NameNode需要多大的内存？

由于NameNode在内存中维护文件系统中每个文件和block的映射关系，就有可能耗尽NameNode的内存。由于内存占用依赖于每个文件的block数目，文件名长度以及文件系统中目录的数目，而且有可能hadoop的不同版本也不一样，如此则很难给出精确的内存使用统计方式。

默认情况下，1000MB内存足够至少1000,000个文件的元数据存储了。

例如，200个节点的集群中每个节点有24TB存储空间，block大小为128MB，block复制因子为3，能存储大概2000,000个block（或更多）：200×24,000,000MB/(128MB×3)。此时，NameNode内存大概需要12000MB。

你可以通过设置hadoop-env.sh中的HADOOP\_NAMENODE\_OPTS直接增加NameNode的内存占用，而不影响其他Hadoop进程。HADOOP\_NAMENODE\_OPTS允许你为NameNode的JVM设置额外的选项，例如，如果你使用的是Sun的JVM，-Xmx2000m表示为NameNode指定2000MB的内存。

如果你更改了NameNode的内存大小，还需要在Secondary NameNode中对应的HADOOP\_SECONDARYNAMENODE\_OPTS中设置相同的参数，因为SecondaryNameNode的内存占用和NameNode差不多一样。

HDFS联邦

namenode在内存中维护着文件系统中每个文件元数据以及和块的映射关系，对于一个拥有很多文件的大的集群，内存会成为系统扩展的限制因素。

从hadoop 2.x中引入的HDFS联邦允许一个集群通过添加NameNode来扩展，每个NameNode管理者一部分文件系统的命名空间。

例如，一个NameNode管理着命名空间中/user下的所有文件映射，另一个NameNode可能管理着/share下的文件映射关系。

对于联邦，每个NameNode管理着一个命名空间卷，该卷包含命名空间的元数据和一个block池，该block池保存着该命名空间中所有文件的block信息。

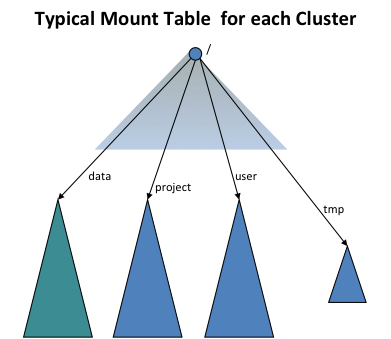
各个命名空间卷是相互独立的，意味着NameNode不用彼此通信，同时一个命名空间的失效也不会影响其他NameNode对命名空间的管理。

block池存储不分区，向所有的NameNode注册，存储来自不同block池的block数据。

要访问联邦的HDFS集群，客户端需要使用客户端侧的挂载表将文件路径映射到NameNode。

这需要通过ViewFileSystem来配置，通过viewfs://URI来访问。

# ViewFs：



## core-site.xml配置

|  |
| --- |
| <configuration>  <property>  <name>fs.defaultFS</name>  <value>viewfs://ClusterX</value>  </property>  <property>/data  <name>fs.viewfs.mounttable.ClusterX.link./data</name>  <value>hdfs://node1:8020/data</value>  </property>  <property>  <name>fs.viewfs.mounttable.ClusterX.link./project</name>  <value>hdfs://node1:8020/project</value>  </property>  <property>  <name>fs.viewfs.mounttable.ClusterX.link./user</name>  <value>hdfs://node2:8020/user</value>  </property>  <property>  <name>fs.viewfs.mounttable.ClusterX.link./tmp</name>  <value>hdfs://node2:8020/tmp</value>  </property>  <property>  <name>fs.viewfs.mounttable.ClusterX.linkFallback</name>  <value>hdfs://node2:8020/home</value>  </property>  <property>  <name>hadoop.tmp.dir</name>  <value>/var/bjsxt/hadoop/federation</value>  </property>  </configuration> |

# 配置：

## 步骤一：

在配置文件中添加dfs.nameservices参数并配置为以逗号分隔的多个NameServiceID。DataNode会据此查找NameNode。

## 步骤二：

对每个NameNode及其对应的SecondaryNameNode、BackupNode、Checkpointer等，在配置文件中添加以下以对应的NameServiceID为后缀的参数：

|  |  |
| --- | --- |
| **Daemon** | **Configuration Parameter** |
| NameNode | dfs.namenode.rpc-address dfs.namenode.servicerpc-address dfs.namenode.http-address dfs.namenode.https-address dfs.namenode.keytab.file dfs.namenode.name.dir dfs.namenode.edits.dir dfs.namenode.checkpoint.dir dfs.namenode.checkpoint.edits.dir |
| Secondary Namenode | dfs.namenode.secondary.http-address dfs.secondary.namenode.keytab.file |
| BackupNode | dfs.namenode.backup.address dfs.secondary.namenode.keytab.file |

## 例如：

|  |
| --- |
| <configuration>  <property>  <name>dfs.nameservices</name>  <value>ns1,ns2</value>  </property>  <property>  <name>dfs.namenode.rpc-address.ns1</name>  <value>node1:8020</value>  </property>  <property>  <name>dfs.namenode.http-address.ns1</name>  <value>node1:9870</value>  </property>  <property>  <name>dfs.namenode.secondary.http-address.ns1</name>  <value>node2:9868</value>  </property>  <property>  <name>dfs.namenode.rpc-address.ns2</name>  <value>node2:8020</value>  </property>  <property>  <name>dfs.namenode.http-address.ns2</name>  <value>node2:9870</value>  </property>  <property>  <name>dfs.namenode.secondary.http-address.ns2</name>  <value>node3:9868</value>  </property>  .... Other common configuration ...  </configuration> |

# 格式化NameNode：

## 步骤一：

使用下面的命令格式化一个NameNode

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/bin/hdfs namenode -format [-clusterId <cluster\_id>] |

取一个唯一的cluster\_id，不能与环境中其他集群的冲突。如果没有提供cluster\_id，系统会自动生成一个。

## 步骤二：

使用如下的命令格式化其他的NameNode：

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/bin/hdfs namenode -format -clusterId <cluster\_id> |

其中的cluster\_id必须和第一步的cluster\_id一致。如果不同，该NameNode将不会是联邦集群的一部分。

# 集群管理：

## 启动和停止：

### 启动：

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/sbin/start-dfs.sh |

### 停止：

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/sbin/stop-dfs.sh |

只要有配置文件，就可以随处启动集群。命令使用配置文件识别集群中的NameNode并在那些节点上启动NameNode进程。DataNode使用workers文件来识别。

### 平衡器：

平衡器需要对多个NameNode的情况做调整。平衡器使用如下的命令运行：

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/bin/hdfs --daemon start balancer [-policy <policy>] |

平衡策略有如下选项：

1、datanode – 默认策略。在DataNode级别做存储的平衡。

2、blockpool – 在block pool级别做存储的平衡，同时也会在DataNode级别做平衡。

需要注意的是：平衡器只对存储进行平衡，而不是命名空间。

### 退出：

需要退出的DataNode需要加入到所有NameNode的排除文件中。每个NameNode将该节点从它的block pool中剔除该节点。当所有的NameNode都将该节点剔除之后，该DataNode节点就是退出状态的。

步骤一：将排除文件分发给所有的NameNode，有如下命令

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/sbin/distribute-exclude.sh <exclude\_file> |

步骤二：刷新所有的NameNode重新加载新的排除文件：

|  |
| --- |
| [hdfs]$ $HADOOP\_HOME/sbin/refresh-namenodes.sh |

上述命令使用HDFS的配置文件识别集群中的NameNode，之后刷新所有的NameNode，加载新的排除文件。

## 集群web控制台：

跟NameNode的web页面类似，联邦情况下，集群web控制台可以在http://<any\_nn\_host:port>/dfsclusterhealth.jsp页面监控联邦集群。每个NameNode都可以访问该页面信息。

集群web控制台提供了如下的信息：

1、集群概要信息，提供了文件的数量，block的数量，总的配置的存储空间，使用的和可用的整个集群的存储信息。

2、NameNode的列表以及概要信息，包括文件数量，block，丢失的block，每个NameNode使用中的存储节点以及不在使用中的节点。同时提供了到各个NameNode web ui的超链接。

3、DataNode的剔除状态。

# java客户端

|  |
| --- |
| **package** com.jiazf.feder.demo;  **import** org.apache.hadoop.conf.Configuration; **import** org.apache.hadoop.fs.FSDataOutputStream; **import** org.apache.hadoop.fs.FileStatus; **import** org.apache.hadoop.fs.FileSystem; **import** org.apache.hadoop.fs.Path; **import** org.apache.hadoop.fs.viewfs.ViewFileSystem; **import** org.apache.hadoop.io.IOUtils;  **import** java.io.FileInputStream; **import** java.io.IOException;  **public class** FederationDemo {   **private static** FileSystem *fs*;  **private static** FileInputStream *fis*;   **public static void** main(String[] args) **throws** Exception {  Configuration conf = **new** Configuration(**true**);  *fs* = ViewFileSystem.*get*(conf);  *main1*();  *main2*();   FileStatus[] fss = *fs*.listStatus(**new** Path(**"/data"**));   **for** (FileStatus fsss :  fss) {  System.***out***.println(fsss.getPath());  }   fss = *fs*.listStatus(**new** Path(**"/user"**));   **for** (FileStatus fsss :  fss) {  System.***out***.println(fsss.getPath());  }   }  **public static void** main1() **throws** IOException {  Configuration conf = **new** Configuration(**true**);  System.***out***.println(*fs*);  FSDataOutputStream fsdos = *fs*.create(**new** Path(**"/user/hadoop.tar.gz"**));  *fis* = **new** FileInputStream(**"E:\\jars\\hadoop\\hadoop-2.5.2-src.tar.gz"**);  IOUtils.*copyBytes*(*fis*, fsdos, conf);   *fis*.close();  fsdos.close();    }   **public static void** main2() **throws** IOException {  Configuration conf = **new** Configuration(**true**);   FSDataOutputStream fsdos = *fs*.create(**new** Path(**"/data/hadoop.tar.gz"**));  *fis* = **new** FileInputStream(**"E:\\jars\\hadoop\\hadoop-2.5.2-src.tar.gz"**);  IOUtils.*copyBytes*(*fis*, fsdos, conf);   *fis*.close();  fsdos.close();  } } |