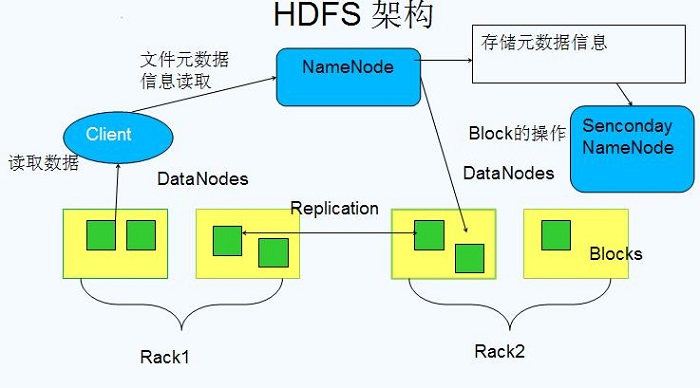
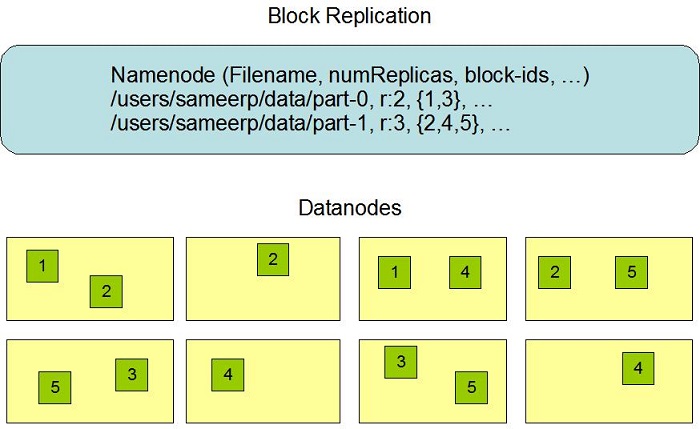
## **[HDFS 原理、架构与特性介绍](http://www.cnblogs.com/davidwang456/p/5015018.html)**

本文主要讲述 HDFS原理-架构、副本机制、HDFS负载均衡、机架感知、健壮性、文件删除恢复机制

1. 当前HDFS架构

HDFS主要包括NameNode、DataNode、Sencondary NameNode以及存储的详细信息。如下图：



1. **NameNode**

Namenode 的目录结构：

|  |
| --- |
| ${ dfs.name.dir}/current /VERSION  /edits  /fsimage  /fstime |

Namenode上保存着 HDFS 的名字空间。文件系统中任何数据的修改，Namenode都会使用一种称为 EditLog 的事务日志记录下来。例如，在 HDFS 中创建一个文件，Namenode就会在Editlog中插入一条记录来表示；同样地，修改文件的副本系数也将往 Editlog 插入一条记录。Namenode在本地操作系统的文件系统中存储这个 Editlog 。整个文件系统的名字空间，包括数据块到文件的映射、文件的属性等，都存储在一个称为 FsImage 的文件中，这个文件也是放在 Namenode 所在的本地文件系统上。

Namenode 在内存中保存着整个文件系统的名字空间和文件数据块映射 (Blockmap) 的映像。这个关键的元数据结构设计得很紧凑，因而一个有 4G 内存的 Namenode 足够支撑大量的文件和目录。当 Namenode启动时，它从硬盘中读取Editlog和FsImage，将所有 Editlog中的事务作用在内存中的FsImage上，并将这个新版本的 FsImage从内存中保存到本地磁盘上，然后删除旧的Editlog因为这个旧的Editlog的事务都已经作用在FsImage上了。这个过程称为一个检查点(checkpoint)。在当前实现中，检查点只发生在Namenode 启动时，在不久的将来将实现支持周期性的检查点。

**2、HDFS 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。当前，HDFS 不支持用户磁盘配额和访问权限控制，也不支持硬链接和软链接。但是HDFS架构并不妨碍实现这些特性。

Namenode负责维护文件系统命名空间，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。应用程序可以设置HDFS 保存的文件的副本数目。文件副本的数目称为文件的副本系数，这个信息也是由 Namenode 保存的。

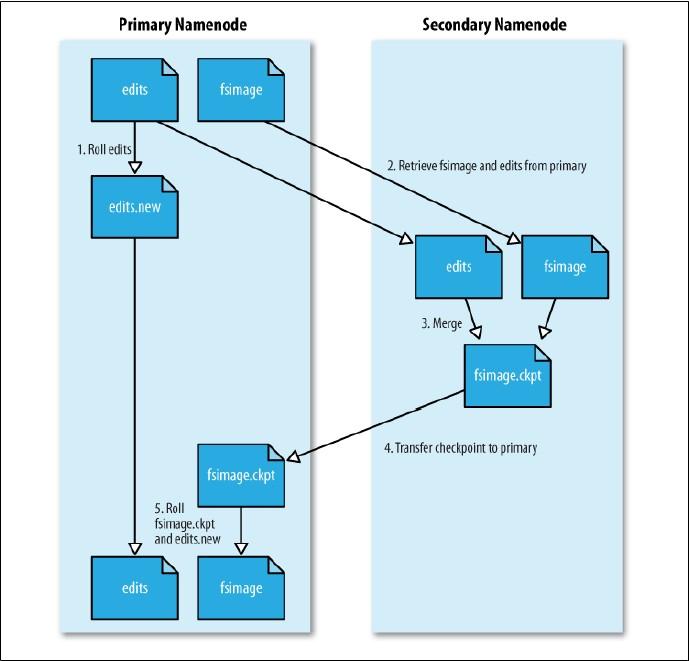
**3、DataNode**

Datanode将HDFS数据以文件形式存储在本地的文件系统中，它并不知道有关HDFS文件的信息。它把每个HDFS数据块存储在本地文件系统的一个单独的文件中。 Datanode 并不在同一个目录创建所有的文件，实际上，它用试探的方法来确定每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录中创建所有的本地文件并不是最优的选择，这是因为本地文件系统可能无法高效地在单个目 录中支持大量的文件。

当一个 Datanode 启动时，它会扫描本地文件系统，产生一个这些本地文件对应的所有HDFS数据块的列表，然后作为报告发送到 Namenode ，这个报告就是块状态报告。

**4、Secondary NameNode**

Secondary NameNode 定期合并 fsimage 和 edits 日志，将 edits 日志文件大小控制在一个限度下。



Secondary NameNode处理流程

(1) namenode 响应 SecondaryNamenode 请求，将editlog推送给SecondaryNamenode，开始重新写一个新的editlog 。

(2) SecondaryNamenode收到来自namenode的fsimage文件和edit log 。

(3) Secondary namenode将fsimage加载到内存，应用editlog，并生成一 个新的 fsimage 文件。

(4) SecondaryNamenode将新的fsimage推送给Namenode 。

(5) Namenode用新的fsimage取代旧的fsimage，在fstime文件中记下检查点发生的时间。

**5、HDFS通信协议**

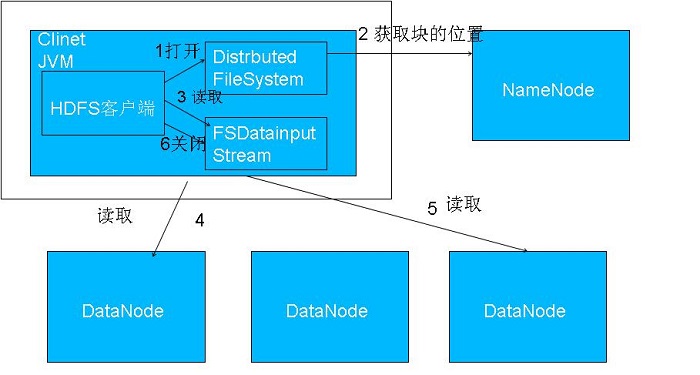
所有的HDFS通讯协议都是构建在TCP/IP协议上。客户端通过一个可配置的端口连接到Namenode，通过 ClientProtocol 与 Namenode交互。而Datanode是使用DatanodeProtocol与Namenode 交互。再设计上，DataNode 通过周期性的向NameNode 发送心跳和数据块来保持和NameNode的通信，数据块报告的信息包括数据块的属性，即数据块属于哪个文件，数据块ID，修改时间等，NameNode、DataNode和数据块的映射关系就是通过系统启动时DataNode的数据块报告建立的。从 ClientProtocol 和 Datanodeprotocol 抽象出一个远程调用(RPC），在设计上，Namenode 不会主动发起 RPC，而是是响应来自客户端和 Datanode 的 RPC 请求。

**6、HDFS的安全模式**

Namenode启动后会进入一个称为安全模式的特殊状态。处于安全模式的Namenode是不会进行数据块的复制的。Namenode从所有的 Datanode接收心跳信号和块状态报告。块状态报告包括了某个 Datanode所有的数据块列表。每个数据块都有一个指定的最小副本数。当 Namenode 检测确认某个数据块的副本数目达到这个最小值，那么该数据块就会被认为是副本安全(safely replicated)的；在一定百分比（这个参数可配置）的数据块被Namenode检测确认是安全之后（加上一个额外的 30 秒等待时间），Namenode 将退出安全模式状态。接下来它会确定还有哪些数据块的副本没有达到指定数目，并将这些数据块复制到其他 Datanode 上。

1. HDFS文件读取的解析

**文件读取流程如下图：**



流程分析

1.使用HDFS提供的客户端开发库Client，向远程的Namenode发起RPC请求；

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都会返回有该block拷贝的DataNode地址；

3.客户端会选取离客户端最接近的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；

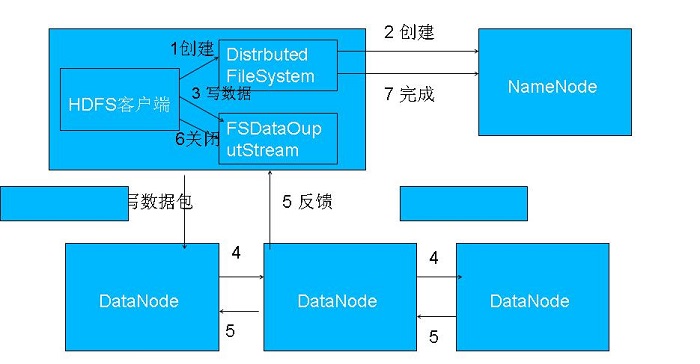
4.读取完当前block的数据后，关闭与当前的DataNode连接，并为读取下一个block寻找最佳的DataNode；

5.当读完列表的block后，且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。

6.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拷贝的datanode继续读。

1. HDFS文件写入的解析

**文件写入流程**



流程分析

1.使用HDFS提供的客户端，向远程的Namenode发起RPC请求；

2.Namenode会检查要创建的文件是否已经存在，创建者是否有权限进行操作，成功则会为文件创建一个记录，否则会让客户端抛出异常；

3.当客户端开始写入文件的时候，会将文件切分成多个packets，并在内部以数据队列”data queue”的形式管理这些packets，并向Namenode申请新的blocks，获取用来存储replicas的合适的datanodes列表，列表的大小根据在Namenode中对replication的设置而定。

4.开始以pipeline（管道）的形式将packet写入所有的replicas中。把packet以流的方式写入第一个datanode，该datanode把该packet存储之后，再将其传递给在此pipeline中的下一个datanode，直到最后一个datanode，这种写数据的方式呈流水线的形式。

5.最后一个datanode成功存储之后会返回一个ack packet，在pipeline里传递至客户端，在客户端的开发库内部维护着”ack queue”，成功收到datanode返回的ack packet后会从”ack queue”移除相应的packet。

6.如果传输过程中，有某个datanode出现了故障，那么当前的pipeline会被关闭，出现故障的datanode会从当前的pipeline中移除，剩余的block会继续剩下的datanode中继续以pipeline的形式传输，同时Namenode会分配一个新的datanode，保持replicas设定的数量。

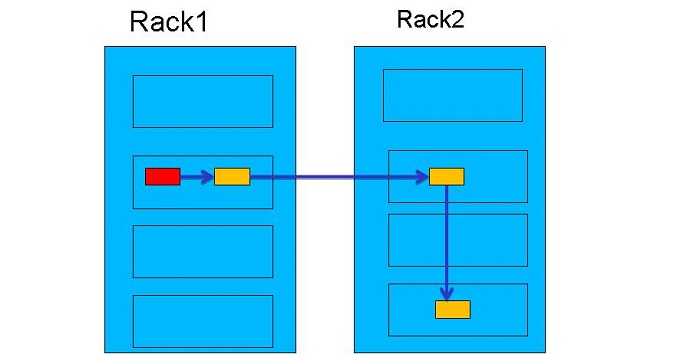
**流水线复制**

当客户端向 HDFS 文件写入数据的时候，一开始是写到本地临时文件中。假设该文件的副本系数设置为 3 ，当本地临时文件累积到一个数据块的大小时，客户端会从 Namenode 获取一个 Datanode 列表用于存放副本。然后客户端开始向第一个 Datanode 传输数据，第一个 Datanode 一小部分一小部分 (4 KB) 地接收数据，将每一部分写入本地仓库，并同时传输该部分到列表中 第二个 Datanode 节点。第二个 Datanode 也是这样，一小部分一小部分地接收数据，写入本地仓库，并同时传给第三个 Datanode 。最后，第三个 Datanode 接收数据并存储在本地。因此，Datanode 能流水线式地从前一个节点接收数据，并在同时转发给下一个节点，数据以流水线的 方式从前一个 Datanode 复制到下一个。

**更细节的原理**

客户端创建文件的请求其实并没有立即发送给集群，事实上，在刚开始阶段HDFS客户端会先将文件数据缓存到本地的一个临时文件。应用程序的写操作被透明地重定向到这个临时文件。当这个临时文件累积的数据量超过一个数据块的大小，客户端才会联系 Namenode。Namenode 将文件名插入文件系统的层次结构中，并且分配一个数据块给它。然后返回Datanode的标识符和目标数据块给客户端。接着客户端将这块数据从本地临时文件上传到指定的 Datanode上。当文件关闭时，在临时文件中剩余的没有上传的数据也会传输到指定的 Datanode 上。然后客户端告诉Namenode文件已经关闭。此时 Namenode 才将文件创建操作提交到日志里进行存储 。如果 Namenode 在文件关闭前宕机了，则该文件将丢失。

1. 副本机制



修改副本数

集群只有三个Datanode，hadoop系统replication=4时，会出现什么情况？

对于上传文件到hdfs上时，当时hadoop的副本系数是几，这个文件的块数副本数就会有几份，无论以后你怎么更改系统副本系统，这个文件的副本数都不会改变，也就说上传到分布式系统上的文件副本数由当时的系统副本数决定，不会受replication的更改而变化，除非用命令来更改文件的副本数。因为dfs.replication实质上是client参数，在create文件时可以指定具体replication，属性dfs.replication是不指定具体replication时的采用默认备份数。文件上传后，备份数已定，修改dfs.replication是不会影响以前的文件的，也不会影响后面指定备份数的文件。只影响后面采用默认备份数的文件。但可以利用hadoop提供的命令后期改某文件的备份数：hadoop fs -setrep -R 1。如果你是在hdfs-site.xml设置了dfs.replication，这并一定就得了，因为你可能没把conf文件夹加入到你的 project的classpath里，你的程序运行时取的dfs.replication可能是hdfs-default.xml里的 dfs.replication，默认是3。可能这个就是造成你为什么dfs.replication老是3的原因。你可以试试在创建文件时，显式设定replication。replication一般到3就可以了，大了意义也不大。

1. HDFS负载均衡

HDFS的数据也许并不是非常均匀的分布在各个DataNode中。一个常见的原因是在现有的集群上经常会增添新的DataNode节点。当新增一个数据块（一个文件的数据被保存在一系列的块中）时，NameNode在选择DataNode接收这个数据块之前，会考虑到很多因素。其中的一些考虑的是：

1.将数据块的一个副本放在正在写这个数据块的节点上。

2.尽量将数据块的不同副本分布在不同的机架上，这样集群可在完全失去某一机架的情况下还能存活。

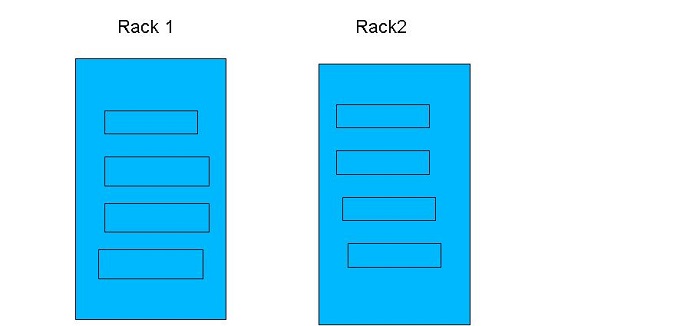
3.一个副本通常被放置在和写文件的节点同一机架的某个节点上，这样可以减少跨越机架的网络I/O。

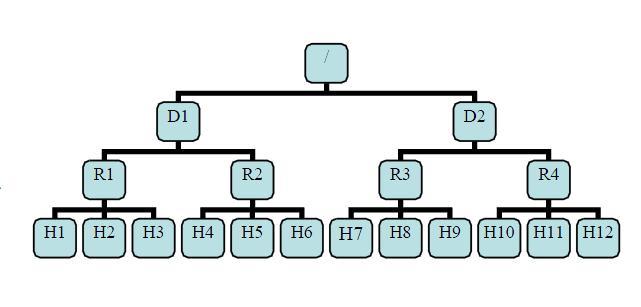
4.尽量均匀地将HDFS数据分布在集群的DataNode中。

1. HDFS机架感知

通常，大型 Hadoop 集群是以机架的形式来组织的，同一个机架上不同节点间的网络状况比不同机架之间的更为理想。另外， NameNode 设法将数据块副本保存在不同的机架上以提高容错性。

而HDFS不能够自动判断集群中各个datanode的网络拓扑情况 Hadoop允许集群的管理员通过配置dfs.network.script 参数来确定节点所处的机架。 文 件提供了 IP->rackid 的翻译。 NameNode 通过这个得到集群中各个 datanode 机器的 rackid 。 如果 topology.script.file.name 没有设定，则每个 IP 都会翻译 成 / default-rack 。





有了机架感知， NameNode 就可以画出上图所示的 datanode 网络拓扑图。 D1,R1 都是交换机，最底层是 datanode 。 则 H1 的 rackid=/D1/R1/H1 ， H1 的 parent 是 R1 ， R1 的是 D1 。 这些 rackid 信息可以通过 topology.script.file.name 配置。有了这些 rackid 信息就可以计算出任意两台 datanode 之间的距离。

distance(/D1/R1/H1,/D1/R1/H1)=0 相同的 datanode

distance(/D1/R1/H1,/D1/R1/H2)=2 同一 rack 下的不同 datanode

distance(/D1/R1/H1,/D1/R1/H4)=4 同一 IDC 下的不同 datanode

distance(/D1/R1/H1,/D2/R3/H7)=6 不同 IDC 下的 datanode

1. HDFS访问

访问方式

HDFS 给应用提供了多种访问方式。用户可以通过 Java API 接口访问，也 可以通过 C 语言的封装 API 访问，还可以通过浏览器的方式访问 HDFS 中的文件。

1. HDFS 健壮性

HDFS 的主要目标就是即使在出错的情况下也要保证数据存储的可靠性。 常见的三种出错情况是：Namenode出错 ,Datanode出错和网络割裂( network partitions)。

磁盘数据错误，心跳检测和重新复制

每个 Datanode节点周期性地向Namenode发送心跳信号。网络割裂可能导致一部分Datanode跟Namenode失去联系。 Namenode 通过心跳信号的缺 失来检测这一情况，并将这些近期不再发送心跳信号Datanode标记为宕机，不会再将新的IO请求发给它们。任何存储在宕机Datanode上的数据将不再有效。Datanode 的宕机可能会引起一些数据块的副本系数低于指定值，Namenode不断地检测这些需要复制的数据块，一旦发现就启动复制操作。在下列情况下，可能需要重新复制：某个Datanode节点失效，某个副本遭到损坏，Datanode 上的硬盘错误，或者文件的副本系数增大。

1. 数据完整性

从某个Datanode获取的数据块有可能是损坏的，损坏可能是由 Datanode的存储设备错误、网络错误或者软件bug造成的。HDFS客户端软件实现了对HDFS文件内容的校验和 (checksum) 检查。当客户端创建一个新的HDFS文件，会计算这个文件每个数据块的校验和，并将校验和作为一个单独的隐藏文件保存在同一个HDFS名字空间下。当客户端获取文件内容后，它会检验从 Datanode 获取的数据跟相应的校验和文件中的校验和是否匹配，如果不匹配，客户端可以选择从其他 Datanode 获取该数据块的副本。

1. HDFS 文件删除恢复机制

当用户或应用程序删除某个文件时，这个文件并没有立刻从 HDFS中删除。实际上， HDFS会将这个文件重命名转移到 /trash 目录。只要文件还在 /trash 目录中，该文件就可以被迅速地恢复。文件在 /trash 中保存的时间是可 配置的，当超过这个时间时， Namenode 就会将该文件从名字空间中删除。 删除文件会使得该文件相关的数据块被释放。注意，从用户删除文件到 HDFS 空闲空间的增加之间会有一定时间的延迟。

只要被删除的文件还在 /trash 目录中，用户就可以恢复这个文件。如果 用户想恢复被删除的文件，他 / 她可以浏览 /trash 目录找回该文件。 /trash 目 录仅仅保存被删除文件的最后副本。 /trash 目录与其他的目录没有什么区别 ，除了一点：在该目录上 HDFS 会应用一个特殊策略来自动删除文件。目前 的默认策略是删除 /trash 中保留时间超过 6 小时的文件。将来，这个策略可以 通过一个被良好定义的接口配置。