Architecture

在一个HA集群中，可以配置两个或以上的NameNode。

一个时间只能有一个Active状态的NameNode，其他的NameNode处于Standby状态。

Active状态的NameNode负责处理客户端的所有操作，Standby状态的NameNode保持与Active状态NameNode的状态同步，准备好随时快速故障切换的准备。

为了让Standby状态的NameNode保持与Active状态的NameNode同步，所有的NameNode都要与JournalNode通信。

当Active的NameNode对命名空间进行了修改，它会将日志写入到JN中的大多数。

The Standby node is capable of reading the edits from the JNs, and is constantly watching them for changes to the edit log.

Standby状态的NameNode监控JN中edits日志的变化。当发现JN中edits发生了改变，就在自己的命名空间中执行这个操作。当发生故障切换的时候，Standby要保证读取并执行了JN中所有的edits日志并在自己的命名空间中执行完毕，才将自己由Standby切换为Active。这样就可以保证在发生故障切换的时候命名空间的元数据是同步的。

为了保证故障切换，Standby状态的NameNode也必须持有集群中block的位置信息。为了打到这一点，hadoop中配置DataNode向所有的NameNode发送block位置信息和心跳。

最关键的是要保证在HA中只能有一个NameNode处于Active状态，否则会导致数据丢失或状态不一致。

为了避免这种所谓“脑裂”的情形，JN在一个时间只允许有一个NameNode写入数据。

在发生故障切换的时候，转换为Active的NameNode接管向JN写数据的角色，同时防止另一个还在Active状态的NameNode写入数据。

## 硬件资源：

NameNode机器：所有Active和Standby NameNode运行的机器在硬件上相同，与非HA集群中的NameNode硬件一样。

JN机器：运行JN的机器。JN进程是一个轻量级的进程，一般有可能运行于已经运行了hadoop其他进程的机器上，比如NameNode/JobTracker/YARN的ResourceManager等。需要注意的是：JN进程最少3个，因为edits要写到JN的大多数上。也可以是3、5、7、9等，这样JN可以在丢失(N-1)/2的情况下正常运行。

在HA集群中，由Standby状态的NameNode进行edits日志和fsimage的合并工作，不能用Secondary NameNode、CheckpointNode或BackupNode，报错。这样在非HA集群中运行Secondary NameNode的硬件在转换为HA集群后移作他用。

## 配置：

跟联邦类似，HA配置向前兼容，允许原来的NameNode在不做改变的情况下接着用。配置设计的初衷是让集群中所有的节点具有相同的配置文件，而不用为不同类型节点的机器部署不同的配置。

配置：

hdfs-site.xml

配置文件中各个属性的顺序无关紧要。dfs.nameservices和dfs.ha.namenodes.[nameserviceID]的配置很重要。

### dfs.nameservices – 新的命名服务的逻辑名称

例如“mycluster”。名称选择随你。它用作其他配置的一部分或者作为HDFS绝对路径的一部分。

需要注意的是：如果你也在使用HDFS的联邦，这个配置也应该包含联邦中其他nameservices，HA或其他的，用逗号隔开。（HA嵌套？）

|  |
| --- |
| <property>  <name>dfs.nameservices</name>  <value>mycluster</value>  </property> |

### dfs.ha.namenodes.[nameservice ID]

nameservice中每个NameNode的唯一标识符

其中配置逗号分隔的NameNode ID。集群中DataNode会以此查找NameNode。例如，如果你想用前面的“mycluster”作为nameservice，并且想配置“nn1”、“nn2”、“nn3”作为NameNode的id，则：

|  |
| --- |
| <property>  <name>dfs.ha.namenodes.mycluster</name>  <value>nn1,nn2, nn3</value>  </property> |

注意：NameNode最少为2个，但是可以配置更多。建议3个，不要超过5个，因为有网络开销。

### dfs.namenode.rpc-address.[nameservice ID].[name node ID]

每个NameNode监听的全限定的RPC地址。

|  |
| --- |
| <property>  <name>dfs.namenode.rpc-address.mycluster.nn1</name>  <value>machine1.example.com:8020</value>  </property>  <property>  <name>dfs.namenode.rpc-address.mycluster.nn2</name>  <value>machine2.example.com:8020</value>  </property>  <property>  <name>dfs.namenode.rpc-address.mycluster.nn3</name>  <value>machine3.example.com:8020</value>  </property> |

注意：你也可以配置“servicerpc-address”

### dfs.namenode.http-address.[nameservice ID].[name node ID]

每个NameNode监听的HTTP全限定地址

|  |
| --- |
| <property>  <name>dfs.namenode.http-address.mycluster.nn1</name>  <value>machine1.example.com:9870</value>  </property>  <property>  <name>dfs.namenode.http-address.mycluster.nn2</name>  <value>machine2.example.com:9870</value>  </property>  <property>  <name>dfs.namenode.http-address.mycluster.nn3</name>  <value>machine3.example.com:9870</value>  </property> |

注意：如果开启了Hadoop的安全特性，则可以为每个NameNode配置https地址

### dfs.namenode.shared.edits.dir

用于标记NameNode要执行读写操作的JN组所在的位置

此处配置的JournalNode组用于共享edits日志存储，由Active的NameNode负责写入，Standby状态的NameNode负责读，以用于跟Active的NameNode保持同步。这个只能配置一次，但是可以配置多个地址：

qjournal://\*host1:port1\*;\*host2:port2\*;\*host3:port3\*/\*journalId\*

Journal ID是当前命名服务中的唯一标识符，它允许单一的一组JournalNode对多个命名系统提供存储服务。建议将nameservice的ID用作journalId。

例如：如果当前集群的JournalNode运行于“node1.example.com”，“node2.example.com”，“node3.example.com”并且nameservice ID是“mycluster”，则应如下配置JournalNode组：

|  |
| --- |
| <property>  <name>dfs.namenode.shared.edits.dir</name>  <value>qjournal://node1.example.com:8485;node2.example.com:8485;node3.example.com:8485/mycluster</value>  </property> |

### dfs.client.failover.proxy.provider.[nameservice ID]

HDFS客户端用以连接Active的NameNode所用到的java类。

HDFS客户端使用该属性指定的类找到Active的NameNode以发送客户端请求。hadoop自带了两个实现：ConfiguredFailoverProxyProvider和RequestHedgingProxyProvider（第一个给所有的namenode发请求，以决定哪个是active的，第二个一直调用active的namenode，直到发生故障转移）。要么自己实现，要么如下配置：

|  |
| --- |
| <property>  <name>dfs.client.failover.proxy.provider.mycluster</name>  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>  </property> |

### dfs.ha.fencing.methods

脚本列表或java类的列表，用于在发生故障切花的时候将原来的Active NameNode围起来。

系统要求一个时间只能有一个Active的NameNode。尤其是，当使用QJM的时候，只有一个NameNode可以向JournalNode写数据，所以文件系统元数据没有发生“脑裂”的可能性。然而，当发生了故障转移，很有可能原来的Active NameNode仍在执行客户端的读请求，而此NameNode如果写JournalNode的话，可能它保有的元数据已经过期了，除非kill该NameNode进程。因此，即使在使用QJM的时候也最好配置一些围栏措施。然而，然而，为了在围栏机制发生错误的时候系统仍可正常运转，最好在该列表的最后配置一个万无一失的方法。需要注意的是，即使你不使用围栏方法，你也得配置一些内容，例如：“shell(/bin/true)”。

围栏方法是一些回车符分隔的列表，在发生故障转移的时候系统会一个一个尝试执行，直至围栏成功。hadoop自带了两个方法：shell和sshfence。

sshfence – SSH到原来的ActiveNameNode并kill掉NameNode进程

sshfence选项会利用SSH登录到目标服务器利用fuser杀死正在监听的NameNode进程。需要免密钥登录到目标服务器。也就是需要配置dfs.ha.fencing.ssh.private-key-files选项，该属性的值是一个逗号分隔的SSH私钥文件。例如：

|  |
| --- |
| <property>  <name>dfs.ha.fencing.methods</name>  <value>sshfence</value>  </property>  <property>  <name>dfs.ha.fencing.ssh.private-key-files</name>  <value>/home/exampleuser/.ssh/id\_rsa</value>  </property> |

如果想使用非标准的用户名或端口来执行SSH，或者配置一个超时时间（ms，超过这个时间被认为是fencing method失败），可以如下配置：

|  |
| --- |
| <property>  <name>dfs.ha.fencing.methods</name>  <value>sshfence([[username][:port]])</value>  </property>  <property>  <name>dfs.ha.fencing.ssh.connect-timeout</name>  <value>30000</value>  </property> |

shell – 执行shell命令以包围Active NameNode。

shell围栏方法执行一个shell命令。配置如下：

|  |
| --- |
| <property>  <name>dfs.ha.fencing.methods</name>  <value>shell(/path/to/my/script.sh arg1 arg2 ...)</value>  </property> |

括号中的命令会在bash的shell中执行，不能包括任何右括号。

命令执行的环境中包含hadoop中配置的所有环境变量，用“\_”代替“.”。fencing mehtod中可以使用如下的环境变量：

|  |  |
| --- | --- |
| $target\_host | 要围栏的节点主机名 |
| $target\_port | 要围栏的节点IPC端口 |
| $target\_address | 上面两个的组合，合并为host:port |
| $target\_nameserviceid | 要围栏的NameNode的nameservice ID |
| $target\_namenodeid | 要围栏的NameNode的namenode ID |

例如：

|  |
| --- |
| <property>  <name>dfs.ha.fencing.methods</name>  <value>shell(/path/to/my/script.sh --nameservice=$target\_nameserviceid $target\_host:$target\_port)</value>  </property> |

如果命令行返回0，fencing method执行成功。如果返回其他编码，fencing method执行失败，执行列表中的下一个命令。

注意：fencing method没有超时时间，如果要使用超时时间，应该在脚本中使用。

### fs.defaultFS – 客户端使用的默认HDFS使用的路径前缀

可以使用新的HA的逻辑URI配置hadoop客户端默认使用的路径。如果你使用“mycluster”作为nameservice ID，则可以使用这个字符串作为HDFS路径的权限部分。在core-site.xml中可以如下配置：

|  |
| --- |
| <property>  <name>fs.defaultFS</name>  <value>hdfs://mycluster</value>  </property> |

### dfs.journalnode.edits.dir –

JournalNode在本地存储状态的路径

JournalNode存储edits或者其他本地状态的本地绝对路径。

为了数据冗余，要么运行多个分立的JournalNode，要么将此路径配置到本地RAID阵列中。例如：

|  |
| --- |
| <property>  <name>dfs.journalnode.edits.dir</name>  <value>/path/to/journal/node/local/data</value>  </property> |

## 部署细节：

当配置完所有必要的选项后，你必须启动所有的JournalNode。命令：hdfs --daemon start journalnode，等待启动过程的完成。

一旦所有的JournalNode都启动成功，需要同步所有HA NamoNode的元数据。

如果在新的HDFS集群中配置，需要首先在一台NameNode主机上运行(hdfs namenode -format)

如果已经格式化过NameNode了，或者从非HA转换到HA集群，需要拷贝NameNode中的元数据到另几个NameNode，在没有格式化的NameNode所在主机执行hdfs namenode -bootstrapStandby命令。运行这个命令也保证了在JournalNode中有足够的edits事务保证启动所有的NameNode。

如果你是从非HA转换到HA集群，你首先应该运行“hdfs namenode -initializeSharedEdits”，用于从NameNode本地的edits目录中初始化JournalNode。

此时你可以正常启动你HA的NameNode了。

然后你就可以通过HTTP在浏览器中访问NameNode的web页面了。需要注意的是：在配置的地址旁边就是NameNode的HA状态（“standby”或者“active”）。当一个HA的NameNode启动的时候默认是standby状态。

## 管理命令：

当HA的NameNode配置完也运行起来了，你可以使用命令行工具来管理HA的HDFS集群了。尤其是，你得熟悉“hdfs haadmin”命令。

|  |
| --- |
| Usage: haadmin  [-transitionToActive <serviceId>]  [-transitionToStandby <serviceId>]  [-failover [--forcefence] [--forceactive] <serviceId> <serviceId>]  [-getServiceState <serviceId>]  [-getAllServiceState]  [-checkHealth <serviceId>]  [-help <command>] |

可以“hdfs haadmin -help <command>”查看帮助

transitionToActive和transitionToStandby - 将NameNode切换到Active或Standby状态

这两个命令不进行围栏操作，最好少用。最好使用“hdfs haadmin -failover”。

failover – 在指定的两个NameNode之间触发一个故障切换

如果第一个NameNode处于Standby状态，这个命令简单地让第二个NameNode处于Active状态，不报错。如果第一个处于Active状态，则尝试将它置于Standby状态。如果失败了，则fencing method会执行dfs.ha.fencing.methods列表中的下一个命令，直到有一个执行成功。在这之后才会将第二个NameNode转换为Active状态。如果没有fencing method成功，第二个NameNode不会转换为Active状态，同时报错。

getServiceState – 返回指定的NameNode处于Active或Standby状态

连接给定的NameNode并获取它的状态，返回“standby”或“active”到标准输出。这个命令用于定时器作业或监控脚本等需要根据NameNode状态执行不同操作的场合。

getAllServiceState – 返回所有NameNode的状态

连接到所有配置的NameNode，在标准输出为每个NameNode打印“standby”或“active”。

checkHealth – 检查给定NameNode的健康状态

连接到指定的NameNode并检查其健康状态。NameNode会进行自我诊断，包括检查内部服务是否正常运行。如果NameNode运行正常，则返回0，非0表示运行不正常。一般监控的时候使用。

需要注意的是，该命令还没有实现，当前如果不是NameNode宕机，只返回成功。

## 自动故障切换：

上述阐述了如何进行手动故障切换。系统不会自动地进行故障切换，即使Active的NameNode失效。本节描述如何配置自动故障切换。

### 组件

自动故障切换添加了两个新的组件：zookeeper投票选举集群，ZKFailoverController进程（简称ZKFC）。

zookeeper是维护协调数据的高可用服务，监控客户端是否失效，通知客户端任何数据变化。HDFS中的自动切换需要用zookeeper做以下工作：

失败检测 – 集群中每个NameNode都跟zookeeper维护一个持久会话。如果NameNode失效，zookeeper的会话就会超时，通知其他NameNode进行故障切换。

选举Active的NameNode – zookeeper提供了选举一个node作为Active NameNode的机制。如果当前NameNode失效，另一个节点在zookeeper建立锁文件表示它作为下一个active的NameNode。

ZKFC是zookeeper的客户端，它监控并管理NameNode的状态。每个NameNode都会运行一个ZKFC进程，ZKFC进程主要做如下的工作：

健康监控 – ZKFC使用健康检查命令周期性地ping本地的NameNode。NameNode返回它的健康状态，ZKFC认为节点健康。如果节点崩溃或处于不健康的状态，健康监控标记它为不健康。

zookeeper会话管理 – 当本地NameNode处于健康状态，ZKFC跟zookeeper维护一个会话。如果本地NameNode处于active状态，它会在zookeeper中保持一个“lock”的znode。该znode利用了zookeeper的瞬态节点。如果会话过期，zookeeper的锁节点自动删除。

基于zookeeper的选举 – 如果本地NameNode健康，ZKFC如果发现没有NameNode保持zookeeper的znode锁文件，它会尝试创建一个锁文件。如果成功，它就赢得了选举，它会运行一个故障切换将自己标记为active的NameNode。故障切换流程跟手动的很像：第一，对上一个active的NameNode执行围栏操作，成功后将本地NameNode切换为active状态。

### zookeeper部署

zookeeper进程一般运行于三个或五个节点。

由于zookeeper本身是一个轻量级的进程，可以运行于HDFS的NameNode运行的硬件之上。

很多情况下选择将三个zookeeper进程部署于YARN的ResourceManager所在的节点。

为了更高的性能和隔离性，最好不要将zookeeper节点数据存储于HDFS元数据所在的节点上。

The setup of ZooKeeper is out of scope for this document. We will assume that you have set up a ZooKeeper cluster running on three or more nodes, and have verified its correct operation by connecting using the ZK CLI.

zookeeper集群的搭建不在本文档中。我们认为你已经在三个节点或更多节点建立了一个zookeeper集群，并且已经通过命令行工具确认了zookeeper的正确性。

### 开始之前

开始配置自动故障切换之前，你必须关闭集群。在集群运行的状态无法从手动故障切换转到自动故障切换。

### 配置自动故障切换：

hdfs-site.xml中添加一个新属性：

|  |
| --- |
| <property>  <name>dfs.ha.automatic-failover.enabled</name>  <value>true</value>  </property> |

core-site.xml中添加一个新属性：

|  |
| --- |
| <property>  <name>ha.zookeeper.quorum</name>  <value>zk1.example.com:2181,zk2.example.com:2181,zk3.example.com:2181</value>  </property> |

该属性列出运行zookeeper服务的主机名和端口号列表。

该属性也可以通过nameservice ID后缀，为指定的nameservice配置自动故障切换。比如：在开启了联邦的集群中，你可以显式的启动指定nameservice的自动故障切换：

dfs.ha.automatic-failover.enabled.my-nameservice-id。

还有其他的一些属性用于控制自动故障切换的行为；但是大多数情况下不是必须的。

### 初始化zookeeper的HA状态

添加了上述的属性之后，下一步就是在zookeeper中进行初始化。运行命令：

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/bin/hdfs zkfc -formatZK |

它会在zookeeper中创建一个znode节点，在该znode节点中存储自动故障切换的数据。

### 使用start-dfs.sh启动集群

由于已经配置并启用了自动故障切换，start-dfs.sh脚本自动在NameNode所在的主机启动一个ZKFC进程。当ZKFC启动了，它们自动选举一个NameNode作为active的节点。

### 手动启动集群

如果你手动管理集群中的服务，你需要在NameNode所在的节点主机手动启动zkfc进程。命令如下：

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/bin/hdfs --daemon start zkfc |

### zookeeper的安全访问

如果你运行的是一个安全的集群，你也向让存储于zookeeper中的信息是安全的，这样就避免了客户端对zookeeper中元数据的修改或故障切换的错误触发。

首先需要在core-site.xml中配置如下：

|  |
| --- |
| <property>  <name>ha.zookeeper.auth</name>  <value>@/path/to/zk-auth.txt</value>  </property>  <property>  <name>ha.zookeeper.acl</name>  <value>@/path/to/zk-acl.txt</value>  </property> |

值中的@表示值没有在这一行，而是在指定的一个磁盘文件中。认证信息也可以通过CredentialProvider读取。

第一个配置文件指定了zookeeper的认证列表，比如：

|  |
| --- |
| digest:hdfs-zkfcs:mypassword |

其中hdfs-zkfcs是zookeeper中的唯一用户名，mypassword用作唯一的密码。

其次，对于认证生成一份zookeeper的ACL：

|  |
| --- |
| [hdfs]$ java -cp $ZK\_HOME/lib/\*:$ZK\_HOME/zookeeper-3.4.2.jar org.apache.zookeeper.server.auth.DigestAuthenticationProvider hdfs-zkfcs:mypassword  output: hdfs-zkfcs:mypassword->hdfs-zkfcs:P/OQvnYyU/nF/mGYvB/xurX8dYs= |

拷贝->之后的部分到文件zk-acls.txt中，前缀是“digest:”。例如：

|  |
| --- |
| digest:hdfs-zkfcs:vlUvLnd8MlacsE80rDuu6ONESbM=:rwcda |

为了让ACL生效，你需要重新运行zkfc -formatZK命令。

完成后，通过ZK CLI验证：

|  |
| --- |
| [zk: localhost:2181(CONNECTED) 1] getAcl /hadoop-ha  'digest,'hdfs-zkfcs:vlUvLnd8MlacsE80rDuu6ONESbM=  : cdrwa |

### 验证自动故障切换

为了验证自动故障切换：

1、首先找到active的NameNode。

2、在active的节点上引发一个失效。例如：你可以kill -9 <nn的PID>模拟一个JVM崩溃。或者你可以重启服务器或者拔掉网点然后再接上。

3、在发生active失效的情况下，几秒之后其他的NameNode之中应该有一个变为active状态。检测失效并触发一个自动故障切换的时间由ha.zookeeper.session-timeout.ms配置，默认是5秒。

4、如果测试没通过，配置有可能出错了。检查zkfc进程的日志以及NameNode进程的日志来检查

## 问答环节：

1、ZKFC和NameNode进程的启动有什么一定的次序吗？

没有。先启动哪个都行

2、我应该监控哪些？

首先，你应该监控每个运行NameNode的节点以保证ZKFC运行正常。在一些zookeeper失效的案例中，例如ZKFC突然退出，需要重启以保证系统还可以自动故障切换。

你也需要监控zookeeper的quorum集群中的每一台主机。如果zookeeper崩溃，就不能自动故障切换了。

3、如果zookeeper宕机，会发生什么？

如果zookeeper集群宕机，不会触发自动故障切换。然而，HDFS还会不受影响地继续运行。当zookeeper重启之后，HDFS会重新连接不受影响。

4、我可以指定哪个NameNode作为主节点吗？

不能。当前版本不支持。一般哪个NameNode启动的早，哪个NameNode作为主节点。你可以选择让集群中的NameNode以一定顺序启动，比如你向做active的NameNode首先启动。

5、在配置自动故障切换之后我如何手动触发故障切换？

即使在自动故障切换配置之后，你还可以使用相同的hdfs haadmin命令触发手动的故障切换。