一、 课程计划

目录

二、 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>
```

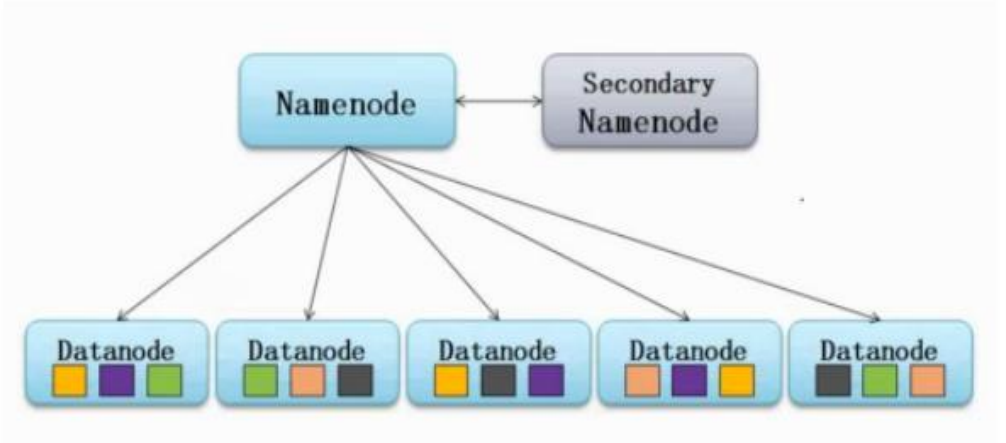
seen_txid

$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． 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：

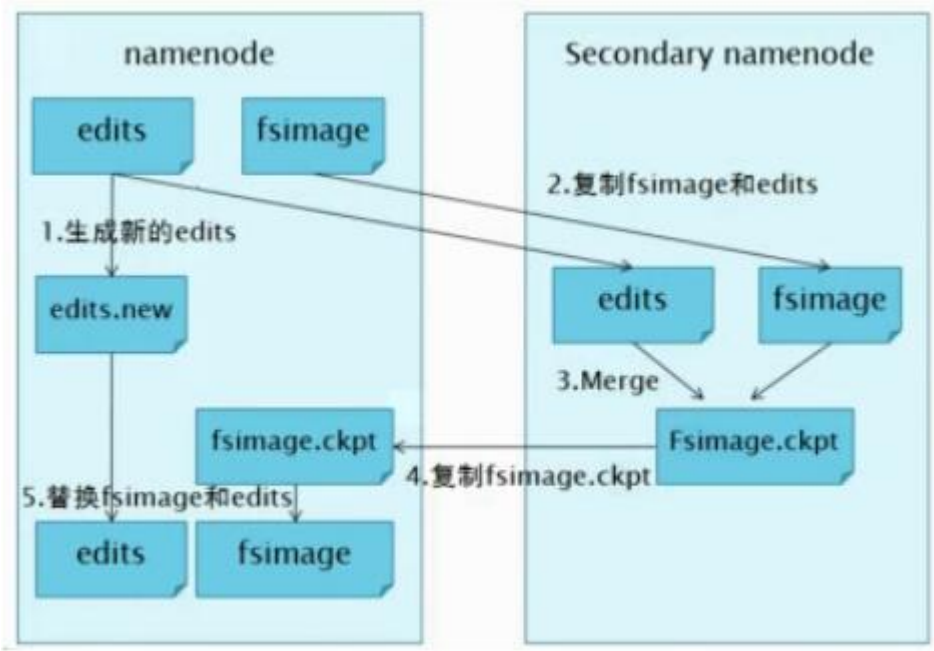
- edit logs 文件会变的很大，怎么去管理这个文件是一个挑战。
- NameNode 重启会花费很长时间，因为有很多改动要合并到 fsimage 文件上。
- 如果 NameNode 挂掉了，那就丢失了一些改动。因为此时的 fsimage 文件非常旧。

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。

4. 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：



4.1. 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。

4.2. 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！**

三、 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：

```
hdfs dfsadmin -safemode get（也可在 web 页面查看安全模式状态）
```

四、 Hadoop Archives

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。

1. 使用方法

1.1. 如何创建 Archives（档案）

```
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
```

1.2. 如何查看 Archives

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
```

1.3. 如何解压 Archives

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． 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；
7. 虽然 HAR 文件可以作为 MR 的输入，但是由于 InputFormat 类并不知道文件已经存档，所以即使在 HAR 文件里处理许多小文件，仍然低效。

五、 Hadoop High Availability

HA(High Available)，高可用，是保证业务连续性的有效解决方案，一般有两个或两个以上的节点，分为活动节点（Active）及备用节点（Standby）。通常把正在执行业务的称为活动节点，而作为活动节点的一个备份的则称为备用节点。当活动节点出现问题，导致正在运行的业务（任务）不能正常运行时，备用节点此时就会侦测到，并立即接续活动节点来执行业务。从而实现业务的不中断或短暂中断。

Hadoop1.X 版本，NN 是 HDFS 集群的单点故障点，每一个集群只有一个 NN，如果这个机器或进程不可用，整个集群就无法使用。为了解决这个问题，出现了一堆针对 HDFS HA 的解决方案（如：Linux HA, VMware FT, shared NAS+NFS, BookKeeper, QJM/Quorum Journal Manager, BackupNode 等）。

在 HA 具体实现方法不同情况下，HA 框架的流程是一致的，不一致的就是如何存储、管理、同步 edits 编辑日志文件。

在 Active NN 和 Standby NN 之间要有个共享的存储日志的地方，Active NN 把 edit Log 写到这个共享的存储日志的地方，Standby NN 去读取日志然后执行，这样 Active 和 Standby NN 内存中的 HDFS 元数据保持着同步。一旦发生主从切换 Standby NN 可以尽快接管 Active NN 的工作。

1. Namenode HA

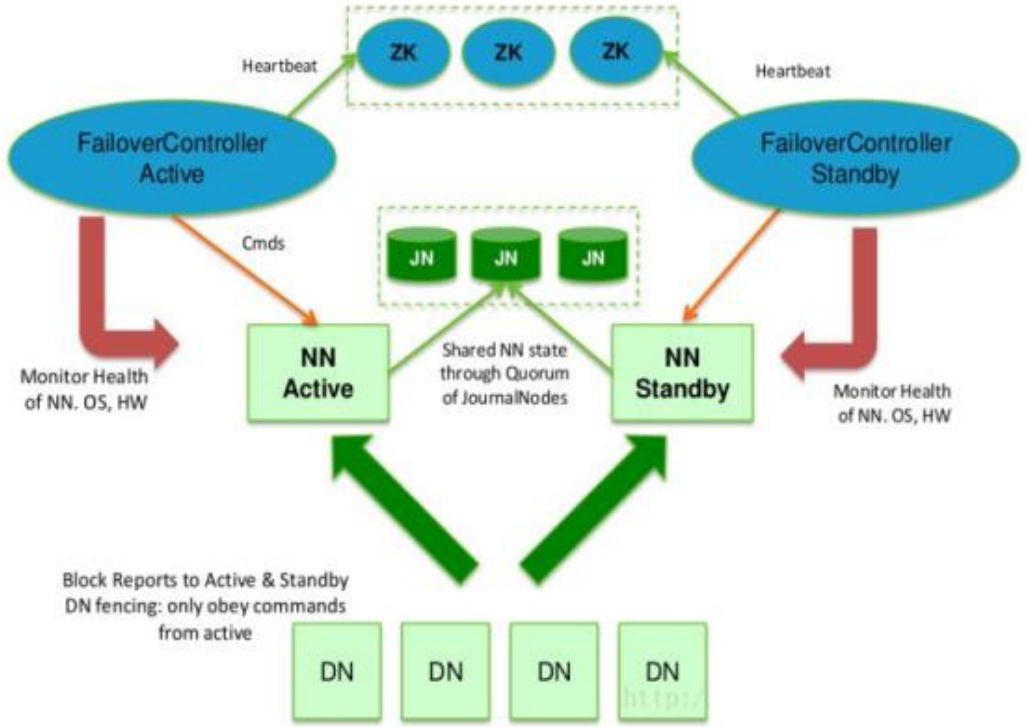
1.1. Namenode HA 详解

hadoop2.x 之后，Clouera 提出了 QJM/Qurom Journal Manager，这是一个基于 Paxos 算法（分布式一致性算法）实现的 HDFS HA 方案，它给出了一种较好的解决思路和方案,QJM 主要优势如下：

不需要配置额外的高共享存储，降低了复杂度和维护成本。

消除 spof(单点故障)。

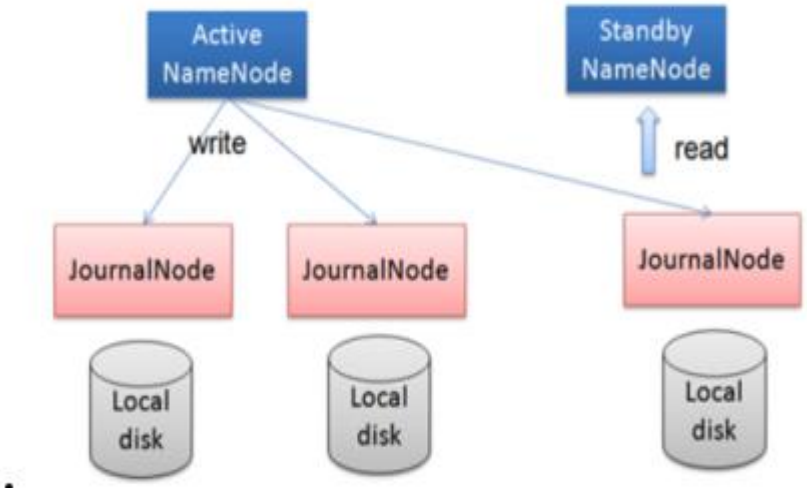
系统鲁棒性(Robust)的程度可配置、可扩展。



基本原理就是用 2N+1 台 JournalNode 存储 EditLog，每次写数据操作有>=N+1 返回成功时即认为该次写成功，数据不会丢失了。当然这个算法所能容忍的是最多有 N 台机器挂掉，如果多于 N 台挂掉，这个算法就失效了。这个原理是基于 Paxos 算法。

在 HA 架构里面 SecondaryNameNode 已经不存在了，为了保持 standby NN 时时的与 Active NN 的元数据保持一致，他们之间交互通过 JournalNode 进行操作同步。

任何修改操作在 Active NN 上执行时，JournalNode 进程同时也会记录修改 log 到至少半数以上的 JN 中，这时 Standby NN 监测到 JN 里面的同步 log 发生变化了会读取 JN 里面的修改 log，然后同步到自己的目录镜像树里面，如下图：



当发生故障时，Active 的 NN 挂掉后，Standby NN 会在它成为 Active NN 前，读取所有的 JN 里面的修改日志，这样就能高可靠的保证与挂掉的 NN 的目录镜像树一致，然后无缝的接替它的职责，维护来自客户端请求，从而达到一个高可用的目的。

在 HA 模式下，datanode 需要确保同一时间有且只有一个 NN 能命令 DN。为此：

每个 NN 改变状态的时候，向 DN 发送自己的状态和一个序列号。

DN 在运行过程中维护此序列号，当 failover 时，新的 NN 在返回 DN 心跳时会返回自己的 active 状态和一个更大的序列号。DN 接收到这个返回则认为该 NN 为新的 active。

如果这时原来的 active NN 恢复，返回给 DN 的心跳信息包含 active 状态和原来的序列号，这时 DN 就会拒绝这个 NN 的命令。

1.2. Failover Controller

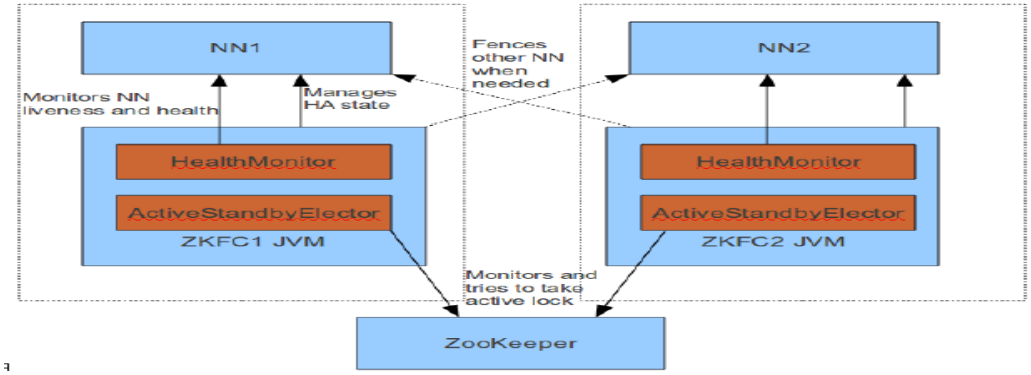
HA 模式下，会将 FailoverController 部署在每个 NameNode 的节点上，作为一个单独的进程用来监视 NN 的健康状态。**FailoverController 主要包括三个组件：**

HealthMonitor： 监控 NameNode 是否处于 unavailable 或 unhealthy 状态。当前通过 RPC 调用 NN 相应的方法完成。

ActiveStandbyElector：监控 NN 在 ZK 中的状态。

ZKFailoverController： 订阅 HealthMonitor 和 ActiveStandbyElector 的事件，并管理 NN 的状态，另外 zkfc 还负责解决 fencing（也就是脑裂问题）。

上述三个组件都在跑在一个 JVM 中，这个 JVM 与 NN 的 JVM 在同一个机器上。但是两个独立的进程。一个典型的 HA 集群，有两个 NN 组成，每个 NN 都有自己的 ZKFC 进程。



ZKFailoverController 主要职责：

- **健康监测：**周期性的向它监控的 NN 发送健康探测命令，从而来确定某个 NameNode 是否处于健康状态，如果机器宕机，心跳失败，那么 zkfc 就会标记它处于一个不健康的状态
- **会话管理：**如果 NN 是健康的，zkfc 就会在 zookeeper 中保持一个打开的会话，如果 NameNode 同时还是 Active 状态的，那么 zkfc 还会在 Zookeeper 中占有一个类型为短暂类型的 znode，当这个 NN 挂掉时，这个 znode 将会被删除，然后备用的 NN 将会得到这把锁，升级为主 NN，同时标记状态为 Active
- 当宕机的 NN 新启动时，它会再次注册 zookeper，发现已经有 znode 锁了，便会自动变为 Standby 状态，如此往复循环，保证高可靠，需要注意，目前仅仅支持最多配置 2 个 NN
- **master 选举：**通过在 zookeeper 中维持一个短暂类型的 znode，来实现抢占式的锁机制，从而判断那个 NameNode 为 Active 状态

2． Yarn HA

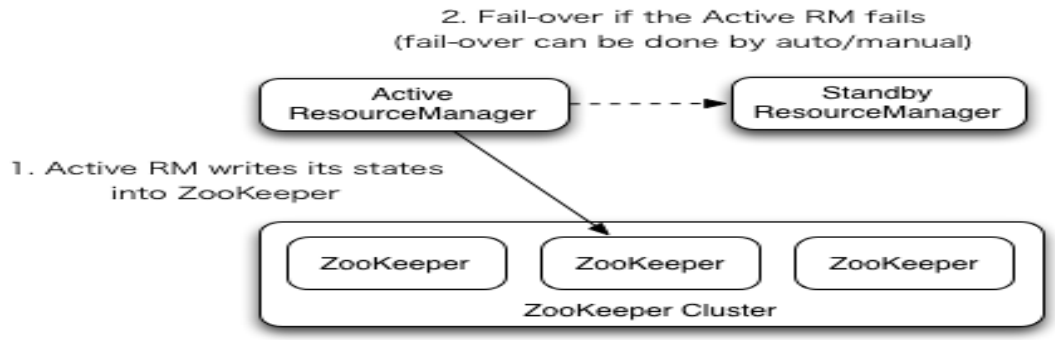
Yarn 作为资源管理系统，是上层计算框架（如 MapReduce,Spark）的基础。在 Hadoop 2.4.0 版本之前，Yarn 存在单点故障（即 ResourceManager 存在单点故障），一旦发生故障，恢复时间较长，且会导致正在运行的 Application 丢失，影响范围较大。从 Hadoop 2.4.0 版本开始，Yarn 实现了 ResourceManager HA，在发生故障时自动 failover，大大提高了服务的可靠性。

ResourceManager（简写为 RM）作为 Yarn 系统中的主控节点，负责整个系统的资源管理和调度，内部维护了各个应用程序的 ApplictionMaster 信息、NodeManager（简写为 NM）信息、资源使用等。由于资源使用情况和 NodeManager 信息都可以通过 NodeManager 的心跳机制重新构建出来，因此只需要对 ApplicationMaster 相关的信息进行持久化存储即可。

在一个典型的 HA 集群中，两台独立的机器被配置成 ResourceManger。在任意时间，有且只允许一个活动的 ResourceManger,另外一个备用。切换分为两种方式：

手动切换：在自动恢复不可用时，管理员可用手动切换状态，或是从 Active 到 Standby，或是从 Standby 到 Active。

自动切换：基于 Zookeeper，但是区别于 HDFS 的 HA，2 个节点间无需配置额外的 ZFKC 守护进程来同步数据。



3． Hadoop HA 集群的搭建

HA 集群搭建的难度主要在于配置文件的编写，**心细，心细，心细**！

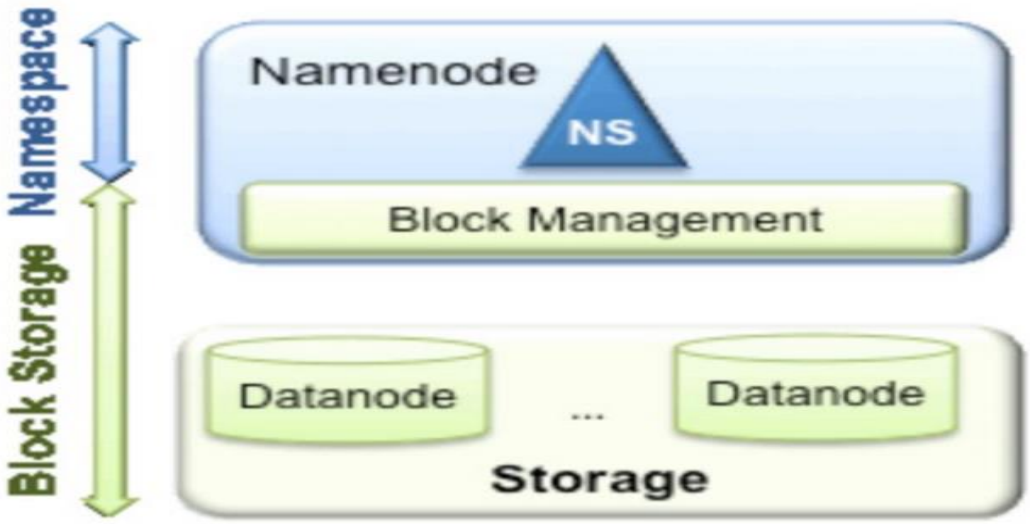
详细的搭建安装步骤请参考附件资料。

六、 Hadoop Federation

1. 背景概述

单 NameNode 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，NameNode 进程使用的内存可能会达到上百 G，NameNode 成为了性能的瓶颈。因而提出了 namenode 水平扩展方案--Federation。

Federation 中文意思为联邦,联盟，是 NameNode 的 Federation,也就是会有多个 NameNode。多个 NameNode 的情况意味着有多个 namespace(命名空间)，区别于 HA 模式下的多 NameNode，它们是拥有着同一个 namespace。既然说到了 NameNode 的命名空间的概念,这里就看一下现有的 HDFS 数据管理架构,如下图所示:

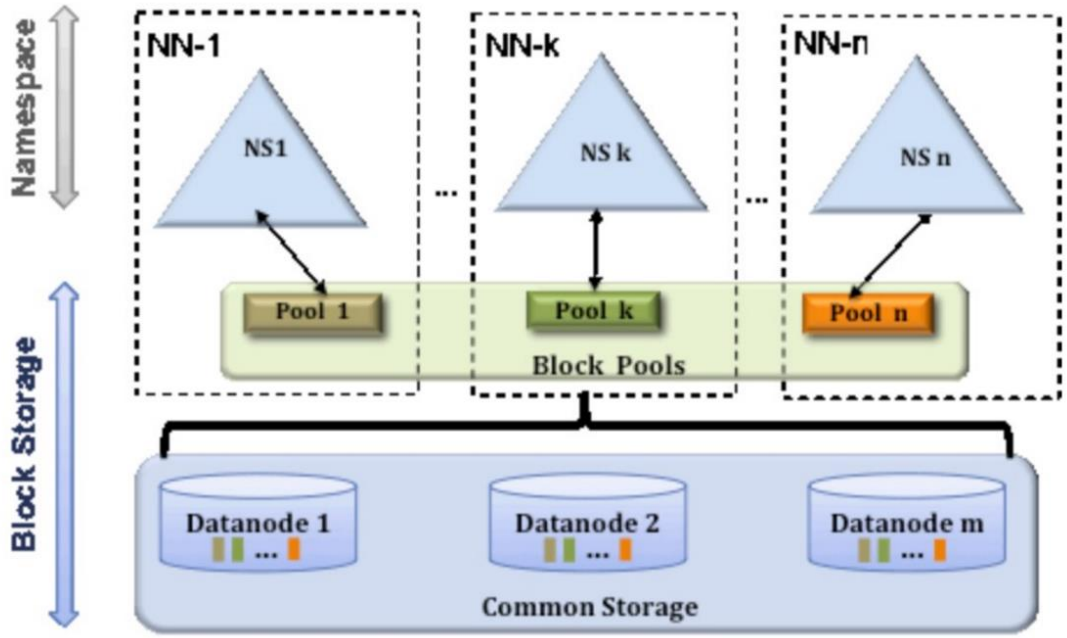


从上图中,我们可以很明显地看出现有的 HDFS 数据管理,数据存储 2 层分层的结构.也就是说,所有关于存储数据的信息和管理是放在 NameNode 这边,而真实数据的存储则是在各个 DataNode 下.而这些隶属于同一个 NameNode 所管理的数据都是在同一个命名空间下的.而一个 namespace 对应一个 block pool。Block Pool 是同一个 namespace 下的 block 的集合.当然这是我们最常见的单个 namespace 的情况,也就是一个 NameNode 管理集群中所有元数据信息的时候.如果我们遇到了之前提到的 NameNode 内存使用过高的问题,这时候怎么办?元数据空间依然还是在不断增大,一味调高 NameNode 的 jvm 大小绝对不是一个持久的办法.这时候就诞生了 HDFS Federation 的机制.

2. Federation 架构设计

HDFS Federation 是解决 namenode 内存瓶颈问题的水平横向扩展方案。

Federation 意味着在集群中将会有多个 namenode/namespace。这些 namenode 之间是联合的，也就是说，他们之间相互独立且不需要互相协调，各自分工，管理自己的区域。分布式的 datanode 被用作通用的数据块存储存储设备。每个 datanode 要向集群中所有的 namenode 注册，且周期性地向所有 namenode 发送心跳和块报告，并执行来自所有 namenode 的命令。



Federation 一个典型的例子就是上面提到的 NameNode 内存过高问题,我们完全可以将上面部分大的文件目录移到另外一个NameNode 上做管理.**更重要的一点在于,这些NameNode 是共享集群中所有的 DataNode 的,它们还是在同一个集群内的。**

这时候在DataNode 上就不仅仅存储一个Block Pool 下的数据了,而是多个(在DataNode 的 datadir 所在目录里面查看 BP-xx.xx.xx.xx 打头的目录)。

概括起来：

多个 NN 共用一个集群里的存储资源，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。

HDFS Federation 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，但是从单个 namenode/namespace 看，仍然存在单点故障：如果某个 namenode 挂掉了，其管理的相应的文件便不可以访问。Federation 中每个 namenode 仍然像之前 HDFS 上实现一样，配有一个 secondary namenode，以便主 namenode 挂掉一下，用于还原元数据信息。

所以一般集群规模真的很大的时候，会采用 HA+Federation 的部署方案。也就是每个联合的 namenodes 都是 ha 的。

3. Federation 示例配置

这是一个包含两个 Namenode 的 Federation 示例配置：

```
<configuration>
  <property>
    <name>dfs.nameservices</name>
    <value>ns1,ns2</value>
  </property>
  <property>
    <name>dfs.namenode.rpc-address.ns1</name>
    <value>nn-host1:rpc-port</value>
  </property>
  <property>
    <name>dfs.namenode.http-address.ns1</name>
    <value>nn-host1:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.secondaryhttp-address.ns1</name>
    <value>snn-host1:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.rpc-address.ns2</name>
    <value>nn-host2:rpc-port</value>
  </property>
  <property>
    <name>dfs.namenode.http-address.ns2</name>
    <value>nn-host2:http-port</value>
  </property>
  <property>
    <name>dfs.namenode.secondaryhttp-address.ns2</name>
    <value>snn-host2:http-port</value>
  </property>

  .... Other common configuration ...
</configuration>
```

七、 了解 CDH

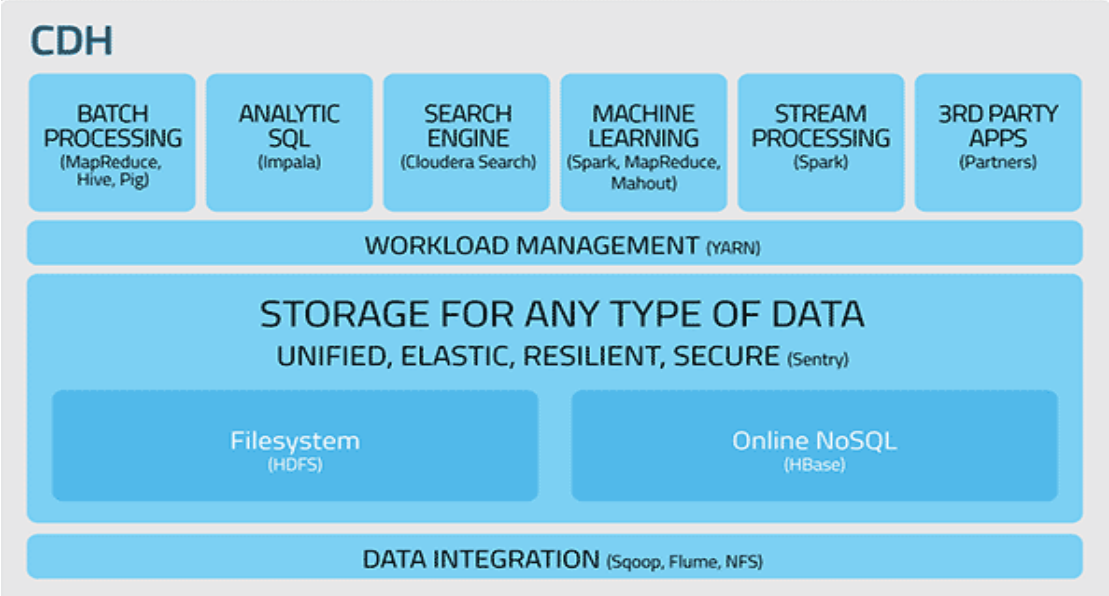
1. 什么是 CDH

hadoop 是 Apache 一个开源项目，所以很多公司在这个基础进行商业化，Cloudera 对 hadoop 做了相应的改变。Cloudera 公司的发行版 hadoop，我们将该版本称为 CDH（Cloudera Distribution Hadoop）。

提供了 Hadoop 的核心功能

- 可扩展存储
- 分布式计算

基于 Web 的用户界面



CDH 的优点：

- 版本划分清晰
- 版本更新速度快
- 支持 Kerberos 安全认证
- 文档清晰
- 支持多种安装方式（Cloudera Manager、YUM、RPM、Tarball）

2. 什么是 CM

Cloudera Manager（CM），也是 Cloudera 公司的产品。简单来说，Cloudera Manager 是一个拥有集群自动化安装、中心化管理、集群监控、报警功能的一个工具（软件）,使得安装 hadoop 集群的时间大大缩短，运维人员数大大降低，极大的提高集群管理的效率。

Cloudera Manager 有四大功能：

（1）管理：对集群进行管理，如添加、删除节点等操作。

（2）监控：监控集群的健康情况，对设置的各种指标和系统运行情况进行全面监控。

（3）诊断：对集群出现的问题进行诊断，对出现的问题给出建议解决方案。

（4）集成：对 hadoop 的多组件进行整合。

利用 CM 可以非常方便快速的搭建 CDH 集群。

详细安装步骤可以看参考资料中的安装手册。