ZK)来做同步锁，但用户可以方便的把这个ZooKeeper FailoverController(以下简称ZKFC)替换为其他的HA方案或leader选举方案。

* [隔离(Fencing)](http://en.wikipedia.org/wiki/Fencing_(computing))，防止[脑裂](http://en.wikipedia.org/wiki/Split-brain_(computing))，就是保证在任何时候只有一个主NN，包括三个方面：
  + 共享存储fencing，确保只有一个NN可以写入edits。
  + 客户端fencing，确保只有一个NN可以响应客户端的请求。
  + DataNode fencing，确保只有一个NN可以向DN下发命令，譬如删除块，复制块，等等。

**四、Hadoop 2.0里Federation的实现方式**

  
图片来源： [HDFS-1052](https://issues.apache.org/jira/browse/HDFS-1052) 设计文档  
图片作者： Sanjay Radia, Suresh Srinivas

这个图过于简明，许多设计上的考虑并不那么直观，我们稍微总结一下

* 多个NN共用一个集群里DN上的存储资源，每个NN都可以单独对外提供服务
* 每个NN都会定义一个存储池，有单独的id，每个DN都为所有存储池提供存储
* DN会按照存储池id向其对应的NN汇报块信息，同时，DN会向所有NN汇报本地存储可用资源情况
* 如果需要在客户端方便的访问若干个NN上的资源，可以使用客户端挂载表，把不同的目录映射到不同的NN，但NN上必须存在相应的目录

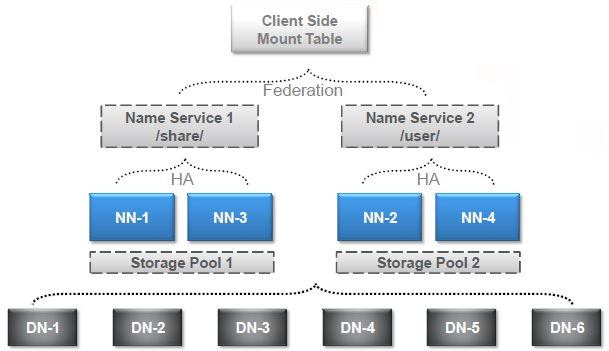
这样设计的好处大致有：

* 改动最小，向前兼容
  + 现有的NN无需任何配置改动.
  + 如果现有的客户端只连某台NN的话，代码和配置也无需改动。
* 分离命名空间管理和块存储管理
  + 提供良好扩展性的同时允许其他文件系统或应用直接使用块存储池
  + 统一的块存储管理保证了资源利用率
  + 可以只通过防火墙配置达到一定的文件访问隔离，而无需使用复杂的Kerberos认证
* 客户端挂载表
  + 通过路径自动对应NN
  + 使Federation的配置改动对应用透明

**五、测试环境**

以上是HA和Federation的简介，对于已经比较熟悉HDFS的朋友，这些信息应该已经可以帮助你快速理解其架构和实现，如果还需要深入了解细节的话，可以去详细阅读设计文档或是代码。这篇文章的主要目的是总结我们的测试结果，所以现在才算是正文开始。

为了彻底搞清HA和Federation的配置，我们直接一步到位，选择了如下的测试场景，结合了HA和Federation：



这张图里有个概念是前面没有说明的，就是NameService。Hadoop 2.0里对NN进行了一层抽象，提供服务的不再是NN本身，而是NameService(以下简称NS)。Federation是由多个NS组成的，每个NS又是由一个或两个(HA)NN组成的。在接下里的测试配置里会有更直观的例子。

图中DN-1到DN-6是六个DataNode，NN-1到NN-4是四个NameNode，分别组成两个HA的NS，再通过Federation组合对外提供服务。Storage Pool 1和Storage Pool 2分别对应这两个NS。我们在客户端进行了挂载表的映射，把/share映射到NS1，把/user映射到NS2，这个映射其实不光是要指定NS，还需要指定到其上的某个目录，稍后的配置中大家可以看到。

下面我们来看看配置文件里需要做哪些改动，为了便于理解，我们先把HA和Federation分别介绍，然后再介绍同时使用HA和Federation时的配置方式，首先我们来看HA的配置：

对于HA中的所有节点，包括NN和DN和客户端，需要做如下更改：

**HA，所有节点，hdfs-site.xml**

<property>

<name>dfs.nameservices</name>

<value>ns1</value>

<description>提供服务的NS逻辑名称，与core-site.xml里的对应</description>

</property>

<property>

<name>dfs.ha.namenodes.${NS\_ID}</name>

<value>nn1,nn3</value>

<description>列出该逻辑名称下的NameNode逻辑名称</description>

</property>

<property>

<name>dfs.namenode.rpc-address.${NS\_ID}.${NN\_ID}</name>

<value>host-nn1:9000</value>

<description>指定NameNode的RPC位置</description>

</property>

<property>

<name>dfs.namenode.http-address.${NS\_ID}.${NN\_ID}</name>

<value>host-nn1:50070</value>

<description>指定NameNode的Web Server位置</description>

</property>

以上的示例里，我们用了${}来表示变量值，其展开后的内容大致如下：

<property>

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn3</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>host-nn1:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>host-nn1:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn3</name>

<value>host-nn3:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn3</name>

<value>host-nn3:50070</value>

</property>

与此同时，在HA集群的NameNode或客户端还需要做如下配置的改动：  
  
**HA，NameNode，hdfs-site.xml**

<property>

<name>dfs.namenode.shared.edits.dir</name>

<value>file:///nfs/ha-edits</value>

<description>指定用于HA存放edits的共享存储，通常是NFS挂载点</description>

</property>

<property>

<name>ha.zookeeper.quorum</name>

<value>host-zk1:2181,host-zk2:2181,host-zk3:2181,</value>

<description>指定用于HA的ZooKeeper集群机器列表</description>

</property>

<property>

<name>ha.zookeeper.session-timeout.ms</name>

<value>5000</value>

<description>指定ZooKeeper超时间隔，单位毫秒</description>

</property>

<property>

<name>dfs.ha.fencing.methods</name>

<value>sshfence</value>

<description>指定HA做隔离的方法，缺省是ssh，可设为shell，稍后详述</description>

</property>

**HA，客户端，hdfs-site.xml**

<property>

<name>dfs.ha.automatic-failover.enabled</name>

<value>true</value>

<description>或者false</description>

</property>

<property>

<name>dfs.client.failover.proxy.provider.${NS\_ID}</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

<description>指定客户端用于HA切换的代理类，不同的NS可以用不同的代理类

以上示例为Hadoop 2.0自带的缺省代理类</description>

</property>

最后，为了方便使用相对路径，而不是每次都使用hdfs://ns1作为文件路径的前缀，我们还需要在各角色节点上修改core-site.xml：  
  
**HA，所有节点，core-site.xml**

<property>

<name>fs.defaultFS</name>

<value>hdfs://ns1</value>

<description>缺省文件服务的协议和NS逻辑名称，和hdfs-site里的对应

此配置替代了1.0里的fs.default.name</description>

</property>

接下来我们看一下如果单独使用Federation，应该如何配置，这里我们假设没有使用HA，而是直接使用nn1和nn2组成了Federation集群，他们对应的NS的逻辑名称分别是ns1和ns2。为了便于理解，我们从客户端使用的core-site.xml和挂载表入手：  
  
**Federation，所有节点，core-site.xml**

<xi:include href=“cmt.xml"/>

<property>

<name>fs.defaultFS</name>

<value>viewfs://nsX</value>

<description>整个Federation集群对外提供服务的NS逻辑名称，

注意，这里的协议不再是hdfs，而是新引入的viewfs

这个逻辑名称会在下面的挂载表中用到</description>

</property>

我们在上面的core-site中包含了一个cmt.xml文件，也就是Client Mount Table，客户端挂载表，其内容就是虚拟路径到具体某个NS及其物理子目录的映射关系，譬如/share映射到ns1的/real\_share，/user映射到ns2的/real\_user，示例如下：  
  
**Federation，所有节点，cmt.xml**

<configuration>

<property>

<name>fs.viewfs.mounttable.nsX.link./share</name>

<value>hdfs://ns1/real\_share</value>

</property>

<property>

<name>fs.viewfs.mounttable.nsX.link./user</name>

<value>hdfs://ns2/real\_user</value>

</property>

</configuration>

注意，这里面的nsX与core-site.xml中的nsX对应。而且对每个NS，你都可以建立多个虚拟路径，映射到不同的物理路径。与此同时，hdfs-site.xml中需要给出每个NS的具体信息：  
  
**Federation，所有节点，hdfs-site.xml**

<property>

<name>dfs.nameservices</name>

<value>ns1,ns2</value>

<description>提供服务的NS逻辑名称，与core-site.xml或cmt.xml里的对应</description>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1</name>

<value>host-nn1:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1</name>

<value>host-nn1:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns2</name>

<value>host-nn2:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns2</name>

<value>host-nn2:50070</value>

</property>

可以看到，在只有Federation且没有HA的情况下，配置的name里只需要直接给出${NS\_ID}，然后value就是实际的机器名和端口号，不需要再.${NN\_ID}。  
  
这里有一个情况，就是NN本身的配置。从上面的内容里大家可以知道，NN上是需要事先建立好客户端挂载表映射的目标物理路径，譬如/real\_share，之后才能通过以上的映射进行访问，可是，如果不指定全路径，而是通过映射+相对路径的话，客户端只能在挂载点的虚拟目录之下进行操作，从而无法创建映射目录本身的物理目录。所以，为了在NN上建立挂载点映射目录，我们就必须在命令行里使用hdfs协议和绝对路径：

hdfs dfs -mkdir hdfs://ns1/real\_share

上面这个问题，我在EasyHadoop的聚会上没有讲清楚，只是简单的说在NN上不要使用viewfs://来配置，而是使用hdfs://，那样是可以解决问题，但是是并不是最好的方案，也没有把问题的根本说清楚。  
  
最后，我们来组合HA和Federation，真正搭建出和本节开始处的测试环境示意图一样的实例。通过前面的描述，有经验的朋友应该已经猜到了，其实HA+Federation配置的关键，就是组合hdfs-site.xml里的dfs.nameservices以及dfs.ha.namenodes.${NS\_ID}，然后按照${NS\_ID}和${NN\_ID}来组合name，列出所有NN的信息即可。其余配置一样。  
  
**HA + Federation，所有节点，hdfs-site.xml**

<property>

<name>dfs.nameservices</name>

<value>ns1, ns2</value>

</property>

<property>

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn3</value>

</property>

<property>

<name>dfs.ha.namenodes.ns2</name>

<value>nn2,nn4</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>host-nn1:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>host-nn1:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns1.nn3</name>

<value>host-nn3:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns1.nn3</name>

<value>host-nn3:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns2.nn2</name>

<value>host-nn2:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns2.nn2</name>

<value>host-nn2:50070</value>

</property>

<property>

<name>dfs.namenode.rpc-address.ns2.nn4</name>

<value>host-nn4:9000</value>

</property>

<property>

<name>dfs.namenode.http-address.ns2.nn4</name>

<value>host-nn4:50070</value>

</property>

对于没有.${NS\_ID}，也就是未区分NS的项目，需要在每台NN上分别使用不同的值单独配置，尤其是NFS位置(dfs.namenode.shared.edits.dir)，因为不同NS必定要使用不同的NFS目录来做各自内部的HA (除非mount到本地是相同的，只是在NFS服务器端是不同的，但这样是非常不好的实践)；而像ZK位置和隔离方式等其实大可使用一样的配置。

除了配置以外，集群的初始化也有一些额外的步骤，譬如，创建HA环境的时候，需要先格式化一台NN，然后同步其name.dir下面的数据到第二台，然后再启动集群 (我们没有测试从单台升级为HA的情况，但道理应该一样)。在创建Federation环境的时候，需要注意保持${CLUSTER\_ID}的值，以确保所有NN能共享同一个集群的存储资源，具体做法是在格式化第一台NN之后，取得其${CLUSTER\_ID}的值，然后用如下命令格式化其他NN：

hadoop namenode -format -clusterid ${CLUSTER\_ID}

当然，你也可以从第一台开始就使用自己定义的${CLUSTER\_ID}值。  
  
如果是HA + Federation的场景，则需要用Federation的格式化方式初始化两台，每个HA环境一台，保证${CLUSTER\_ID}一致，然后分别同步name.dir下的元数据到HA环境里的另一台上，再启动集群。

Hadoop 2.0中的HDFS客户端和API也有些许更改，命令行引入了新的hdfs命令，hdfs dfs就等同于以前的hadoop fs命令。API里引入了新的ViewFileSystem类，可以通过它来获取挂载表的内容，如果你不需要读取挂载表内容，而只是使用文件系统的话，可以无视挂载表，直接通过路径来打开或创建文件。代码示例如下：

ViewFileSystem fsView = (ViewFileSystem) ViewFileSystem.get(conf);

MountPoint[] m = fsView.getMountPoints();

for (MountPoint m1 : m)

System.out.println( m1.getSrc() );

// 直接使用/share/test.txt创建文件

// 如果按照之前的配置，客户端会自动根据挂载表找到是ns1

// 然后再通过failover proxy类知道nn1是Active NN并与其通信

Path p = new Path("/share/test.txt");

FSDataOutputStream fos = fsView.create(p);

**六、HA测试方案和结果**  
  
Federation的测试主要是功能性上的，能用就OK了，这里的测试方案只是针对HA而言。我们设计了两个维度的测试矩阵：系统失效方式，客户端连接模型  
  
**系统失效有两种：**

1. 终止NameNode进程：ZKFC主动释放锁

模拟机器OOM、死锁、硬件性能骤降等故障

1. NN机器掉电：ZK锁超时

模拟网络和交换机故障、以及掉电本身

**客户端连接也是两种：**

1. 已连接的客户端(持续拷贝96M的文件，1M每块)

通过增加块的数目，我们希望客户端会不断的向NN去申请新的块；一般是在第一个文件快结束或第二个文件刚开始拷贝的时候使系统失效。

1. 新发起连接的客户端(持续拷贝96M的文件，100M每块)

因为只有一个块，所以在实际拷贝过程中失效并不会立刻导致客户端或DN报错，但下一次新发起连接的客户端会一开始就没有NN可连；一般是在第一个文件快结束拷贝时使系统失效。

针对每一种组合，我们反复测试10－30次，每次拷贝5个文件进入HDFS，因为时间不一定掐的很准，所以有时候也会是在第三或第四个文件的时候才使系统失效，不管如何，我们会在结束后从HDFS里取出所有文件，并挨个检查文件MD5，以确保数据的完整性。

**测试结果如下：**

* ZKFC主动释放锁
  + 5-8秒切换(需同步edits)
  + 客户端偶尔会有重试(~10%)
  + 但从未失败
* ZK锁超时
  + 15-20s切换(超时设置为10s)
  + 客户端重试几率变大(~75%)
  + 且偶有失败(~15%)，但仅见于已连接客户端
* 可确保数据完整性
  + MD5校验从未出错
  + 失败时客户端有Exception

**我们的结论是：Hadoop 2.0里的HDFS HA基本可满足高可用性**

**扩展测试**  
我们另外还(试图)测试Append时候NN失效的情形，因为Append的代码逻辑非常复杂，所以期望可以有新的发现，但是由于复杂的那一段只是在补足最尾部块的时候，所以必须在测试程序一运行起来就关掉NN，测了几次，没发现异常情况。另外我们还使用HBase进行了测试，由于WAL只是append，而且HFile的compaction操作又并不频繁，所以也没有遇到问题。

**七、HA推荐配置及其他**

**HA推荐配置**

* ha.zookeeper.session-timeout.ms ＝ 10000
  + ZK心跳是2000
  + 缺省的5000很容易因为网络拥塞或NN GC等导致误判
  + 为避免电源闪断，不要把start-dfs.sh放在init.d里
* dfs.ha.fencing.methods ＝ shell(/path/to/the/script)
  + STONITH (Shoot The Other Node In The Head)不一定可行，当没有网络或掉电的时候，是没法shoot的
  + 缺省的隔离手段是sshfence，在掉电情况下就无法成功完成，从而切换失败
  + 唯一能保证不发生脑裂的方案就是确保原Active无法访问NFS
    - 通过script修改NFS上的iptables，禁止另一台NN访问
    - 管理员及时介入，恢复原Active，使其成为Standby。恢复iptables

**客户端重试机制**  
代码可在org.apache.hadoop.io.retry.RetryPolicies.FailoverOnNetworkExceptionRetry里找到。目前的客户端在遇到以下Exception时启动重试：

// 连接失败

ConnectException

NoRouteToHostException

UnKnownHostException

// 连到了Standby而不是Active

StandbyException

其重试时间间隔的计算公式为：  
RAND(0.5~1.5) \* min (2^retryies \* baseMillis, maxMillis)  
baseMillis = dfs.client.failover.sleep.base.millis，缺省500  
maxMillis = dfs.client.failover.sleep.max.millis，缺省15000  
最大重试次数：dfs.client.failover.max.attempts，缺省15

**未尽事宜**  
关于那15%失败的情况，我们从日志和代码分析，基本确认是HA里的问题，就是Standby NN在变为Active NN的过程中，会试图重置文件的lease的owner，从而导致LeaseExpiredException: Lease mismatch，客户端遇到这个异常不会重试，导致操作失败。这是一个非常容易重现的问题，相信作者也知道，可能是为了lease安全性也就是数据完整性做的一个取舍吧：宁可客户端失败千次，不可lease分配错一次，毕竟，客户端失败再重新创建文件是一个很廉价且安全的过程。另外，与MapReduce 2.0 (YARN)的整合测试我们也没来得及做，原因是我们觉得YARN本身各个组件的HA还不完善，用它来测HDFS的HA有点本末倒置。

**关于天云趋势**

* 天云趋势由天云科技和趋势科技共同投资成立于2010年3月
* 趋势科技是Hadoop的重度使用者:
  + 2006年开始使用, 用于处理Web Reputation和Email Reputation等
  + 五个数据中心, 近1000个节点
  + 日均处理3.6T日志数据
  + 亚洲最早的主要代码贡献者
  + HBase 0.92新功能的主要开发者(coprocessors, security)
* 天云趋势的BigData业务
  + TCloud BigData Platform, 集部署、监控、配置优化、数据处理以及扩展SDK库于一体的集群管理和开发平台
  + 丰富的垂直行业开发经验， 包括电信、电力、银行等
  + 为多家大型传统软件开发厂商提供咨询服务
  + Hadoop管理员、开发者培训，也可为企业内部进行定制化培训
* 详情请访问：<http://www.tcloudcomputing.com.cn>

原文地址：<http://www.sizeofvoid.net/hadoop-2-0-namenode-ha-federation-practice-zh/>