

# Hadoop分布式文件系统：架构和设计

by Dhruba Borthakur

## 目录

|                               |   |
|-------------------------------|---|
| 1 引言 .....                    | 3 |
| 2 前提和设计目标 .....               | 3 |
| 2.1 硬件错误 .....                | 3 |
| 2.2 流式数据访问 .....              | 3 |
| 2.3 大规模数据集 .....              | 3 |
| 2.4 简单的一致性模型 .....            | 3 |
| 2.5 “移动计算比移动数据更划算” .....      | 4 |
| 2.6 异构软硬件平台间的可移植性 .....       | 4 |
| 3 Namenode 和 Datanode .....   | 4 |
| 4 文件系统的名字空间 (namespace) ..... | 5 |
| 5 数据复制 .....                  | 6 |
| 5.1 副本存放：最最开始的一步 .....        | 7 |
| 5.2 副本选择 .....                | 7 |
| 5.3 安全模式 .....                | 7 |
| 6 文件系统元数据的持久化 .....           | 8 |
| 7 通讯协议 .....                  | 8 |
| 8 健壮性 .....                   | 9 |
| 8.1 磁盘数据错误，心跳检测和重新复制 .....    | 9 |
| 8.2 集群均衡 .....                | 9 |
| 8.3 数据完整性 .....               | 9 |

|      |                |    |
|------|----------------|----|
| 8.4  | 元数据磁盘错误 .....  | 10 |
| 8.5  | 快照 .....       | 10 |
| 9    | 数据组织 .....     | 10 |
| 9.1  | 数据块 .....      | 10 |
| 9.2  | Staging .....  | 10 |
| 9.3  | 流水线复制 .....    | 11 |
| 10   | 可访问性 .....     | 11 |
| 10.1 | DFSShell ..... | 11 |
| 10.2 | DFSAdmin ..... | 12 |
| 10.3 | 浏览器接口 .....    | 12 |
| 11   | 存储空间回收 .....   | 12 |
| 11.1 | 文件的删除和恢复 ..... | 12 |
| 11.2 | 减少副本系数 .....   | 13 |
| 12   | 参考资料 .....     | 13 |

## 1. 引言

Hadoop分布式文件系统(HDFS)被设计成适合运行在通用硬件(commodity hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS是一个高度容错性的系统，适合部署在廉价的机器上。HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS放宽了一部分POSIX约束，来实现流式读取文件系统数据的目的。HDFS在最开始是作为Apache Nutch搜索引擎项目的基础架构而开发的。HDFS是Apache Hadoop Core项目的一部分。这个项目的地址是<http://hadoop.apache.org/core/>。

## 2. 前提和设计目标

### 2.1. 硬件错误

硬件错误是常态而不是异常。HDFS可能由成百上千的服务器所构成，每个服务器上存储着文件系统的部分数据。我们面对的现实是构成系统的组件数目是巨大的，而且任一组件都有可能失效，这意味着总是有一部分HDFS的组件是不工作的。因此错误检测和快速、自动的恢复是HDFS最核心的架构目标。

### 2.2. 流式数据访问

运行在HDFS上的应用和普通的应用不同，需要流式访问它们的数据集。HDFS的设计中更多的考虑到了数据批处理，而不是用户交互处理。比之数据访问的低延迟问题，更关键的在于数据访问的高吞吐量。POSIX标准设置的很多硬性约束对HDFS应用系统不是必需的。为了提高数据的吞吐量，在一些关键方面对POSIX的语义做了一些修改。

### 2.3. 大规模数据集

运行在HDFS上的应用具有很大的数据集。HDFS上的一个典型文件大小一般都在G字节至T字节。因此，HDFS被调节以支持大文件存储。它应该能提供整体上高的数据传输带宽，能在一个集群里扩展到数百个节点。一个单一的HDFS实例应该能支撑数以千万计的文件。

### 2.4. 简单的一致性模型

HDFS应用需要一个“一次写入多次读取”的文件访问模型。一个文件经过创建、写入

和关闭之后就不需要改变。这一假设简化了数据一致性问题，并且使高吞吐量的数据访问成为可能。Map/Reduce应用或者网络爬虫应用都非常适合这个模型。目前还有计划在将来扩充这个模型，使之支持文件的附加写操作。

### 2.5. “移动计算比移动数据更划算”

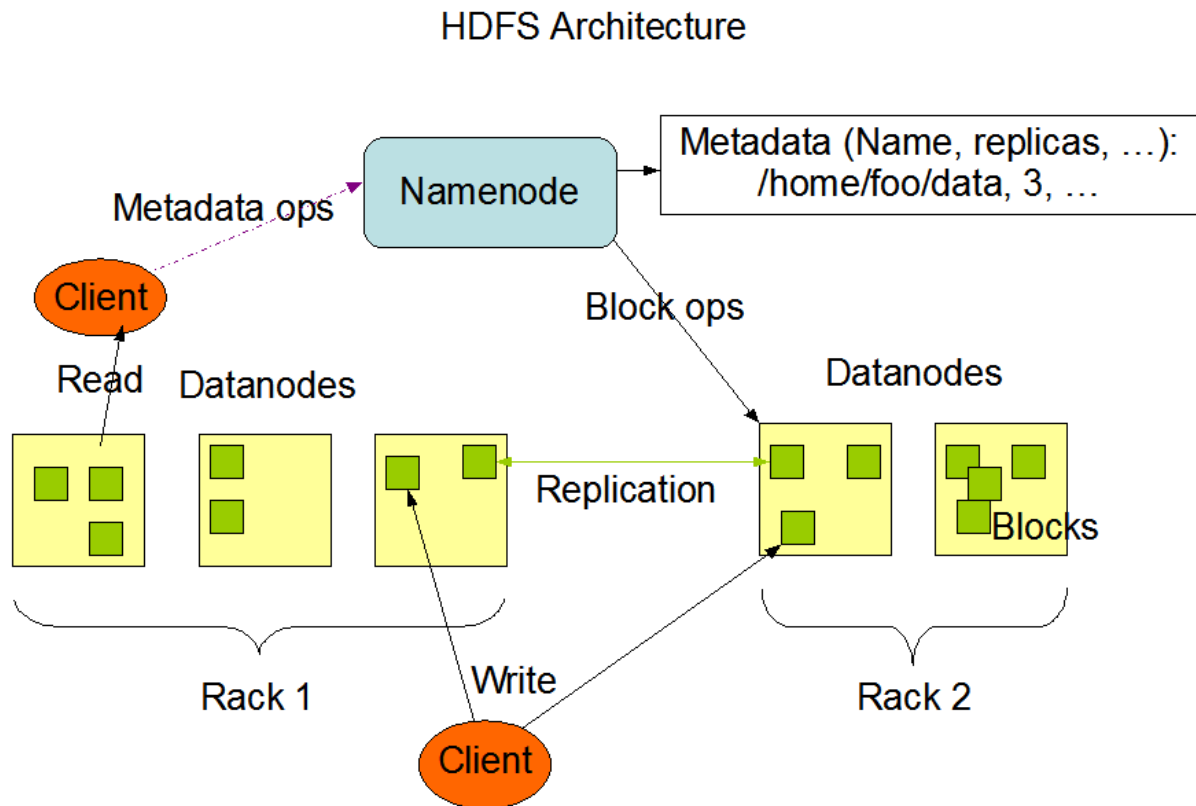
一个应用请求的计算，离它操作的数据越近就越高效，在数据达到海量级别的时候更是如此。因为这样就能降低网络阻塞的影响，提高系统数据的吞吐量。将计算移动到数据附近，比之将数据移动到应用所在显然更好。HDFS为应用提供了将它们自己移动到数据附近的接口。

### 2.6. 异构软硬件平台间的可移植性

HDFS在设计的时候就考虑到平台的可移植性。这种特性方便了HDFS作为大规模数据应用平台的推广。

## 3. Namenode 和 Datanode

HDFS采用master/slave架构。一个HDFS集群是由一个Namenode和一定数目的Datanodes组成。Namenode是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。集群中的Datanode一般是一个节点一个，负责管理它所在节点上的存储。HDFS暴露了文件系统的名字空间，用户能够以文件的形式在上面存储数据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在了一组Datanode上。Namenode执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体Datanode节点的映射。Datanode负责处理文件系统客户端的读写请求。在Namenode的统一调度下进行数据块的创建、删除和复制。



Namenode和Datanode被设计成可以在普通的商用机器上运行。这些机器一般运行着GNU/Linux操作系统(OS)。HDFS采用Java语言开发，因此任何支持Java的机器都可以部署Namenode或Datanode。由于采用了可移植性极强的Java语言，使得HDFS可以部署到多种类型的机器上。一个典型的部署场景是一台机器上只运行一个Namenode实例，而集群中的其它机器分别运行一个Datanode实例。这种架构并不排斥在一台机器上运行多个Datanode，只不过这样的情况比较少见。

集群中单一Namenode的结构大大简化了系统的架构。Namenode是所有HDFS元数据的仲裁者和管理者，这样，用户数据永远不会流过Namenode。

#### 4. 文件系统的名字空间 (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。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。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。

## 5. 数据复制

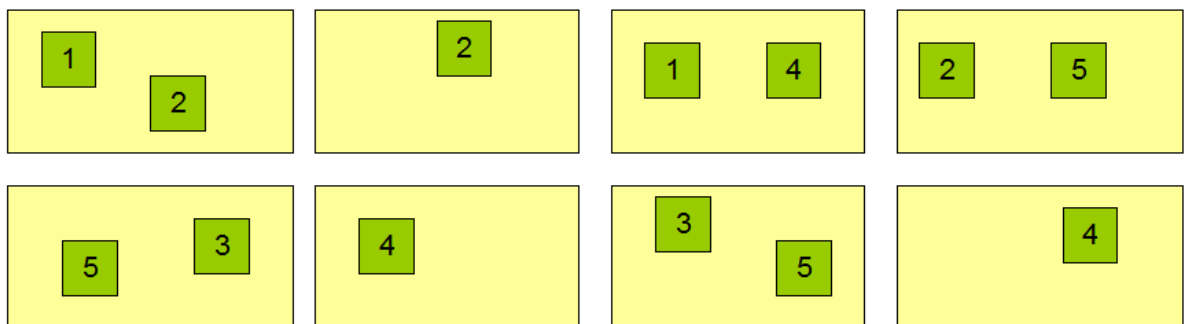
HDFS被设计成能够在一个大集群中跨机器可靠地存储超大文件。它将每个文件存储成一系列的数据块，除了最后一个，所有的数据块都是同样大小的。为了容错，文件的所有数据块都会有副本。每个文件的数据块大小和副本系数都是可配置的。应用程序可以指定某个文件的副本数目。副本系数可以在文件创建的时候指定，也可以在之后改变。HDFS中的文件都是一次性写入的，并且严格要求在任何时候只能有一个写入者。

Namenode全权管理数据块的复制，它周期性地从集群中的每个Datanode接收心跳信号和块状态报告(Blockreport)。接收到心跳信号意味着该Datanode节点工作正常。块状态报告包含了一个该Datanode上所有数据块的列表。

### Block Replication

```
Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
```

### Datanodes



### 5.1. 副本存放：最最开始的一步

副本的存放是HDFS可靠性和性能的关键。优化的副本存放策略是HDFS区别于其他大部分分布式文件系统的重要特性。这种特性需要做大量的调优，并需要经验的积累。HDFS采用一种称为机架感知(rack-aware)的策略来改进数据的可靠性、可用性和网络带宽的利用率。目前实现的副本存放策略只是在这个方向上的第一步。实现这个策略的短期目标是验证它在生产环境下的有效性，观察它的行为，为实现更先进的策略打下测试和研究的基础。

大型HDFS实例一般运行在跨越多个机架的计算机组成的集群上，不同机架上的两台机器之间的通讯需要经过交换机。在大多数情况下，同一个机架内的两台机器间的带宽会比不同机架的两台机器间的带宽大。

通过一个[机架感知](#)的过程，Namenode可以确定每个Datanode所属的机架id。一个简单但没有优化的策略就是将副本存放在不同的机架上。这样可以有效防止当整个机架失效时数据的丢失，并且允许读数据的时候充分利用多个机架的带宽。这种策略设置可以将副本均匀分布在集群中，有利于当组件失效情况下的负载均衡。但是，因为这种策略的一个写操作需要传输数据块到多个机架，这增加了写的代价。

在大多数情况下，副本系数是3，HDFS的存放策略是将一个副本存放在本地机架的节点上，一个副本放在同一机架的另一个节点上，最后一个副本放在不同机架的节点上。这种策略减少了机架间的数据传输，这就提高了写操作的效率。机架的错误远远比节点的错误少，所以这个策略不会影响到数据的可靠性和可用性。于此同时，因为数据块只放在两个（不是三个）不同的机架上，所以此策略减少了读取数据时需要的网络传输总带宽。在这种策略下，副本并不是均匀分布在不同的机架上。三分之一的副本在一个节点上，三分之二的副本在一个机架上，其他副本均匀分布在剩下的机架中，这一策略在不损害数据可靠性和读取性能的情况下改进了写的性能。

当前，这里介绍的默认副本存放策略正在开发的过程中。

### 5.2. 副本选择

为了降低整体的带宽消耗和读取延时，HDFS会尽量让读取程序读取离它最近的副本。如果在读取程序的同一个机架上有一个副本，那么就读取该副本。如果一个HDFS集群跨越多个数据中心，那么客户端也将首先读本地数据中心的副本。

### 5.3. 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。

## 6. 文件系统元数据的持久化

Namenode上保存着HDFS的名字空间。对于任何对文件系统元数据产生修改的操作, 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。

Datanode将HDFS数据以文件的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。

## 7. 通讯协议

所有的HDFS通讯协议都是建立在TCP/IP协议之上。客户端通过一个可配置的TCP端口连



接到Namenode，通过ClientProtocol协议与Namenode交互。而Datanode使用DatanodeProtocol协议与Namenode交互。一个远程过程调用(RPC)模型被抽象出来封装ClientProtocol和Datanodeprotocol协议。在设计上，Namenode不会主动发起RPC，而是响应来自客户端或Datanode的RPC请求。

## 8. 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)。

### 8.1. 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。

### 8.2. 集群均衡

HDFS的架构支持数据均衡策略。如果某个Datanode节点上的空闲空间低于特定的临界点，按照均衡策略系统就会自动地将数据从这个Datanode移动到其他空闲的Datanode。当对某个文件的请求突然增加，那么也可能启动一个计划创建该文件新的副本，并且同时重新平衡集群中的其他数据。这些均衡策略目前还没有实现。

### 8.3. 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。

## 8.4. 元数据磁盘错误

FsImage和Editlog是HDFS的核心数据结构。如果这些文件损坏了，整个HDFS实例都将失效。因而，Namenode可以配置成支持维护多个FsImage和Editlog的副本。任何对FsImage或者Editlog的修改，都将同步到它们的副本上。这种多副本的同步操作可能会降低Namenode每秒处理的名字空间事务数量。然而这个代价是可以接受的，因为即使HDFS的应用是数据密集的，它们也非元数据密集的。当Namenode重启的时候，它会选取最近的完整的FsImage和Editlog来使用。

Namenode是HDFS集群中的单点故障(single point of failure)所在。如果Namenode机器故障，是需要手工干预的。目前，自动重启或在另一台机器上做Namenode故障转移的功能还没实现。

## 8.5. 快照

快照支持某一特定时刻的数据的复制备份。利用快照，可以让HDFS在数据损坏时恢复到过去一个已知正确的时间点。HDFS目前还不支持快照功能，但计划在将来的版本进行支持。

# 9. 数据组织

## 9.1. 数据块

HDFS被设计成支持大文件，适用HDFS的是那些需要处理大规模的数据集的应用。这些应用都是只写入数据一次，但却读取一次或多次，并且读取速度应能满足流式读取的需要。HDFS支持文件的“一次写入多次读取”语义。一个典型的数据块大小是64MB。因而，HDFS中的文件总是按照64M被切分成不同的块，每个块尽可能地存储于不同的Datanode中。

## 9.2. Staging

客户端创建文件的请求其实并没有立即发送给Namenode，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。

上述方法是对在HDFS上运行的目标应用进行认真考虑后得到的结果。这些应用需要进行文件的流式写入。如果不采用客户端缓存，由于网络速度和网络堵塞会对吞吐量造成比较大的影响。这种方法并不是没有先例的，早期的文件系统，比如AFS，就用客户端缓存来提高性能。为了达到更高的数据上传效率，已经放松了POSIX标准的要求。

### 9.3. 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。

## 10. 可访问性

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。通过WebDAV协议访问的方式正在开发中。

### 10.1. DFSShell

HDFS以文件和目录的形式组织用户数据。它提供了一个命令行的接口(DFSShell)让用户与HDFS中的数据进行交互。命令的语法和用户熟悉的其他shell(例如 bash, csh)工具类似。下面是一些动作/命令的示例：

| 动作                 | 命令                            |
|--------------------|-------------------------------|
| 创建一个名为 /foodir 的目录 | bin/hadoop dfs -mkdir /foodir |
| 创建一个名为 /foodir 的目录 | bin/hadoop dfs -mkdir /foodir |

|                               |  |
|-------------------------------|--|
| 查看名为 /foodir/myfile.txt 的文件内容 | bin/hadoop dfs -cat /foodir/myfile.txt |
|-------------------------------|--|

DFSAdmin 可以用在那些通过脚本语言和文件系统进行交互的应用程序上。

## 10.2. DFSAdmin

DFSAdmin 命令用来管理HDFS集群。这些命令只有HDFS的管理员才能使用。下面是一些动作/命令的示例:

| 动作                         | 命令   |
|----------------------------|--|
| 将集群置于安全模式                  | bin/hadoop dfsadmin -safemode enter            |
| 显示Datanode列表               | bin/hadoop dfsadmin -report                    |
| 使Datanode节点 datanodename退役 | bin/hadoop dfsadmin -decommission datanodename |

## 10.3. 浏览器接口

一个典型的HDFS安装会在一个可配置的TCP端口开启一个Web服务器用于暴露HDFS的名字空间。用户可以用浏览器来浏览HDFS的名字空间和查看文件的内容。

## 11. 存储空间回收

### 11.1. 文件的删除和恢复

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。

## 11.2. 减少副本系数

当一个文件的副本系数被减小后，Namenode会选择过剩的副本删除。下次心跳检测时会将该信息传递给Datanode。Datanode遂即移除相应的数据块，集群中的空闲空间加大。同样，在调用setReplication API结束和集群中空闲空间增加间会有一定的延迟。

## 12. 参考资料

HDFS Java API: <http://hadoop.apache.org/core/docs/current/api/>

HDFS 源代码: [http://hadoop.apache.org/core/version\\_control.html](http://hadoop.apache.org/core/version_control.html)