

Hadoop分布式文件系统

提纲

Hadoop原理

HDFS

- HDFS文件系统原理
- HDFS读写过程

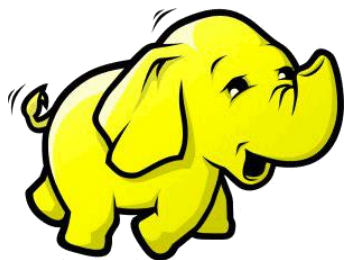
Hadoop浏览器界面

Hadoop常用命令

Hadoop文件系统

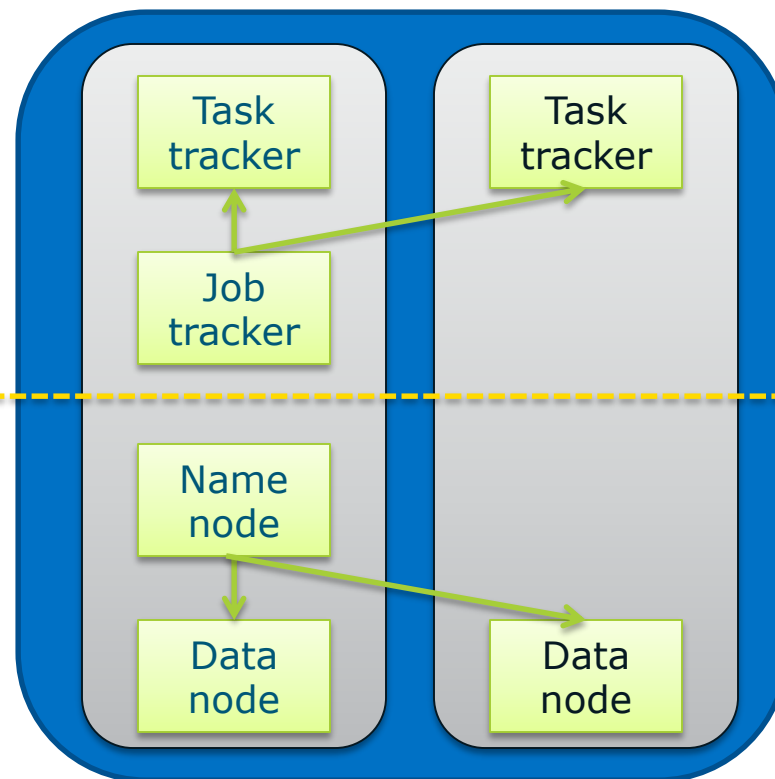
- 压缩

Hadoop 架构



Map Reduce
Layer

HDFS
Layer



Reference:
http://en.wikipedia.org/wiki/File:Hadoop_1.png

HDFS与Google文件系统GFS

GFS的设计目的：为了存储Google内部大量的数据，主要是全球互联网的数据，需要极大的容量，为搜索引擎提供后备的存储支持。

Hadoop文件系统HDFS的设计思想来源于GFS，HDFS的基本结构与GFS一致。

HDFS的基本假定

一个分布式文件系统存储大量的数据

- 建立在大规模的廉价x86集群之上
- 硬件模块会出错，出错可能同时发生

“适量”的大文件

- 文件数量可能在百万级
- 文件很大，数百GB大小很常见

读写特性：写入一次，多次读取。写入过程可能是并发的

读的过程是连续的读取，一次将一个文件全部内容读一遍

- 针对MapReduce优化

整个系统对于吞吐率的要求非常高，但是对于延迟不敏感

- 面向批处理

HDFS的特点

基于本地文件系统之上，用户态

存储海量信息（TB~PB），支持很大单个文件。

通过复制提供高可靠性

- 单个或者多个节点不工作，对系统不会造成任何影响，数据仍然可用。

很高的系统吞吐量。

水平扩展：简单加入更多服务器就能够扩展容量和吞吐量

- 最大的实用集群:4000个节点。

针对MapReduce优化。

- HDFS对顺序读进行了优化
- 尽可能根据数据的本地局部性进行访问与计算。

HDFS的基本设计

数据块：文件被划分为固定大小的数据块进行存储

- 数据块（缺省为64MB）远远大于一般文件系统数据块的大小
 - 减少元数据的量
 - 有利于顺序读写（在磁盘上数据顺序存放）

可靠性：数据通过副本的方式保存在多个数据节点（DataNode）上

- 默认3个副本。
- 副本选择会考虑机架信息以防整个机架同时掉电

系统设计简化：用单个节点(NameNode)来保存文件系统元数据和管理/协调

HDFS的基本设计（2）

数据缓存：DataNode没有数据缓存

- 由于文件的访问是扫描式的，不具有局部性

访问方式

- 读、写、文件改名、删除等
- 文件内容不允许覆盖更新overwrite
- 提供一个特殊的访问接口：追加append

HDFS系统结构中的主要模块

NameNode :

- 单台服务器，系统中的单点
- NameNode管理所有文件系统的元数据以及协调管理客户端对于数据的访问
- 管理集群节点和各种操作（如负载均衡）

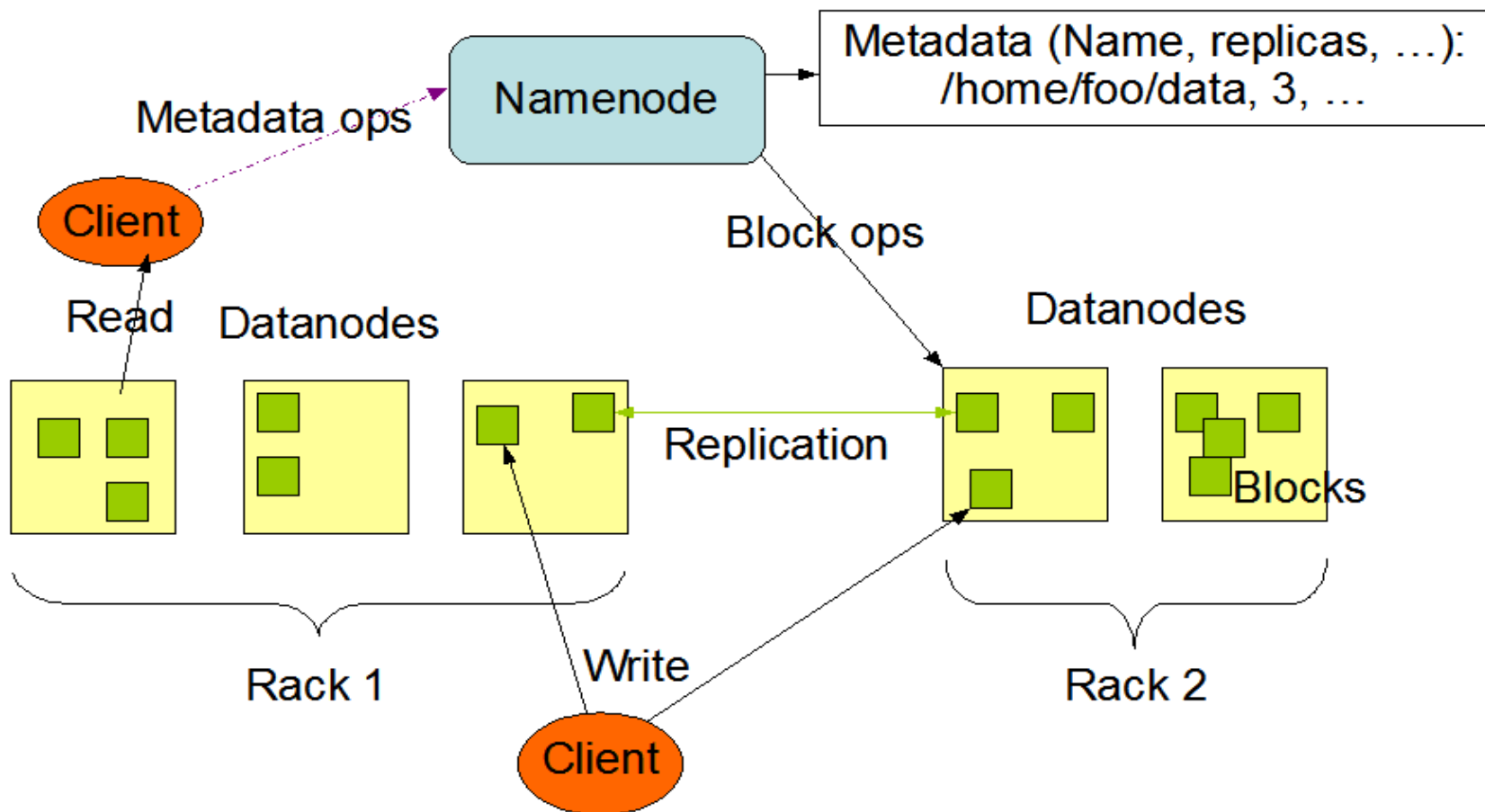
DataNode :

- 运行在集群中的绝大多数节点之上，每个节点一个DataNode进程
- 管理文件系统中的数据存储
- 从NameNode中接收命令，并对数据进行组织管理上的操作
- 响应文件系统客户端的读写访问命令，提供数据服务

数据通信的协议使用TCP/IP协议，使用RPC进行远程信息访问请求

HDFS体系结构

HDFS Architecture

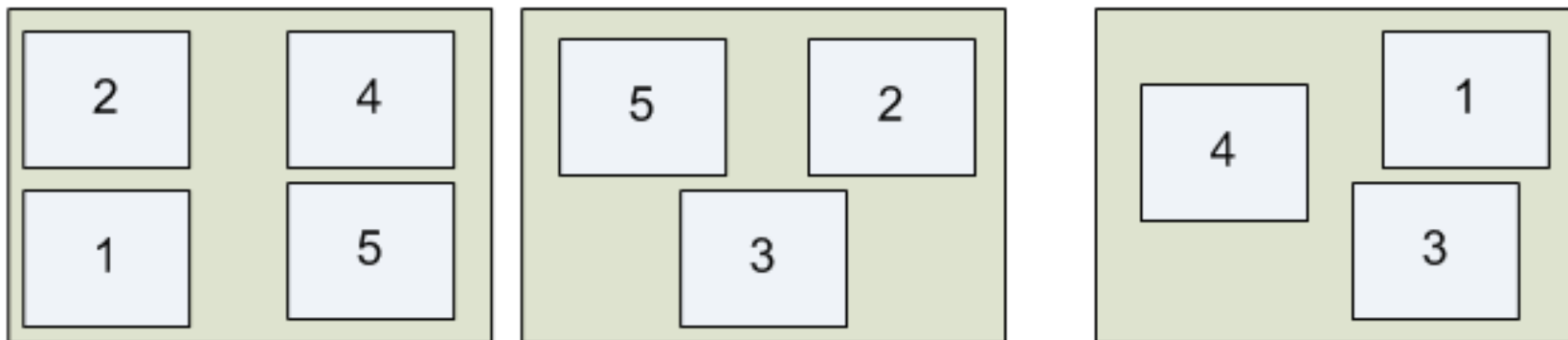


HDFS数据分布设计

NameNode:
Stores metadata only

METADATA:
/user/aaron/foo → 1, 2, 4
/user/aaron/bar → 3, 5

DataNodes: Store blocks from files



NameNode

管理HDFS的元数据：所有的元数据都保存在内存中以支持快速访问

- 文件和目录信息（路径、拥有者、权限等）
- 所有文件到数据块之间的映射关系
- 每个数据块副本的位置

元数据被持久化到磁盘(文件名fsimage)，NameNode启动时会先读取这个信息

- 持久化数据不包括数据块副本的位置

记录元数据修改：修改内存同时，将修改添加到磁盘log文件(文件名edits)

管理DataNode结点

- 负载均衡
- 垃圾搜集
- 过期副本的删除

NameNode的内存需求

NameNode需要大量内存保存元数据，推荐>16G内存

- 每项元数据需要150~200字节内存
 - 文件元数据：文件名、权限等
 - 每个数据块的信息
- 因此，HDFS喜欢适量数目的大文件

DataNode

每个数据块在DataNode所在节点上存为一个本地文件

- 以blk_xxxxxxx标识，无HDFS文件信息
- 还有一个伴生的checksum文件（每64K一个检验）

DataNode的职责：

- 负责管理它所在结点上存储数据的读写
- 周期性地向Namenode发送心跳信号和文件块状态报告
- 执行数据的写流水线

NameNode启动和Safe Mode

当NameNode启动时，它首先获取最新的元数据

1. 从fsimage中读取快照
2. 应用edits中的操作日志，从而得到最新的元数据状态
3. 用新的元数据覆盖fsimage，并清空edits文件

然后，NameNode会进入Safe Mode

- 等待每个DataNode报告情况，更新数据块和DataNode的映射
- 当绝大多数数据块（缺省99.9%）达到最小复制份数时，退出Safe Mode

退出安全模式的时候才进行副本复制操作，以及允许数据的写入

Secondary NameNode

很大的NameNode edits日志文件

- NameNode只有在启动时才合并fsimage和edits，所以久而久之日志文件可能会变得非常庞大，特别是对大型的集群
- 日志文件太大导致下次NameNode启动会花很长时间

Secondary NameNode

- 不是standby Namenode
- 定期合并fsimage和edits日志，然后将更新后的fsimage回写到NameNode上。从而，控制edits文件大小
 - 工作原理和NameNode的启动一样，因此内存需求和NameNode相同
- 通常Secondary NameNode和NameNode运行在不同的机器上

HDFS可靠性

DataNode定期（缺省3s）发送心跳信息至NameNode

- 超过一定时间NameNode没有收到某个DataNode的心跳，就认为其已经死亡
- NameNode检查该DataNode上的数据块列表，将副本过少的数据块进行重新分布

DataNode定期（缺省每小时）发送block report至NameNode

DataNode定期对其所拥有的数据块进行数据检验

关于NameNode单点的讨论

单个NameNode可能存在的问题

- 性能的瓶颈：一个节点不能够负载所有的数据流量。实际运维不存在
- 可靠性的瓶颈：一个节点出现失效，整个系统无法运行

解决方法：

1. 高可靠、高性能的服务器

- 使用多个元数据目录，同时存放多份fsimage和edits
- 磁盘用RAID10或者RAID 5

2. Standby NameNode

- 标准服务器HA方案：DRBD+Pacemaker

Hadoop 2.0对NameNode的改进

Federation

- 引入多个相互无关的NameNode，共享DataNode

内置HA方案

- 使用Zookeeper进行选举Active NameNode
- 多个NameNode都可见的存储
 - 共享存储，例如NFS
 - Bookkeeper
 - Quorum Journal Manager

HDFS的副本放置策略

数据副本默认是3份复制

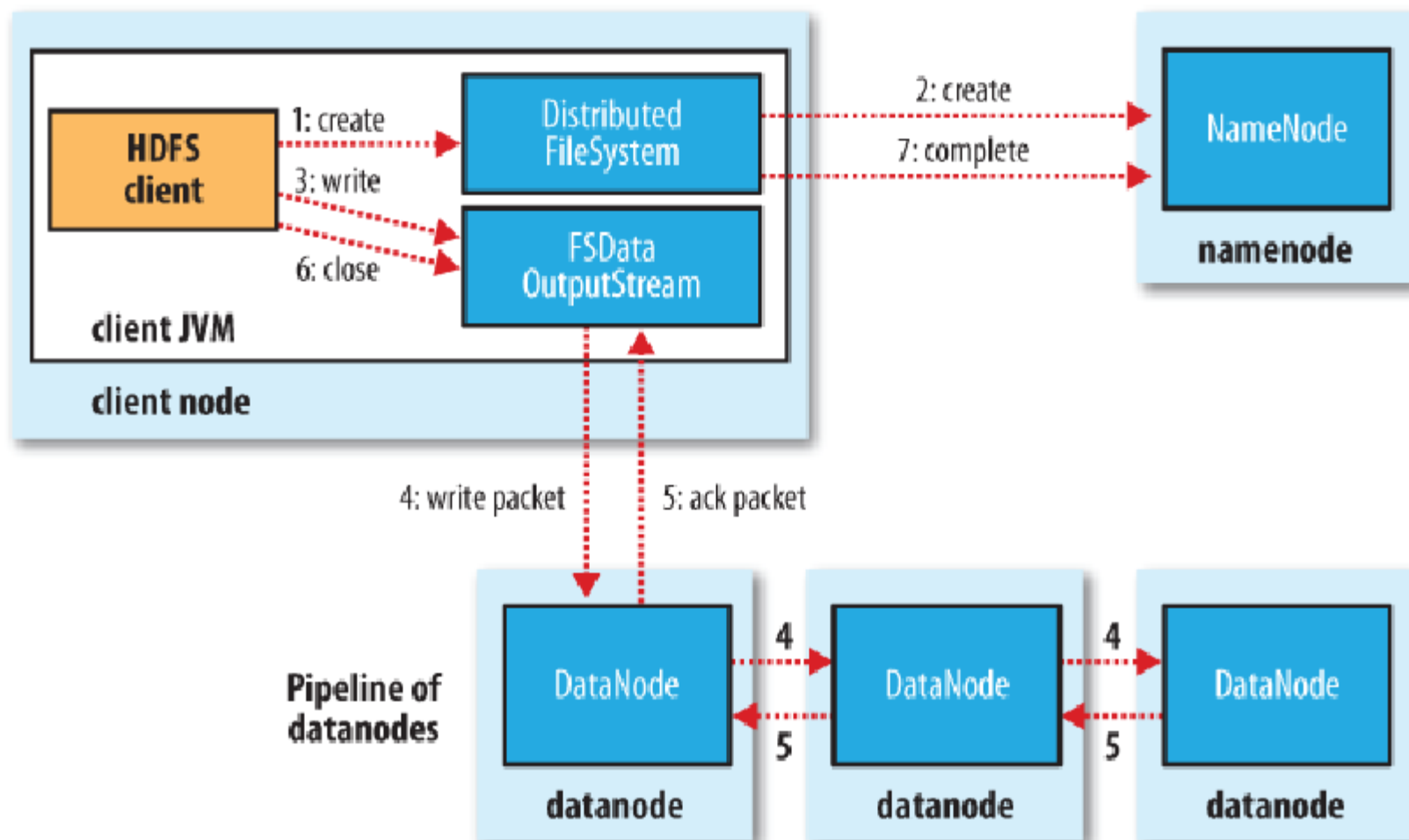
对于副本的选择来说，需要兼顾高可靠性以及高性能

- 数据副本的基本原则是不能把副本放置在一个节点上
- 不能把所有副本放在同一个机架上，以防止机架停电
- 为了提高性能，副本应该靠近客户端

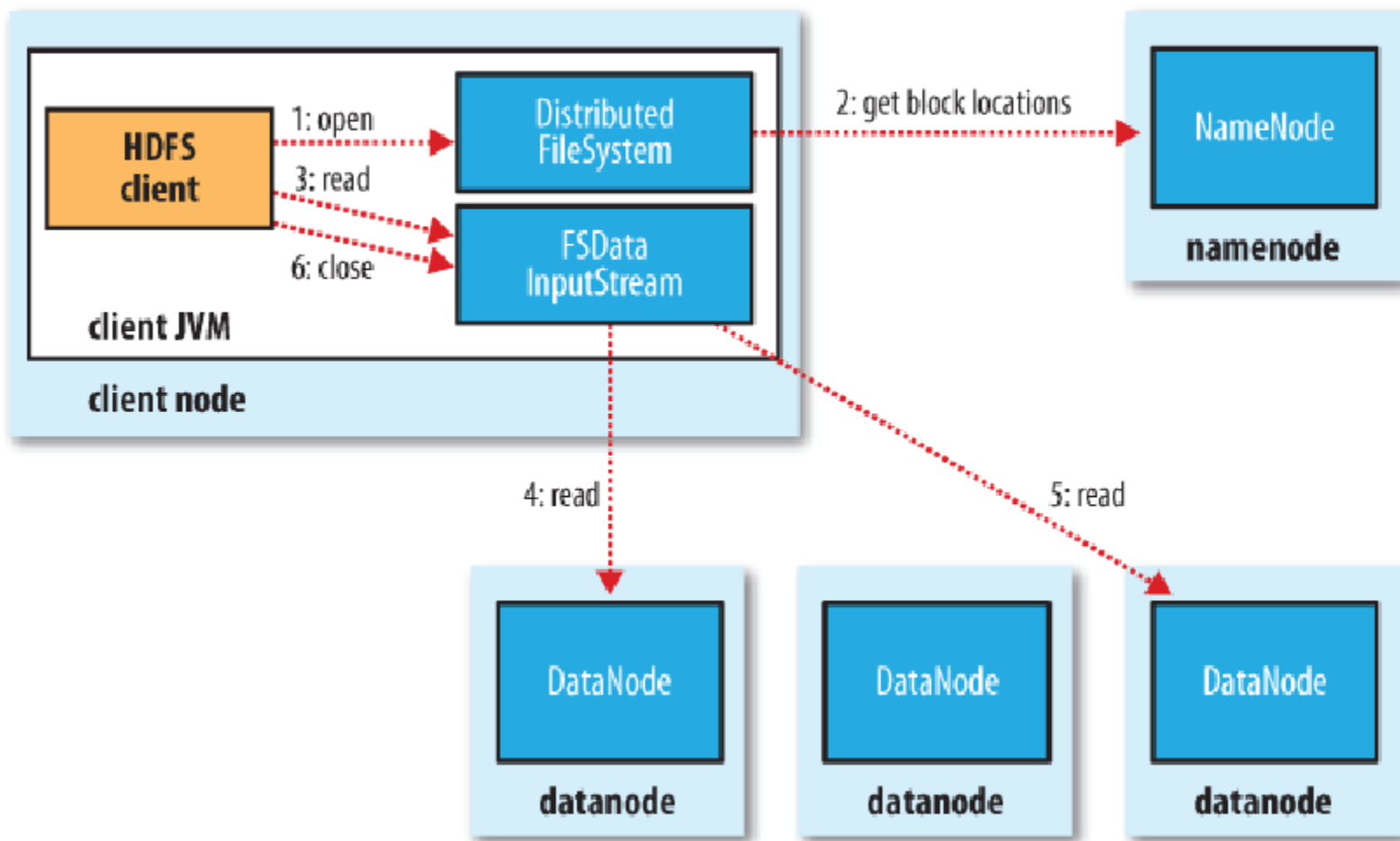
HDFS的副本放置策略

- 第一份副本会优先放在本地DataNode（如果客户端在DataNode上的话）。
- 第二份副本会优先放在本地机架中的DataNode。
- 第三份副本会放在其他机架中的DataNode。

数据写入



数据读取



HDFS数据块读取策略

对于一个数据块来说，因为每个副本都是完整的，所以读取任何一份都可以

HDFS使用静态策略来决定读哪个副本

- 根据客户端与副本的距离远近（本机、同机架、跨机架）
- 当集群异构时容易出现瓶颈

HDFS的权限管理

类似于标准的UNIX文件权限管理

- `rwXrwXrwX`
- 判断十分简单，就是拿发出指令的用户名对文件进行权限检查

HDFS自己没有用户数据库

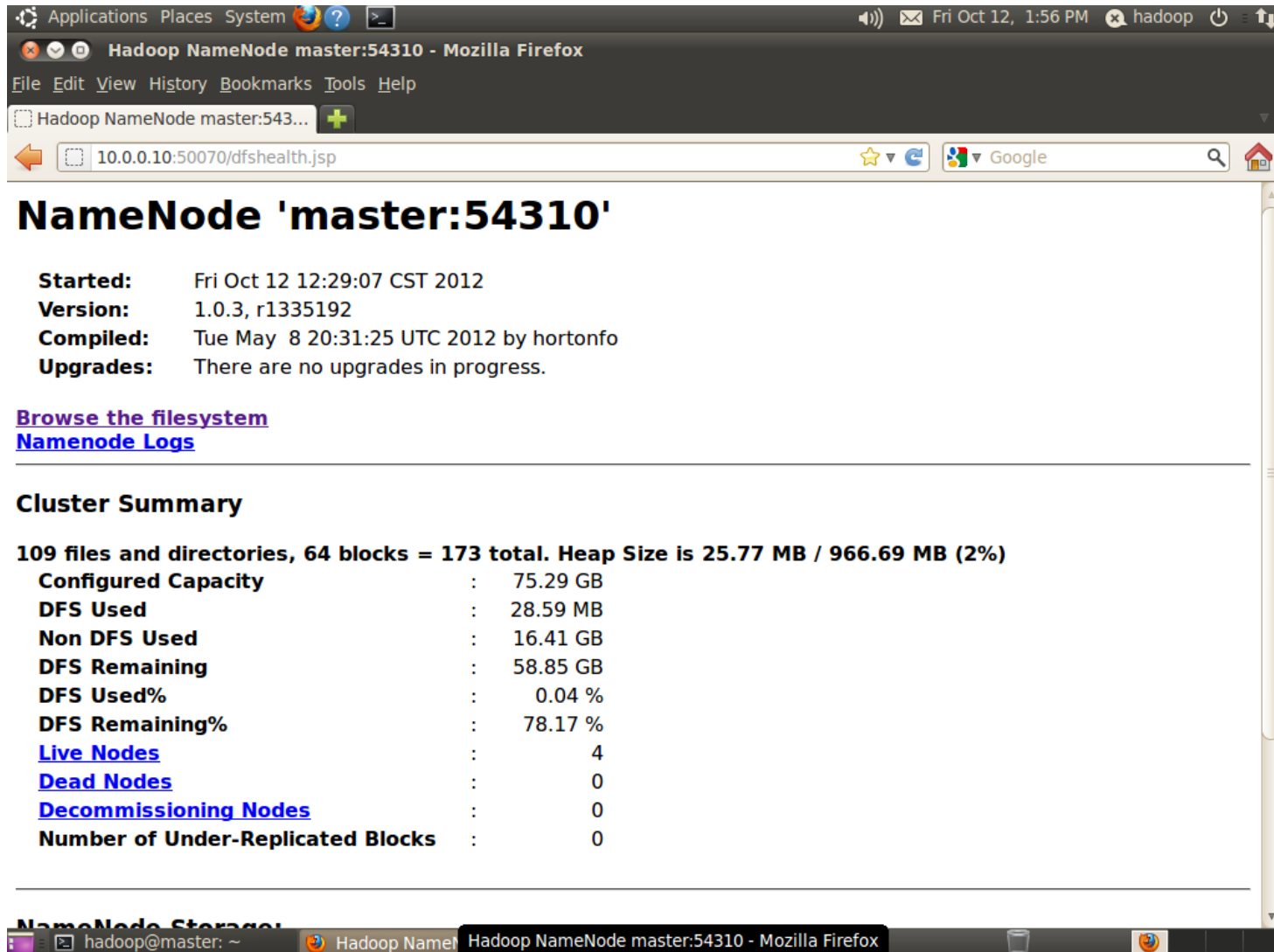
只有在使用Kerberos验证的情况下，才能提供足够的安全性

关于HDFS文件系统的总结

HDFS实际上是演示了如何在市场上可见的硬件水平上构建一个大规模的处理系统

- 从设计上就内建错误容忍机制
- 对大文件的优化，特别是数据追加以及读取
- 不局限于现有的文件系统接口，为了应用进行接口的扩展
- 尽量使用简化的设计，如单个的主服务器NameNode，简化系统的结构，便于理解与维护

浏览器界面 (http://NameNode:50070)



The screenshot shows a Mozilla Firefox browser window with the title "Hadoop NameNode master:54310 - Mozilla Firefox". The address bar shows the URL "10.0.0.10:50070/dfshealth.jsp". The page content includes the following information:

NameNode 'master:54310'

Started: Fri Oct 12 12:29:07 CST 2012
Version: 1.0.3, r1335192
Compiled: Tue May 8 20:31:25 UTC 2012 by hortonfo
Upgrades: There are no upgrades in progress.

[Browse the filesystem](#)
[Namenode Logs](#)

Cluster Summary

109 files and directories, 64 blocks = 173 total. Heap Size is 25.77 MB / 966.69 MB (2%)

Configured Capacity	:	75.29 GB
DFS Used	:	28.59 MB
Non DFS Used	:	16.41 GB
DFS Remaining	:	58.85 GB
DFS Used%	:	0.04 %
DFS Remaining%	:	78.17 %
Live Nodes	:	4
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

The bottom of the browser window shows the taskbar with the terminal window "hadoop@master: ~" and the browser window "Hadoop NameNode master:54310 - Mozilla Firefox".

浏览文件系统的内容

Firefox Web Browser
Browse the World Wide Web

File Edit View History Bookmarks Tools Help

HDFS:/

slave1:50075/browseDirectory.jsp?namenodeInfoPort=50070&dir=/
Google

Contents of directory /

Goto : go

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
hbase	dir				2012-08-20 08:52	rwxr-xr-x	hadoop	supergroup
home	dir				2012-08-20 08:25	rwxr-xr-x	hadoop	supergroup
tmp	dir				2012-08-20 03:02	rwxrwxr-x	hadoop	supergroup
user	dir				2012-08-20 02:56	rwxr-xr-x	hadoop	supergroup

[Go back to DFS home](#)

Local logs

[Log](#) directory

This is [Apache Hadoop](#) release 1.0.3

hadoop@master: ~ HDFS:/ - Mozilla Firefox

查看自己目录中的内容



Contents of directory [/user/hadoop](#)

Goto :

[Go to parent directory](#)

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
hello.txt	file	0.01 KB	1	64 MB	2012-10-12 12:30	rw-r--r--	hadoop	supergroup
input	dir				2012-08-20 01:11	rw-r--r--	hadoop	supergroup
output	dir				2012-08-20 01:14	rw-r--r--	hadoop	supergroup
output2	dir				2012-08-20 01:24	rw-r--r--	hadoop	supergroup
output3	dir				2012-08-20 01:31	rw-r--r--	hadoop	supergroup
warehouse	dir				2012-08-20 02:46	rw-r--r--	hadoop	supergroup

[Go back to DFS home](#)

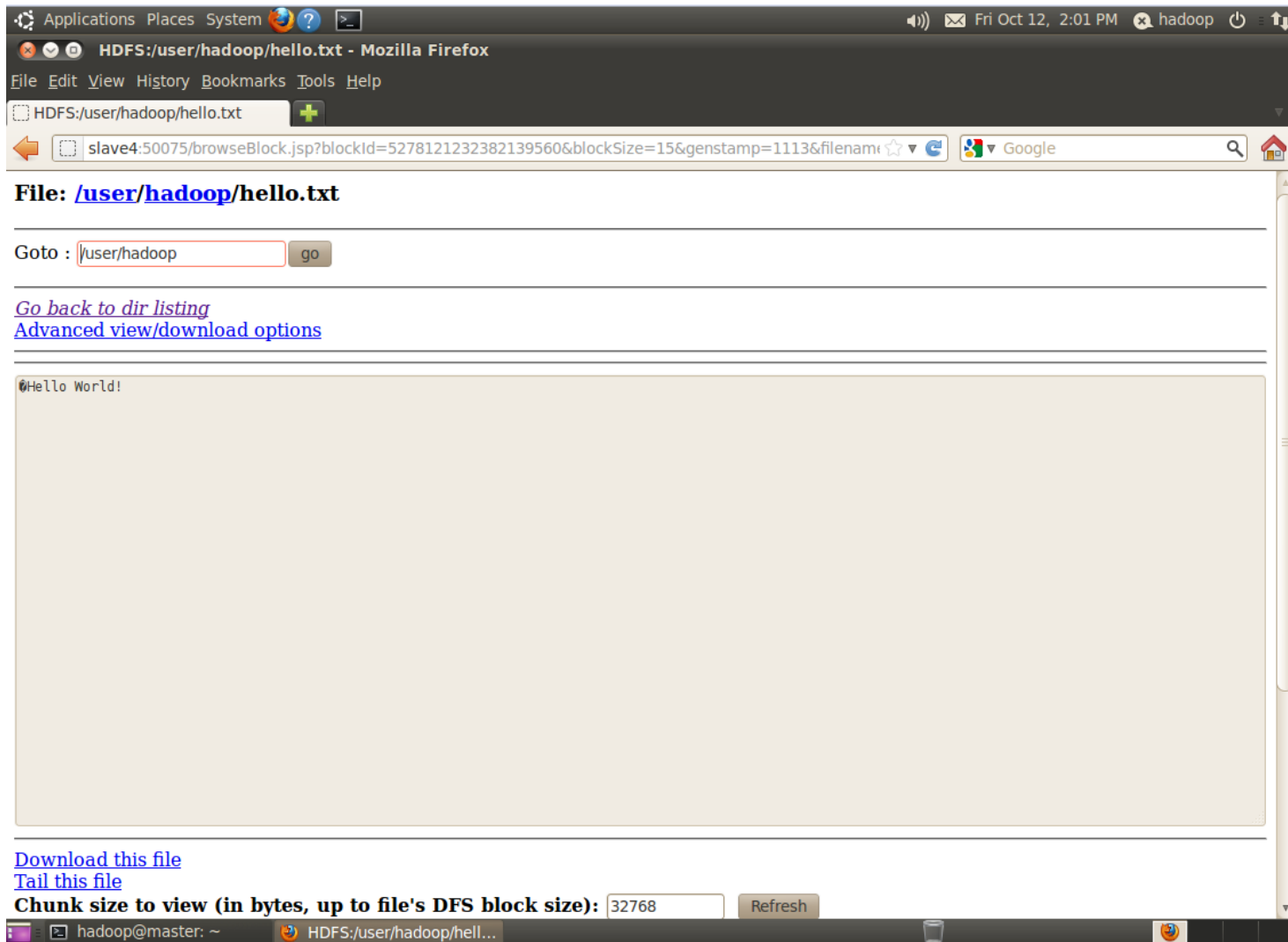
Local logs

[Log](#) directory

This is [Apache Hadoop](#) release 1.0.3



查看文件内容



HDFS简单交互交互

```
someone@anynode:hadoop$ hadoop fs -ls / 列根目录中的内容  
Found 2 items  
drwxr-xr-x - hadoop supergroup 0 2008-09-20 19:40 /hadoop  
drwxr-xr-x - hadoop supergroup 0 2008-09-20 20:08 /tmp
```

```
hadoop fs -mkdir /user 建目录
```

```
hadoop fs -mkdir /user/someone 建用户目录，依据HDFS用户以及权限模型需  
要在/user下面为每一个用户建立目录
```

```
hadoop fs -put /tmp/interestingFile.txt /user/yourUserName/ 从本地复制  
数据到hdfs文件中，-put等同于-copyFromLocal
```

```
hadoop fs -put directory destination Put上传整个目录
```

HDFS交互命令行

-ls path	列目录信息，包括文件名，权限，拥有者，大小以及修改时间
-lsr path	与-ls类似，同时还要列出子目录中的内容
-du path	现实某一个目录的磁盘使用情况（每个文件，单位为字节数）
-dus path	与-du类似，但是打印一个概要信息
-mv src dest	在HDFS内部移动目录或者文件
-cp src dest	在HDFS内部复制文件或者目录

HDFS交互命令行

-rm path	删除文件或者空的目录
-rmr path	删除文件或者目录，删除目录时递归删除所有子目录以及文件
-put localSrc dest	拷贝本地文件或者目录到HDFS中
-copyFromLocal localSrc dest	等同于-copy
-moveFromLocal localSrc dest	拷贝数据到HDFS，并且在成功后删除本地数据

命令行

-get [-crc] src localDest	从HDFS中拷贝文件或者目录到本地
-getmerge src localDest [addnl]	将HDFS目录中所有的文件取出，所有内容合并写入到一个本地文件中，用以将计算结果进行合并
-cat filename	打印输出文件的内容
-copyToLocal [-crc] src localDest	相当于-get
-moveToLocal [-crc] src localDest	拷贝到本地成功后删除HDFS中的内容
-mkdir path	类似mkdir -p进行目录创建，如果父目录没有则主动创建

命令行

-setrep [-R] [-w] rep path	设置文件的目标副本数目
-touchz path	建立一个大小为0文件，并具有当前的时间戳
-test -[ezd] path	测试path是否存在(e)，是否大小为0(z)，是否是目录(d)；测试成功返回1，失败返回0
-stat [format] path	依据format指定的格式打印目录信息
-tail [-f] file	打印文件最后1KB的数据

命令行

-chmod [-R] mode,mode,... path...	设置目录属性，-R用于递归设置，设置方法与Linux下一致，使用3位八进制，或者{augo}+/-{rwxX}
-chown [-R] [owner][:[group]] path...	设置目录的拥有者属性
-chgrp [-R] group path...	设置目录的拥有组属性
-help cmd	打印命令的帮助信息

负载均衡

balancer用来做负载均衡，默认的均衡参数是10%（节点间的负载差异）范围内

`hadoop balancer -threshold 5`

balancer可以在线工作，配置参数

`dfs.balance.bandwidthPerSec`可以用来控制负载均衡所能够使用的带宽 (bytes/sec for each node)

分布式拷贝

分布式拷贝支持集群内或集群间的拷贝，用MapReduce来拷贝的。

```
hadoop distcp \  
hdfs://SomeNameNode:9000/foo/bar/ \  
hdfs://OtherNameNode:2000/baz/quux/
```

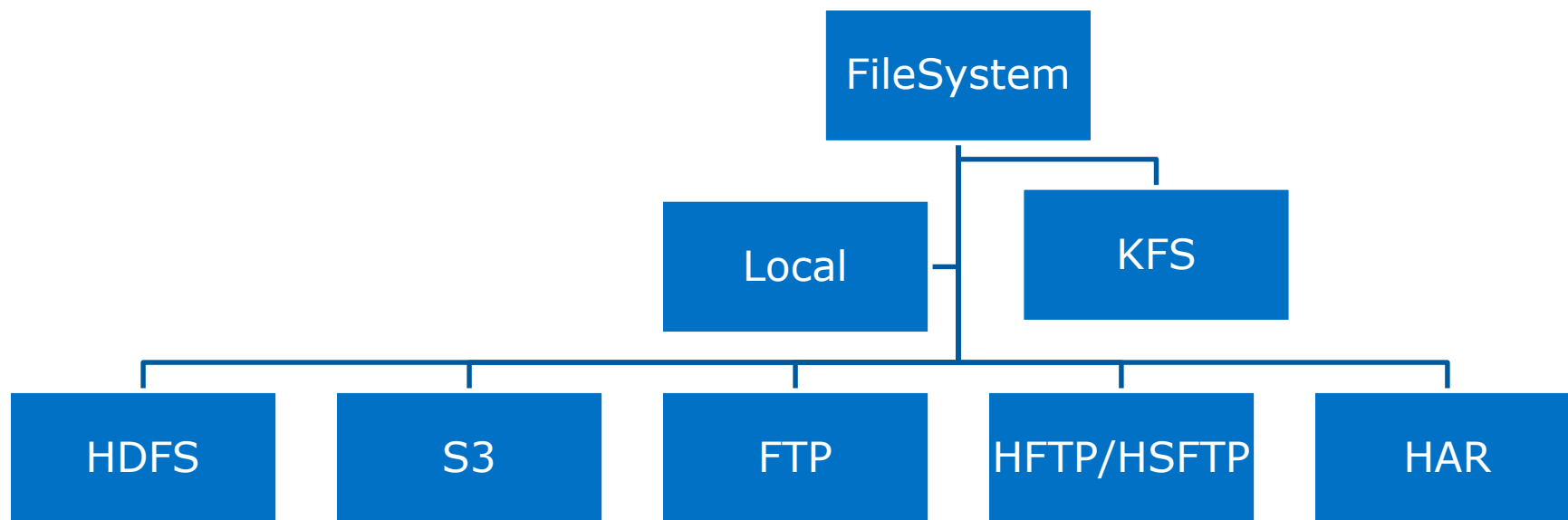
文件系统检查工具FSCK

`hadoop fsck [path] [options]`

- `-move` (to `/lost+found`) or `-delete` //把损坏的文件移走或删除
- `-files`: 列出被检查的文件
- `-blocks`: 打印block报告
- `-openforwrite`: 被打开的待写入文件
- `-locations`: 每个分片的位置

`sudo -u hdfs hadoop fsck <path> [options]`

Hadoop文件系统接口



Create specific FileSystem instance by configuration file

FileSystem Usage

```
Configuration conf = new Configuration();  
  
try{  
  
    FileSystem fs = FileSystem.get(conf);  
  
    FSDataOutputStream out = fs.create(new Path(file));  
  
    out.writeUTF(message);  
  
    out.close();  
  
}finally{  
  
    fs.close();  
  
}
```


压缩

Hadoop支持对文件的压缩

- 减少储存的内容
- 减少网络传输

压缩以Codec形式表达

可以利用Codec接口实现了对文件和MapReduce过程的加密

压缩格式

Compression Format	Tool	Algorithm	Filename Extension	Multiple Files	Splittable
DEFLATE	N/A	DEFLATE	.deflate	NO	NO
gzip	gzip	DEFLATE	.gz	NO	NO
ZIP	zip	DEFLATE	.zip	YES	YES, at file boundaries
bzip2	bzip2	bzip2	.bz2	NO	YES
LZO	lzop	LZO	.lzo	NO	NO
Snappy	N/A	Snappy	.snappy	NO	YES

压缩代码示例

```
String codecClassname = args[0];  
Class<?> codecClass = Class.forName(codecClassname);  
Configuration conf = new Configuration();  
CompressionCodec codec = (CompressionCodec)  
    ReflectionUtils.newInstance(codecClass, conf);  
CompressionOutputStream out =  
codec.createOutputStream(System.out);  
IOUtils.copyBytes(System.in, out, 4096, false);  
out.finish();
```

SequenceFile

用于保存大量的键值对

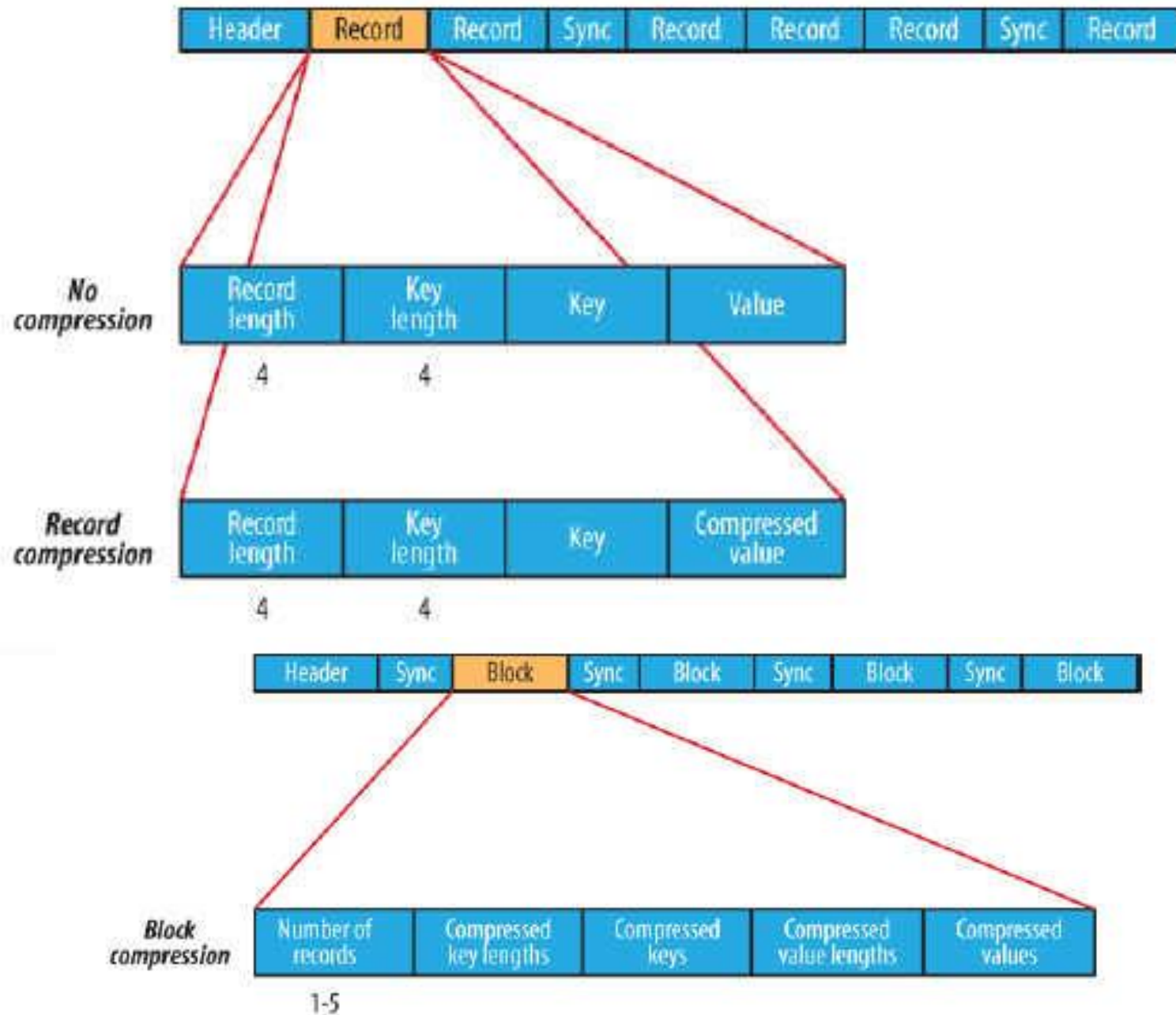
- 不是Map
- 不提供按键方式访问

一些应用场景：

1. Log文件：

- 键：时间戳
- 值：log内容

2. 用于保存很多的小文件



SequenceFile代码示例

```
FileSystem fs=FileSystem.get(conf);
```

```
SequenceFile.Writer
```

```
writer=SequenceFile.createWriter(fs, conf, new  
Path(file), LongWritable.class, UserWritable.class);
```

```
writer.append(key, value);
```

```
IOUtils.closeStream(writer);
```

```
fs.close();
```

MapFile

MapFile = 排序的SequenceFile + 内存Index

支持快速的查找

键类型必须继承自**WritableComparable**

值类型必须继承自**Writable**

小心内存占用和排序开销，仅支持小数据量

