**HDFS**

## 一、基础知识

**1 HDFS 的产生背景？**

随着数据量越来越大，在一个操作系统管辖的范围内存不下了，那么就分配到更多的操 作系统管理的磁盘中，但是不方便管理和维护，迫切需要一种系统来管理多台机器上的文件，这就是分布式文件管理系统。**HDFS只是分布式文件管理系统中的一种。**

**DFS或分布式文件系统**

分布式文件系统讨论管理数据，即跨多台计算机或服务器的文件或文件夹。换句话说，DFS是一种文件系统，允许我们将数据存储在群集中的多个节点或机器上，并允许多个用户访问数据。所以基本上，它与您的机器中可用的文件系统具有相同的用途，例如用于具有NTFS（新技术文件系统）的Windows或用于具有HFS（分层文件系统）的Mac。唯一的区别是，在分布式文件系统的情况下，您将数据存储在多台机器而不是单台机器上。即使文件存储在整个网络中，DFS也可以组织和显示数据，使坐在机器上的用户感觉所有数据都存储在该机器中。

**当数据集的大小超过一台独立的物理计算机的存储能力时，就有必要对它进行分区（Partition）并存储到若干台单独的计算机上。管理网络中跨多台计算机存储的文件系统成为分布式文件系统（distributed filesystem）。该系统架构与网络之上，势必会引入网络编程的复杂性，因此分布式文件系统比普通磁盘文件系统更为复杂。例如：使文件系统能够容忍节点故障且不丢失任何数据，就是一个极大的挑战。**

**2 HDFS 到底是什么？**

HDFS（Hadoop Distributed File System）是Hadoop的核心子项目，是一个可以运行在普通硬件设备上的分布式文件系统，是分布式计算中数据存储和管理的基础，是基于流 数据模式访问和处理超大文件的需求而开发的。它所具有的高容错、高可靠性、高可扩展性、高吞吐率等特征为海量数据提供了不怕故障的存储，给超大数据集（Large Data Set） 的应用处理带来了很多便利。

Hadoop分布式文件系统或HDFS是基于Java的分布式文件系统，允许您在Hadoop集群中的多个节点上存储大量数据。因此，如果您安装Hadoop，您将HDFS作为底层存储系统来存储分布式环境中的数据。

**举个例子来理解它。想象一下，你有十台机器或十台电脑，每台机器上有1TB的硬盘。现在，HDFS表示，如果您将Hadoop作为平台安装在这十台机器上，您将获得HDFS作为存储服务。Hadoop分布式文件系统以这样的方式分发，即每台机器都有自己的存储空间来存储任何类型的数据。**

**3 HDFS 从何而来？**

HDFS 源于 Google 在2003年10月份发表的GFS（Google File System） 论文。 它其实就是 GFS 的一个克隆版本。

**4 HDFS的设计目标**

1）、硬件故障是常态，而不是异常

2）、适合流式数据访问

3）、适合大规模数据集

4）、简单的一致性模型

5）、移动计算比移动数据更划算

6）、轻便的访问异构的软硬件平台

**5 HDFS的设计**

1. 流式数据访问

HDFS构建思路决定为：一次写入、多次读取是最高效的访问模式。

1. 商用硬件

Hadoop不需要部署在高昂且高可靠的硬件上，HDFS的设计使得数据的靠可性和安全性增加。

1. 低时间延迟的数据访问

HDFS不适合此种方式，对于低延迟的访问需求，可以选择HBase。

1. 大量的小文件

Namenode用于存储文件系统的元数据，首相与Namenode的内存，对于大量的小文件，Namenode的管理压力过大。

1. 多用户写入，任意修改文件

HDFS不支持多用户写入操作，也不支持在文件任意位置进行修改，数据的添加在文件的末尾。

## 二、HDFS的架构和数据存储原理

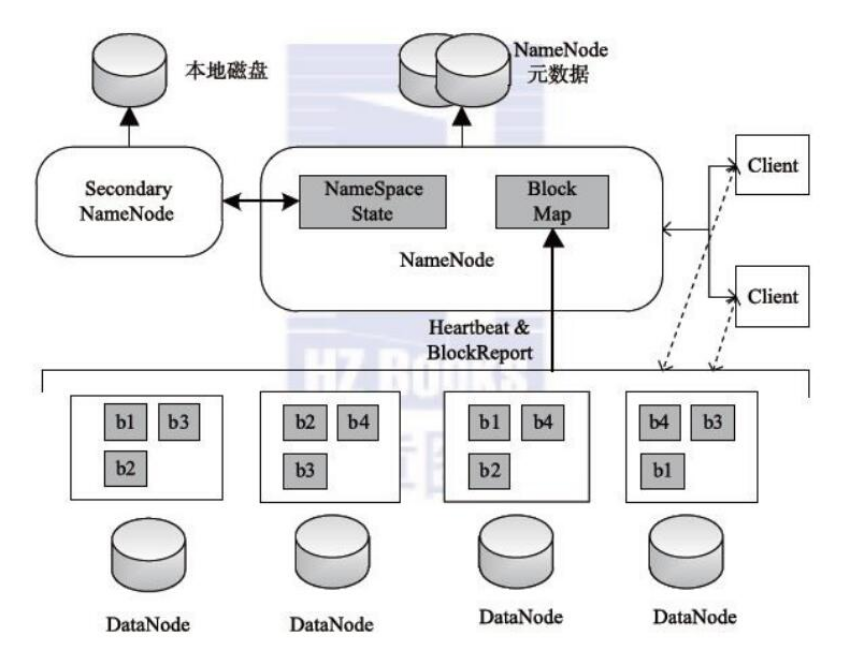
**1 HDFS是一个主/从（Mater/Slave）体系结构**

从最终用户的角度来看，它就像传统的 文件系统一样，可以通过目录路径对文件执行CRUD（Create、Read、Update和Delete） 操作。但由于分布式存储的性质，HDFS集群拥有一个NameNode和一些DataNode。

**2 NameNode管理文件系统的元数据**，DataNode存储实际的数据。客户端通过同NameNode和DataNodes的交互访问文件系统。客户端联系NameNode以获取文件的元数 据，而真正的文件I/O操作是直接和DataNode进行交互的。

**3 HDFS的架构图**

这种架构主要由四个部分组成，分别为HDFS Client、NameNode、DataNode和Secondary NameNode。下面我们分别介绍这四个组成部分。



**4 Client：就是客户端。**

1）、文件切分。文件上传 HDFS 的时候，Client 将文件切分成 一个一个的Block，然后进行存储。

2）、与 NameNode 交互，获取文件的位置信息。

3）、与 DataNode 交互，读取或者写入数据。

4）、Client 提供一些命令来管理 HDFS，比如启动或者关闭HDFS。

5）、Client 可以通过一些命令来访问 HDFS。

**5 NameNode：就是 master，它是一个主管、管理者。**

1）、管理 HDFS 的名称空间。

2）、管理数据块（Block）映射信息

3）、配置副本策略

4）、处理客户端读写请求。

**6 DataNode：就是Slave。**

NameNode 下达命令，DataNode 执行实际的操作。

1）、存储实际的数据块。

2）、执行数据块的读/写操作。

**7 Secondary NameNode：并非 NameNode 的热备。**

当NameNode 挂掉的时候，它并不能马上替换 NameNode 并提供服务。

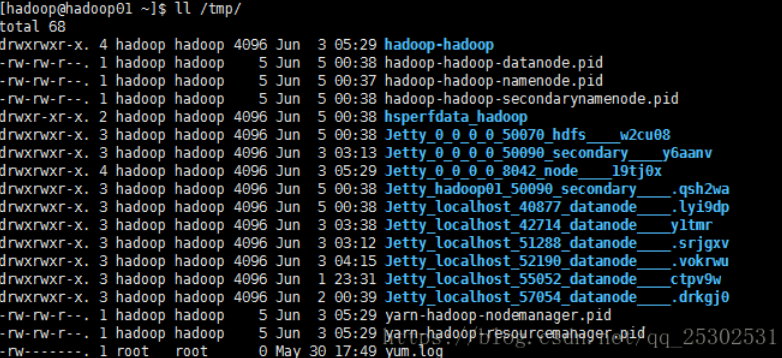
1）、辅助 NameNode，分担其工作量。

2）、定期合并 fsimage和fsedits，并推送给NameNode。

3）、在紧急情况下，可辅助恢复 NameNode。

**8 pid进程详解**

Hadoop进程的pid文件默认存放在/tmp目录中，linux默认情况下是一个月清空一次。若将pid文件删除了，对我们的HDFS运行没有影响，但是当停止集群时，若将namenode的pid删除，会发生错误显示：no namenode to stop。这时会存在残留进程，下次启动时可能启动不起来，不利于集群维护。(jps显示的是hsperfdata文件中的内容)



解决方法是不修改Linux系统内容，而是更改Hadoop的pid进程的默认存储路径，进入etc配置文件，修改hadoop-env.sh文件。

cd /opt/software/hadoop-2.8.1/etc/hadoop

# The directory where pid files are stored. /tmp by default.

# NOTE: this should be set to a directory that can only be written to by

# the user that will run the hadoop daemons. Otherwise there is the

# potential for a symlink attack.

export HADOOP\_PID\_DIR=${HADOOP\_PID\_DIR}

export HADOOP\_SECURE\_DN\_PID\_DIR=${HADOOP\_PID\_DIR}

## 三、核心知识

**1、 HDFS 具有以下优点**

**(1) 高容错性**

1) 数据自动保存多个副本。它通过增加副本的形式，提高容错性。

2) 某一个副本丢失以后，它可以自动恢复，这是由 HDFS 内部机制实现的，我们不必关

心。

**(2) 适合批处理**

1) 它是通过移动计算而不是移动数据。

2) 它会把数据位置暴露给计算框架。

**(3) 适合大数据处理**

1) 数据规模：能够处理数据规模达到 GB、TB、甚至PB级别的数据。

2) 文件规模：能够处理百万规模以上的文件数量，数量相当之大。

3) 节点规模：能够处理10K节点的规模。

**(4) 流式数据访问**

1) 一次写入，多次读取，不能修改，只能追加。

2) 它能保证数据的一致性。

**(5) 可构建在廉价机器上**

1) 它通过多副本机制，提高可靠性。

2) 它提供了容错和恢复机制。比如某一个副本丢失，可以通过其它副本来恢复。

**2、HDFS 的劣势，并不适合所有的场合**

(1) 不适合低延时数据访问

1) 比如毫秒级的来存储数据，这是不行的，它做不到。

2) 它适合高吞吐率的场景，就是在某一时间内写入大量的数据。但是它在低延时的情况 下是不行的，比如毫秒级以内读取数据，这样它是很难做到的。

改进策略

(2) 无法高效的对大量小文件进行存储

1) 存储大量小文件的话，它会占用 NameNode大量的内存来存储文件、目录和块信息。这样是不可取的，因为NameNode的内存总是有限的。

2) 小文件存储的寻道时间会超过读取时间，它违反了HDFS的设计目标。 改进策略

(3) 并发写入、文件随机修改

1) 一个文件只能有一个写，不允许多个线程同时写。

2) 仅支持数据 append（追加），不支持文件的随机修改。

**3 HDFS 如何读取文件？**

HDFS的文件读取原理，主要包括以下几个步骤：

1）、首先调用FileSystem对象的open方法，其实获取的是一个DistributedFileSystem的 实例。

2）、DistributedFileSystem通过RPC(远程过程调用)获得文件的第一批block的locations，同一block按照重复数会返回多个locations，这些locations按照hadoop拓 扑结构排序，距离客户端近的排在前面。

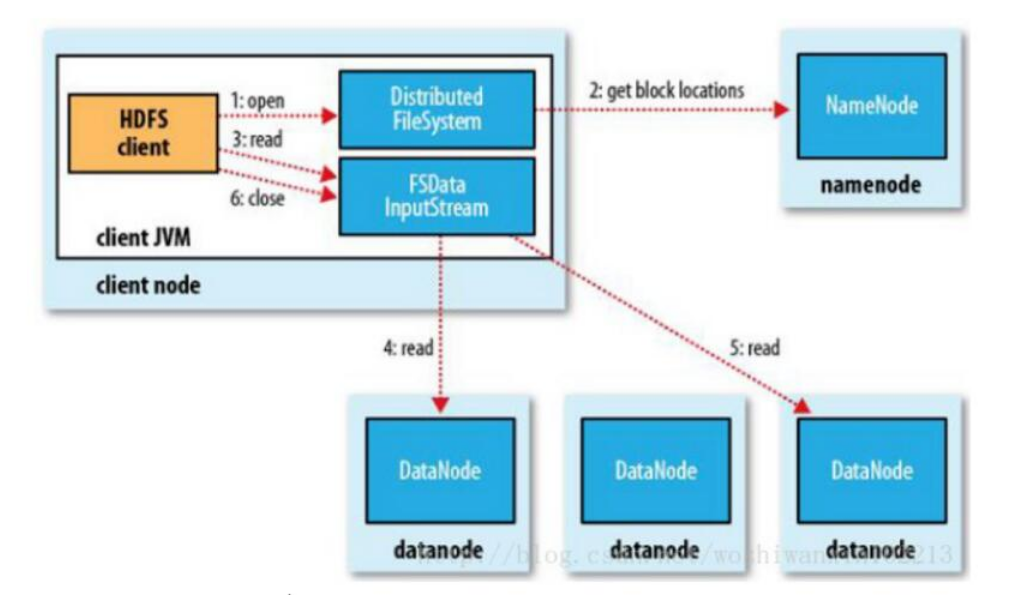
3）、前两步会返回一个FSDataInputStream对象，该对象会被封装成 DFSInputStream 对象，DFSInputStream可以方便的管理datanode和namenode数据流。客户端调用read方 法，DFSInputStream就会找出离客户端最近的datanode并连接datanode。

4）、数据从datanode源源不断的流向客户端。

5）、如果第一个block块的数据读完了，就会关闭指向第一个block块的datanode连接， 接着读取下一个block块。这些操作对客户端来说是透明的，从客户端的角度来看只是 读一个持续不断的流。

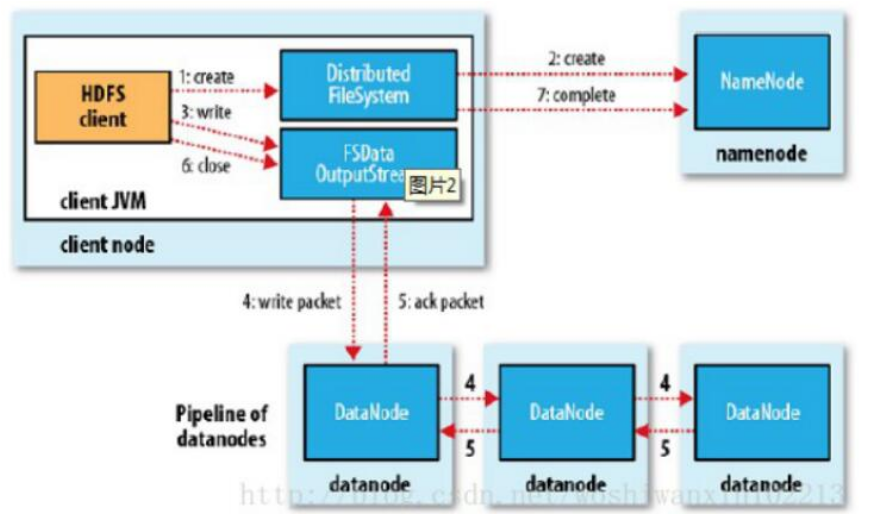
6）、如果第一批block都读完了，DFSInputStream就会去namenode拿下一批blocks的

location，然后继续读，如果所有的block块都读完，这时就会关闭掉所有的流。



**4 HDFS 如何写入文件？**

HDFS的文件写入原理，主要包括以下几个步骤：



1）.客户端通过调用 DistributedFileSystem 的create方法，创建一个新的文件。

2）.DistributedFileSystem 通过 RPC（远程过程调用）调用 NameNode，去创建一个没有blocks关联的新文件。创建前，NameNode 会做各种校验，比如文件是否存在， 客户端有无权限去创建等。如果校验通过，NameNode 就会记录下新文件，否则就会抛出IO异常。

3）.前两步结束后会返回 FSDataOutputStream 的对象，和读文件的时候相似，FSDataOutputStream 被封装成 DFSOutputStream，DFSOutputStream 可以协调

NameNode和 DataNode。客户端开始写数据到DFSOutputStream,DFSOutputStream会把数据切成一个个小packet，然后排成队列data queue。

4）.DataStreamer 会去处理接受 data queue，它先问询 NameNode 这个新的 block 最适合存储的在哪几个DataNode里，比如重复数是3，那么就找到3个最适合的DataNode，把它们排成一个 pipeline。DataStreamer 把 packet 按队列输出到管道的第一个 DataNode 中，第一个 DataNode又把 packet 输出到第二个 DataNode 中，以此类推。

5）.DFSOutputStream 还有一个队列叫 ack queue，也是由 packet 组成，等待

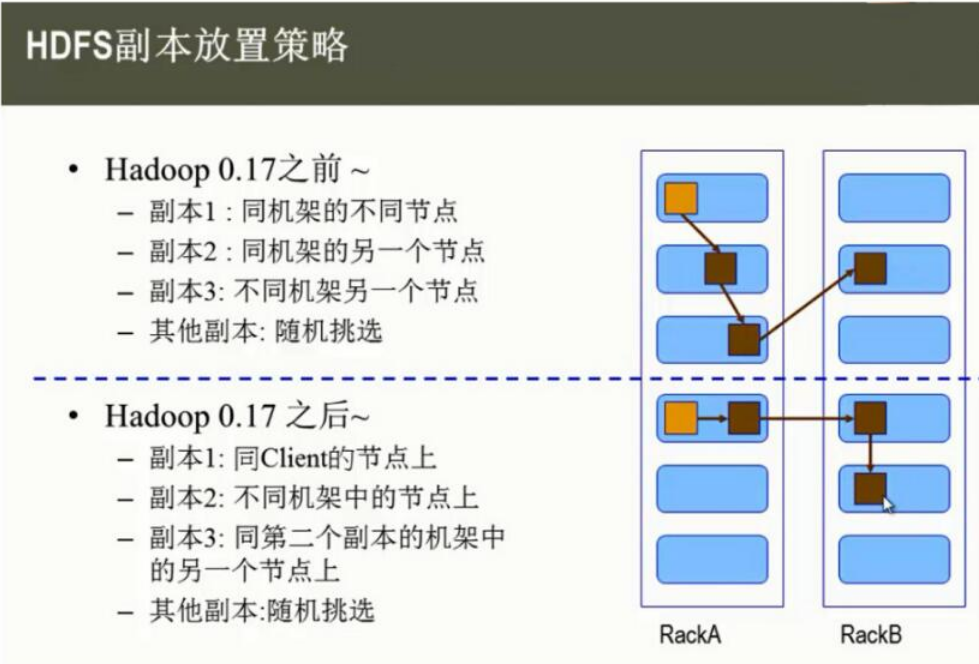
DataNode的收到响应，当pipeline中的所有DataNode都表示已经收到的时候，这时akc queue才会把对应的packet包移除掉。

6）.客户端完成写数据后，调用close方法关闭写入流。

7）.DataStreamer 把剩余的包都刷到 pipeline 里，然后等待 ack 信息，收到最后一个

ack 后，通知 DataNode 把文件标示为已完成。

**5 HDFS 副本存放策略**



1. **namenode 如何选择在哪个 datanode 存储副本（replication）？**

这里需要对可靠性、写入带宽和读取带宽进行权衡。 Hadoop 对 datanode 存储副本有自己的副本策略，在其发展过程中一共有两个版本的副本策略，分别如下所示。

**（2）Hadoop 0.17之前的副本策略**

第一个副本：存储在同机架的不同节点上。

第二个副本：存储在同机架的另外一个节点上。第三个副本：存储在不同机架的另外一个节点。其它副本：选择随机存储。

**（3）Hadoop 0.17 之后的副本策略**

第一个副本：存储在同 Client 相同节点上。第二个副本：存储在不同机架的节点上。

第三个副本：存储在第二个副本机架中的另外一个节点上。其它副本：选择随机存储。

**6 hdfs dfs命令**

l-help[cmd]  //显示命令的帮助信息

l-ls(r)<path>  //显示当前目录下所有文件

l-du(s)<path>  //显示目录中所有文件大小

l-count[-q]<path>  //显示目录中文件数量

l-mv<src><dst>  //移动多个文件到目标目录

l-cp <src><dst>  //复制多个文件到目标目录

l-rm(r)  //删除文件(夹)

l-put<localsrc><dst>  //本地文件复制到hdfs

l-copyFromLocal  //同put

l-moveFromLocal  //从本地文件移动到hdfs

l-get[-ignoreCrc]<src><localdst>  //复制文件到本地，可以忽略crc校验

l-getmerge <src><localdst>  //将源目录中的所有文件排序合并到一个文件中

l-cat<src>  //在终端显示文件内容

l-text<src>  //在终端显示文件内容

l-copyToLocal [-ignoreCrc]<src><localdst>  //复制到本地

l-moveToLocal <src><localdst>

l-mkdir<path>  //创建文件夹

l-touchz<path>  //创建一个空文件

1）、查询

使用命令：

hdfs dfs -ls / 这条执行会列出/目录下的文件和目录

hdfs dfs -ls -R /这条会列出/目录下的左右文件，由于有-R参数，会在文件夹和子文件夹下执行ls操作。

hadoop fs -ls / 查看

hadoop fs -lsr

hadoop fs -mkdir /user/haodop 创建文件夹

hadoop fs -put a.txt /user/hadoop 上传到hdfs

hadoop fs -get /user/hadoop/a.txt 从hdfs下载

hadoop fs -cp src dst 复制

hadoop fs -mv src dst 移动

hadoop fs -cat /user/hadoop/a.txt 查看文件内容

hadoop fs -rm /user/hadoop/a.txt 删除文件

hadoop fs -rmr /user/hadoop 删除文件夹

hadoop fs -text /user/hadoop/a.txt 查看文件内容

hadoop fs -copyFromLocal localsrc dst 与hadoop fs -put功能类似

hadoop fs -moveFromLocal localsrc dst 将本地文件上传到hdfs，同时删除本地文件

**hadoop fs：使用面最广，可以操作任何文件系统。**

**hadoop dfs与hdfs dfs：只能操作HDFS文件系统相关（包括与Local FS间的操作），前者已经Deprecated，一般使用后者。**

2）、添加文件夹

使用命令；

hdfs dfs -mkdir -p /xytest/testdata001/

这里需要注意的是，root用户是否拥有hdfs 的权限，如果有，我们可以继续操作，如果没有，我们可以直接使用hdfs用户，默认安装完cdh后，用户hdfs拥有所有的权限。如果对于权限有不明白的，推荐看一下我的另一篇文章：

http://blog.csdn.net/wild46cat/article/details/69664376

3）、增加文件

使用命令：

hdfs dfs -copyFromLocal ~/123.txt /xytest/testdata001/

4）、查看hdfs文件中的内容

使用命令：

hdfs dfs -cat /xytest/testdata001/123.txt

或者，可以把hdfs中的文件copy到本地

使用命令：

hdfs dfs -copyToLocal /xytest/testdata001/123.txt ~/222.txt

5）、删除文件

使用命令：

hdfs dfs -rm -f /xytest/testdata001/123.txt

6）、删除文件夹

使用命令：

hdfs dfs -rm -r /xytest/testdata001

## 四、HDFS的新特性HA

**1 HDFS的HA机制**

Hadoop 2.2.0 版本之前，NameNode是HDFS集群的单点故障点，每一个集群只有一个NameNode ，如果这个机器或者进程不可用，整个集群就无法使用，直到重启

NameNode或者新重启一个NameNode节点 。影响HDFS集群不可用主要包括以下两种情况。

(1) 类似机器跌宕这样的意外情况将导致集群不可用，只有重启NameNode之后才可使用。

(2) 计划内的软件或硬件升级（NameNode节点）将导致集群在短时间范围内不可用。

HDFS的高可用性（HA ,High Availability）就可以解决上述问题，通过提供选择运行在同一集群中的一个热备用的 "主/备"两个冗余NameNode，允许在机器宕机或系统维护的时候， 快速转移到另一个NameNode。

**2 典型的HA集群**

一个典型的HA集群，两个单独的机器配置为NameNodes，在任何时候，一个NameNode 处于活动状态，另一个处于待机状态，活动的NameNode负责处理集群中所有客户端的操 作，待机时仅仅作为一个slave，保持足够的状态，如果有必要提供一个快速的故障转移。

为了保持备用节点与活动节点状态的同步，目前的实现需要两个节点同时访问一个共享存储 设备（例如从NASNFS挂载）到一个目录。将有可能在未来的版本中放宽此限制。

当活动节点对命名空间进行任何修改，它将把修改记录写到共享目录下的一个日志文件，备 用节点会监听这个目录，当发现更改时，它会把修改内容同步到自己的命名空间。备用节点 在故障转移时，它将保证已经读取了所有共享目录内的更改记录，保证在发生故障前的状态 与活动节点保持完全一致。为了提供快速的故障转移，必须保证备用节点有最新的集群中块的位置信息，为了达到这一 点，Datanode节点需要配置两个nameNode的位置，同时发送块的位置信息和心跳信息到 两个nameNode。任何时候只有一个namenode处于活动状态，对于HA集群的操作是至关重要的，否则两个节 点之间的状态就会产生冲突，数据丢失或其它不正确的结果，为了达到这个目的或者所谓的“裂脑场景”出现，管理员必须为共享存储配置至少一个（fencing）方法。在宕机期间，如果不能确定之间的活动节点已经放弃活动状态，fencing进程负责中断以前的活动节点编辑存储的共享访问。这可以防止任何进一步的修改命名空间，允许新的活动节点安全地进行故障转移。

**3 HA架构**

HA架构解释如下：

1）、只有一个NameNode是Active的，并且只有这个ActiveNameNode能提供服务，改变NameNode。以后可以考虑让StandbyNameNode提供读服务。

2）、提供手动Failover，在升级过程中，Failover在NameNode-DataNode之间写不变的情况下才能生效。

3）、在之前的NameNode重新恢复之后，不能提供failback。

4）、数据一致性比Failover更重要。

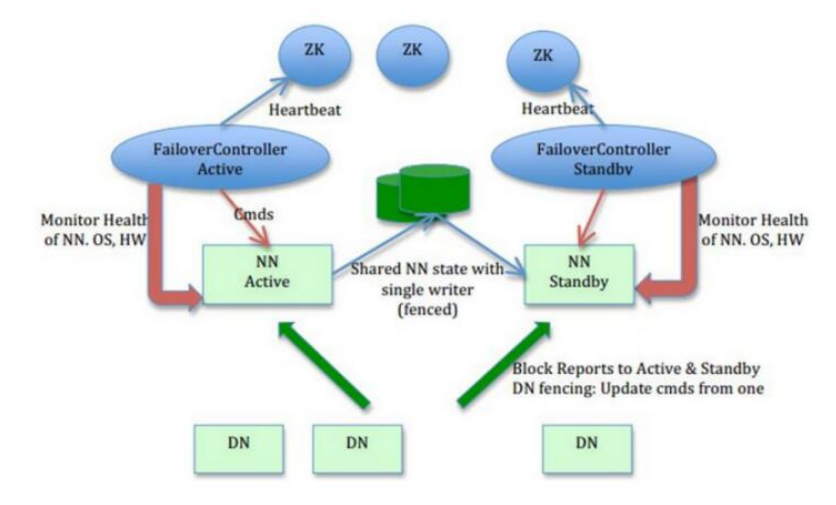
5）、尽量少用特殊的硬件。

6）、HA的设置和Failover都应该保证在两者操作错误或者配置错误的时候，不得导致数据损 坏。

7）、NameNode的短期垃圾回收不应该触发Failover。

8）、DataNode会同时向NameNodeActive和NameNodeStandby汇报块的信息。

NameNodeActive和NameNodeStandby通过NFS备份MetaData信息到一个磁盘上面。



**4 为什么会有HA机制**

**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

A、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。

B、Backup NameNode (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。

C、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。

D、Facebook AvatarNode。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。

E、还有若干解决方案，基本都是依赖外部的HA机制，譬如DRBD，Linux HA，VMware的

FT等等。

**2）、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。

**HA方案解决的是单点故障问题，而Fdederation解决的是单命名空间问题。集群的全部元数据都存放在一个名称节点上，当集群足够大的时候，这个名称节点就成了性能的瓶颈。而且这种设计不能进行 的隔离，用户的所有操作都必须由这一个节点来处理。HDFS Federation就是使HDFS支持多个命名空间，并且允许在HDFS中同时存在多个NameNode。**

## 五、HDFS的新特性Federation

**1 单个Namenode的HDFS架构的局限性**

**1）. Namespace（命名空间）的限制**

由于Namenode在内存中存储所有的元数据（metadata），因此单个Namenode所能存储 的对象（文件+块）数目受到Namenode所在JVM的heap size的限制。50G的heap能够存储20亿（200 million）个对象，这20亿个对象支持4000个datanode，12PB的存储（假设文件平均大小为40MB）。 随着数据的飞速增长，存储的需求也随之增长。单个datanode从4T增长到36T，集群的尺寸增长到8000个datanode。存储的需求从12PB增长到大于100PB。

**2）. 性能的瓶颈**

由于是单个Namenode的HDFS架构，因此整个HDFS文件系统的吞吐量受限于单个

Namenode的吞吐量。毫无疑问，这将成为下一代MapReduce的瓶颈。

**3）. 隔离问题**

由于HDFS仅有一个Namenode，无法隔离各个程序，因此HDFS上的一个实验程序就很有可 能影响整个HDFS上运行的程序。那么在HDFS Federation中，可以用不同的Namespace来 隔离不同的用户应用程序，使得不同Namespace Volume中的程序相互不影响。

**4）. 集群的可用性**

在只有一个Namenode的HDFS中，此Namenode的宕机无疑会导致整个集群不可用。

**5. Namespace和Block Management的紧密耦合**

当前在Namenode中的Namespace和Block Management组合的紧密耦合关系会导致如果想要实现另外一套Namenode方案比较困难，而且也限制了其他想要直接使用块存储的应用。

**6）. 为什么纵向扩展目前的Namenode不可行？比如将Namenode的Heap空间扩大到**

**512GB。**

这样纵向扩展带来的第一个问题就是启动问题，启动花费的时间太长。当前具有50GB Heap

Namenode的HDFS启动一次大概需要30分钟到2小时，那512GB的需要多久？ 第二个潜在的问题就是Namenode在Full GC时，如果发生错误将会导致整个集群宕机。 第三个问题是对大JVM Heap进行调试比较困难。优化Namenode的内存使用性价比比较低。

**2 为什么要引入Federation**

引入Federation的最主要原因是简单，其简单性是与真正的分布式Namenode相比而言的。

Federation能够快速的解决了大部分单Namenode HDFS的问题。

Federation是简单鲁棒的设计，由于联盟中各个Namenode之间是相互独立的。Federation 整个核心设计实现大概用了3.5个月。大部分改变是在Datanode、Config和Tools，而

Namenode本身的改动非常少，这样Namenode原先的鲁棒性不会受到影响。比分布式的

Namenode简单，虽然这种实现的扩展性比起真正的分布式的Namenode要小些，但是可以 迅速满足需求。另外一个原因是Federation良好的向后兼容性，已有的单Namenode的部署 配置不需要任何改变就可以继续工作。**因此Federation（联盟）是未来可选的方案之一。在Federation架构中可以无缝的支持目前单Namenode架构中的配置。**

**3 HDFS的Federation机制**

1. **HDFS Federation使用了多个独立的Namenode/namespace来使得HDFS的命名服务能够水平扩展。**

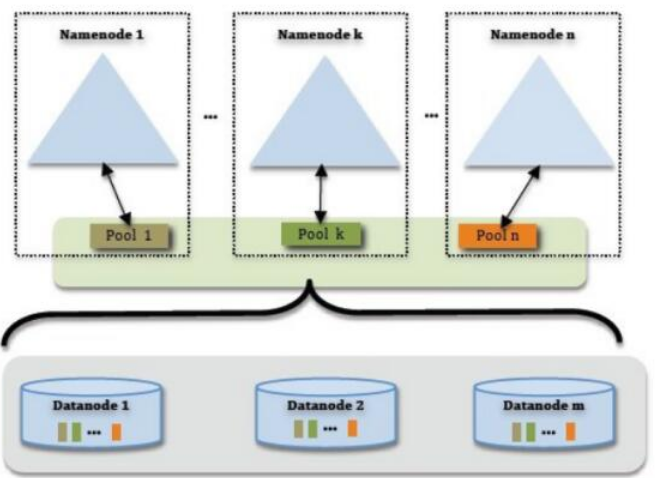
在HDFS Federation中的Namenode之间是联盟关系，他们之间相互独立且不需要相互协调。HDFS Federation中的Namenode提供了命名空间和块管理功能。HDFS Federation中的datanode被所有的Namenode用作公共存储块的地方。每一个datanode都会向所在集群中所有的Namenode注册，并且会周期性的发送心跳和块信息报 告，同时处理来自Namenode的指令。

**（2）Federation HDFS与当前HDFS的比较及改进**

当前HDFS只有一个命名空间（Namespace），它使用全部的块。而Federation HDFS中有多个独立的命名空间（Namespace），并且每一个命名空间使用一个块池（block pool）。

当前HDFS中只有一组块。而Federation HDFS中有多组独立的块。块池（block pool）就是属于同一个命名空间的一组块。当前HDFS由一个Namenode和一组datanode组成。而Federation HDFS由多个Namenode和一组datanode，每一个datanode会为多个块池（block pool）存储块。1.Block Pool(块池)。所谓Block pool(块池)就是属于单个命名空间的一组block(块)。每一个datanode为所有的block pool存储块。Datanode是一个物理概念，而block pool是一个重新将block划分的逻辑概念。同一个datanode中可以存着属于多个block pool的多个块。Block pool允许一个命名空间在不通知其他命名空间的情况下为一个新的block创建Block ID。同时，一个Namenode失效不会影响其下的datanode为其他Namenode的服务。 当datanode与Namenode建立联系并开始会话后自动建立Block pool。每个block都有一个唯一的标识， 这个标识我们称之为扩展的块ID（Extended Block ID）= BlockID+BlockID。这个扩展的块ID在HDFS集群之间都是唯一的，这为以后集群归并创造了条件。

Datanode中的数据结构都通过块池ID（BlockPoolID）索引，即datanode中的BlockMap，storage等都通过BPID索引。 在HDFS中，所有的更新、回滚都是以Namenode和BlockPool为单元发生的。**即同一HDFS Federation中不同的Namenode/BlockPool之间没有什么关系。** Hadoop V0.23版本中Block Pool的管理功能依然放在了Namenode中，将来的版本中会将Block Pool的管理功能移动的新的功能节点中。



**4 Datanode的改进**

在datanode中，对应于每个Namnode都有一条相应的线程。每个datanode会去每一个

Namenode注册，并且周期性的给所有的Namenode发送心跳及datanode的使用报告。

Datanode还会给Namenode发送其所在的block pool的block report（块报告）。由于有多个Namenode同时存在，因此任何一个Namenode都可以随时动态加入、删除和更新。

**5 Federation中的其他方面的改进**

提供了工具，对于Namenode的初始化和退役的监控和管理。 允许在datanode级别或者block pool级别的负载均衡。 Datanode的后台守护进程，为Federation所做的磁盘和目录扫描。 提供了显示Namenode的Block pool的使用状态的Web UI。 还提供了对全部集群存储使用状态的UI展示。 在Web UI中列出了所有的Namenode及其细节，如Namenode-BlockPoolID和存储的使用状态，失去联系的、活的和死的块信息。还有前往各个Namenode Web UI的链接。 Datanode退役状态的展示。

**6 多命名空间的管理问题**

在一个集群中需要唯一的命名空间还是多个命名空间，核心问题命名空间中数据的共享和访 问的问题。使用全局唯一的命名空间是解决数据共享和访问的一种方法。在多命名空间下， 我们还可以使用Client Side Mount Table方式做到数据共享和访问。

**7 Namespace Volume（命名空间卷）**

一个Namespace和它的Block Pool合在一起称作Namespace Volume。Namespace

Volume是一个独立完整的管理单元。当一个Namenode/Namespace被删除，与之相对应 的Block Pool也也被删除。在升级时每一个Namespace Volume也会整体作为一个单元。

**8 ClusterID**

在HDFS Federation中添加了Cluster ID用来区分集群中的每个节点。当格式化一个

Namenode时，这个ClusterID会自动生成或者手动提供。在格式化同一集群中其他

Namenode时会用到这个ClusterID。

**9 HDFS Federation对老版本的HDFS是兼容的**

这种兼容性可以使得已有的Namenode配置不需要任何改变继续工作。

**HDFS Federation的不足**

**HDFS Federation并没有完全解决单点故障问题。虽然namenode/namespace存在多个，但对于单个namenode来说，仍然存在单点故障。**

**如果某个namenode挂掉了，其管理的相应文件便不可以访问。**

**Federation中每个namenode仍然像之前一样，配有一个secondary namenode，以便主namenode挂掉后，用于还原元数据信息**

## 六、Java和HDFS

**1 Jar包引入，pom.xml：**

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-common</artifactId>

<version>2.8.0</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-hdfs</artifactId>

<version>2.8.0</version>

</dependency>

**2 将本地文件上传到hdfs服务器:**

/\*\*

\* 上传文件到hdfs上

\*/

@Test

public void upload() throws IOException {

Configuration conf = new Configuration();

conf.set("fs.defaultFS","hdfs://hzq:9000");

FileSystem fs = FileSystem.get(conf);

fs.copyFromLocalFile(new Path("/home/hzq/jdk1.8.tar.gz"),new Path("/demo"));

}

解析：在开发中我没有引入“core-site.xml”配置文件，所以在本地调用时使用conf进行配置“conf.set("fs.defaultFS","hdfs://hzq:9000");“，下面雷同。

**3 将hdfs上文件下载到本地：**

/\*\*

\* 将hdfs上文件下载到本地

\*/

@Test

public void download() throws IOException {

Configuration conf = new Configuration();

conf.set("fs.defaultFS","hdfs://hzq:9000");

FileSystem fs = FileSystem.newInstance(conf);

fs.copyToLocalFile(new Path("/java/jdk1.8.tar.gz"),new Path("/home/hzq/"));

}

**4 删除hdfs上指定文件：**

/\*\*

\* 删除hdfs上的文件

\* @throws IOException

\*/

@Test

public void removeFile() throws IOException {

Configuration conf = new Configuration();

conf.set("fs.defaultFS","hdfs://hzq:9000");

FileSystem fs = FileSystem.newInstance(conf);

fs.delete(new Path("/demo/jdk1.8.tar.gz"),true);

}

**5 在hdfs上创建文件夹：**

/\*\*

\* 在hdfs更目录下面创建test1文件夹

\* @throws IOException

\*/

@Test

public void mkdir() throws IOException {

Configuration conf = new Configuration();

conf.set("fs.defaultFS","hdfs://hzq:9000");

FileSystem fs = FileSystem.newInstance(conf);

fs.mkdirs(new Path("/test1"));

}

**6 列出hdfs上所有的文件或文件夹：**

@Test

public void listFiles() throws IOException {

Configuration conf = new Configuration();

conf.set("fs.defaultFS","hdfs://hzq:9000");

FileSystem fs = FileSystem.newInstance(conf);

// true 表示递归查找 false 不进行递归查找

RemoteIterator<LocatedFileStatus> iterator = fs.listFiles(new Path("/"), true);

while (iterator.hasNext()){

LocatedFileStatus next = iterator.next();

System.out.println(next.getPath());

}

System.out.println("----------------------------------------------------------");

FileStatus[] fileStatuses = fs.listStatus(new Path("/"));

for (int i = 0; i < fileStatuses.length; i++) {

FileStatus fileStatus = fileStatuses[i];

System.out.println(fileStatus.getPath());

}

}

**7 综合实例**

package captain.hadoop.hdfs;

import java.io.FileInputStream;

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IOUtils;

//利用HDFS的Java接口实现目录的创建、删除，文件的上传、下载。

public class HDFSDemo {

public static final String HDFS\_PATH = "hdfs://hadoop:9000";

public static void main(String[] args) throws Exception{

//注意需要root权限，需将"root"作为其参数

FileSystem fs = FileSystem.get(new URI(HDFS\_PATH), new Configuration(), "root");

//创建文件夹

makeDirectory(fs);

//上传文件

uploadData(fs);

//另一种上传文件的方式：直接调用FileSystem的实例方法

//fs.copyFromLocalFile(new Path("D:/Eclipse\_Workspace/Hadoop/log.txt"), new Path("/log"));

//下载文件

downloadData(fs);

//删除文件(夹)

deleteFile(fs);

}

//删除文件函数

private static void deleteFile(FileSystem fs) throws Exception {

fs.delete(new Path("/testHDFS"), true);

}

//下载文件函数

private static void downloadData(FileSystem fs) throws Exception {

FSDataInputStream in = fs.open(new Path("/testHDFS/log"));

IOUtils.copyBytes(in, System.out, 1024, true);

}

//创建目录函数

private static void makeDirectory(FileSystem fs) throws Exception {

fs.mkdirs(new Path("/testHDFS"));

}

//上传文件函数

private static void uploadData(FileSystem fs) throws Exception {

FileInputStream in = new FileInputStream("D:/Eclipse\_Workspace/Hadoop/log.txt");

FSDataOutputStream out = fs.create(new Path("/testHDFS/log"));

IOUtils.copyBytes(in, out, 4096, true);

}

}