HDFS 常用操作

通过web了解Hadoop的活动

- 通过用浏览器和http访问jobtracker所在节点的50030端口监控jobtracker
- 通过用浏览器和http访问namenode所在节点的50070端口监控集群

http://192.168.0.2:50030/jobtracker.jsp

Quick Links

backup01 Hadoop Map/Reduce Administration

State: RUNNING
Started: Tue Jul 30 17:46:51 HKT 2013
Version: 1.1.2, r1440782
Compiled: Thu Jan 31 02:03:24 UTC 2013 by hortonfo
Identifier: 201307301746
SafeMode: OFF

Cluster Summary (Heap Size is 45.44 MB/888.94 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	1	2	0	0	0	0	4	4	4.00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)

Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

http://192.168.0.2:50070/dfshealth.jsp

NameNode 'backup01:9000'

Started: Tue Jul 30 17:46:47 HKT 2013
Version: 1.1.2, r1440782
Compiled: Thu Jan 31 02:03:24 UTC 2013 by hortonfo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

23 files and directories, 10 blocks = 33 total. Heap Size is 60.94 MB / 888.94 MB (6%)

Configured Capacity	:	1.05 TB
DFS Used	:	146.35 KB
Non DFS Used	:	62.85 GB
DFS Remaining	:	1013.16 GB
DFS Used%	:	0 %
DFS Remaining%	:	94.16 %
Live Nodes	:	2
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

NameNode Storage:

Storage Directory	Type	State
/home/huang/hadoop-1.1.2/tmp/dfs/name	IMAGE_AND_EDITS	Active

观看日志

Directory: /logs/

hadoop-huang-jobtracker-backup01.log	6012 bytes	Jul	31,	2013	5:50:09 PM
hadoop-huang-jobtracker-backup01.log.2013-07-22	5921057 bytes	Jul	22,	2013	10:39:02 PM
hadoop-huang-jobtracker-backup01.log.2013-07-23	176 bytes	Jul	23,	2013	10:38:26 PM
hadoop-huang-jobtracker-backup01.log.2013-07-24	176 bytes	Jul	24,	2013	10:38:29 PM
hadoop-huang-jobtracker-backup01.log.2013-07-25	176 bytes	Jul	25,	2013	10:38:31 PM
hadoop-huang-jobtracker-backup01.log.2013-07-26	176 bytes	Jul	26,	2013	10:38:33 PM
hadoop-huang-jobtracker-backup01.log.2013-07-27	176 bytes	Jul	27,	2013	10:38:35 PM
hadoop-huang-jobtracker-backup01.log.2013-07-28	176 bytes	Jul	28,	2013	10:38:37 PM
hadoop-huang-jobtracker-backup01.log.2013-07-29	176 bytes	Jul	29,	2013	10:38:39 PM
hadoop-huang-jobtracker-backup01.log.2013-07-30	32213 bytes	Jul	30,	2013	5:47:23 PM
hadoop-huang-jobtracker-backup01.out	0 bytes	Jul	30,	2013	5:46:50 PM
hadoop-huang-jobtracker-backup01.out.1	0 bytes	Jul	30,	2013	5:44:05 PM
hadoop-huang-jobtracker-backup01.out.2	0 bytes	Jul	30,	2013	5:42:11 PM
hadoop-huang-jobtracker-backup01.out.3	0 bytes	Jul	22,	2013	10:38:22 PM
hadoop-huang-jobtracker-backup01.out.4	0 bytes	Jul	22,	2013	9:01:24 PM
hadoop-huang-jobtracker-backup01.out.5	0 bytes	Jul	22,	2013	8:35:13 PM
hadoop-huang-namenode-backup01.log	47796 bytes	Jul	31,	2013	5:55:27 PM
hadoop-huang-namenode-backup01.log.2013-07-22	2026115 bytes	Jul	22,	2013	11:43:27 PM
hadoop-huang-namenode-backup01.log.2013-07-23	43614 bytes	Jul	23,	2013	11:43:46 PM
hadoop-huang-namenode-backup01.log.2013-07-24	43615 bytes	Jul	24,	2013	11:44:07 PM
hadoop-huang-namenode-backup01.log.2013-07-25	43614 bytes	Jul	25,	2013	11:44:28 PM
hadoop-huang-namenode-backup01.log.2013-07-26	43613 bytes	Jul	26,	2013	11:44:49 PM
hadoop-huang-namenode-backup01.log.2013-07-27	43620 bytes	Jul	27,	2013	11:45:10 PM
hadoop-huang-namenode-backup01.log.2013-07-28	43614 bytes	Jul	28,	2013	11:45:30 PM
hadoop-huang-namenode-backup01.log.2013-07-29	43612 bytes	Jul	29,	2013	11:45:54 PM
hadoop-huang-namenode-backup01.log.2013-07-30	81047 bytes	Jul	30,	2013	11:55:22 PM
hadoop-huang-namenode-backup01.out	0 bytes	Jul	30,	2013	5:46:45 PM
hadoop-huang-namenode-backup01.out.1	0 bytes	Jul	30,	2013	5:44:00 PM
hadoop-huang-namenode-backup01.out.2	0 bytes	Jul	30,	2013	5:42:06 PM

HDFS 主要用途

- 提供分布式存储机制，提供可线性增长的海量存储能力
- 任何节点操作都可以
- 自动数据冗余，无须使用Raid，无须另行备份
- 为进一步分析计算提供数据基础

HDFS设计基础与目标

- 硬件错误是常态。因此需要冗余
- 流式数据访问。即数据批量读取而非随机读写，Hadoop擅长做的是数据分析而不是事务处理
- 大规模数据集
- 简单一致性模型。为了降低系统复杂度，对文件采用一次性写多次读的逻辑设计，即是文件一经写入，关闭，就再也不能修改
- 程序采用“数据就近”原则分配节点执行

HDFS的关键运作机制

也记录着每个文件中各个数据块所在的数据节点信息。（临时记录，数据节点可能会重建）

子节点才是HDFS真正的存储和检索地点，如果想在主节点做整个集群数据的索引并检索的话，请考虑可行性，毕竟HDFS不擅长做巨型索引。

构中，有两
以管理者-工

主
维护着文件
树内的所有文
树和整棵
和目录。

命名空间镜像文件（永久）

编辑日志文件（永久）

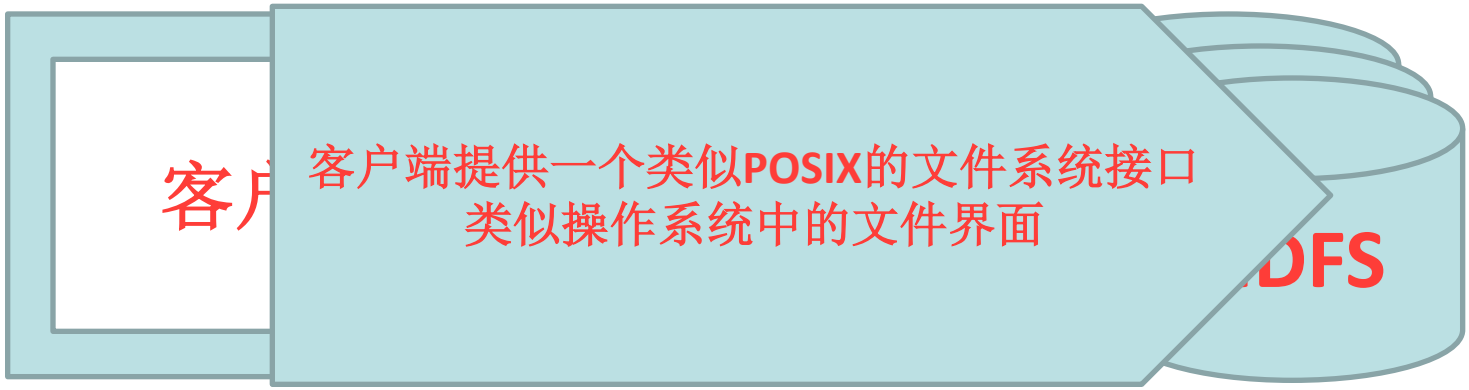
节点
文
系统的工作节点

本地化的文件数据块

自身存储的数据块列表

HDFS的关键运作机制

客户端（client）代表用户通过与namenode和datanode交互访问整个文件系统。可以是具体程序，也可以是应用。



HDFS的关键运作机制

没有datanode，文件系统不会崩溃，文件系统只是无法存储文件，也不会丢失数据。

没有namenode，文件系统会崩溃，文件系统上的所有文件将丢失（无法读出，因为无法定位元数据块的位置，也就无法根据datanode的块来重构文件）。

解决方案一

备份组成文件系统元数据持久状态的文件。操作方法是在写入本地磁盘的同时，写入一个远程挂载的网络文件系统。

解决方案二

在运行集群时，运行一个辅助namenode，但不能用作namenode，辅助主节点是用来定期通过编辑日志合并命名空间镜像，防止编辑日志过大。主节点失效时数据会部分丢失。

HDFS文件操作

- 命令行方式
- API方式

SHELL 命令行

- Shell

命令	功能
hadoop -fs ls	文件夹列表
hadoop -fs copyFromLocal	从本地拷贝至HDFS
hadoop -fs mkdir	建立目录
hadoop -fs rm	删除文件
hadoop -fs rmdir	删除目录
hadoop -fs cp	复制文件
……	……

Usage:hadoop [--config confdir] COMMAND

注意，hadoop没有当前目录的概念，也没有cd命令

关键运行机制及API

- Hadoop API被分成如下几种主要的包：

org.apache.hadoop.conf　定义了系统参数的配置文件处理API。

org.apache.hadoop.fs　定义了抽象的文件系统API。

org.apache.hadoop.Hdfs　HDFS，Hadoop的分布式文件系统实现。

org.apache.hadoop.io　定义了通用的I/O API，用于针对网络，数据库，文件等数据对象做读写操作。

org.apache.hadoop.ipc　用于网络服务端和客户端的工具，封装了网络异步I/O的基础模块。

org.apache.hadoop.mapreduce　Hadoop分布式计算系统（MapReduce）模块的实现，包括任务的分发调度等。

HDFS API

- org.apache.hadoop.metrics 定义了用于性能统计信息的API，主要用于mapred和dfs模块。
- org.apache.hadoop.record 定义了针对记录的I/O API类以及一个记录描述语言翻译器，用于简 化将记录序列化成语言中性的格式（language-neutral manner）。
- org.apache.hadoop.tools 定义了一些命令行的工具。
- org.apache.hadoop.util 定义了一些公用的API。
- org.apache.hadoop.Secruity 用户和用户组信息

HDFS API： org.apache.hadoop.fs

org.apache.hadoop.fs.FileSystem （implements java.io.Closeable）

- org.apache.hadoop.fs.FilterFileSystem
- org.apache.hadoop.fs.ChecksumFileSystem
- org.apache.hadoop.fs.InMemoryFileSystem
- org.apache.hadoop.fs.LocalFileSystem
- org.apache.hadoop.fs.HarFileSystem
- org.apache.hadoop.fs.RawLocalFileSystem
- 抽象文件系统的基本要素和基本操作。最显著的一个特点就是，FileSystem文件系统是基于流式数据访问的，并且，可以基于命令行的方式来对文件系统的文件进行管理与操作。

org.apache.hadoop.HDFS

- **org.apache.hadoop.hdfs.server.protocol.NamenodeProtocol**

Secondary NameNod)与Namenode进行通信所需进行的操作

- **org.apache.hadoop.hdfs.server.protocol.InterDatanodeProtocol**

Datanode之间的通信

- **org.apache.hadoop.hdfs.server.protocol.DatanodeProtocol**

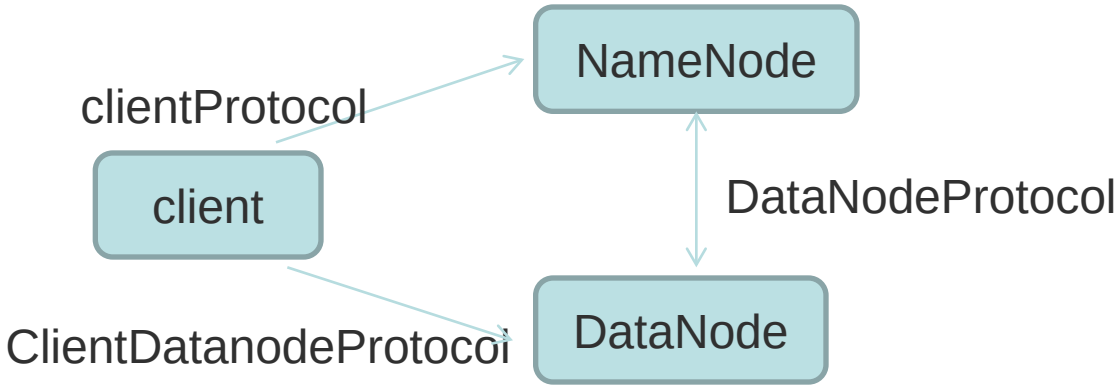
一个HDFS Datanode用户与Namenode进行通信的协议

- **org.apache.hadoop.hdfs.protocol.ClientProtocol**

客户端进程与Namenode进程进行通信

- **org.apache.hadoop.hdfs.protocol.ClientDatanodeProtocol**

客户端进程与datenode进程进行通信



Namenode主要实现了ClientProtocol，DatanodeProtocol，NamenodeProtocol

上传本地文件到HDFS的指定位置上

hdfs.copyFromLocalFile(path srcPath, path dstPath)

- **eg: 1.Configuration config = new Configuration();**

  2.FileSystem hdfs = FileSystem.get(config);

  3.Path srcPath = new Path(srcFile);

  4.Path dstPath = new Path(dstFile);

  5.hdfs.copyFromLocalFile(srcPath, dstPath);

获取到指定文件src的全部块的信息

获取到指定文件src的全部块的信息返回LocatedBlocks，包括文件长度、组成文件的块及其存储位置（所在的Datanode数据结点）

```
public LocatedBlocks  getBlockLocations(String src, long offset, long length)
```

- 对指定文件执行追加写操作，返回信息，可以定位到追加写入最后部分块的信息

```
public LocatedBlock append(String src, String clientName)
```

- 设置副本因子，为一个指定的文件修改块副本因子

```
public boolean setReplication(String src, short replication)
```

- 为已经存在的目录或者文件，设置给定的操作权限

```
public void setPermission(String src, FsPermission permission)
```

- 设置文件或目录属主

```
public void setOwner(String src, String username, String groupname)
```

- 客户端放弃对指定块的操作

```
public void abandonBlock(Block b, String src, String holder)
```

- 客户端向一个当前为写操作打开的文件写入数据块

```
public LocatedBlock addBlock(String src, String clientName)
```

- 客户端完成对指定文件的写操作，并期望能够写完，在写完以后关闭文件

```
public boolean complete(String src, String clientName)
```

- 客户端向Namenode报告corrupted块的信息(块在Datanode上的位置信息)

```
public void reportBadBlocks(LocatedBlock[] blocks)
throws IOException
```

- 在文件系统命令空间中重命名一个文件或目录

```
public boolean rename(String src, String dst)
```

- 删除文件或目录src

```
public boolean delete(String src)
```

- 删除文件或目录src，根据recursive选项来执行

```
public boolean delete(String src, boolean recursive) throws
  IOException;
```

- 创建目录src，并赋予目录src指定的nasked权限

```
public boolean mkdirs(String src, FsPermission masked) throws
  IOException;
```

- 获取指定目录src中的文件列表

```
public FileStatus[] getListing(String src) throws IOException;
```

文件读取

- 在客户端DFSClient中，有一个 DFSClient.DFSInputStream类。当读取一个文件的时候，会生成一个DFSInputStream的实例
- DFSInputStream的实例调用 ClientProtocol定义getBlockLocations接口，取得一个 LocatedBlocks类的对象，这个对象包含一组LocatedBlock，那里面有所规定位置中包含的所有数据块信息，以及数据块对应的所有数据服务器的位置信息
- 读取开始， DFSInputStream的Read方法
- 如读取错误，客户端向Namenode报告corrupted块的信息
- public void reportBadBlocks(LocatedBlock[] blocks)

文件存入

- DFSClient也有一个 DFSClient.DFSOutputStream类，写入开始，会创建此类的实例
- DFSOutputStream会从NameNode上拿一个LocatedBlock
- 写入开始，调用DFSOutputStream的Write方法

创建一个由src路径指定的空文件

- 在制定的文件系统命名空间中创建一个文件入口（entry），在命名空间中创建一个文件入口。该方法将创建一个由src路径指定的空文件

```
public void create(String src, FsPermission masked, String
    clientName, boolean overwrite, short replication, long
    blockSize)
```

- hdfs.create(path path) 创建HDFS文件

```
1.Configuration config = new Configuration();
2.FileSystem hdfs = FileSystem.get(config);
3.Path path = new Path(fileName);
4.FSDataOutputStream outputStream = hdfs.create(path);
5.outputStream.write(buff, 0, buff.length);
```

ClientProtocol（系统管理相关接口）

- 监听客户端，Namenode监听到某个客户端发送的心跳状态

  `public void renewLease(String clientName)`

- 获取文件系统的状态统计数据

  `public long[] getStats()`

- 注：返回的数组：

```
public int GET_STATS_CAPACITY_IDX = 0;
public int GET_STATS_USED_IDX = 1;
public int GET_STATS_REMAINING_IDX = 2;
public int GET_STATS_UNDER_REPLICATED_IDX = 3;
public int GET_STATS_CORRUPT_BLOCKS_IDX = 4;
public int GET_STATS_MISSING_BLOCKS_IDX = 5;
```

ClientProtocol（系统管理相关接口）

- 安全模式开关操作

```
public boolean setSafeMode(FSConstants.SafeModeAction action)
```

- 保存FsImage映像，同时将更新同步到EditLog中，要求具有超级权限，并且在安全模式下进行。

```
public void saveNamespace()
```

- 持久化文件系统元数据，将Namenode结点上的数据结构写入到指定的文件中，如果指定文件已经存在，则追加到该文件中

```
metaSave(String filename)
```

以上还只是ClientProtocol相关的提供给客户端相关的类，还有NameNode相关的类，如文件系统(FSNamesystem、 INode)等。

怎样添加节点？

1. 在新节点安装好hadoop
2. 把namenode的有关配置文件复制到该节点
3. 修改masters和slaves文件，增加该节点
4. 设置ssh免密码进出该节点
5. 单独启动该节点上的datanode和tasktracker（hadoop-daemon.sh start datanode/tasktracker）
6. 运行start-balancer.sh进行数据负载均衡
7. 是否要重启集群？

负载均衡

- 作用：当节点出现故障，或新增加节点时，数据块分布可能不均匀，负载均衡可以重新平衡各个datanode上数据块的分布

```
[huang@Backup02 bin]$
[huang@Backup02 bin]$ ./start-balancer.sh
starting balancer, logging to /home/huang/hadoop-1.1.2/libexec/../logs/hadoop-huang-balancer-Backup02.out
[huang@Backup02 bin]$
```

HDFS的缺点

1. 访问时延

不太适合于那些要求低延时（数十毫秒）访问的应用程序，HDFS的设计主要是为了用于大吞吐量的数据，这是以一定延时为代价的。HDFS单Master设计时，所有对文件的请求都要通过它，当请求多时，必然会有延时。当前，对于那些有低延时要求的应用程序，HBase会是一个更好的选择。同时，可以使用缓存或多Master设计以降低客户端的数据请求压力，从而减少延时。如果要降低时延还可以对HDFS内部程序进行修改，以权衡大吞吐量与低延时的关系。

HDFS的缺点

2. 对大量小文件的处理

因为NameNode把文件系统的元数据放置在内存中，所以文件系统所能容纳的文件数目是由NameNode的内存大小来决定的。

一般来说，每一个文件、文件夹和Block需要占据150字节左右的空间，所以，如果有100万个文件，每一个文件占据一个Block，至少需要300MB内存。

就当前情况来说，数百万的文件还是可行的，当文件数扩展到数十亿时，当前的硬件水平就不容易实现了。

HDFS的缺点

对大量小文件的处理

还有一个问题就是，因为Map任务的数量默认是由splits来决定的，所以用MapReduce处理大量的小文件时，就会产生过多的Map任务，线程管理开销将会增加作业时间。

举个例子，处理10000MB的文件，若每个split为10MB，那就会有1000个Map任务，会有很大的线程开销；若每个split为100MB，则只有100个Map任务，每个Map任务将会有更多的事情要做，而线程的管理开销也将减小很多。

HDFS的缺点

为了处理好小文件，建议使用以下方法：

1）利用SequenceFile、MapFile、Har等方式归档小文件。这个方法的原理就是把小文件进行归档管理，HBase就是基于此的。对于这种方法，如果想找回原来的小文件内容，就必须得知道其与归档文件的映射关系。

2）横向扩展。既然一个Hadoop集群能管理的小文件数量有限，那就把几个Hadoop集群拖在一个虚拟服务器后面，形成一个大的Hadoop集群。Google就这么处理。

3）多Master设计。正在研发中的GFS II要改为分布式多Master设计，还要支持Master的Failover，而且Block大小也要改为1M，要有意调优处理小文件。

HDFS的缺点

3. 多用户写，任意文件修改

目前Hadoop只支持单用户写，不支持并发多用户写。可以使用Append操作在文件的末尾添加数据，但不支持在文件的任意位置进行修改。这些特性的加入将会降低Hadoop运行的效率。

可以利用Chubby、ZooKeeper之类的分布式协调服务来解决一致性问题。