第三次作业：开源系统的软件体系结构的建模与分析

选题：Hadoop HDFS分析

# 项目简介

Hadoop是一个由Apache基金会所开发的分布式系统基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力进行高速运算和存储。Hadoop HDFS（Hadoop Distributed File System）是一种分布式文件系统，设计用于在商用硬件上运行。HDFS是Hadoop的旗舰文件系统，用于分布式存储具有流数据访问模式的非常大的文件，在商用硬件集群上运行，是Hadoop项目的两大核心之一，另一个核心是MapReduce，整个Hadoop的体系结构主要是通过HDFS来实现分布式存储的底层支持的，并且它会通过MapReduce来实现分布式并行任务处理的程序支持。

Hadoop不需要昂贵，高度可靠的硬件。它被设计为在商用硬件集群（可从多个供应商处获得的常用硬件）上运行，对于这些集群，集群中节点故障的可能性很高，至少对于大型集群而言。HDFS旨在在面对此类故障时继续工作而不会对用户造成明显的中断。

项目代码行数：413632

项目类数量：2070

项目包数量：259

# 二、需求分析

## 2.1功能需求：

1.基本功能，数据可以在HDFS上被存储、删除、修改以及处理。

2.权限管理功能，管理员可以设置用户不同的权限，根据不同权限用户对数据以及数据的处理得到一定的分级限制。

3.数据备份，当有足够的数据备份时，才能确保当某些节点故障时能够及时的恢复数据。也就是说需要具有很强的容错能力，Hadoop HDFS需要在大规模的商用硬件集群上稳定运行，而这些机器并一定都是高可靠，高性能的，当这些机器发生故障时，HDFS需要自动进行故障检测，并及时依靠备份恢复数据。

4.需要具备对数据的安全保护，防止非法操作恶意破坏数据或者删除数据等操作，同时也要防止非法获取他人数据。

对于性能方面的需求：

1.需要具有可扩展性，可靠地存储和处理大量数据。用户可以自由的增加、删除、处理数据，同时集群中可以自如地加入或者减少机器。

2.处理数据需要有高效率，能够并行高效地处理数据。

3.需要具有可移植性，软件可能跨异构商品硬件和操作系统，需要尽量兼容不同型号的设备和系统。

4.需要实现一定的经济性，尽量节省集群中机器之间的传输以及运算的消耗。

## 2.2主要质量属性：

### 可用性

场景：当系统某些节点发生故障，导致数据的部分丢失或失效

刺激源：系统中的节点

刺激：发生故障

制品：故障中的系统

环境：系统非正常状态下

响应：启用数据副本，恢复系统正常

响应度量：在短时间内恢复系统正常工作，数据不会丢失

### 可修改性

场景：系统管理员可以修改用户的权限，将用户划分不同权限等级，控制对数据的访问

刺激源：系统管理员

刺激：修改权限的请求

制品：用户权限列表

环境：管理员工作界面

响应：系统对用户权限做出相应修改

响应度量：能100%处理管理员的合法修改请求

### 性能

场景：大量用户同时使用系统，并发出服务请求

刺激源：大量用户

刺激：海量的服务请求

制品：系统客户端

环境：系统正常工作时

响应：对用户的请求进行回应

响应度量：在10s内回应用户的请求

### 安全性

场景： 某些恶意用户

刺激源： 恶意用户

刺激： 伪造用户信息，发送删除任务或者数据的指令请求

制品： 系统客户端

环境：产品正常工作时

响应：在执行操作前，对用户进行身份验证，验证用户的是否具备执行相应操作的权利，不合法则拒绝服务请求

响应度量：，有效拦截99%恶意用户的请求

### 可测试性

场景： 测试人员测试各系统的功能

刺激源： 测试人员

刺激： 对产品功能的测试操作

制品： 测试结果报告

环境：软件测试

响应：系统提供了自带测试类可供测试使用

响应度量：测试类能完成大部分的测试工作

### 易用性

场景：HDFS系统提供了客户端，用户不需要了解底层实现，不需要知道文件系统元数据和存储位于不同的服务器上，或者该块具有多个副本，就可以轻松使用该软件，进行对文件的管理。

刺激源：使用客户端的用户

刺激：读写、删除等请求

制品：HDFS客户端

环境：正常使用操作

响应：根据收到的用户输入的指令（例如读取或写入数据），HDFS做出相应的操作

响应度量：能100%执行用户输入的正确的指令

## 2.3 其它需求

分布式文件系统通过网络为用户提供远程文件服务，它的设计目标是要使得用户感知不到其访问的是存储在远程服务器中的文件。因此，分布式文件系统的设计特别强调系统对用户的透明性。系统的透明性需求体现在多个方面：

访问透明：用户程序不必考虑文件的分布问题，能够通过完全相同的编程接口对本地文件和远程文件进行访问和操作；

位置透明：用户程序能够看到统一的文件名字空间，对文件进行访问的路径名与其在远程服务器中的存放位置无关；

移动透明：用户程序无需关心文件的存放位置发生移动所产生的影响，这一需求的满足与位置透明性的实现相关；；

性能透明：用户程序始终能够获得高效的文件访问能力，即使远程服务器上的文件服务负载在一定范围内发生了变化；

扩展透明：用户程序导致的文件服务需求的增大或者网络规模的扩张等问题，能够通过系统的自动扩展得到有效的解决；

满足用户的透明性需求对于分布式文件系统设计非常关键，直接影响了用户对远程文件的访问体验。除此以外，还有其他一些设计需求，包括分布式文件系统需要具有高可用性，能够支持异构客户端的并发访问，能够提供文件数据的多个拷贝并保证文件数据的一致性和安全性等。

# 软件体系结构设计

## 3.1系统设计体系

系统体系图：



HDFS是传统的Master-Slave架构：一个集群由一个Master节点和若干个Slave节点组成。在HDFS中，Master节点称为NameNode，Slave节点称为DataNode。

NameNode：

HDFS集群有两种类型的节点以主 - 工模式运行：名称节点（主节点）和多个数据节点（工作者）。namenode管理文件系统命名空间。它维护文件系统树以及树中所有文件和目录的元数据。此信息以两个文件的形式持久存储在本地磁盘上：命名空间映像和编辑日志。namenode还知道给定文件的所有块所在的datanode; 但是，它不会持久存储块位置，因为此信息是在系统启动时从数据节点重建的。

文件和目录由NameInode在inode上表示。Inode记录属性，如权限，修改和访问时间，命名空间和磁盘空间配额。文件内容被拆分为大块（通常为128兆字节，但用户可以逐个文件选择），并且文件的每个块都在多个DataNode上独立复制。NameNode维护命名空间树以及块到DataNode的映射。

如果没有namenode，则无法使用文件系统。事实上，如果运行namenode的机器被删除，文件系统上的所有文件都将丢失，因为无法知道如何从数据节点上的块重建文件。因此，重要的是使namenode适应失败。

DataNode：

Datanodes是文件系统的主力。它们在被告知（通过客户端或名称节点）时存储和检索块，并且它们定期向名称节点报告它们存储的块列表。

DataNode上的每个块副本由本地本机文件系统中的两个文件表示。第一个文件包含数据本身，第二个文件记录块的元数据，包括数据的校验和和生成标记。数据文件的大小等于块的实际长度，并且不需要额外的空间来将其四舍五入到传统文件系统中的标称块大小。因此，如果块是半满的，则它只需要本地驱动器上整个块的一半空间。

在启动期间，每个DataNode都连接到NameNode并执行握手。握手的目的是验证命名空间ID和DataNode的软件版本。如果其中任何一个与NameNode的匹配，则DataNode会自动关闭。

格式化时，名称空间ID将分配给文件系统实例。命名空间ID持久存储在群集的所有节点上。具有不同命名空间ID的节点将无法加入群集，从而保护文件系统的完整性。允许新初始化且没有任何名称空间ID的DataNode加入集群并接收集群的名称空间ID。

握手后，DataNode向NameNode注册。DataNode持久存储其唯一的存储ID。存储ID是DataNode的内部标识符，即使使用不同的IP地址或端口重新启动，也可以识别它。存储ID在第一次向NameNode注册时分配给DataNode，之后永远不会更改。

DataNode通过发送块报告来识别其拥有的块副本到NameNode。块报告包含块ID，生成戳记以及服务器托管的每个块副本的长度。在DataNode注册后立即发送第一个块报告。每小时发送一次后续块报告，并为NameNode提供块副本在群集上的位置的最新视图。

在正常操作期间，DataNode将心跳发送到NameNode以确认DataNode正在运行，并且它所托管的块副本可用。默认心跳间隔为3秒。如果NameNode在十分钟内没有从DataNode接收到心跳，则NameNode会认为DataNode已停止服务，并且该DataNode托管的块副本不可用。然后，NameNode计划在其他DataNode上创建这些块的新副本。

来自DataNode的心跳还包含有关总存储容量，正在使用的存储容量以及当前正在进行的数据传输的数量的信息。这些统计信息用于NameNode的块分配和负载平衡决策。

NameNode不直接向DataNode发送请求。它使用对心跳的回复来向DataNode发送指令。这些指令包括将块复制到其他节点，删除本地块副本，重新注册和发送立即块报告以及关闭节点的命令。

这些命令对于维护整体系统完整性非常重要，因此即使在大型集群上也必须保持心跳频繁。NameNode可以每秒处理数千个心跳，而不会影响其他NameNode操作。

NameNode与DataNode的关系

名称节点和数据节点是设计为跨异构操作系统在商用机器上以分离方式运行的软件组件。HDFS是使用Java编程语言构建的; 因此，任何支持Java编程语言的机器都可以运行HDFS。典型的安装集群具有运行名称节点和可能一个数据节点的专用机器。群集中的每台其他计算机都运行一个数据节点。

数据节点不断循环，向名称节点询问指令。名称节点无法直接连接到数据节点; 它只是从数据节点调用的函数返回值。每个数据节点都维护一个打开的服务器套接字，以便客户端代码或其他数据节点可以读取或写入数据 该服务器套接字的主机或端口由名称节点知道，该节点向感兴趣的客户端或其他数据节点提供信息。名称节点维护和管理对文件系统名称空间的更改。

客户端：

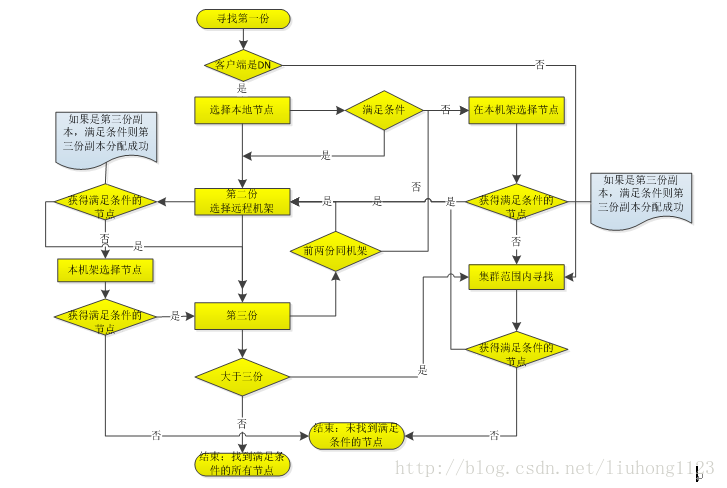
用户应用程序使用HDFS客户端访问文件系统，HDFS客户端是一个导出HDFS文件系统接口的库。与大多数传统文件系统一样，HDFS支持读取，写入和删除文件的操作，以及创建和删除目录的操作。用户通过命名空间中的路径引用文件和目录。用户应用程序不需要知道文件系统元数据和存储位于不同的服务器上，或者该块具有多个副本。

当应用程序读取文件时，HDFS客户端首先向NameNode请求托管文件块副本的DataNode列表。该列表按客户端的网络拓扑距离排序。客户端直接联系DataNode并请求传输所需的块。当客户端写入时，它首先要求NameNode选择DataNodes来托管文件第一个块的副本。客户端从节点到节点组织管道并发送数据。当第一个块被填满时，客户端请求选择新的DataNode来托管下一个块的副本。组织新的管道，客户端发送文件的其他字节。

连接交互：

使用TCP/IP ， 用户与NameNode之间使用Client Protocol ，NameNode与DataNode之间使用DataNode Protocol

### HDFS块分配策略

块分配策略流程图

注：上图提到的远程机架与本地机架是相对于找到的第一个副本的目标节点来说。

一个客户端a机器发起请求分配块请求，NN接收到请求后，执行如下块分配流程：

1）   如果a不是一个DataNode，则在集群范围内随机选择一个节点作为目标节点，否则执行下面的2,3步骤；

2）  判断a机器是否符合存储数据块的目标节点，如果符合，第一个块副本分配完毕；

3）如果a机器不符合作为目标节点，则在于与a机器同机架范围内寻找，如果找到目标节点，第一个块副本分配完毕；

4）如果在同一个机架内未找到符合要求的目标节点，则在集群内随机查找，找到则第一个块副本分配完毕，否则未找到符合条件的块，块分配失败；

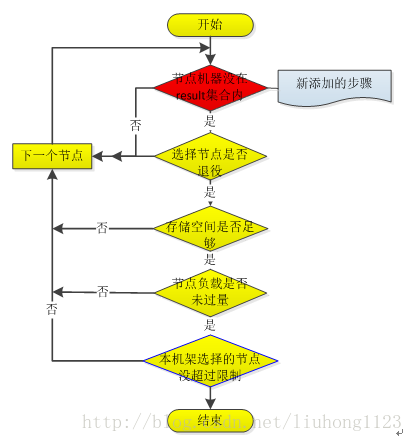
5）如果已经成功分配第一个块副本，则与a不同机架的远程机架内寻找目标节点，如果符合，第二个块副本分配完毕；

6）如果在远程机架内未找到符合要求的目标节点，在与a相同的本机架寻找，如果找到则第二个块副本分配完毕；否则未找到符合条件的块，第二份块分配失败；

7）如果前2个块副本分配成功，则准备分配第三个副本的目标节点，首先会判断前两份是否在同一个机架，如果是，则在远程机架寻找目标节点，找到则第三份副本分配完毕；如果前两份在不同机架，则在与a相同机架内寻找，如果找到则第三份副本分配完毕，否则在集群范围寻找，找到则第三份分配完毕，否则第三份分配失败

8）如果块副本大于三分，则在集群范围内随机寻找节点

在上面的块分配流程图中，反复出现满足条件的节点判断，要判断一个节点是否满足条件，需要经过如下流程，流程图如下：



当在一个范围内找到一个节点后，还需要经过如上的条件判断，才能确定一个DataNode进程是否可以作为目标节点：

1）  如果没有节点机器被选择，则该节点可以作为备选节点，否则需要判断下一个DataNode是否符合要求；（这样就防止同一个块副本存储到同一台机器）

2）  然后判断节点是否退役，存储空间是否足够，负载是否大于2倍平均负载，本机架选择的节点是否超过限制，如果均满足，则该datanode符合要求，否则需要判断下一个DataNode是否符合要求；

### HDFS存储策略

**介绍**

归档存储（Archival Storage）是将不断增长的存储容量与计算容量分离的解决方案。密度更高、存储成本更低、计算能力更低的节点正在变得可用，可以用作集群中的冷存储。根据策略，可以将数据从热节点移动到冷节点。向冷存储添加更多节点可以独立于集群中的计算容量来增加存储。

Heterogeneous Storage和归档存储（Archival Storage）提供的框架将HDFS架构推广到包含SSD和memory在内的其他类型的存储介质。用户为了更好的性能可以选择将数据存储到SSD或者memory。

**存储类型和存储策略**

存储类型：ARCHIVE, DISK, SSD and RAM\_DISK

Heterogeneous Storage (HDFS-2832)的第一个阶段，将datanode 存储模型从可能对应于多个物理存储介质的单个存储转变成存储集合，其中每个存储对应于一个物理存储介质。它还添加了存储类型，DISK和SSD的概念，其中DISK是默认的存储类型。

为了支持归档存储，增加了一种新的存储类型ARCHIVE，它具有较高的存储密度（PB级别的存储容量），但是计算能力较小。

添加了另一种新的存储类型RAM \_ DISK，用于支持在内存中写入单个副本文件。

存储策略：Hot, Warm, Cold, All\_SSD, One\_SSD and Lazy\_Persist

提出了一种新的存储策略概念，使文件可以根据存储策略存储在不同的存储类型中。

**目前有以下集中存储策略：**

①Hot - 用于存储和计算。受欢迎且仍用于处理的数据将保留在此策略中。当数据块处于hot状态时，所有复制副本都存储在DISK中。

②Cold -用于有限计算的存储。不再使用的数据或者需要归档的数据将从hot存储移动到cold存储。当数据处于cold状态时，所有副本将会存储到ARCHIVE。

③Warm -部分hot部分cold。当块是warm时，一部分副本存储到DISK，其余副本存储到ARCHIVE。

④All\_SSD -存储所有副本到SSD。

⑤One\_SSD - 用于存储一个副本到SSD，其他副本存储到DISK。

⑥Lazy\_Persist - 用于内存中写入单个副本的块。副本首先写入RAM\_DISK，然后延迟保存到磁盘。

更正式地说，存储策略由以下字段组成:

①Policy ID

②Policy name

③块放置的存储类型列表

④用于创建文件的备用存储类型列表

⑤用于放置副本的备用存储类型列表

如果有足够的空间，将根据#3中指定的存储类型列表存储数据块副本。当#3中部分存储类型空间不足时，#4 and #5中指定的备用的存储类型列表将会替换空间不够的存储类型来用于文件创建和副本复制。

### Block之副本放置策略

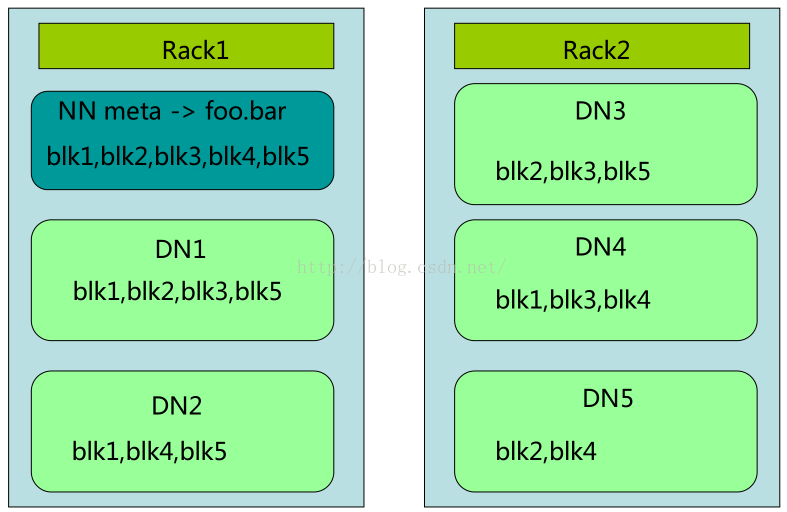
第一副本：放置在上传文件DataNode，如果是集群外提交，由NameNode选择一台磁盘不太满，CPU不太忙的节点。

第二副本：放置在于第一副本不同的机架的节点上

第三副本：与第二个副本相同集群的节点

第四副本   随机节点

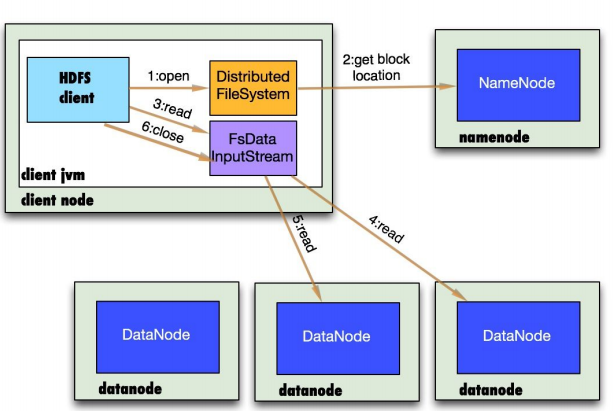
总之：第一个副本存放节点位置是随机不固定,只要保证三个副本节点不同时存放在同一个机架上即可。如bk1第一个副本副本在DN1(属于Back1),第二个在DN2(属于Back1)中,而第三个在DN4(属于Back2)中.同样bk5第一个副本在DN3(属于Back2)第二个副本在DN1(属于Back1)二第三个副本在DN2(属于Back1)。这些都符合上面的条件



### HDFS读写策略

1. **读文件操作**

HDFS client上传数据到HDFS时，首先，在本地缓存数据，当数据达到一个block大小时，请求NameNode分配一个block。 NameNode会把block所在的DataNode的地址告诉HDFS client。 HDFS client会直接和DataNode通信，把数据写到DataNode节点一个block文件中。



1. 首先调用FileSystem对象的open方法，其实是一个DistributedFileSystem的实例。

2. DistributedFileSystem通过rpc获得文件的第一批block的locations，同一个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。

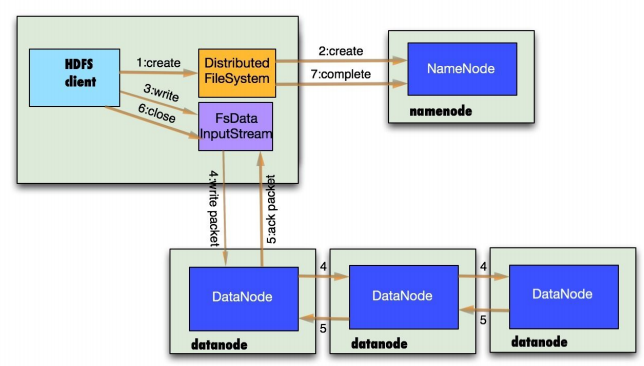
3. 前两步会返回一个FSDataInputStream对象，该对象会被封装DFSInputStream对象，DFSInputStream可 以方便的管理datanode和namenode数据流。客户端调用read方法，DFSInputStream最会找出离客户端最近的datanode 并连接。

4. 数据从datanode源源不断的流向客户端。

5. 如果第一块的数据读完了，就会关闭指向第一块的datanode连接，接着读取下一块。这些操作对客户端来说是透明的，客户端的角度看来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block的locations，然后继续读，如果所有的块都读完，这时就会关闭掉所有的流。

1. **写文件操作**



1. 客户端通过调用DistributedFileSystem的create方法创建新文件。

2. DistributedFileSystem通过RPC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，与读文件的时候相似， 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的packet，然后排成队 列data quene。

4. DataStreamer会去处理接受data quene，它先询问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），把他们排成一个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还有一个对列叫ack quene，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 quene才会把对应的packet包移除掉。

如果在写的过程中某个datanode发生错误，会采取以下几步：

1) pipeline被关闭掉；

2)为了防止防止丢包ack quene里的packet会同步到data quene里；

3)把产生错误的datanode上当前在写但未完成的block删掉；

4)block剩下的部分被写到剩下的两个正常的datanode中；

5)namenode找到另外的datanode去创建这个块的复制。当然，这些操作对客户端来说是无感知的。

6. 客户端完成写数据后调用close方法关闭写入流。

7. DataStreamer把剩余得包都刷到pipeline里，然后等待ack信息，收到最后一个ack后，通知datanode把文件标视为已完成。

注意：客户端执行write操作后，写完的block才是可见的，正在写的block对客户端是不可见的，只有 调用sync方法，客户端才确保该文件的写操作已经全部完成，当客户端调用close方法时，会默认调用sync方法。是否需要手动调用取决你根据程序需 要在数据健壮性和吞吐率之间的权衡。

## 3.2系统模块

### 设计模式

根据HDFS项目的源码结构，可以分为四大模块，分别为httpfs、rbf、client、nfs，其中用到了诸多设计模式，例如httpfs模块中中使用了单例设计模式，rbf模块中有命令模式，client模块中有代理模式，在之后的模块分析中会做具体阐述。

### Httpfs（Hadoop HDFS over HTTP）

HttpFS是提供REST HTTP接口的服务器，可以支持全部HDFS文件系统操作（读和写），它通过webhdfs REST HTTP API来进行交互。这个功能由cloudera提供给Apache主分支。

HttpFS可以用于在不同的Hadoop版本间传输数据（避免了RPC版本问题），例如使用Hadoop DistCP。

HTTP可以用于访问防火墙内的HDFS集群数据（HttpFS可以作为一个网关角色，是唯一可以穿过防火墙访问内部集群数据的系统）。

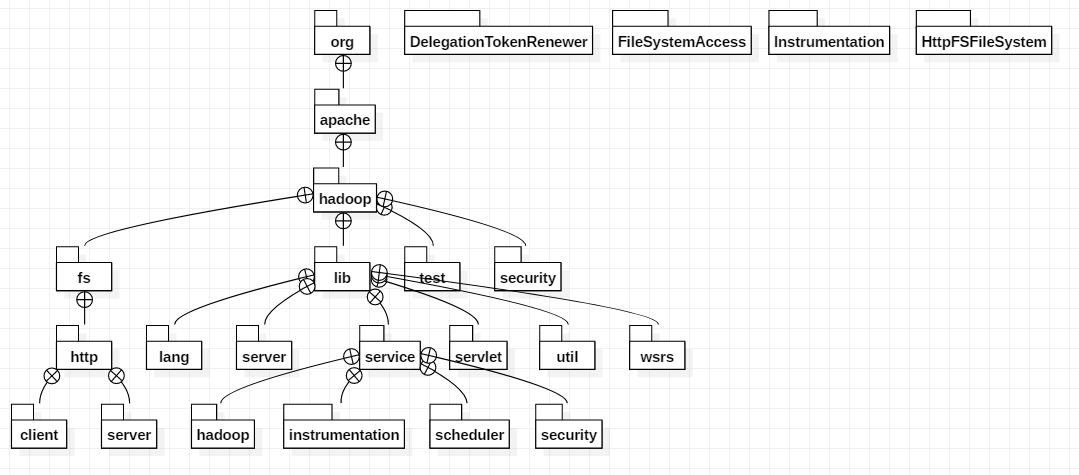
HttpFS可以通过HTTP工具（比如curl和wget）和各种编程语言的HTTP包（不局限Java）来访问数据。

webhdfs 客户端文件系统实现可以使用HDFS文件系统命令行命令（如hadoop dfs）以及使用HDFS Java API来访问HttpFS。

HttpFS内置安全特性支持Hadoop伪身份验证和HTTP SPNEGO Kerberos 和其他插件式（pluggable ）验证机制。它还提供了Hadoop代理用户的支持。

HttpFS本质上是一个代理服务，它部署在能够完全访问HDFS集群的网络内，对于文件CURD的操作全部提交给HttpFS服务，然后由它去更HDFS集群交互，所以客户端不需要能够访问所有HDFS主机。

使用逆向软件得到的httpfs包结构，如图：



其中：

* Server：启动web app，提供restful服务的相关类
* HttpFSServerWebApp.java用单例的模式提供对Server的访问。

单例模式（Singleton Pattern）是 Java 中最简单的设计模式之一。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。

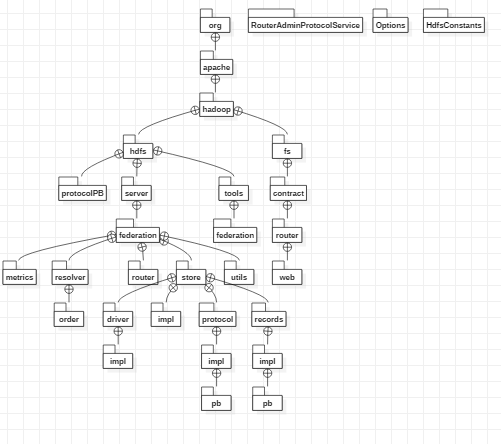
优点：1、在内存里只有一个实例，减少了内存的开销，尤其是频繁的创建和销毁实例。2、避免对资源的多重占用。

缺点**：**没有接口，不能继承，与单一职责原则冲突，一个类应该只关心内部逻辑，而不关心外面怎么样来实例化。

* server ：Server的framework类，包括Server/Service相关接口和抽象父类
* Service的具体实现类；大部分restful接口都是通过代理给这些Service实现的。
* Servlet：和servlet规范联系紧密的类，包括filter和listener

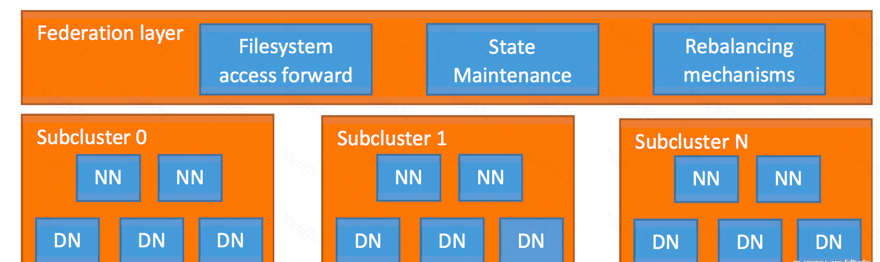
### Rbf（Router-Based Federation）

逆向得到的RBF包结构



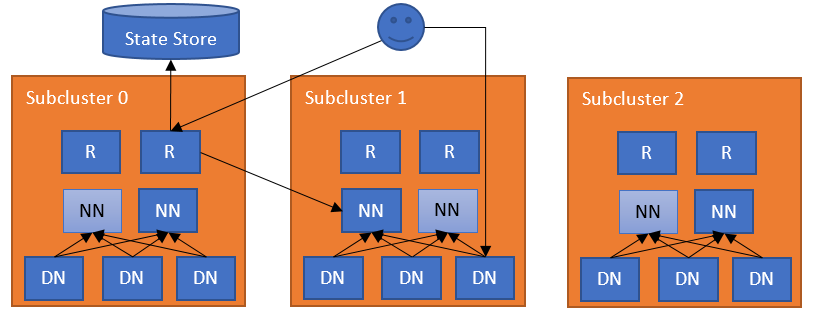
HDFS 的 Master/Slave 架构，使得其具有单点瓶颈，即随着业务数据的大规模膨胀，Master 节点在元数据存储与提供服务上都会存在瓶颈。为了克服 HDFS 单点瓶颈存在的扩展性、性能、隔离问题，社区提出了Federation方案来进行解决。将大的集群切分成小的子集群来进行管理。但是使用该方案之后，暴露给客户的问题就是，同一个集群出现了多个命名空间（namespace），客户需要知道读写的数据在哪个命名空间下才可以进行操作。

为了解决上述问题，社区提出了Router-Based Federation 方案，对外提供了 Router 服务，包含在 Federation layer 中，如下图所示。这个 Router 服务将允许用户透明地访问任何子集群，让子集群独立管理自己的 Blockpool。为了实现这些目标，Federation layer 必须将 Block 访问引导至适当的子群集。同时，它具有可扩展性，高可用性和容错性。



Federation layer 包含多个组件。Router 是一个与 NameNode 具有相同接口的组件，根据 State Store 的元数据信息将客户端请求转发给正确的子集群。State Store 组件包含了远程挂载表（具有 ViewFS 特性，但在客户端之间共享）和有关 SubCluster 的负载/空间信息。

下图架构中显示每个子集群增加了 Router（标记为“R”）和逻辑集中式（但物理分布式）的状态存储（State Store），以及每个 SubCluster 的 Namenodes（“NN”）和 Datanodes（“DN”）。 最简单的配置在每个NameNode计算机上部署路由器。路由器监视本地NameNode并将状态检测到状态存储。当常规DFS客户端联系任何路由器以访问联合文件系统中的文件时，路由器会检查状态存储中的挂载表（即本地缓存）以找出包含该文件的子集群。然后，它检查State Store中的Membership表（即本地缓存），以查找负责子集群的NameNode。在识别出正确的NameNode之后，路由器代理请求。客户端直接访问Datanodes。这种方法与 YARN Federation（YARN-2915）具有相同的架构。



rbf 模块中分为 ProtocalPB 、 server 、tool 三个子模块

#### ProtocolPB：

这个子模块主要描述了客户端client与NameNode通信的协议，这个协议必须是安全可靠的。

该模块只有三个类，RouterAdminProtocolPB（负责调用通信协议）、RouterAdminProtocolServerSideTranslatorPB（负责服务器端的数据转换，将PB格式的数据转换为通用协议数据类型）、RouterAdminProtocolTranslatorPB（处理NameNode的数据转化协议）

#### Server

该模块描述了RBF的一些核心服务，其中包含了多个包，类也较多。

##### Metric：

关于RBF的一些统计信息。该包使用了JMX（Java Management Extensions，即Java管理扩展，是Java平台上为应用程序、设备、系统等植入管理功能的框架）进行开发，为RBF收集统计路由信息、存储状态信息提供良好接口，同时服务远程过程调用。Metrics中的几个JAVA文件分别实现了上述几个功能。

##### Resolver：

该包中含有数据解析程序。包中包含HDFS使用的独立数据解析程序。 数据解析器从群集中收集数据，包括从状态存储中收集数据，解析器公开了HDFS联合使用的API，以收集聚合的缓存数据，以便在实时请求处理中使用。

##### Router：

这个包实现了HDFS联邦集群中路由器的核心服务。路由器是RBF中的重要组件，router的作用如下：

1.向客户端提供一个全局 Namenode 接口并负责转发请求正确的子群集中的 Active Namenode；

2.在 State Store 中维护关于 Namenode 的信息。

对于 Namenode 信息的维护，Router 定期检查一个 Namenode 的状态和向 State Store 报告其高可用性（HA）状态和负载/空间状态。 为了提高 Namenode HA 的性能，Router 使用 State Store 中的高可用性状态信息,以将请求转发到最有可能处于活动状态的 Namenode。

为了与用户和管理员进行交互，Router 公开了多个接口。包括 RPC、Admin、WebUI 。

RPC 实现了客户端与 HDFS 交互的最常见接口。 目前仅支持使用普通 MapReduce，Spark 和 Hive ( on Tez，Spark 和MapReduce）。一些高级特性，如快照、加密和分层存储在未来版本实现。 所有未实现的功能都会抛出异常。

Admin 为管理员实现的一个 RPC 接口，包括从子集群获取信息、添加/删除条目到 mout table。也可以通过命令行获取和修改 Federation 信息。WebUI 实现了一个可视化 Federation 状态，模仿了当前的 Namenode UI，除此之外，还包含 mout table，每个子集群的成员信息以及 Router 的状态。

##### Store

主要负责一些记录的存储。Store包中最重要的就是state store，state store主要负责维护以下几个数据：

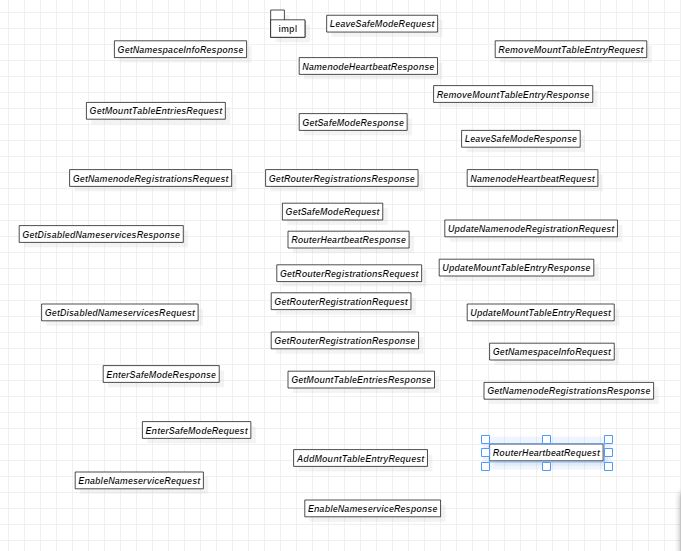
1.子集群的块访问负载，可用磁盘空间，HA 状态等状态；

2.文件夹/文件和子集群之间的映射，即远程 Mount Table；

3。Router 的状态。State Store 的后端存储是可配置的。 既可以可以存储在文件中，也可以存在 ZooKeeper 中。

4. Membership 反映了 Federation 中的 Namenode 的状态。包括有关子集群的信息，例如存储量和节点数量。Router 定期检测一个或多个 Namenode 的信息。

在Store包中的子包 protocol中提供了丰富的可调用的API来维护和使用这些存储数据。



从这些类名可以清晰看到它们的作用，例如RouterHeartbeatRequest是用来记录路由器心跳请求的。

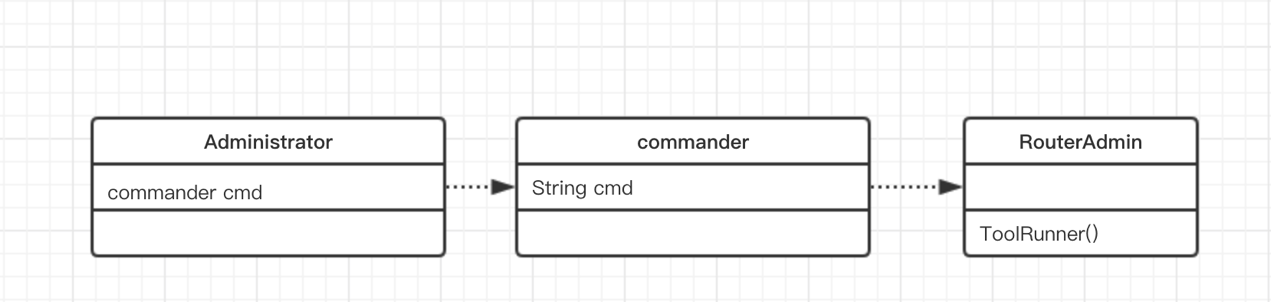
在这一块的代码设计中，可以看到适配器模式的应用，主要是接口适配器，通过一个适配器类可以很好的将使用者引入到合适接口中使用。

#### tool：

这个模块实现了一个命令行的工具，使具有权限的用户通过命令行对Federation进行管理。

该模块下只有一个包Federation，包中只有一个类RouterAdmin，该类就是对输入的命令行进行拆分，判断命令的正确性，执行命令，或者对使用者进行提示。

这里显而易见用到了命令模式(command将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可取消的操作。)



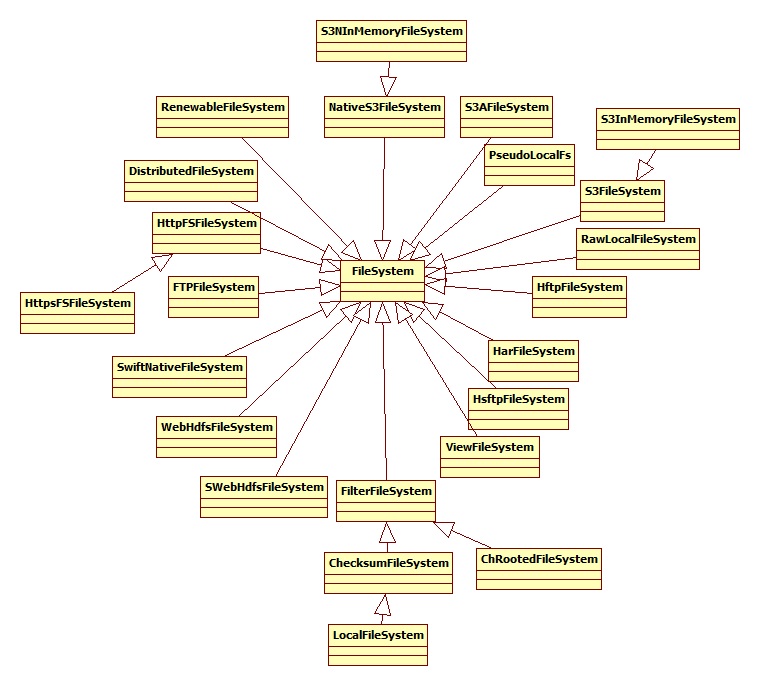
### Client

Client（客户端）通过与 NameNode和DataNode 交互访问HDFS中的文件。Client 提供了一个类似 POSIX 的文件系统接口供用户调用。

client是用户和HDFS进行交互的手段，HDFS提供了各种各样的客户端，包括命令行接口、Java API, Thrift接口、C语言库、用户空间文件系统〔Filesystem in Userspace,FUSE)等。

HDFS Client就是用来访问这个hadoop文件系统的机器. 在实际的开发环境中，在集群环境中开发往往存在很多安全隐患，例如集群文件被误删等等，所以一般的开发工作都是本地完成开发的。本地做MR开发时，由于没有hadoop环境，所以调试工作往往变的很难进行，所以在本地搭建一个hadoop client，不仅能提供本地调试环境，还能从直接从本地访问到hdfs 数据和提交任务到hadoop环境中。你可以在本地运行MR，不登陆服务器就能查看数据。

1. **HDFS文件系统层次结构**



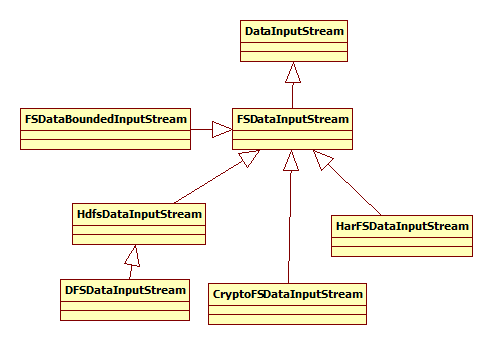
1. **HDFS client 读文件操作**

HDFS client上传数据到HDFS时，首先，在本地缓存数据，当数据达到一个block大小时，请求NameNode分配一个block。 NameNode会把block所在的DataNode的地址告诉HDFS client。 HDFS client会直接和DataNode通信，把数据写到DataNode节点一个block文件中。

如果在读数据的时候， DFSInputStream和datanode的通讯发生异常，就会尝试正在读的block的排序第二近的datanode,并且会记录哪个 datanode发生错误，剩余的blocks读的时候就会直接跳过该datanode。 DFSInputStream也会检查block数据校验和，如果发现一个坏的block,就会先报告到namenode节点，然后 DFSInputStream在其他的datanode上读该block的镜像。

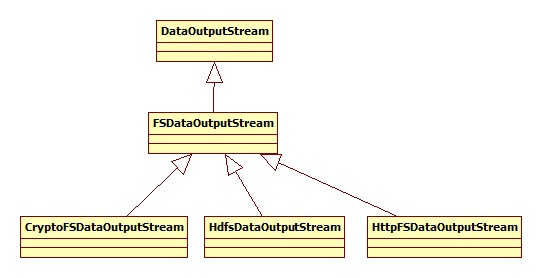
该设计就是客户端直接连接datanode来检索数据并且namenode来负责为每一个block提供最优的datanode， namenode仅仅处理block location的请求，这些信息都加载在namenode的内存中，hdfs通过datanode集群可以承受大量客户端的并发访问。

输入/输出流和文件系统相对应，输入流的结构如下：



1. **HDFS client 写文件操作**

输出流的结构如下：



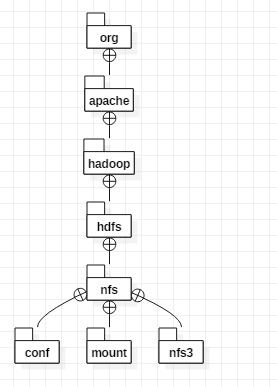
使用到的设计模式：代理模式（为其他对象提供一个代理以控制对这个对象的访问）

ClientServiceDelegate：代理了客户端服务，代理的原因是因为要做一些安全检查，状态记录。

### Nfs

hdfs是分布式系统，要想访问hdfs上的文件，可以用java api或者hadoop shell等工具，如果想操作hdfs文件系统就像操作本地文件系统一样的便捷，可以将hdfs文件系统挂载到本地的一个目录上，nfs挂载，apache hadoop2.x以后的版本中自带了一个nfs3的插件服务，满足了用户文件系统挂载的需求。

逆向得到的包结构：



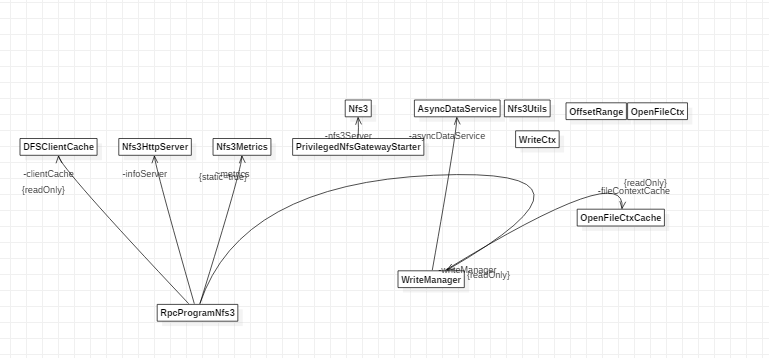
该部分主要分为三个包，conf、mount以及nfs3

Conf包主要是实现了NFS的密钥生成与配置，主要是为了安全性，NfsConfigKey类设定了密钥的格式以及参数，NfsConfiguration类对密钥进行了配置与生成。

Mount包主要实现的是一个挂载的守护进程，它可以保持活跃，相应NFS客户端的请求连接。Mountd类主要实现了调用的接口，RpcProgramMountd类主要实现了实际守护进程的编码，主要对请求连接的客户端进行安全认证，通过后可以返回目录和文件等。

Nf3这个包为NFS的核心部分，实现了具体的文档打开、传输，以及客户端连接等的功能。

类关系如下：

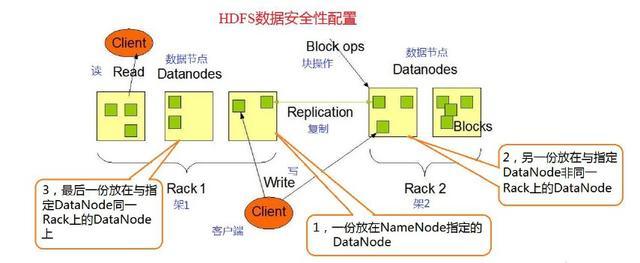


# 设计特色分析

## 4.1分布式文件系统HDFS设计特点

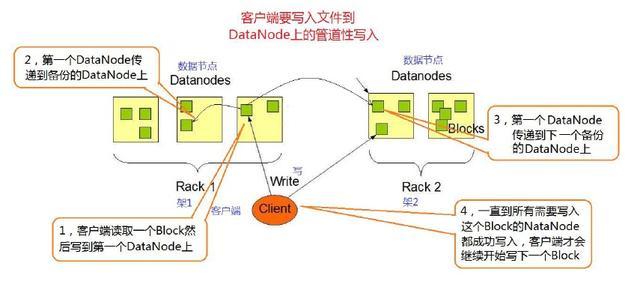
**1）Block的配置：**

默认不配置。一个Block会有三份备份，一份放在NameNode指定的DataNode，另一份放在与指定DataNode非同一Rack上的DataNode，最后一份放在与指定DataNode同一Rack上的DataNode上。备份无非就是为了数据安全，考虑同一Rack的失败情况以及不同Rack之间数据拷贝性能问题就采用这种配置方式。



**2）数据管道性写入：**

当客户端要写入文件到DataNode上，首先客户端读取一个Block然后写到第一个DataNode上，然后由第一个DataNode传递到备份的DataNode上，一直到所有需要写入这个Block的NataNode都成功写入，客户端才会继续开始写下一个Block。



**3）数据校验：**

采用CRC32 作数据校验。在文件Block 写入的时候除了写入数据还会写入校验信息，在读取的时候需要校验后再读入。

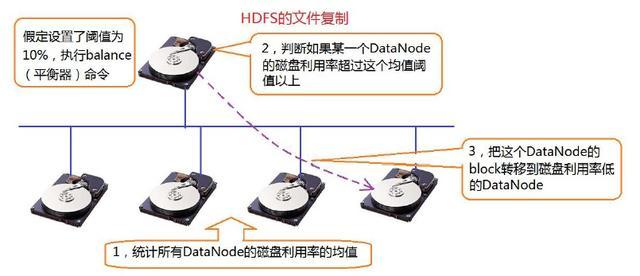
**4）健康监测：**

心跳检测DataNode 的健康状况，如果发现问题就采取数据备份的方式来保证数据的安全性。

**5）数据复制**

如果DataNode 失败、需要平衡DataNode 的存储利用率和需要平衡DataNode 数据交互压力等情况。使用HDFS 的balancer（平衡器）命令，可以配置一个Threshold（阈值）来平衡每一个DataNode 磁盘利用率。

例如，设置了阈值为10%，那么执行balance（平衡器）命令的时候，首先统计所有DataNode的磁盘利用率的均值，然后判断如果某一个DataNode 的磁盘利用率超过这个均值阈值以上，那么将会把这个DataNode 的block 转移到磁盘利用率低的DataNode，这对于新节点的加入来说十分有用。



**6）NameNod 是单点：**

如果失败的话，任务处理信息将会纪录在本地文件系统和远端的文件系统中。

**7）安全模式：**

在分布式文件系统启动的时候，开始的时候会有安全模式，当分布式文件系统处于安全模式的情况下，文件系统中的内容不允许修改也不允许删除，直到安全模式结束。

安全模式主要是为了系统启动的时候检查各个DataNode 上数据块的有效性，同时根据策略必要的复制或者删除部分数据块。

运行期通过命令也可以进入安全模式。在实践过程中，系统启动的时候去修改和删除文件也会有安全模式不允许修改的出错提示，只需要等待一会儿即可。

**8）高效：**

分布式文件系统的高效数据交互实现以及MapReduce 结合LocalData（本地数据）处理的模式，为高效处理海量的信息作了基础准备。

**9）CPU 占用：**

分布式计算程序的运行优先级非常低，一般不会对正常使用造成影响，但有部分大型软件中部分组件也可能运行于同等低优先级，这时情况下，可以手动暂停计算或是设置为不在使用计算机的时候进行计算。

## 4.2HDFS与NFS比较

NFS（networkfilesystem）是最早商用并流行至今的分布式文件系统。NFS体现了之前主流分布式文件系统的典型特征，HDFS则代表了新兴的面向云计算的分布式文件系统，它们在满足系统设计需求方面各有侧重，具体的比较见表。HDFS很好地满足了系统设计的透明性需求，在系统的扩展性、可靠性等方面具有优势，而这也正是云计算对文件服务的主要需求，因此，HDFS以及与之具有相似设计思想的GFS、KFS（KOSMOSdistributedfilesystem）等分布式文件系统在当前发挥了巨大的作用并拥有广阔的前景。以这类新兴的分布式文件系统为基础，分布式计算模型、分布式数据库等云计算关键技术也有了新的发展，例如，MapReduce计算架构能够部署在HDFS系统上实现将计算步骤推向数据节点的计算模式，HBase分布式数据库可以借助HDFS的管理能力而无需传统的数据库管理系统。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  |  | NFS | HDFS | | 系统透明性 | 访问透明 | 提供与传统的本地文件系统访问类似的接口，实现访问透明 | 未提供与传统的POSIX完全兼容的接口，访问透明性不高 | | 位置透明 | 远程文件系统的目录直接挂接在客户端文件系统上，支持位置透明 | 通过文件文本名字和统一资源标识符访问文件，支持位置透明 | | 移动透明 | 没有统一的名字空间，文件不能在服务器侧移动，不支持移动透明 | Namenode节点管理和维护统一的文件名字空间，实现移动透明 | | 性能透明 | 提供服务器侧和客户端的缓存机制，文件访问性能较高 | 支持多个文件数据块的并行访问，文件访问性能高 | | 扩展透明 | 需要用户手工设置远程文件访问的挂接位置，扩展不够透明 | 根据负载规模增删Datanode，无需用户参与，实现扩展透明 | | 文件并发更新 | | 使用网络文件锁管理机制进行并发控制，但会受NFS协议的影响 | 不支持文件的并发更新，采用租期机制限定对文件数据块的访问 | | 文件内容复制 | | 读文件可被复制到多台服务器上，但是不支持可写文件的复制 | 文件块被自动复制为多份拷贝并被分散存放到多个Datanode上 | | 文件数据一致 | | 采用单一拷贝机制，定时审验客户端缓存中的数据信息 | 采用简单的一致性协议，通过原子化操作对文件进行修改 | | 文件服务容错 | | 采用NFS无状态协议，异常处理和容错功能完全由客户端实现 | 在Datanode、文件、数据块等多个层次上提供冗余，实现容错 | | 文件服务兼容 | | 支持具有多种操作系统和硬件平台的客户端和服务器 | 支持运行Linux操作系统的标准PC服务器作为客户端和服务器 | | 文件服务安全 | | 采用安全的远程过程调用机制保证用户权限识别及数据安全传输 | 采用基于安全套接字的网络安全机制实现文件的安全访问 | |

HDFS很好地满足了系统设计的透明性需求，在系统的扩展性、可靠性等方面具有优势，而这也正是云计算对文件服务的主要需求，因此，HDFS以及与之具有相似设计思想的GFS、KFS（KOSMOSdistributedfilesystem）等分布式文件系统在当前发挥了巨大的作用并拥有广阔的前景。以这类新兴的分布式文件系统为基础，分布式计算模型、分布式数据库等云计算关键技术也有了新的发展，例如，MapReduce计算架构能够部署在HDFS系统上实现将计算步骤推向数据节点的计算模式，HBase分布式数据库可以借助HDFS的管理能力而无需传统的数据库管理系统。

但是，从表中也可以发现HDFS当前存在的一些问题，如当文件服务的负载过大时，NFS能够以增加新的服务器的方式缓解文件服务的压力，而HDFS虽然可以通过增加Datanode对存储容量进行动态扩容，但是系统中惟一的用于管控系统统一名字空间的Namenode会成为性能和可用性的瓶颈。对于这一问题有多种改进思路。例如，可以通过部署Namenode集群的方式供客户端从多台服务器上获得文件的元数据［4］，但是Namenode节点间的高效同步机制会增加系统复杂度；还可以采用在线动态生成文件元数据而无需专用管理节点的方式，但是其性能和效率还有待提高。另外，还需要特别注意的是，HDFS虽然在当前的面向云计算（主要是互联网领域的相关应用）的应用场景中取得了较大的成功，但是它也具有一定的局限性，例如不适合用于进行低延时的文件数据交互等。因此，文件服务的提供者需要根据实际的用户需求，从文件系统的扁平文件服务、目录服务和客户端等方面的技术要点进行综合考察，才能最终选择出真正适用的文件系统。

## 4.3总结

通过对HDFS提供文件服务的体系架构进行建模分析，可以发现，无论是在服务器侧提供的扁平文件服务和目录服务，还是在客户端运行的客户端模块，HDFS在设计和实现时都充分考虑了海量数据在存储和管理等方面的需求，并最终获得了系统的高可扩展性、高可靠性和高性能，满足了云计算领域的相关用户需求。因此，HDFS一直以来都受到了学术界和产业界的广泛关注，拥有广阔的应用前景。但是，通过对比、分析也可以发现，HDFS在一些方面仍然有待完善，可以向传统的分布式文件系统借鉴相关的方法和技术。同时，HDFS也不是万能的，文件服务的提供者必须根据实际需要选择合适的文件系统。

# 五、组内分工情况

|  |  |
| --- | --- |
| 小组成员 | 工作内容 |
| 林秋成 | 项目简介、需求分析功能需求以及质量属性、体系结构总体架、RBF模块分析、NFS模块分析、设计特色部分分析、思维导图部分工作 |
| 李达 | 需求分析中的其他需求、系统设计体系、体系结构策略分析、Httpfs模块具体分析、HDFS设计特色、与NFS比较的优势以及需要完善的方面。 |
| 赖汝锋 | hdfs存储策略分析、读写策略分析、  hdfs client模块分析、hdfs思维导图 |

# 参考文献

[1] Vivek HJ. HDFS Architecture [EB/OL]. <http://www.corejavaguru.com/bigdata/hadoop/hdfs-architecture>

[2] The Apache Software Foundation. HDFS Architecture [EB/OL]. <https://hadoop.apache.org/docs/r3.1.2/>

[3] java06051515. Router-Based HDFS Federation 在滴滴大数据的应用[EB/OL]. <http://blog.itpub.net/31559758/viewspace-2374710/>

[4] Feng Wang,Baohua Lei. Modeling and Analysis of Hadoop Distributed File System[J]. Telecommunications Science, 2010, 26(12): 95-99.

[5] 行者无疆\_路过. HDFS笔记 [EB/OL]. <https://blog.csdn.net/woshiwanxin102213/article/details/19990487>

[6] 我不是李寻欢. HDFS的特点 [EB/OL]. <https://blog.csdn.net/qq_39532946/article/details/76036706>

[7] liuhong1123. HDFS块分配策略[EB/OL]. <https://blog.csdn.net/liuhong1123/article/details/12949487>

[8] 顺顺顺子. HDFS分配策略学笔记[EB/OL]. <https://blog.csdn.net/xiaoshunzi111/article/details/48153063>

[9] Hadoop Documentation.

<http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>

[10] Robert Chansler, Hairong Kuang, Sanjay Radia, Konstantin Shvachko, and Suresh Srinivas. [The Hadoop Distributed File System](http://www.aosabook.org/en/hdfs.html)

<http://www.aosabook.org/en/hdfs.html>

[11] Tom White. Hadoop: The Definitive Guide. O’Reilly Media(2012-05), pp 94-96

# 附件：体系结构的分解过程

