大数据离线阶段

03

一、 课程大纲

目录

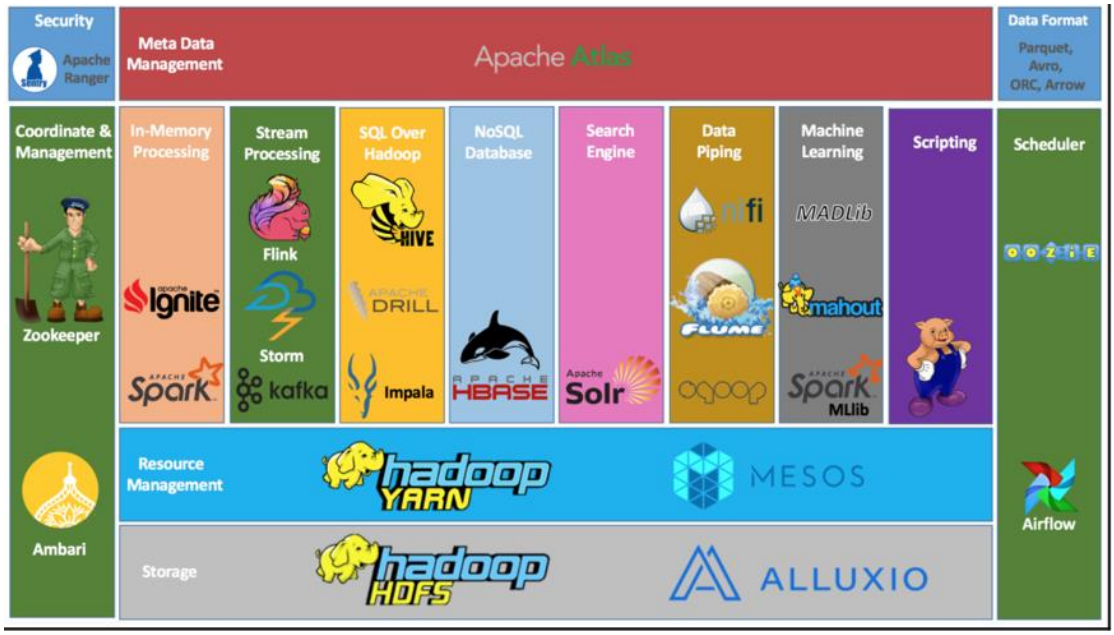
二、 HDFS 入门

1. HDFS 基本概念

1.1. HDFS 介绍

HDFS 是 Hadoop Distribute File System 的简称，意为：Hadoop 分布式文件系统。是 Hadoop 核心组件之一，作为最底层的分布式存储服务而存在。

分布式文件系统解决的问题就是大数据存储。它们是横跨在多台计算机上的存储系统。分布式文件系统在大数据时代有着广泛的应用前景，它们为存储和处理超大规模数据提供所需的扩展能力。



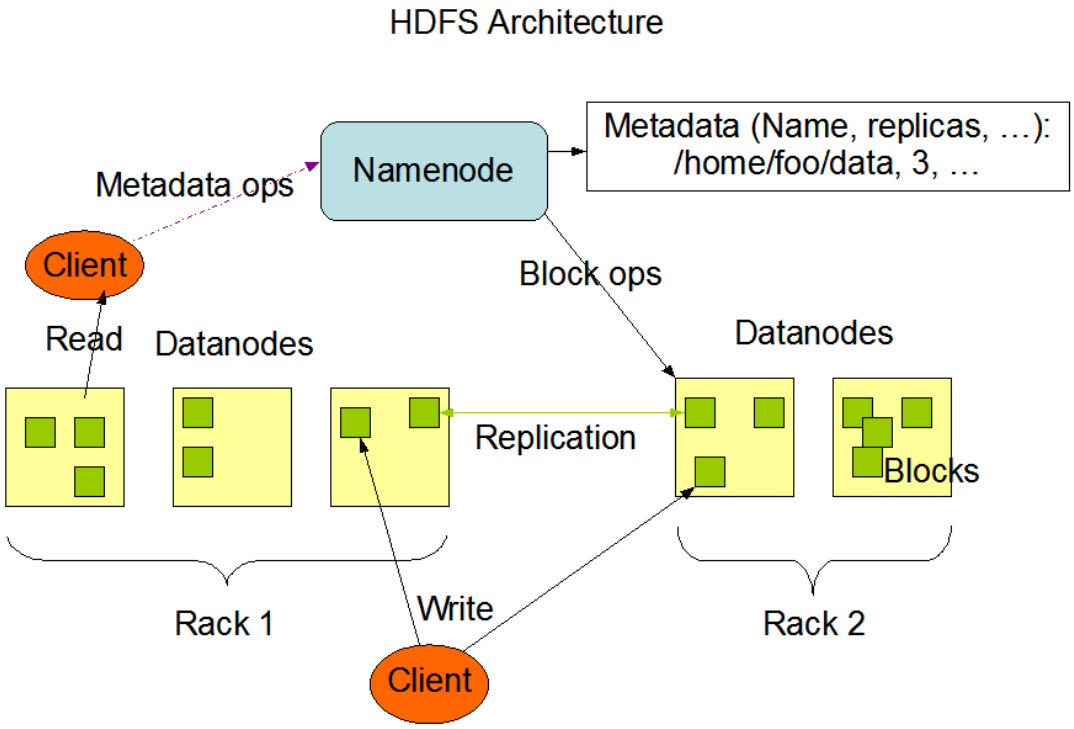
1.2. HDFS 设计目标

1) 硬件故障是常态， HDFS 将有成百上千的服务器组成，每一个组成部分都有可能出现故障。因此故障的检测和自动快速恢复是 HDFS 的核心架构目标。
2) HDFS 上的应用与一般的应用不同，它们主要是以流式读取数据。HDFS 被设计成适合批量处理，而不是用户交互式的。相较于数据访问的反应时间，更注重数据访问的高吞吐量。
3) 典型的 HDFS 文件大小是 GB 到 TB 的级别。所以，HDFS 被调整成支持大文件。它应该提供很高的聚合数据带宽，一个集群中支持数百个节点，一个集群中还应该支持千万级别的文件。
4) 大部分 HDFS 应用对文件要求的是 write-one-read-many 访问模型。一个文件一旦创建、写入、关闭之后就不需要修改了。这一假设简化了数据一致性问题，使高吞吐量的数据访问成为可能。
5) 移动计算的代价比之移动数据的代价低。一个应用请求的计算，离它操作的数据越近就越高效，这在数据达到海量级别的时候更是如此。将计算移动到数据附近，比之将数据移动到应用所在显然更好。
6) 在异构的硬件和软件平台上的可移植性。这将推动需要大数据集的应用更广泛地采用 HDFS 作为平台。

2. HDFS 重要特性

首先，它是一个文件系统，用于存储文件，通过统一的命名空间目录树来定位文件；

其次，它是分布式的，由很多服务器联合起来实现其功能，集群中的服务器有各自的角色。



2.1. master/slave 架构

HDFS 采用 master/slave 架构。一般一个 HDFS 集群是有一个 Namenode 和一定数目的 Datanode 组成。

Namenode 是 HDFS 集群主节点，Datanode 是 HDFS 集群从节点，两种角色各司其职，共同协调完成分布式的文件存储服务。

2.2. 分块存储

HDFS 中的文件在物理上是分块存储（block）的，块的大小可以通过配置参数来规定，默认大小在 hadoop2.x 版本中是 128M。

2.3. 名字空间（NameSpace）

HDFS 支持传统的层次型文件组织结构。用户或者应用程序可以创建目录，然后将文件保存在这些目录里。文件系统名字空间的层次结构和大多数现有的文件系统类似：用户可以创建、删除、移动或重命名文件。

Namenode 负责维护文件系统的名字空间，任何对文件系统名字空间或属性的修改都将被 Namenode 记录下来。

HDFS 会给客户端提供一个统一的抽象目录树，客户端通过路径来访问文件，形如：hdfs://namenode:port/dir-a/dir-b/dir-c/file.data。

2.4. Namenode 元数据管理

我们把目录结构及文件分块位置信息叫做元数据。Namenode 负责维护整个 hdfs 文件系统的目录树结构，以及每一个文件所对应的 block 块信息（block 的 id，及所在的 datanode 服务器）。

2.5. Datanode 数据存储

文件的各个 block 的具体存储管理由 datanode 节点承担。每一个 block 都可以在多个 datanode 上。Datanode 需要定时向 Namenode 汇报自己持有的 block 信息。

2.6. 副本机制

为了容错，文件的所有 block 都会有副本。每个文件的 block 大小和副本系数都是可配置的。应用程序可以指定某个文件的副本数目。副本系数可以在文件创建的时候指定，也可以在之后改变。

副本数量也可以通过参数设置 dfs.replication，默认是 3。

2.7. 一次写入，多次读出

HDFS 是设计成适应一次写入，多次读出的场景，且不支持文件的修改。

正因为如此，HDFS 适合用来做大数据分析的底层存储服务，并不适合用来做.网盘等应用，因为，修改不方便，延迟大，网络开销大，成本太高。

3. HDFS 基本操作

3.1. Shell 命令行客户端

Hadoop 提供了文件系统的 shell 命令行客户端，使用方法如下：

```
hadoop  fs  <args>
```

文件系统 shell 包括与 Hadoop 分布式文件系统（HDFS）以及 Hadoop 支持的其他文件系统（如本地 FS，HFTP FS，S3 FS 等）直接交互的各种类似 shell 的命令。所有 FS shell 命令都将路径 URI 作为参数。

URI 格式为 scheme://authority/path。对于 HDFS，该 scheme 是 hdfs，对于本地 FS，该 scheme 是 file。scheme 和 authority 是可选的。如果未指定，则使用配置中指定的默认方案。

对于 HDFS,命令示例如下：

```
hadoop fs -ls  hdfs://namenode:host/parent/child
hadoop fs -ls  /parent/child
```

fs.defaultFS 中有配置

对于本地文件系统，命令示例如下：

```
hadoop fs -ls file:///root/
```

如果使用的文件系统是 HDFS，则使用 hdfs dfs 也是可以的，此时

```
hadoop fs <args>  =  hdfs dfs <args>
```

3.2. Shell 命令选项

选项名称	使用格式	含义
-ls	-ls <路径>	查看指定路径的当前目录结构
-lsr	-lsr <路径>	递归查看指定路径的目录结构
-du	-du <路径>	统计目录下个文件大小
-dus	-dus <路径>	汇总统计目录下文件(夹)大小
-count	-count [-q] <路径>	统计文件(夹)数量
-mv	-mv <源路径> <目的路径>	移动
-cp	-cp <源路径> <目的路径>	复制
-rm	-rm [-skipTrash] <路径>	删除文件/空白文件夹
-rmr	-rmr [-skipTrash] <路径>	递归删除
-put	-put <多个 linux 上的文件> <hdfs 路径>	上传文件
-copyFromLocal	-copyFromLocal <多个 linux 上的文件> <hdfs 路径>	从本地复制
-moveFromLocal	-moveFromLocal <多个 linux 上的文件> <hdfs 路径>	从本地移动
-getmerge	-getmerge <源路径> <linux 路径>	合并到本地
-cat	-cat <hdfs 路径>	查看文件内容
-text	-text <hdfs 路径>	查看文件内容
-copyToLocal	-copyToLocal [-ignoreCrc] [-crc] [hdfs 源路径] [linux 目的路径]	从本地复制
-moveToLocal	-moveToLocal [-crc] <hdfs 源路径> <linux 目的路径>	从本地移动
-mkdir	-mkdir <hdfs 路径>	创建空白文件夹
-setrep	-setrep [-R] [-w] <副本数> <路径>	修改副本数量
-touchz	-touchz <文件路径>	创建空白文件
-stat	-stat [format] <路径>	显示文件统计信息
-tail	-tail [-f] <文件>	查看文件尾部信息
-chmod	-chmod [-R] <权限模式> [路径]	修改权限
-chown	-chown [-R] [属主][:[属组]] 路径	修改属主
-chgrp	-chgrp [-R] 属组名称 路径	修改属组
-help	-help [命令选项]	帮助

3.3. Shell 常用命令介绍

-ls

使用方法：hadoop fs -ls [-h] [-R] <args>

功能：显示文件、目录信息。

示例：hadoop fs -ls /user/hadoop/file1

-mkdir

使用方法：hadoop fs -mkdir [-p] <paths>

功能：在 hdfs 上创建目录，-p 表示会创建路径中的各级父目录。

示例：hadoop fs -mkdir -p /user/hadoop/dir1

-put

使用方法：hadoop fs -put [-f] [-p] [-|<localsrc1> ..]. <dst>

功能：将单个 src 或多个 srcs 从本地文件系统复制到目标文件系统。

-p：保留访问和修改时间，所有权和权限。

-f：覆盖目的地（如果已经存在）

示例：hadoop fs -put -f localfile1 localfile2 /user/hadoop/hadoopdir

-get

使用方法：hadoop fs -get [-ignorecrc] [-crc] [-p] [-f] <src> <localdst>

-ignorecrc：跳过对下载文件的 CRC 检查。

-crc：为下载的文件写 CRC 校验和。

功能：将文件复制到本地文件系统。

示例：hadoop fs -get hdfs://host:port/user/hadoop/file localfile

-appendToFile

使用方法：hadoop fs -appendToFile <localsrc> ... <dst>

功能：追加一个文件到已经存在的文件末尾

示例：hadoop fs -appendToFile localfile /hadoop/hadoopfile

-cat

使用方法：hadoop fs -cat [-ignoreCrc] URI [URI ...]

功能：显示文件内容到 stdout

示例：hadoop fs -cat /hadoop/hadoopfile

-tail

使用方法：hadoop fs -tail [-f] URI

功能：将文件的最后一千字节内容显示到 stdout。

-f 选项将在文件增长时输出附加数据。

示例：hadoop fs -tail /hadoop/hadoopfile

-chgrp

使用方法：hadoop fs -chgrp [-R] GROUP URI [URI ...]

功能：更改文件组的关联。用户必须是文件的所有者，否则是超级用户。

-R 将使改变在目录结构下递归进行。

示例：hadoop fs -chgrp othergroup /hadoop/hadoopfile

-chmod

功能：改变文件的权限。使用-R 将使改变在目录结构下递归进行。

示例：hadoop fs -chmod 666 /hadoop/hadoopfile

-chown

功能：改变文件的拥有者。使用-R 将使改变在目录结构下递归进行。

示例：hadoop fs -chown someuser:somegrp /hadoop/hadoopfile

-cp

功能：从 hdfs 的一个路径拷贝 hdfs 的另一个路径

示例： hadoop fs -cp /aaa/jdk.tar.gz /bbb/jdk.tar.gz.2

-mv

功能：在 hdfs 目录中移动文件

示例： hadoop fs -mv /aaa/jdk.tar.gz /

-getmerge

功能：合并下载多个文件

示例：比如 hdfs 的目录 /aaa/下有多个文件:log.1, log.2,log.3,...

```
hadoop fs -getmerge /aaa/log.* ./log.sum
```

-rm

功能：删除指定的文件。只删除非空目录和文件。-r 递归删除。

示例：hadoop fs -rm -r /aaa/bbb/

-df

功能：统计文件系统的可用空间信息

示例：hadoop fs -df -h /

-du

功能：显示目录中所有文件大小，当只指定一个文件时，显示此文件的大小。

示例：hadoop fs -du /user/hadoop/dir1

-setrep

功能：改变一个文件的副本系数。-R 选项用于递归改变目录下所有文件的副本系数。

示例：hadoop fs -setrep -w 3 -R /user/hadoop/dir1

三、 HDFS 基本原理

1. NameNode 概述

a、 NameNode 是 HDFS 的核心。

b、 NameNode 也称为 Master。

c、 NameNode 仅存储 HDFS 的元数据：文件系统中所有文件的目录树，并跟踪整个集群中的文件。

d、 NameNode 不存储实际数据或数据集。数据本身实际存储在 DataNodes 中。

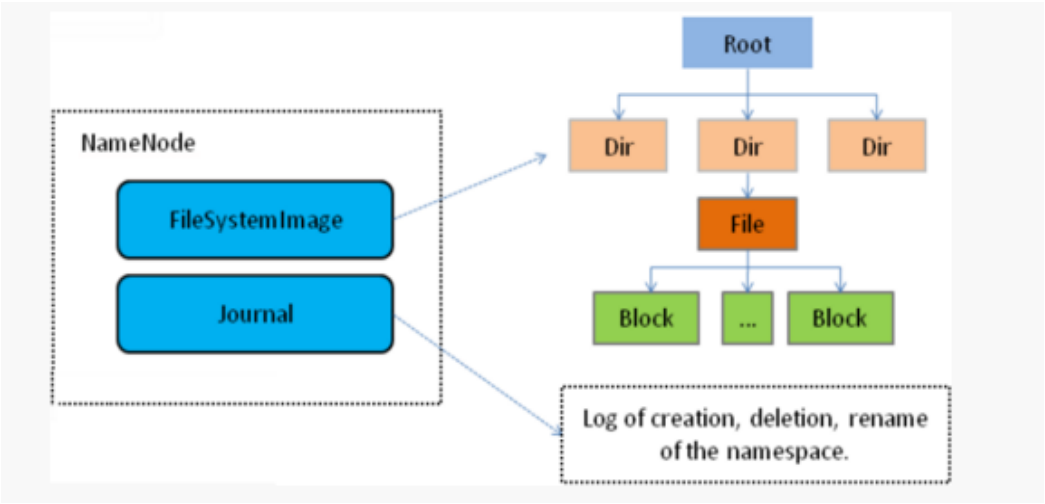
e、 NameNode 知道 HDFS 中任何给定文件的块列表及其位置。使用此信息 NameNode 知道如何从块中构建文件。

f、 NameNode 并不持久化存储每个文件中各个块所在的 DataNode 的位置信息，这些信息会在系统启动时从数据节点重建。

g、 NameNode 对于 HDFS 至关重要，当 NameNode 关闭时，HDFS / Hadoop 集群无法访问。

h、 NameNode 是 Hadoop 集群中的单点故障。

i、 NameNode 所在机器通常会配置有大量内存（RAM）。



2. DataNode 概述

a、 DataNode 负责将实际数据存储在 HDFS 中。

b、 DataNode 也称为 Slave。

c、 NameNode 和 DataNode 会保持不断通信。

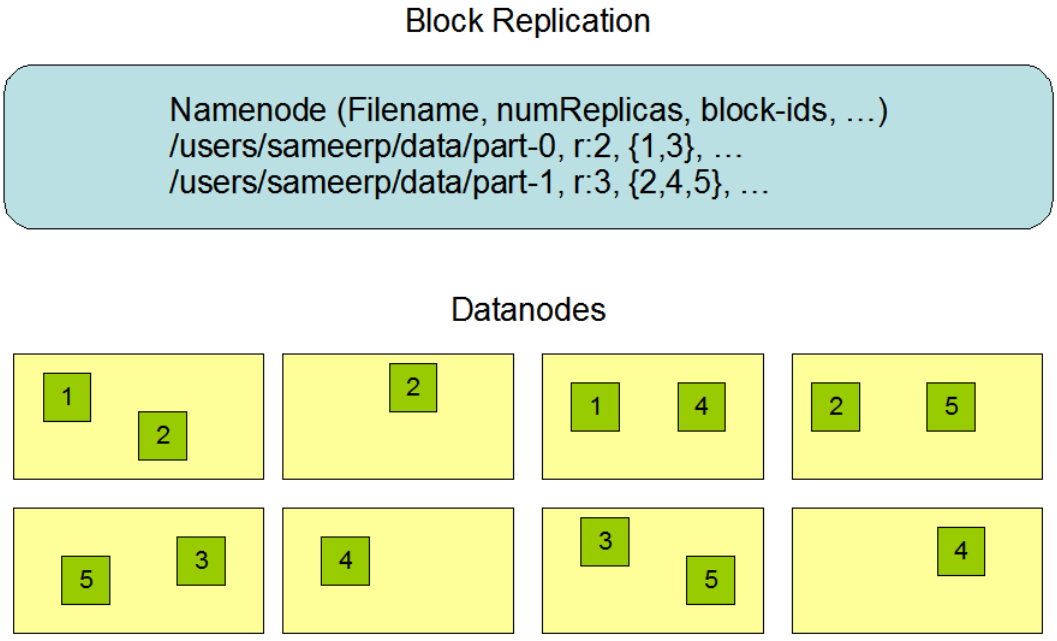
d、 DataNode 启动时，它将自己发布到 NameNode 并汇报自己负责持有的块列表。

e、 当某个 DataNode 关闭时，它不会影响数据或群集的可用性。NameNode 将安排由其他 DataNode 管理的块进行副本复制。

f、 DataNode 所在机器通常配置有大量的硬盘空间。因为实际数据存储在 DataNode 中。

g、 DataNode 会定期（dfs.heartbeat.interval 配置项配置，默认是 3 秒）向 NameNode 发送心跳，如果 NameNode 长时间没有接受到 DataNode 发送的心跳， NameNode 就会认为该 DataNode 失效。

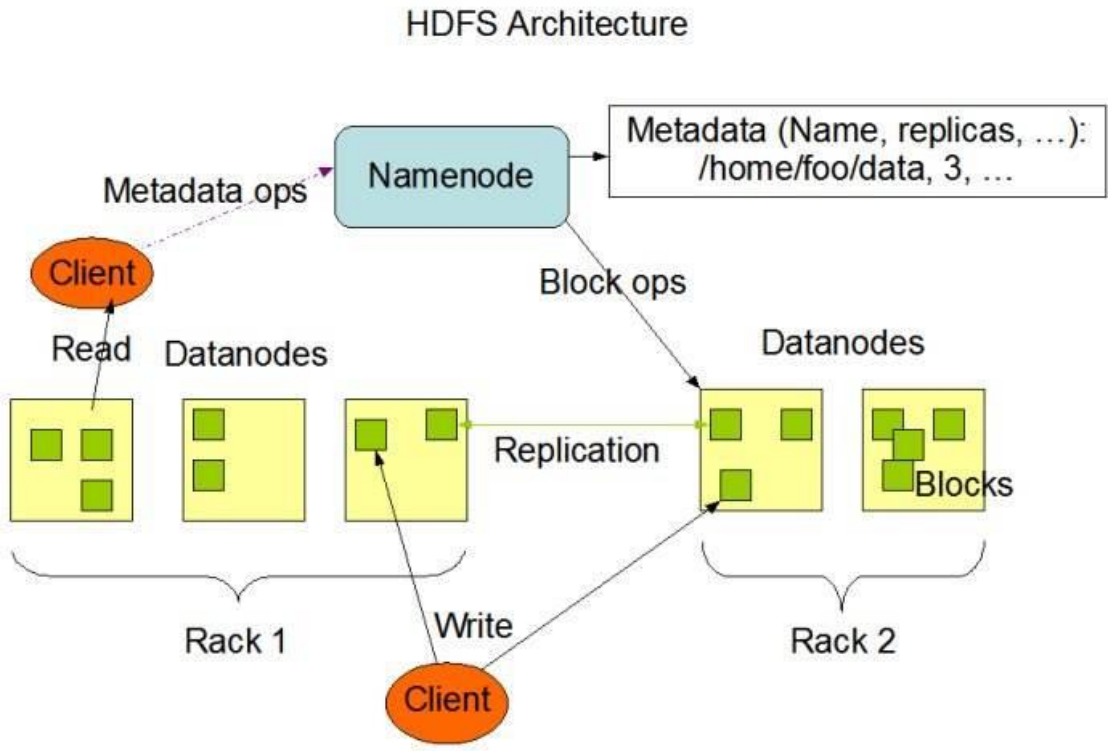
h、 block 汇报时间间隔取参数 dfs.blockreport.intervalMsec,参数未配置的话默认为 6 小时.



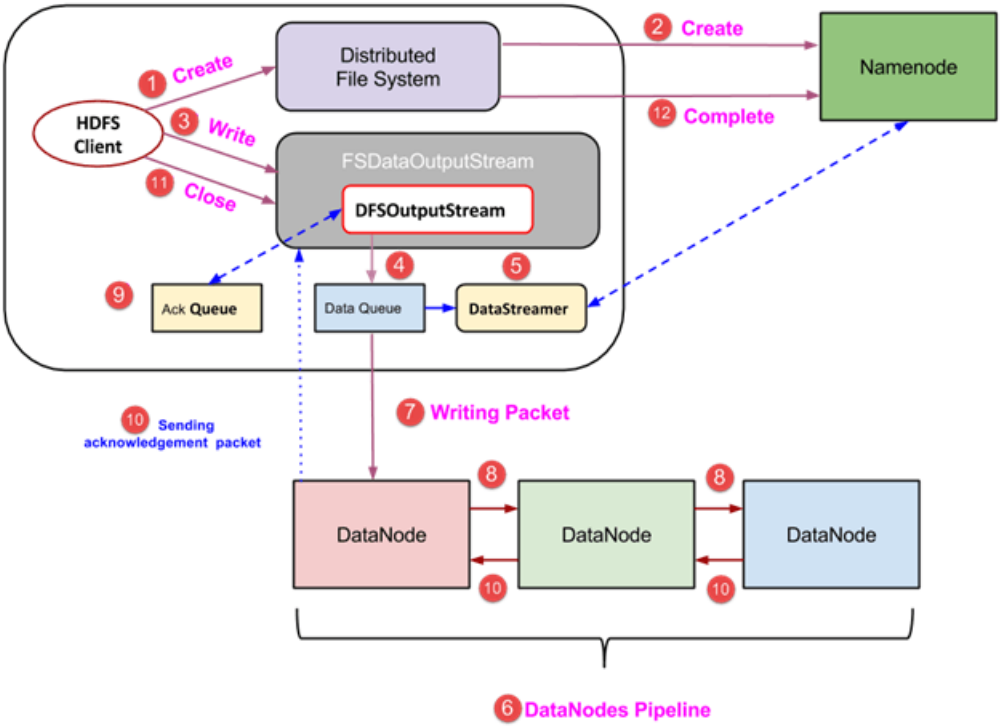
3． HDFS 的工作机制

NameNode 负责管理整个文件系统元数据；DataNode 负责管理具体文件数据块存储；Secondary NameNode 协助 NameNode 进行元数据的备份。

HDFS 的内部工作机制对客户端保持透明，客户端请求访问 HDFS 都是通过向 NameNode 申请来进行。



3.1. HDFS 写数据流程



详细步骤解析：

1、 client 发起文件上传请求，通过 RPC 与 NameNode 建立通讯，NameNode 检查目标文件是否已存在，父目录是否存在，返回是否可以上传；

2、 client 请求第一个 block 该传输到哪些 DataNode 服务器上；

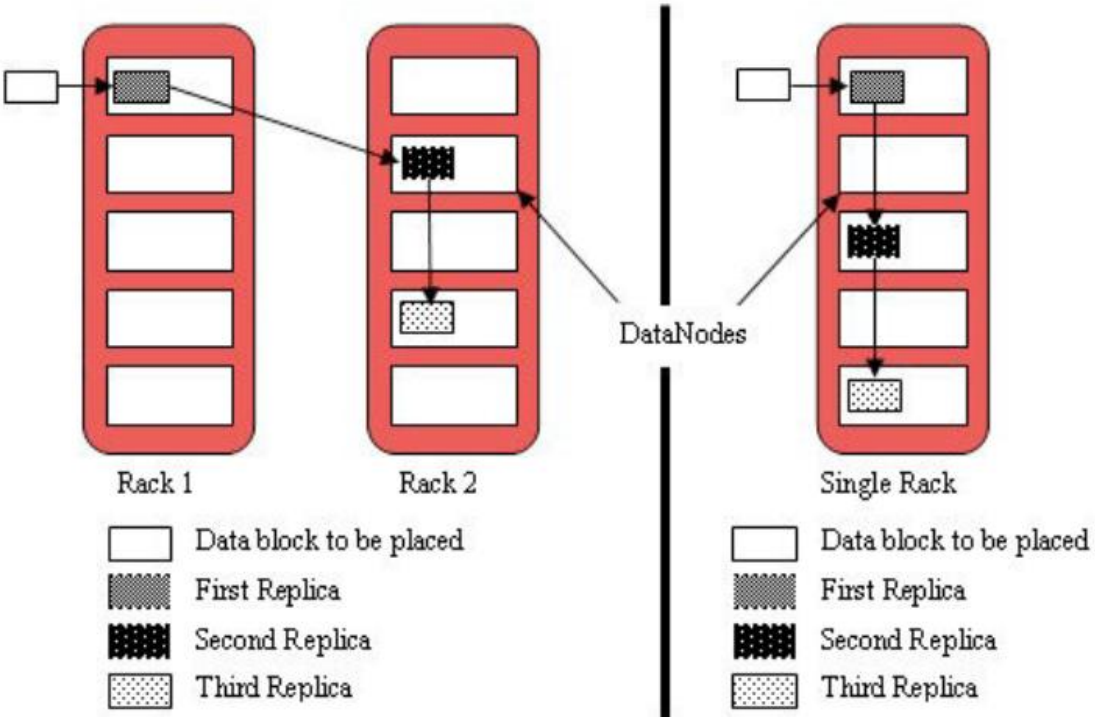
3、 NameNode 根据配置文件中指定的备份数量及副本放置策略进行文件分配，返回可用的 DataNode 的地址，如：A，B，C；

注：默认存储策略由 BlockPlacementPolicyDefault 类支持。也就是日常生活中提到最经典的 ***3 副本策略。***

1st replica 如果写请求方所在机器是其中一个 datanode,则直接存放在本地,否则随机在集群中选择一个 datanode.

2nd replica 第二个副本存放于不同第一个副本的所在的机架.

3rd replica 第三个副本存放于第二个副本所在的机架,但是属于不同的节点



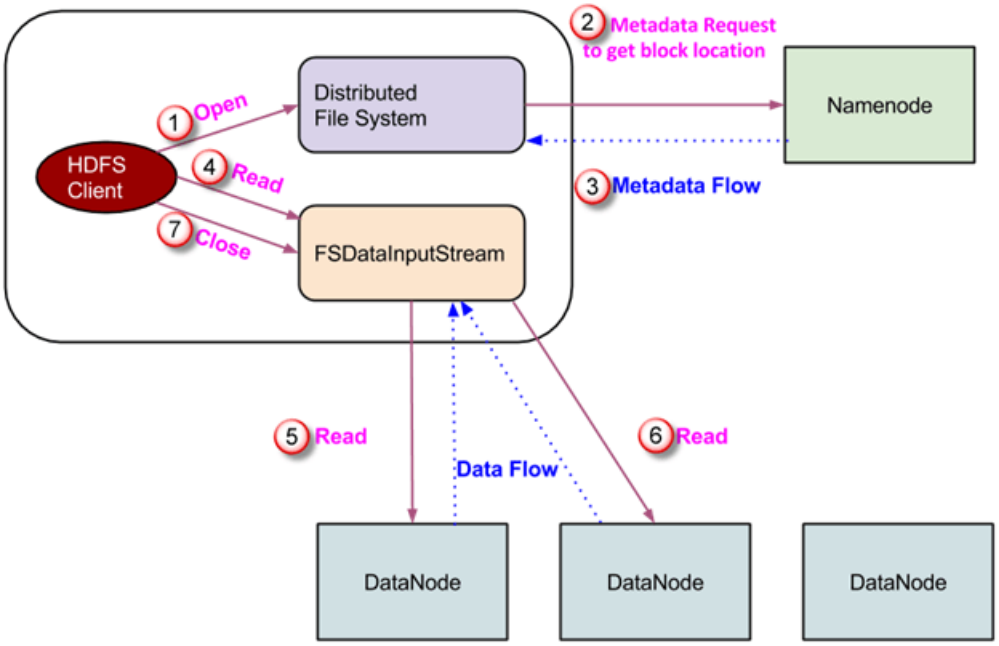
4、 client 请求 3 台 DataNode 中的一台 A 上传数据（本质上是一个 RPC 调用，建立 pipeline），A 收到请求会继续调用 B，然后 B 调用 C，将整个 pipeline 建立完成，后逐级返回 client；

5、 client 开始往 A 上传第一个 block（先从磁盘读取数据放到一个本地内存缓存），以 packet 为单位（默认 64K），A 收到一个 packet 就会传给 B，B 传给 C；A 每传一个 packet 会放入一个应答队列等待应答。

6、 数据被分割成一个个 packet 数据包在 pipeline 上依次传输，在 pipeline 反方向上，逐个发送 ack（命令正确应答），最终由 pipeline 中第一个 DataNode 节点 A 将 pipeline ack 发送给 client；

7、 当一个 block 传输完成之后，client 再次请求 NameNode 上传第二个 block 到服务器。

3.2. HDFS 读数据流程



详细步骤解析：

1、 Client 向 NameNode 发起 RPC 请求，来确定请求文件 block 所在的位置；

2、 NameNode 会视情况返回文件的部分或者全部 block 列表，对于每个 block，NameNode 都会返回含有该 block 副本的 DataNode 地址；

3、 这些返回的 DN 地址，会按照集群拓扑结构得出 DataNode 与客户端的距离，然后进行排序，排序两个规则：网络拓扑结构中距离 Client 近的排靠前；心跳机制中超时汇报的 DN 状态为 STALE，这样的排靠后；

4、 Client 选取排序靠前的 DataNode 来读取 block，如果客户端本身就是 DataNode，那么将从本地直接获取数据；

5、 底层上本质是建立 FSDataInputStream，重复的调用父类 DataInputStream 的 read 方法，直到这个块上的数据读取完毕；一旦到达块的末尾，DFSInputStream 关闭连接并继续定位下一个块的下一个 DataNode；

6、 当读完列表的 block 后，若文件读取还没有结束，客户端会继续向 NameNode 获取下一批的 block 列表；一旦客户端完成读取，它就会调用 close() 方法。

7、 读取完一个 block 都会进行 checksum 验证，如果读取 DataNode 时出现错误，客户端会通知 NameNode，然后再从下一个拥有该 block 副本的 DataNode 继续读。

8、 NameNode 只是返回 Client 请求包含块的 DataNode 地址，并不是返回请求块的数据；

9、 最终读取来所有的 block 会合并成一个完整的最终文件。

四、 HDFS 其他功能

1. 不同集群之间的数据复制

在我们实际工作当中，极有可能会遇到将测试集群的数据拷贝到生产环境集群，或者将生产环境集群的数据拷贝到测试集群，那么就需要我们在多个集群之间进行数据的远程拷贝，hadoop 自带也有命令可以帮我们实现这个功能。

1.1. 集群内部文件拷贝 scp

```
cd /export/softwares/
scp -r jdk-8u141-linux-x64.tar.gz root@node2:/export/
```

1.2. 跨集群之间的数据拷贝 distcp

```
bin/hadoop distcp hdfs://node1:8020/jdk-8u141-linux-x64.tar.gz
hdfs://cluster2:9000/
```

2. Archive 档案的使用

HDFS 并不擅长存储小文件，因为每个文件最少一个 block，每个 block 的元数据都会在 NameNode 占用内存，如果存在大量的小文件，它们会吃掉 NameNode 节点的大量内存。

Hadoop Archives 可以有效的处理以上问题，它可以把多个文件归档成为一个文件，归档成一个文件后还可以透明的访问每一个文件。

2.1. 如何创建 Archive

```
Usage: hadoop archive -archiveName name -p <parent> <src>* <dest>
```

其中-archiveName 是指要创建的存档的名称。比如 test.har，archive 的名字的扩展名应该是*.har。 -p 参数指定文件存档文件（src）的相对路径。

举个例子：-p /foo/bar a/b/c e/f/g

这里的/foo/bar 是 a/b/c 与 e/f/g 的父路径，

所以完整路径为/foo/bar/a/b/c 与/foo/bar/e/f/g

例如：如果你只想存档一个目录/input 下的所有文件:

```
hadoop archive -archiveName test.har -p /input  /outputdir
```

这样就会在/outputdir 目录下创建一个名为 test.har 的存档文件。

```
[root@node-21 ~]# hadoop fs -ls /input
Found 3 items
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 /input/1.txt
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 /input/2.txt
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 /input/3.txt
```

```
[root@node-21 ~]# hadoop fs -ls /outputdir
Found 1 items
drwxr-xr-x   - root supergroup          0 2017-09-06 14:59 /outputdir/test.har
```

2.2. 如何查看 Archive

首先我们来看下创建好的 har 文件。使用如下的命令：

```
hadoop fs -ls /outputdir/test.har
```

```
[root@node-21 ~]# hadoop fs -ls /outputdir/test.har
Found 4 items
-rw-r--r--   2 root supergroup          0 2017-09-06 14:59 /outputdir/test.har/_SUCCESS
-rw-r--r--   5 root supergroup        245 2017-09-06 14:59 /outputdir/test.har/_index
-rw-r--r--   5 root supergroup         23 2017-09-06 14:59 /outputdir/test.har/_masterindex
-rw-r--r--   2 root supergroup          6 2017-09-06 14:59 /outputdir/test.har/part-0
```

这里可以看到 har 文件包括：两个索引文件，多个 part 文件（本例只有一个）以及一个标识成功与否的文件。part 文件是多个原文件的集合，根据 index 文件去找到原文件。

例如上述的三个小文件 1.txt 2.txt 3.txt 内容分别为 1，2，3。进行 archive 操作之后，三个小文件就归档到 test.har 里的 part-0 一个文件里。

```
[root@node-21 ~]# hadoop fs -cat /input/1.txt
1
```

```
[root@node-21 ~]# hadoop fs -cat /outputdir/test.har/part-0
1
2
3
```

archive 作为文件系统层暴露给外界。所以所有的 fs shell 命令都能在 archive 上运行，但是要使用不同的 URI。Hadoop Archives 的 URI 是：

```
har://scheme-hostname:port/archivepath/fileinarchive
```

scheme-hostname 格式为 hdfs-域名:端口，如果没有提供 scheme-hostname，它会使用默认的文件系统。这种情况下 URI 是这种形式：

```
har:///archivepath/fileinarchive
```

如果用 har uri 去访问的话，索引、标识等文件就会隐藏起来，只显示创建档案之前的原文件：

```
[root@node-21 ~]# hadoop fs -ls har://hdfs-node-21:9000/outputdir/test.har
Found 3 items
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 har://hdfs-node-21:9000/outputdir/test.har/1.txt
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 har://hdfs-node-21:9000/outputdir/test.har/2.txt
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 har://hdfs-node-21:9000/outputdir/test.har/3.txt
```

```
[root@node-21 ~]# hadoop fs -ls har:///outputdir/test.har
Found 3 items
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 har:///outputdir/test.har/1.txt
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 har:///outputdir/test.har/2.txt
-rw-r--r--   2 root supergroup          2 2017-09-06 14:58 har:///outputdir/test.har/3.txt
```

```
[root@node-21 ~]# hadoop fs -cat har:///outputdir/test.har/1.txt
1
```

2.3. 如何解压 Archive

按顺序解压存档（串行）：

```
Hadoop fs -cp har:///user/zoo/foo.har/dir1  hdfs:/user/zoo/newdir
```

要并行解压存档，请使用 DistCp：

```
hadoop distcp har:///user/zoo/foo.har/dir1  hdfs:/user/zoo/newdir
```

2.4. Archive 注意事项

1. Hadoop archives 是特殊的档案格式。一个 Hadoop archive 对应一个文件系统目录。Hadoop archive 的扩展名是*.har；
2. 创建 archives 本质是运行一个 Map/Reduce 任务，所以应该在 Hadoop 集群上运行创建档案的命令；
3. 创建 archive 文件要消耗和原文件一样多的硬盘空间；
4. archive 文件不支持压缩，尽管 archive 文件看起来像已经被压缩过；
5. archive 文件一旦创建就无法改变，要修改的话，需要创建新的 archive 文件。事实上，一般不会再对存档后的文件进行修改，因为它们是定期存档的，比如每周或每日；
6. 当创建 archive 时，源文件不会被更改或删除；

五、 HDFS 元数据管理机制

1. 元数据管理概述

HDFS 元数据，按类型分，主要包括以下几个部分：

1、文件、目录自身的属性信息，例如文件名，目录名，修改信息等。

2、文件记录的信息的存储相关的信息，例如存储块信息，分块情况，副本个数等。

3、记录 HDFS 的 Datanode 的信息，用于 DataNode 的管理。

按形式分为内存元数据和元数据文件两种，分别存在内存和磁盘上。

HDFS 磁盘上元数据文件分为两类，用于持久化存储：

fsimage 镜像文件：是元数据的一个持久化的检查点，包含 Hadoop 文件系统中的所有目录和文件元数据信息，但不包含文件块位置的信息。文件块位置信息只存储在内存中，是在 datanode 加入集群的时候，namenode 询问 datanode 得到的，并且间断的更新。

Edits 编辑日志：存放的是 Hadoop 文件系统的所有更改操作（文件创建，删除或修改）的日志，文件系统客户端执行的更改操作首先会被记录到 edits 文件中。

fsimage 和 edits 文件都是经过序列化的，在 NameNode 启动的时候，它会将 fsimage 文件中的内容加载到内存中，之后再执行 edits 文件中的各项操作，使得内存中的元数据和

实际的同步，存在内存中的元数据支持客户端的读操作，也是最完整的元数据。

当客户端对 HDFS 中的文件进行新增或者修改操作，操作记录首先被记入 edits 日志文件中，当客户端操作成功后，相应的元数据会更新到内存元数据中。因为 fsimage 文件一般都很大（GB 级别的很常见），如果所有的更新操作都往 fsimage 文件中添加，这样会导致系统运行的十分缓慢。

HDFS 这种设计实现着手于：一是内存中数据更新、查询快，极大缩短了操作响应时间；二是内存中元数据丢失风险颇高（断电等），因此辅佐元数据镜像文件（fsimage）+编辑日志文件（edits）的备份机制进行确保元数据的安全。

NameNode 维护整个文件系统元数据。因此，元数据的准确管理，影响着 HDFS 提供文件存储服务的能力。

2. 元数据目录相关文件

在 Hadoop 的 HDFS 首次部署好配置文件之后，并不能马上启动使用，而是先要对文件系统进行格式化。需要在 NameNode（NN）节点上进行如下的操作：

```
$HADOOP_HOME/bin/hdfs namenode -format
```

在这里要注意两个概念，一个是文件系统，此时的文件系统在物理上还不存在；二就是此处的格式化并不是指传统意义上的本地磁盘格式化，而是一些清除与准备工作。

格式化完成之后，将会在$dfs.namenode.name.dir/current 目录下创建如下的文件结构，这个目录也正是 namenode 元数据相关的文件目录：

```
current/
|-- VERSION
|-- edits_*
|-- fsimage_0000000000008547077
|-- fsimage_0000000000008547077.md5
`-- seen_txid
```

其中的 dfs.namenode.name.dir 是在 hdfs-site.xml 文件中配置的，默认值如下：

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file://${hadoop.tmp.dir}/dfs/name</value>
</property>

hadoop.tmp.dir是在core-site.xml中配置的，默认值如下
<property>
  <name>hadoop.tmp.dir</name>
  <value>/tmp/hadoop-${user.name}</value>
  <description>A base for other temporary directories.</description>
</property>
```

dfs.namenode.name.dir 属性可以配置多个目录，各个目录存储的文件结构和内容都完全一样，相当于备份，这样做的好处是当其中一个目录损坏了，也不会影响到 Hadoop 的元数据，特别是当其中一个目录是 NFS（网络文件系统 Network File System，NFS）之上，即使你这台机器损坏了，元数据也得到保存。

下面对$dfs.namenode.name.dir/current/目录下的文件进行解释。

VERSION

```
namespaceID=934548976
clusterID=CID-cdff7d73-93cd-4783-9399-0a22e6dce196
cTime=0
storageType=NAME_NODE
blockpoolID=BP-893790215-192.168.24.72-1383809616115
layoutVersion=-47
```

namespaceID/clusterID/blockpoolID 这些都是 HDFS 集群的唯一标识符。标识符被用来防止 DataNodes 意外注册到另一个集群中的 namenode 上。这些标识在联邦（federation）部署中特别重要。联邦模式下，会有多个 NameNode 独立工作。每个的 NameNode 提供唯一的命名空间（namespaceID），并管理一组唯一的文件块池（blockpoolID）。clusterID 将整个集群结合在一起作为单个逻辑单元，在集群中的所有节点上都是一样的。

storageType 说明这个文件存储的是什么进程的数据结构信息（如果是 DataNode，storageType=DATA_NODE）；

cTime NameNode 存储系统创建时间，首次格式化文件系统这个属性是 0，当文件系统升级之后，该值会更新到升级之后的时间戳；

layoutVersion 表示 HDFS 永久性数据结构的版本信息，是一个负整数。

补充说明：

格式化集群的时候，可以指定集群的 cluster_id，但是不能与环境中其他集群有冲突。如果没有提供 cluster_id，则会自动生成一个唯一的 ClusterID。

```
$HADOOP_HOME/bin/hdfs namenode -format -clusterId <cluster_id>
```

seen_txid

$dfs.namenode.name.dir/current/seen_txid 非常重要，是存放 transactionId 的文件，format 之后是 0，它代表的是 namenode 里面的 edits_*文件的尾数，namenode 重启的时候，会按照 seen_txid 的数字，循序从头跑 edits_0000001~到 seen_txid 的数字。所以当你的 hdfs 发生异常重启的时候，一定要比对 seen_txid 内的数字是不是你 edits 最后的尾数。

Fsimage & edits

$dfs.namenode.name.dir/current 目录下在 format 的同时也会生成 fsimage 和 edits 文件，及其对应的 md5 校验文件。

3. Fsimage、Edits

3.1. 概述

fsimage 文件其实是 Hadoop 文件系统元数据的一个永久性的检查点，其中包含 Hadoop 文件系统中的所有目录和文件 idnode 的序列化信息；

fsimage 包含 Hadoop 文件系统中的所有目录和文件 idnode 的序列化信息；对于文件来说，包含的信息有修改时间、访问时间、块大小和组成一个文件块信息等；而对于目录来说，包含的信息主要有修改时间、访问控制权限等信息。

edits 文件存放的是 Hadoop 文件系统的所有更新操作的路径，文件系统客户端执行的所以写操作首先会被记录到 edits 文件中。

NameNode 起来之后，HDFS 中的更新操作会重新写到 edits 文件中，因为 fsimage 文件一般都很大（GB 级别的很常见），如果所有的更新操作都往 fsimage 文件中添加，这样会导致系统运行的十分缓慢，但是如果往 edits 文件里面写就不会这样，每次执行写操作之后，且在向客户端发送成功代码之前，edits 文件都需要同步更新。如果一个文件比较大，使得写操作需要向多台机器进行操作，只有当所有的写操作都执行完成之后，写操作才会返回成功，这样的好处是任何的操作都不会因为机器的故障而导致元数据的不同步。

3.2. 内容查看

fsimage、edits 两个文件中的内容使用普通文本编辑器是无法直接查看的，幸运的是 hadoop 为此准备了专门的工具用于查看文件的内容，这些工具分别为 **oev** 和 **oiv**，可以使用 hdfs 调用执行。

oev 是 offline edits viewer（离线 edits 查看器）的缩写，该工具只操作文件因而并不需要 hadoop 集群处于运行状态。

```
hdfs oev -i edits_0000000000000000081-0000000000000000089 -o edits.xml
```

-i,--inputFile <arg>

-o,--outputFile <arg> Name of output file.

在输出文件中，每个 RECORD 记录了一次操作,示例如下：

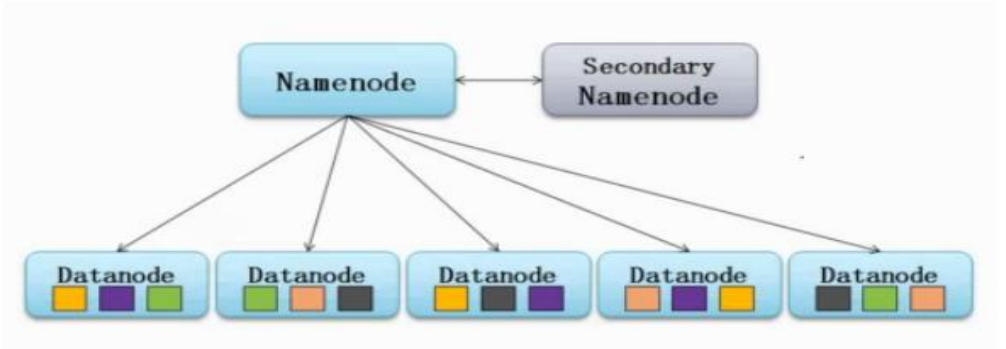
```
<RECORD>
  <OPCODE>OP_DELETE</OPCODE>
  <DATA>
    <TXID>88</TXID>
    <LENGTH>0</LENGTH>
    <PATH>/user/hive/test</PATH>
    <TIMESTAMP>1413794973949</TIMESTAMP>
    <RPC_CLIENTID>a52277d8-a855-41ee-9ca2-a5d0bc7d298a</RPC_CLIENTID>
    <RPC_CALLID>3</RPC_CALLID>
  </DATA>
</RECORD>
```

oiv 是 offline image viewer 的缩写，用于将 fsimage 文件的内容转储到指定文件中以便于阅读，该工具还提供了只读的 WebHDFS API 以允许离线分析和检查 hadoop 集群的命名空间。oiv 在处理非常大的 fsimage 文件时是相当快的，如果该工具不能够处理 fsimage，它会直接退出。该工具不具备向后兼容性，比如使用 hadoop-2.4 版本的 oiv 不能处理 hadoop-2.3 版本的 fsimage，只能使用 hadoop-2.3 版本的 oiv。同 oev 一样，就像它的名称所提示的（offline），oiv 也不需要 hadoop 集群处于运行状态。

```
hdfs oiv -i fsimage_0000000000000000115 -p XML -o fsimage.xml
```

```
<INodeSection>
        <lastInodeId>16418</lastInodeId>
        <inode>
                <id>16385</id>
                <type>DIRECTORY</type>
                <name></name>
                <mtime>1412832662162</mtime>
                <permission>hadoop:supergroup:rwxr-xr-x</permission>
                <nsquota>9223372036854775807</nsquota>
                <dsquota>-1</dsquota>
        </inode>
```

六、 secondary namenode



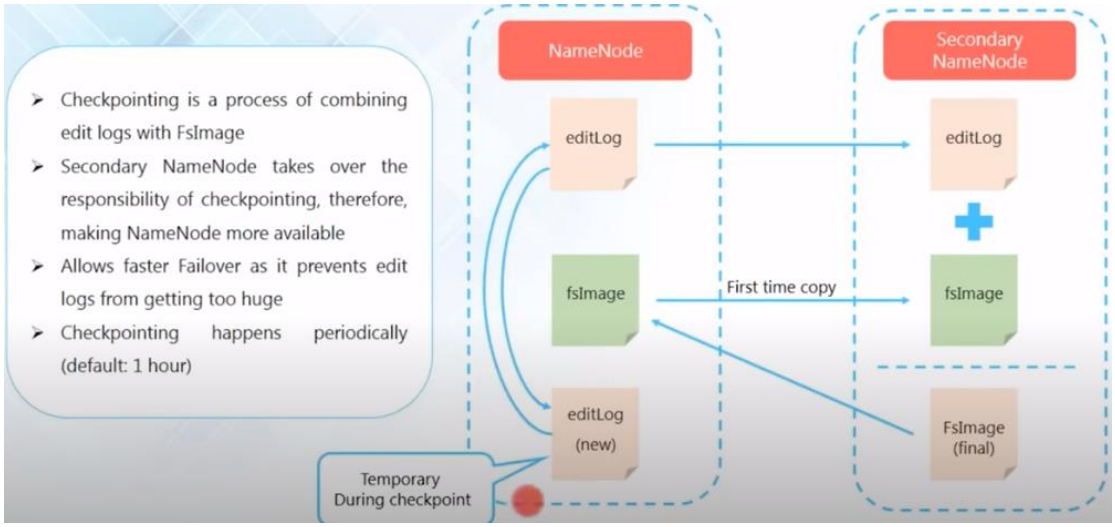
NameNode 职责是管理元数据信息，DataNode 的职责是负责数据具体存储，那么 SecondaryNameNode 的作用是什么？对很多初学者来说是非常迷惑的。它为什么会出现在 HDFS 中。从它的名字上看，它给人的感觉就像是 NameNode 的备份。但它实际上却不是。

大家猜想一下，当 HDFS 集群运行一段事件后，就会出现下面一些问题：

- edit logs 文件会变的很大，怎么去管理这个文件是一个挑战。
- NameNode 重启会花费很长时间，因为有很多改动要合并到 fsimage 文件上。
- 如果 NameNode 挂掉了，那就丢失了一些改动。因为此时的 fsimage 文件非常旧。

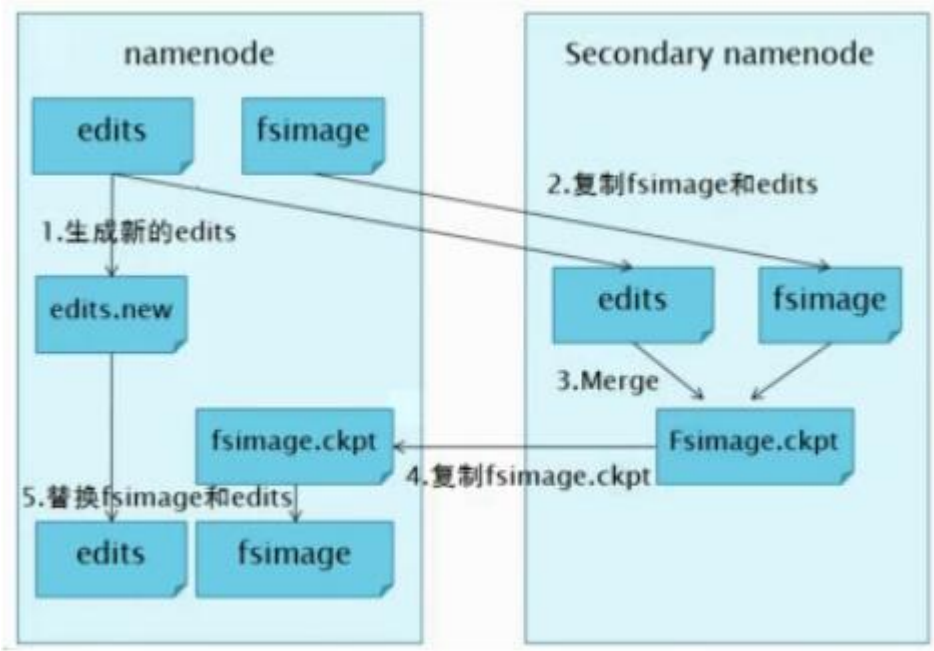
因此为了克服这个问题，我们需要一个易于管理的机制来帮助我们减小 edit logs 文件的大小和得到一个最新的 fsimage 文件，这样也会减小在 NameNode 上的压力。这跟 Windows 的恢复点是非常像的，Windows 的恢复点机制允许我们对 OS 进行快照，这样当系统发生问题时，我们能够回滚到最新的一次恢复点上。

SecondaryNameNode 就是来帮助解决上述问题的，它的职责是合并 NameNode 的 edit logs 到 fsimage 文件中。



1. Checkpoint

每达到触发条件，会由 secondary namenode 将 namenode 上积累的所有 edits 和一个最新的 fsimage 下载到本地，并加载到内存进行 merge（这个过程称为 checkpoint），如下图所示：



1.1. Checkpoint 详细步骤

- NameNode 管理着元数据信息，其中有两类持久化元数据文件：edits 操作日志文件和 fsimage 元数据镜像文件。新的操作日志不会立即与 fsimage 进行合并，也不会刷到 NameNode 的内存中，而是会先写到 edits 中(因为合并需要消耗大量的资源)，操作成功之后更新至内存。
- 有 dfs.namenode.checkpoint.period 和 dfs.namenode.checkpoint.txns 两个配置，只要达到这两个条件任何一个，secondarynamenode 就会执行 checkpoint 的操作。
- 当触发 checkpoint 操作时，NameNode 会生成一个新的 edits 即上图中的 edits.new 文件，同时 SecondaryNameNode 会将 edits 文件和 fsimage 复制到本地(HTTP GET 方式)。
- secondarynamenode 将下载下来的 fsimage 载入到内存，然后一条一条地执行 edits 文件中的各项更新操作，使得内存中的 fsimage 保存最新，这个过程就是 edits 和 fsimage 文件合并，生成一个新的 fsimage 文件即上图中的 Fsimage.ckpt 文件。
- secondarynamenode 将新生成的 Fsimage.ckpt 文件复制到 NameNode 节点。

- 在 NameNode 节点的 edits.new 文件和 Fsimage.ckpt 文件会替换掉原来的 edits 文件和 fsimage 文件，至此刚好是一个轮回，即在 NameNode 中又是 edits 和 fsimage 文件。
- 等待下一次 checkpoint 触发 SecondaryNameNode 进行工作，一直这样循环操作。

1.2. Checkpoint 触发条件

Checkpoint 操作受两个参数控制，可以通过 core-site.xml 进行配置：

```
<property>
  <name> dfs.namenode.checkpoint.period</name>
   <value>3600</value>
    <description>
        两次连续的 checkpoint 之间的时间间隔。默认 1 小时
    </description>
</property>
<property>
  <name>dfs.namenode.checkpoint.txns</name>
  <value>1000000</value>
  <description>
    最大的没有执行 checkpoint 事务的数量，满足将强制执行紧急 checkpoint，即使尚未达到检查点周期。默认设置为 100 万。
</description>
</property>
```

从上面的描述我们可以看出，SecondaryNamenode 根本就不是 Namenode 的一个热备，其只是将 fsimage 和 edits 合并。其拥有的 fsimage 不是最新的，因为在他从 NameNode 下载 fsimage 和 edits 文件时候，新的更新操作已经写到 edit.new 文件中去了。而这些更新在 SecondaryNamenode 是没有同步到的!当然，**如果 NameNode 中的 fsimage 真的出问题了，还是可以用 SecondaryNamenode 中的 fsimage 替换一下 NameNode 上的 fsimage，虽然已经不是最新的 fsimage，但是我们可以将损失减小到最少！**

七、 HDFS 安全模式

1. 安全模式概述

安全模式是 HDFS 所处的一种特殊状态，在这种状态下，文件系统只接受读数据请求，而不接受删除、修改等变更请求，是一种保护机制，用于保证集群中的数据块的安全性。

在 NameNode 主节点启动时，HDFS 首先进入安全模式，集群会开始检查数据块的完整性。DataNode 在启动的时候会向 namenode 汇报可用的 block 信息，当整个系统达到安全标准时，HDFS 自动离开安全模式。

假设我们设置的副本数（即参数 dfs.replication）是 5，那么在 Datanode 上就应该有 5 个副本存在，假设只存在 3 个副本，那么比例就是 3/5=0.6。在配置文件 hdfs-default.xml 中定义了一个最小的副本的副本率（即参数 dfs.namenode.safemode.threshold-pct）0.999。

我们的副本率 0.6 明显小于 0.99，因此系统会自动的复制副本到其他的 DataNode，使得副本率不小于 0.999.如果系统中有 8 个副本，超过我们设定的 5 个副本，那么系统也会删除多余的 3 个副本。

如果 HDFS 处于安全模式下，不允许 HDFS 客户端进行任何修改文件的操作，包括上传文件，删除文件，重命名，创建文件夹，修改副本数等操作。

2. 安全模式配置

与安全模式相关主要配置在 hdfs-site.xml 文件中，主要有下面几个属性：

dfs.namenode.replication.min：每个数据块最小副本数量，默认为 1. 在上传文件时，达到最小副本数，就认为上传是成功的。

dfs.namenode.safemode.threshold-pct：达到最小副本数的数据块的百分比。默认为 0.999f。当小于这个比例，那就将系统切换成安全模式，对数据块进行复制；当大于该比例时，就离开安全模式，说明系统有足够的数据块副本数，可以对外提供服务。小于等于 0 意味不进入安全模式，大于 1 意味一直处于安全模式。

dfs.namenode.safemode.min.datanodes：离开安全模式的最小可用 datanode 数量要求，默认为 0.也就是即使所有 datanode 都不可用，仍然可以离开安全模式。

dfs.namenode.safemode.extension：当集群可用 block 比例，可用 datanode 都达到要求之后，如果在 extension 配置的时间段之后依然能满足要求，此时集群才离开安全模式。单位为毫秒，默认为 30000.也就是当满足条件并且能够维持 30 秒之后，离开安全模式。这个配置主要是对集群稳定程度做进一步的确认。避免达到要求后马上又不符合安全标准。

总结一下，要离开安全模式，需要满足以下条件：

1）达到副本数量要求的 block 比例满足要求；

2）可用的 datanode 节点数满足配置的数量要求；

3） 1、2 两个条件满足后维持的时间达到配置的要求

3. 安全模式命令

手动进入安全模式

```
hdfs dfsadmin -safemode enter
```

手动进入安全模式对于集群维护或者升级的时候非常有用，因为这时候 HDFS 上的数据是只读的。手动退出安全模式可以用下面命令：

```
hdfs dfsadmin -safemode leave
```

如果你想获取到集群是否处于安全模式，可以用下面的命令获取：