

数据存储

HDFS特点

HDFS特点

- 海量数据存储
- 数据块
- 校验和
- 回收站
- 序列化
- 联邦
- 元数据保护
- 简单一致性模型
- 类Linux文件权限
- 流式数据访问
- 心跳机制
- 安全模式
- DataNode缓存
- 快照
- 容错性
- 小文件处理
- 压缩性能

安全模式

Namenode启动时经过一个“安全模式”阶段

dfs.safemode.threshold.pct

安全模式阶段不会产生数据写

安全模式阶段Namenode收集各个datanode的报告，当数据块达到最小副本数以上时，会被认为是“安全”的，再过若干时间，安全模式结束。达不到比例，hadoop开始写副本直到写到达安全比例。

```
[grid@h3 hadoop-0.20.2]$ bin/hadoop dfsadmin -safemode enter
Safe mode is ON
[grid@h3 hadoop-0.20.2]$
[grid@h3 hadoop-0.20.2]$ bin/hadoop dfsadmin -safemode leave
Safe mode is OFF
[grid@h3 hadoop-0.20.2]$
```

DataNode缓存

- 利用DataNode的可用内存集中地缓存、管理数据, 并暴露给外界应用程序使用
- MapReduce、Hive等应用程序也可以申请内存进行缓存, 然后直接从数据节点的内存中读取内容, 通过完全避免磁盘操作极大提高效率。如Hive正在开发的ORC文件实现

快照

- 存储某个时间点的映像，需要时可以使数据重返这个时间点的状态。（oracle的flashback功能）
- Hadoop1不支持快照，Hadoop2已经解决。

- 删除的文件不能被存储
 - 垃圾错误且不容易被理解
 - 垃圾只基于删除的CLI工作
- 没有时间点恢复
- 没有定期快照从中恢复
 - 没有管理员/用户快照

Hadoop1

- 文件系统的时间点图像
- 只读
- 写时复制
- 防范用户错误
- 备份
- 实验/测试设置

Hadoop2

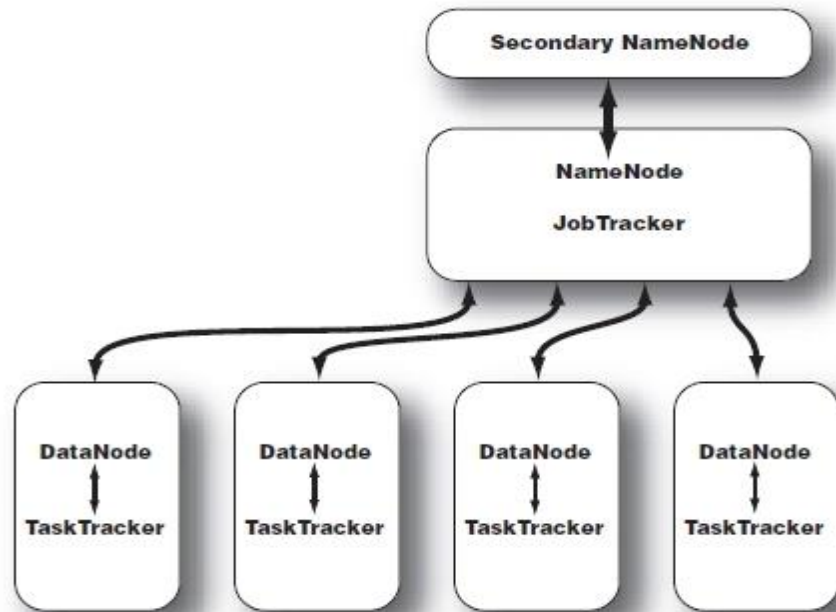
周期性快照备份

拍摄快照于

- 每15分钟一次，并保持24小时
- 每1小时一次，并保持两天
- 每天一次，并保持14天
- 每周一次，并保持3个月
- 每个月一次，并保持一年

Namenode单点

- **瓶颈问题：** Hadoop1所有metadata操作都通过NN，NN是性能的瓶颈；NN在内存中存储所有元数据metadata，单个NN所能存储的文件块数目受到NN JVM的**heap size**限制。假设文件平均大小为40MB，50G的heap能够存储20亿个对象，20亿个对象支持4000个DN即12PB的存储。单个NN内存容量有限，使得Hadoop集群节点数量被限制在2000个左右，能支持的文件系统大小被限制在10-50PB，最多能支持的文件数量大约为1.5亿左右。
- **单点故障源（SPOF）：** 单一Jobtracker的设计严重制约了整个Hadoop可扩展性和可靠性NN和Jobtracker是整个系统中明显的单点故障源（SPOF）



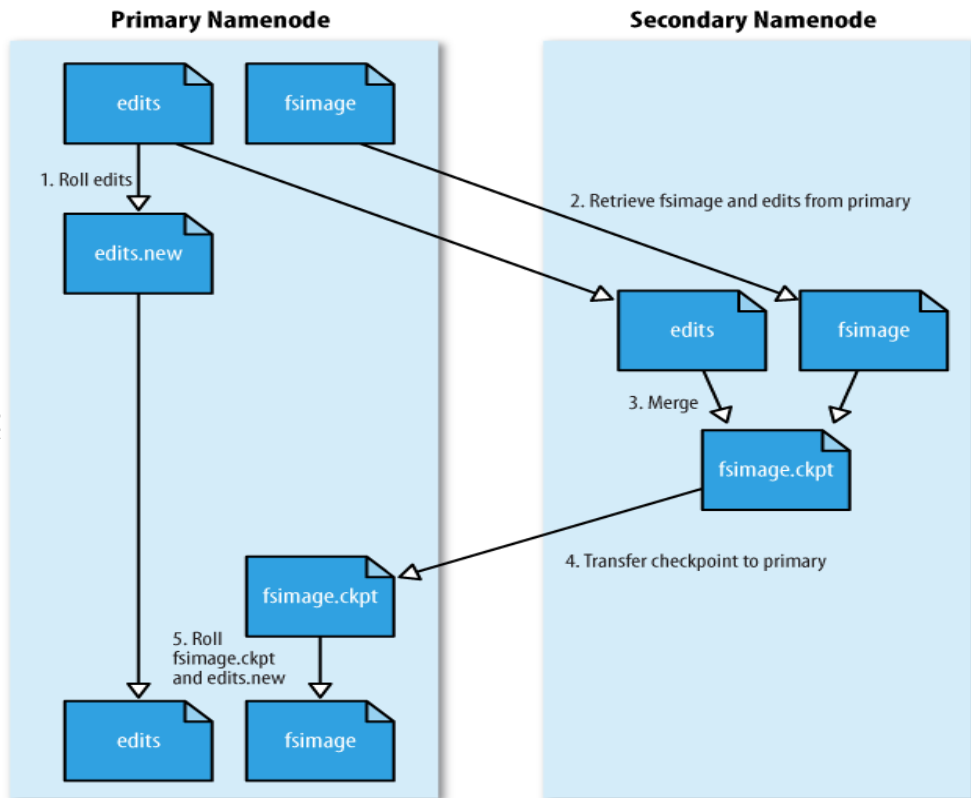
Namenode单点

- **性能开销问题：** Hadoop1 M/R应用程序需要通过DN来访问HDFS，涉及到额外的进程切换和网络传输开销；DN的块汇报对NN性能造成严重影响。如集群有1800个DN，每个DN有3T存储，整个集群大约有1.8P有效存储（ $1800 \times 3T / 3$ ，假设每个数据块有3份），每个DN上有大约50000个左右的block（假设block大小是64M，有的block并没有达到64M大小），假设DN每小时发送一次块汇报，则NN每两秒会收到一次块汇报，每个块汇报包含50000条数据，处理这些数据无疑会占用相当资源。
- **隔离问题：** Hadoop1仅有一个NN，无法隔离各个程序，HDFS上的一个实验程序就很有可能影响整个HDFS上运行的程序。HDFS2引入联邦解决了大部分单个NN HDFS的问题。
- **扩展性问题：** Hadoop1无法按业务区分NN，并对其进行扩展。

Secondary Namenode

- SecondaryNN在检查点定期合并FSImage和FSEditlog，控制FSEditlog日志文件大小上限。避免NN启动时启动时间过长，降低NN压力，同时作为主NN的冷备份。
- 每个集群都有一个SecondaryNN，SecondaryNN通常要运行在单独服务器
- SecondaryNN保存最后一次checkpoint结果，存储结构和主NN一样，主NN可以随时来读取。
- Checkpoint配置参数:
- fs.checkpoint.period，默认1小时，指定连续2次创建Checkpoint的最大时间间隔。
- fs.checkpoint.size，默认64MB，当FSEditlog大小到达该设置值，即使创建Checkpoint的最大时间间隔未到也强制促其执行创建Checkpoint。

FSImage / FSEditlog



Secondary NameNode

配置文件: masters文件

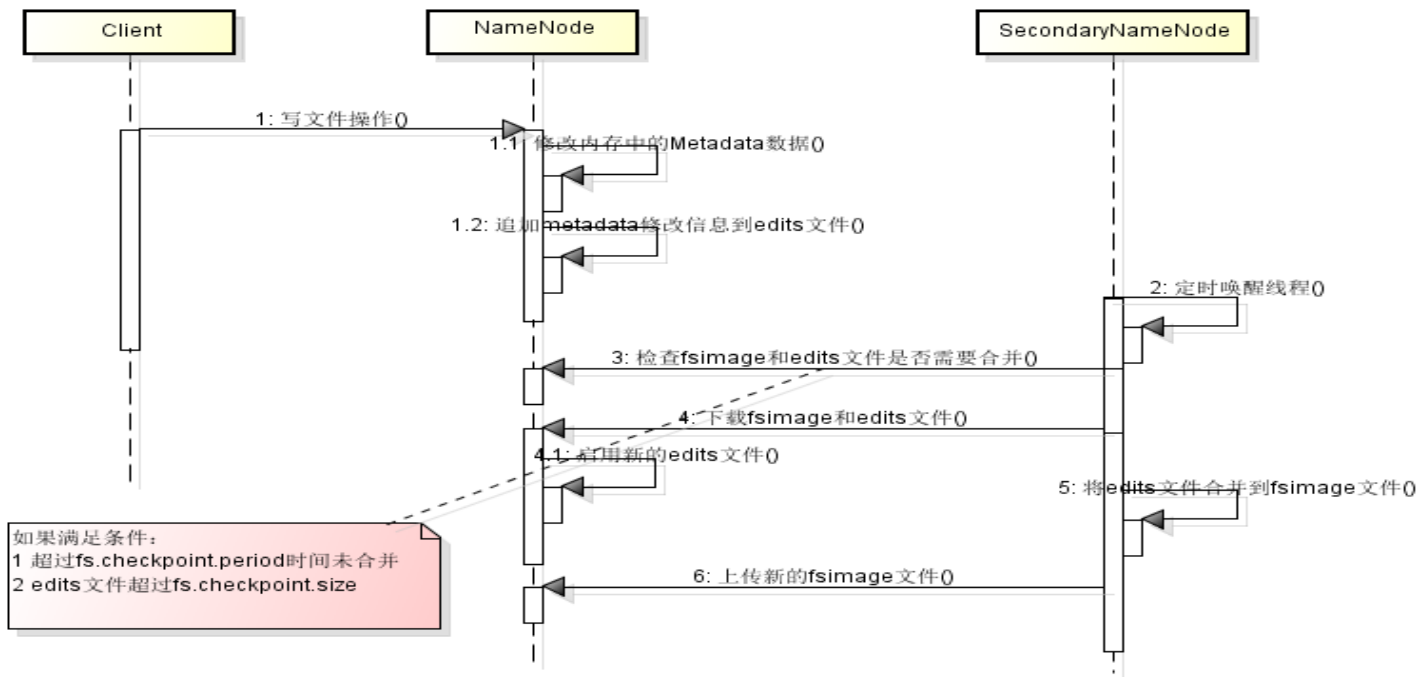
启动secondaryNameNode

```
bin/hadoop secondarynamenode -checkpoint
```

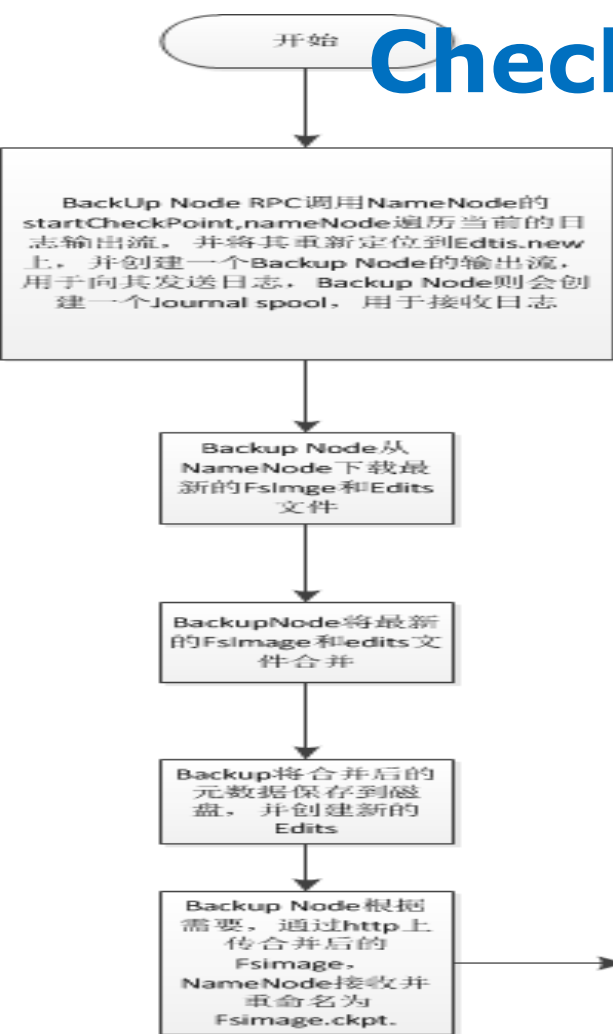
```
bin/hadoop secondarynamenode -checkpoint force
```

checkpoint可以解决重启NameNode时间过长的弊端。

```
bin/hadoop namenode -importCheckpoint
```



Checkpoint node、Backup Node



- SecondaryNN这个名字给人带来混淆, Hadoop1.0.4开始使用CheckPointNode。SecondaryNN和CheckPointNode都只提供一个fsimage更新和检查点备份, 并不提供NN服务, 当NN宕机的时候就会引起HDFS集群不可用。CheckPointNode和SecondaryNN的作用以及配置完全相同, 启动命令不同

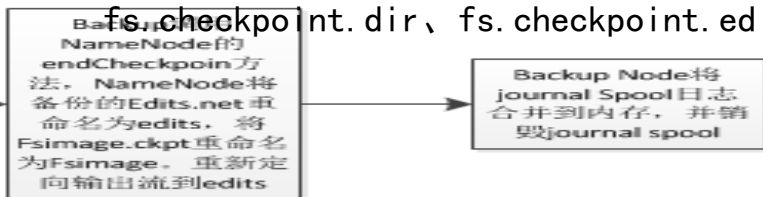
`hadoop secondarynamenode -checkpoint`

`hdfs namenode -checkpoint`

- BackupNode提供一个真正意义上的备用节点, NN所有写操作都会实时将更新Log发送给BackupNode, BackupNode据此更新本机fsimage和edits文件, 并在内存中维护和NN一样的Metadata数据。BackupNode具备了热备功能, 但没有failover, 是阶段性的checkpoint, 无法保证完整性。
- Namenode和BackupNode都要配置这些选项:

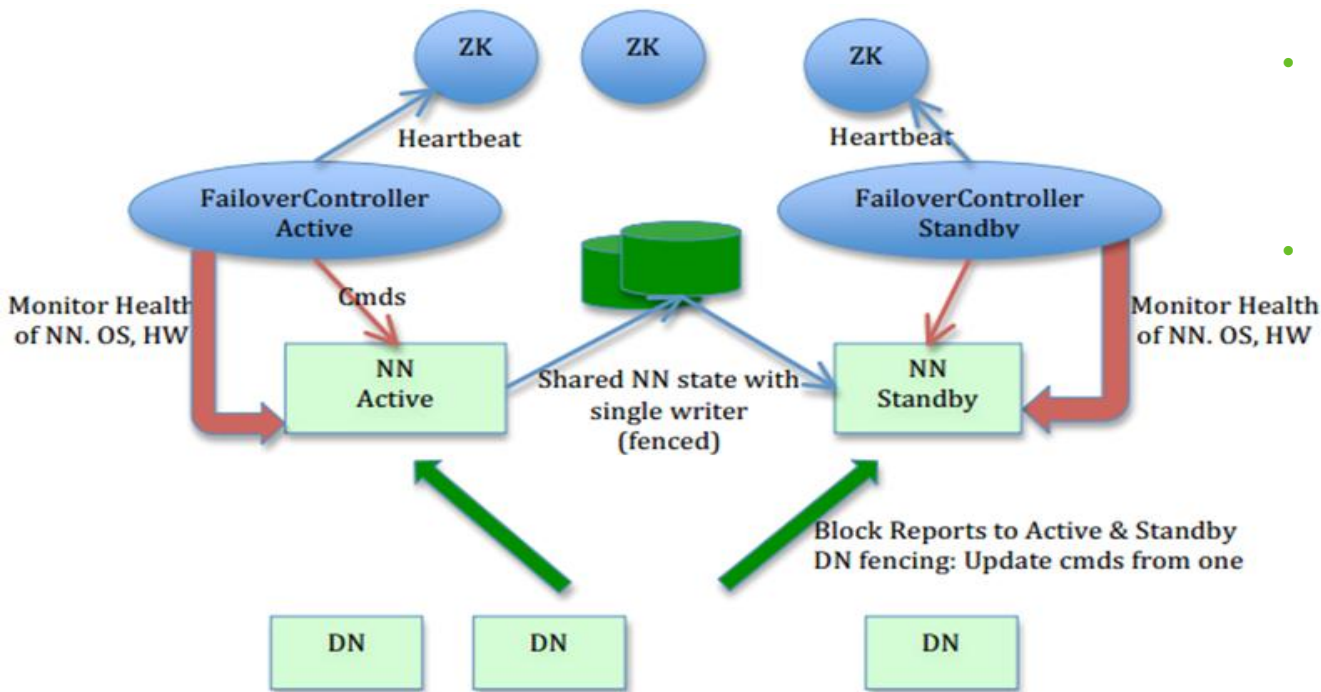
`hdfs-site.xml`: `dfs.backup.address`、`dfs.backup.http.address`

`core-site.xml`: `fs.checkpoint.period`、`fs.checkpoint.size`、`fs.checkpoint.dir`、`fs.checkpoint.edits.dir`



Avatarnode、 Namenode HA

- Avatarnode由Facebook出品，只是Hot Standby，并没有自动切换，当主NN失效的时候，需要管理员确认，然后手动把对外提供服务的虚拟IP映射到Standby NN。
- 利用共享存储来在两个NN间同步edits信息。转移了单点故障的位置。Journal node充当共享存储的功能。DN同时向两个NN汇报块信息。让StandbyNN保持集群最新状态。



- 用FailoverController watchdog进程监视和控制NN进程，防止因NN Full GC挂起无法发送heartbeat
- 防止脑裂：主备切换时由于切换不彻底等原因导致Slave误以为出现两个active master，通常采用Fencing机制：
 - 共享存储fencing，确保只有一个NN可以写入edits
 - 客户端fencing，确保只有一个NN可以响应客户端的请求
 - DN fencing，确保只有一个NN可以向DN下发删除等命令

小文件处理

- HDFS和MapReduce针对**大数据文件**来设计的，小文件指文件大小小于HDFS上block大小的文件。每一个小文件占用一个Block, 每一个block的元数据都存储在namenode的内存，小文件处理上效率低下，十分消耗内存资源。
 - **namenode内存限制**：假设元数据大小约占150byte，如果有1000 0000个小文件，每个文件占用一个Block，namenode需要2G空间。如果存储1亿个文件，则namenode需要20G空间, 这样namenode内存容量严重制约了集群的扩展；
 - **namdeNode性能问题**：HDFS最初为流式访问大文件开发，如果访问大量小文件，需要不断从一个DN跳到另一个DN，严重影响性能；
 - **Slot问题**：处理大量小文件速度远远小于处理同等大小的大文件的速度。每一个小文件要占用一个slot，而task启动将耗费大量时间甚至大部分时间都耗费在启动task和释放task上。

小文件处理

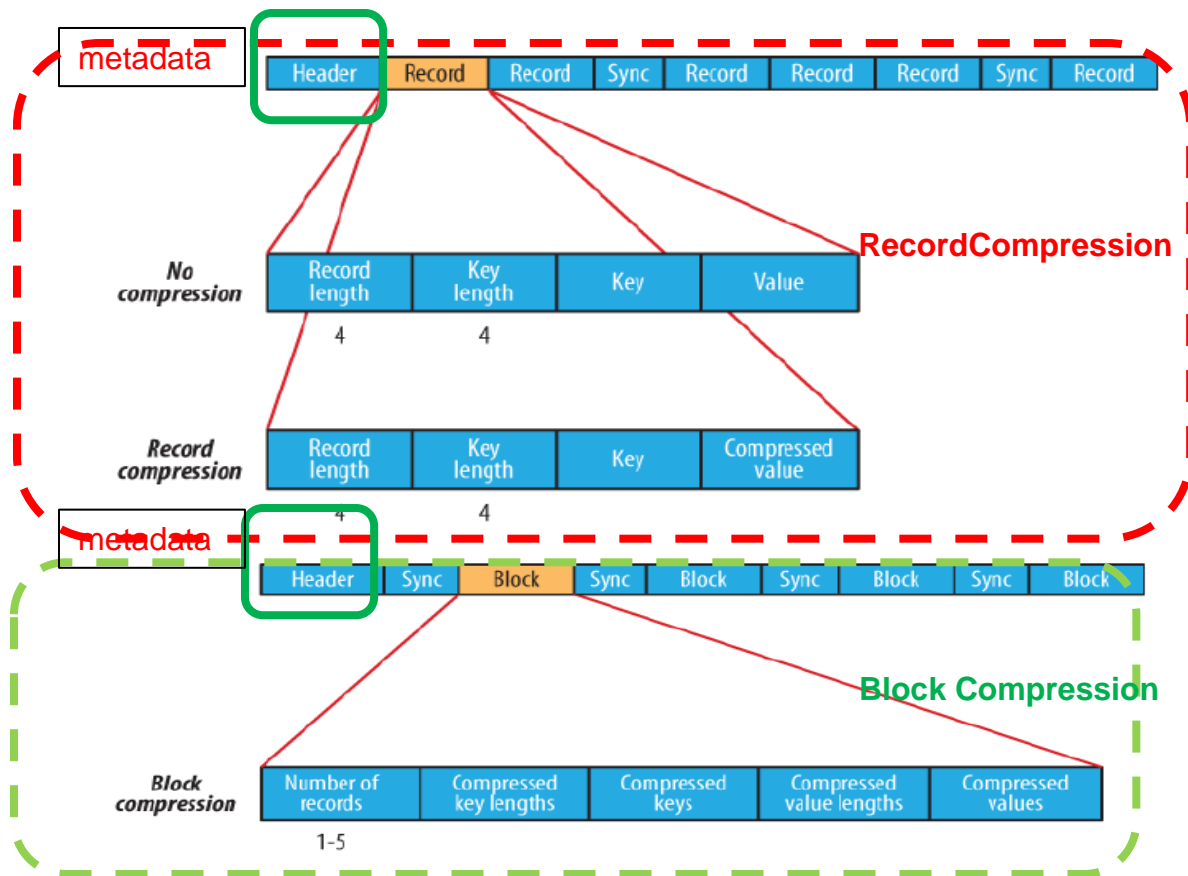
- 1、选择一个容器将这些小文件组织起来统一存储。在原有HDFS基础上添加一个小文件处理模块，当一个文件到达时，判断是否属于小文件，如果是，则交给小文件处理模块处理，否则，交给通用文件处理模块处理。小文件处理模块的设计思想是先将很多小文件合并成一个大文件，然后为这些小文件建立索引，以便进行快速存取和访问。
- 2、Sequence File：提供了二进制键/值对的永久存储的数据结构，由一系列的二进制key/value组成，如果key为小文件名，value为文件内容，则将大批小文件合并成一个大文件。Hadoop提供了SequenceFile，包括Writer，Reader和SequenceFileSorter类进行写，读和排序操作通过SequenceFile类型将小文件包装起来，可以获得更高效的存储和处理。

小文件处理

- SequenceFile文件最开始是Header部分，包含了record的key-value的数据类型、sync标记所采用的字符、有关压缩的细节、用户自定义的metadata。

SequenceFile内置的压缩方式有两种：

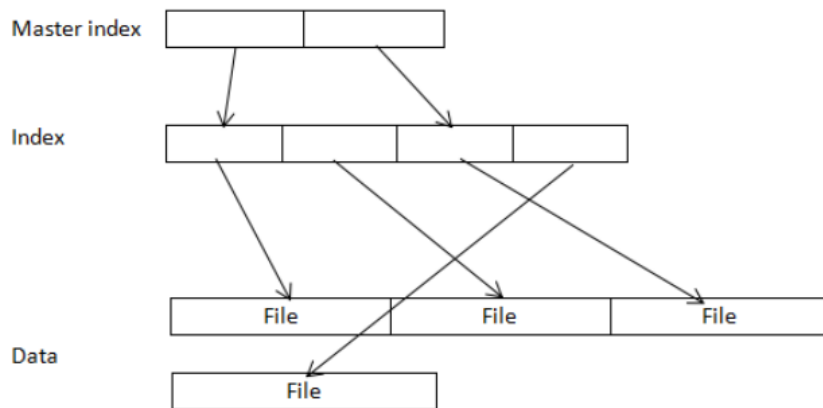
Record Compression和Block Compression。



小文件处理

- 3、使用Hadoop Archive高效地将小文件放入HDFS块中文件存档，能够将多个小文件打包成一个HAR文件，减少namenode内存使用的同时，仍然允许对文件进行透明的访问。
- 使用HAR时需要注意，第一，对小文件进行存档后，原文件并不会自动被删除，需要用户自己删除；第二，创建HAR文件的过程实际上是在运行一个mapreduce作业，因而需要有一个hadoop集群运行此命令。第三，一旦创建，Archives便不可改变。要增加或移除里面的文件，必须重新创建归档文件。第四，要归档的文件名中不能有空格，否则会抛出异常，可以将空格用-替换

HAR File Layout



对某个目录`/foo/bar`下的所有小文件存档成`/outputdir/ zoo.har`:

```
hadoop archive -archiveName zoo.har -p /foo/bar /outputdir
```

HDFS文件压缩

- zip默认的压缩器。
- bzip可以分片压缩，但是速度不行。影响性能。
- lzop压缩。可分片，分片表示该压缩算法支持切分，可以搜索数据流中的任意位置并进一步往下读数据，压缩解压速度快，压缩算法都要在执行速度和压缩比上做一个权衡，支持分片的索引机制，提高并行化的能力。切分压缩尤其适合MapReduce，lzop使用lzo压缩库来提供服务，与gzip工具相比，它的最大优势就是极快的压缩速度和解压速度（在相同的压缩比例的前提下）。由于分布式计算，所以需要支持对压缩数据进行分片，也就是Hadoop的InputSplit，这样才能分配给多台机器并行处理。输入文件建议先用lzop压缩，然后用hadoop-lzo*.jar包对lzo文件建立索引，使大压缩包具有可分片的能力。压缩好了put到hdfs里面，put进去以后还没有分片，还要借助于lzo工具建分片索引。Lzo原生并不支持分片，需要增加索引以后才可以分片。
- gzip比zip压缩比大，但是压缩慢，不支持分割机制。以牺牲本地化为代价：一个map任务将处理16个HDFS块。大都不是map的本地数据。与此同时，因为map任务少，所以作业分割的粒度不够细，从而导致运行时间变长。

压缩算法	原始文件大小	压缩后文件大小	压缩速度	解压缩速度
gzip	8.3GB	1.8GB	17.5MB/s	58MB/s
bzip2	8.3GB	1.1GB	2.4MB/s	9.5MB/s
LZO	8.3GB	2.9GB	49.3MB/S	74.6MB/s

HDFS文件压缩

```
leon@Ubuntu: time lzop test.data
```

```
real    0m7.429s
```

```
user    0m5.260s
```

```
sys     0m1.100s
```

```
485M test.data.lzo
```

```
leon@Ubuntu: time gzip test.data
```

```
real    1m9.639s
```

```
user    1m1.615s
```

```
sys     0m0.881s
```

```
293M test.data.gz
```

在MapReduce中使用压缩

- 大数据量shuffle阶段费时多，压缩可以减少存储文件所需的存储空间；加速数据在网络和磁盘上的传输。压缩算法都要在执行速度和压缩比上做一个权衡
- **输入的文件压缩：**输入的文件是压缩过的，那么在被MapReduce读取时会被自动解压，根据文件扩展名来决定应该使用哪个压缩解码器
- **map作业输出结果的压缩：**map作业输出被写入磁盘并通过网络传输到reducer节点，如果使用LZO之类的快速压缩，能得到更好的性能，因为传输的数据量大大减少了。
- **MapReduce作业的输出的压缩：**对于map和reduce的输出也需要压缩，但可以不用重建索引。使用sequencefile（hadoop内建）块级压缩，压缩解压器还是选择lzop（当然也可以选择其他不可分片的压缩器）。sequencefile本身是分块的，sequencefile格式的文件，再配上lzo的压缩格式，就可实现lzo文件方式的splitable。

在MapReduce中使用压缩

- **MapReduce作业的输出的压缩：**在作业配置文件中将 `mapred.output.compress` 属性设置为 `true`。将 `mapred.output.compression.codec` 属性设置为使用的压缩编码/解码器的类名。如果为输出使用了一系列文件，可以设置 `mapred.output.compression.type` 属性来控制压缩类型，默认为 `RECORD`，它压缩单独的记录。将它改为 `BLOCK`，则可以压缩一组记录
- `set hive.exec.compress.output=true;`
- `set mapred.output.compress=true;`
- `set mapred.output.compression.codec=com.hadoop.compression.lzo.LzoCodec;`
- `set mapred.output.compression.type=BLOCK;`
- `set io.seqfile.compression.type=BLOCK;`

HDFS访问方式

- HDFS Shell命令
- HDFS Java API
- HDFS REST API
- HDFS Fuse: 实现了fuse协议
- HDFS Viewfs
- HDFS NFS
- HDFS lib hdfs: C/C++访问接口
- HDFS 其他语言编程API
 - ✓ 使用thrift实现
 - ✓ 支持C++、Python、php、C#等语言

HDFS Shell命令—文件操作命令

```
[hadoop@chinahadoop-1 hadoop-2.7.3]$ bin/hdfs dfs
```

```
Usage: hadoop fs [generic options]
```

```
[-appendToFile <localsrc> ... <dst>]
```

```
[-cat [-ignoreCrc] <src> ...]
```

```
[-checksum <src> ...]
```

```
[-chgrp [-R] GROUP PATH...]
```

```
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
```

```
[-chown [-R] [OWNER][:[GROUP]] PATH...]
```

```
[-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
```

```
[-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
```

```
[-count [-q] [-h] <path> ...]
```

```
[-cp [-f] [-p | -p[topax]] <src> ... <dst>]
```

```
[-createSnapshot <snapshotDir> [<snapshotName>]]
```

```
[-deleteSnapshot <snapshotDir> <snapshotName>]
```

```
[-df [-h] [<path> ...]]
```

```
[-du [-s] [-h] <path> ...]
```

```
[-expunge]
```

```
[-find <path> ... <expression> ...]
```

```
[-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
```

```
[-getfacl [-R] <path>]
```

```
[-getfattr [-R] {-n name | -d} [-e en] <path>]
```

```
[-getmerge [-nl] <src> <localdst>]
```

```
[-help [cmd ...]]
```

```
[-ls [-d] [-h] [-R] [<path> ...]]
```

```
[-mkdir [-p] <path> ...]
```

```
[-moveFromLocal <localsrc> ... <dst>]
```

```
[-moveToLocal <src> <localdst>]
```

```
[-mv <src> ... <dst>]
```

```
[-put [-f] [-p] [-l] <localsrc> ... <dst>]
```

```
[-renameSnapshot <snapshotDir> <oldName> <newName>]
```

```
[-rm [-f] [-r|-R] [-skipTrash] <src> ...]
```

```
[-rmdir [--ignore-fail-on-non-empty] <dir> ...]
```

➤ 将本地文件上传到HDFS

✓ `hdfs fs -copyFromLocal /local/data /hdfs/data`

➤ 删除文件/目录

✓ `hdfs fs -rmr/hdfs/data`

➤ 创建目录

✓ `hdfs fs -mkdir/hdfs/data`

HDFS Shell命令—管理命令

```
[hadoop@chinahadoop-1 hadoop-2.7.3]$ bin/hdfs dfsadmin
```

Usage: hdfs dfsadmin

Note: Administrative commands can only be run as the HDFS superuser.

```
[-report [-live] [-dead] [-decommissioning]]
[-safemode <enter | leave | get | wait>]
[-saveNamespace]
[-rollEdits]
[-restoreFailedStorage true|false|check]
[-refreshNodes]
[-setQuota <quota> <dirname>...<dirname>]
[-clrQuota <dirname>...<dirname>]
[-setSpaceQuota <quota> [-storageType <storagetype>] <dirname>...<dirname>]
[-clrSpaceQuota [-storageType <storagetype>] <dirname>...<dirname>]
[-finalizeUpgrade]
[-rollingUpgrade [<query|prepare|finalize>]]
[-refreshServiceAcl]
[-refreshUserToGroupsMappings]
[-refreshSuperUserGroupsConfiguration]
[-refreshCallQueue]
[-refresh <host:ipc_port> <key> [arg1..argn]]
[-reconfig <datanode|...> <host:ipc_port> <start|status>]
[-printTopology]
[-refreshNamenodes datanode_host:ipc_port]
[-deleteBlockPool datanode_host:ipc_port blockpoolId [force]]
[-setBalancerBandwidth <bandwidth in bytes per second>]
[-fetchImage <local directory>]
[-allowSnapshot <snapshotDir>]
[-disallowSnapshot <snapshotDir>]
[-shutdownDatanode <datanode_host:ipc_port> [upgrade]]
[-getDatanodeInfo <datanode_host:ipc_port>]
[-metasave filename]
[-triggerBlockReport [-incremental] <datanode_host:ipc_port>]
[-help [cmd]]
```

- ✓ start-all.sh
- ✓ start-dfs.sh
- ✓ start-yarn.sh
- ✓ hadoop-deamon(s).sh
 - 单独启动某个服务
- ✓ hadoop-deamon.sh start namenode
- ✓ hadoop-deamons.sh start namenode

HDFS Shell命令—文件管理命令fsck

- 检查hdfs中文件的健康状况
- 查找缺失的块以及过少或过多副本的块
- 查看一个文件的所有数据块位置
- 删除损坏的数据块

```
[hadoop@chinahadoop-1 hadoop-2.7.3]$ bin/hdfs fsck
Usage: hdfs fsck <path> [-list-corruptfileblocks] [-move | -delete | -openforwrite] [-files [-blocks [-locations | -racks]]] [-includeSnapshots] [-storagepolicies] [-block
Id <blk_Id>]
    <path> start checking from this path
    -move move corrupted files to /lost+found
    -delete delete corrupted files
    -files print out files being checked
    -openforwrite print out files opened for write
    -includeSnapshots include snapshot data if the given path indicates a snapshottable directory or there are snapshottable directories under it
    -list-corruptfileblocks print out list of missing blocks and files they belong to
    -blocks print out block report
    -locations print out locations for every block
    -racks print out network topology for data-node locations
    -storagepolicies print out storage policy summary for the blocks
    -blockId print out which file this blockId belongs to, locations (nodes, racks) of this block, and other diagnostics info (under replicated, corrupted or not,
etc)
```

HDFS Shell命令—文件管理命令fsck

```
[hadoop@chinahadoop-1 ~]$ bin/hdfs fsck /home/hadoop/data/input/text/data_wide_1.txt -files -blocks -locations
Connecting to namenode via http://chinahadoop-1:50070/fsc?ugi=hadoop&files=1&blocks=1&locations=1&path=%2Fhome%2Fhadoop%2Fdata%2Finput%2Ftext%2Fdata_wide_1.txt
FSCK started by hadoop (auth:SIMPLE) from /115.28.132.226 for path /home/hadoop/data/input/text/data_wide_1.txt at Sat Dec 03 14:20:24 CST 2016
/home/hadoop/data/input/text/data_wide_1.txt 1738917506 bytes, 13 block(s): OK
0. BP-1191294997-10.144.130.145-1475046603859:blk_1073742376_1552 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK], DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK]]
1. BP-1191294997-10.144.130.145-1475046603859:blk_1073742377_1553 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK]]
2. BP-1191294997-10.144.130.145-1475046603859:blk_1073742378_1554 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK]]
3. BP-1191294997-10.144.130.145-1475046603859:blk_1073742379_1555 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK], DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK]]
4. BP-1191294997-10.144.130.145-1475046603859:blk_1073742380_1556 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK]]
5. BP-1191294997-10.144.130.145-1475046603859:blk_1073742381_1557 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK], DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK]]
6. BP-1191294997-10.144.130.145-1475046603859:blk_1073742382_1558 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK], DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK]]
7. BP-1191294997-10.144.130.145-1475046603859:blk_1073742383_1559 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK], DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK]]
8. BP-1191294997-10.144.130.145-1475046603859:blk_1073742384_1560 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK]]
9. BP-1191294997-10.144.130.145-1475046603859:blk_1073742385_1561 len=134217728 repl=3 [DatanodeInfoWithStorage[115.28.132.226:50010,DS-4433315b-2699-4d9a-acdd-1a69cb394638,DISK], DatanodeInfoWithStorage[115.28.45.102:50010,DS-e12e6f97-705e-4eb8-b0fd-fe2c28dc4cc7,DISK], DatanodeInfoWithStorage[115.28.48.35:50010,DS-566d809e-aafc-4208-97c3-0ed87d864395,DISK]]
```

HDFS Shell命令—数据均衡器balancer

➤ 数据块重分布

✓ `bin/start-balancer.sh -threshold <percentage of disk capacity>`

➤ percentage of disk capacity

✓ HDFS达到平衡状态的磁盘使用率偏差值

✓ 值越低各节点越平衡，但消耗时间也更长

HDFS Shell命令—设置目录份额

- 限制一个目录最多使用磁盘空间
 - ✓ `bin/hadoop dfsadmin -setSpaceQuota 1t /user/username`
- 限制一个目录包含的最多子目录和文件数目
 - ✓ `bin/hadoop dfsadmin -setQuota 10000 /user/username`

HDFS Shell命令—增加/移除节点

➤ 加入新的datanode

- ✓ 步骤1：将已存在datanode上的安装包（包括配置文件等） 拷贝到新datanode上；
- ✓ 步骤2：启动新datanode： `sbin/hadoop-daemon.sh start datanode`

➤ 移除旧datanode

- ✓ 步骤1：将datanode加入黑名单，并更新黑名单，在NameNode上，将datanode的host或者ip加入配置选项 `dfs.hosts.exclude` 指定的文件中
- ✓ 步骤2：移除datanode： `bin/hadoop dfsadmin -refreshNodes`

HDFS Java API介绍

- Configuration类：该类的对象封装了配置信息，这些配置信息来自core-*.xml;
- FileSystem类：文件系统类，可使用该类的方法对文件/目录进行操作。一般通过FileSystem的静态方法，get获得一个文件系统对象；
- FSDataInputStream和FSDataOutputStream类：HDFS中输入输出流。分别通过FileSystem的open方法和create方法获得。

以上类均来自java包：org.apache.hadoop.fs

HDFS Java程序举例

➤ 将本地文件拷贝到HDFS上;

```
Configuration config = new  
Configuration();
```

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

```
Path srcPath = new Path(srcFile);
```

```
Path dstPath = new  
Path(dstFile);
```

```
hdfs.copyFromLocalFile(srcPath,  
dstPath);
```

➤ 创建HDFS文件;

```
//byte[] buff – 文件内容
```

```
Configuration config = new  
Configuration();
```

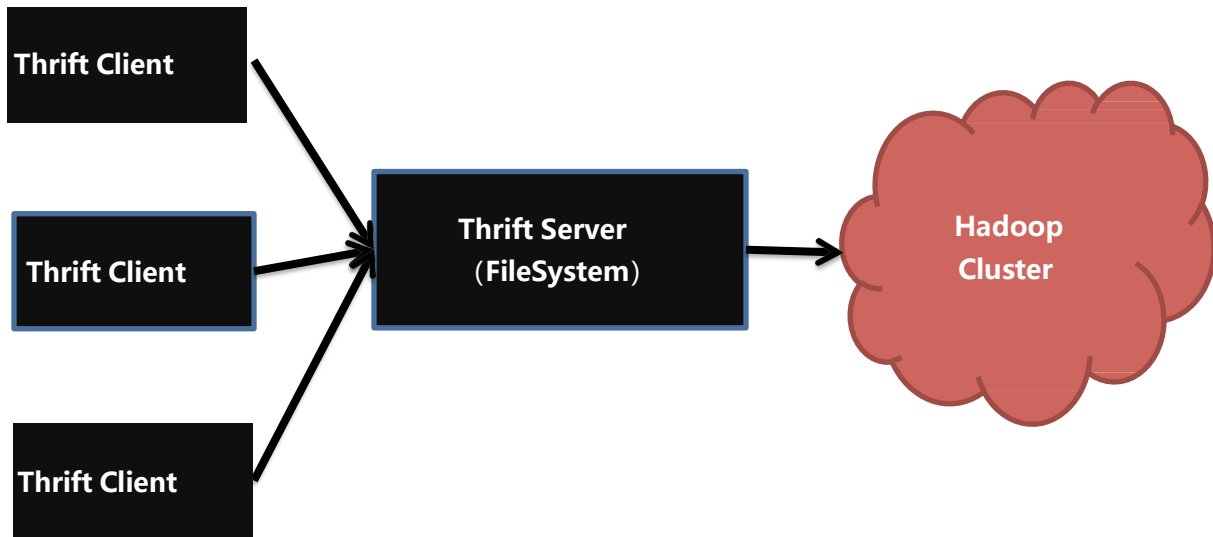
```
FileSystem hdfs =  
FileSystem.get(config);
```

```
Path path = new Path(fileName);
```

```
FSDaOutputStream outputStream =  
hdfs.create(path);
```

```
outputStream.write(buff, 0, buff.length);
```

HDFS 多语言API—借助thrift



hadoopfs.thrift接口定义

```
service ThriftHadoopFileSystem
{

    // set inactivity timeout period. The period is specified in seconds.
    // if there are no RPC calls to the HadoopThrift server for this much
    // time, then the server kills itself.
    void setInactivityTimeoutPeriod(1:i64 periodInSeconds),

    // close session
    void shutdown(1:i32 status),

    // create a file and open it for writing
    ThriftHandle create(1:Pathname path) throws (1:ThriftIOException ouch),

    // create a file and open it for writing
    ThriftHandle createFile(1:Pathname path, 2:i16 mode,
                           3:bool overwrite, 4:i32 bufferSize,
                           5:i16 block_replication, 6:i64 blocksize)
                           throws (1:ThriftIOException ouch),

    // returns a handle to an existing file for reading
    ThriftHandle open(1:Pathname path) throws (1:ThriftIOException ouch),

    // returns a handle to an existing file for appending to it.
    ThriftHandle append(1:Pathname path) throws (1:ThriftIOException ouch),

    // write a string to the open handle for the file
    bool write(1:ThriftHandle handle, string data) throws (1:ThriftIOException ouch),

    // read some bytes from the open handle for the file
    string read(1:ThriftHandle handle, i64 offset, i32 size) throws (1:ThriftIOException ouch),

    // close file
    bool close(1:ThriftHandle out) throws (1:ThriftIOException ouch),
```

PHP语言访问HDFS

```
$transport = new TSocket(HDFS_HOST, HDFS_PORT);
$transport->setRecvTimeout(60000);
$transport->setSendTimeout(60000);
$protocol = new TBinaryProtocol($transport);
$client = new ThriftHadoopFileSystemClient($protocol);
logv("connect hdfs");
$transport->open();
logv("testing existent of `%s'", $remote_uri);
$remote_path = new Pathname(array('pathname' => $remote_uri));
$remote_file = null;
try {
    $remote_file = $client->listStatus($remote_path);
} catch(Exception $e) { }
if (!$remote_file)
    logv("could not open `%s'", $remote_uri);
```

Python语言访问HDFS

```
def connect(self):
    try:
        # connect to hdfs thrift server
        self.transport = TSocket.TSocket(self.server_name, self.server_port)
        self.transport = TTransport.TBufferedTransport(self.transport)
        self.protocol = TBinaryProtocol.TBinaryProtocol(self.transport)

        # Create a client to use the protocol encoder
        self.client = ThriftHadoopFileSystem.Client(self.protocol)
        self.transport.open()

        # tell the HadoopThrift server to die after 60 minutes of inactivity
        self.client.setInactivityTimeoutPeriod(60*60)
        return True

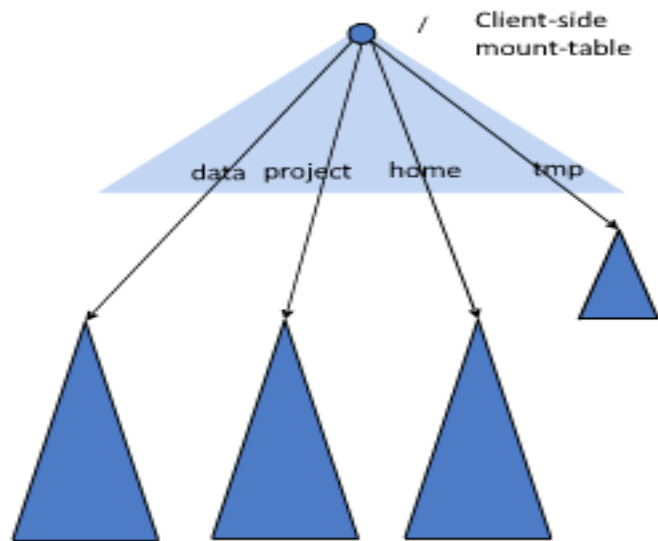
    except Thrift.TException, tx:
        print "ERROR in connecting to ", self.server_name, ":", self.server_port
        print '%s' % (tx.message)
        return False

    def do_create(self, name):
        if name == "":
            print " ERROR usage: create <pathname>"
            print
            return 0

        # Create the file, and immediately closes the handle
        path = Pathname();
        path.pathname = name;
        status = self.client.create(path)
        self.client.close(status)
        return 0
```

viewfs访问HDFS

- 一个集群中需要唯一的命名空间还是多个命名空间，核心问题命名空间中数据的共享和访问的问题。使用全局唯一的命名空间是解决数据共享和访问的一种方法。
- 多命名空间下，可以使用**Client Side Mount Table**方式做到数据共享和访问。
- 深色三角形代表独立的命名空间，浅色三角形代表从客户角度去访问下方的子命名空间。各个深色的命名空间Mount到浅色的表中，访问不同的挂载点来访问不同的命名空间，如Linux系统中访问不同挂载点一样。
- 将各个命名空间挂载到全局mount - table中，就可以做将数据到全局共享；同样的命名空间挂载到个人的mount-table中，这就成为应用程序可见的命名空间视图。

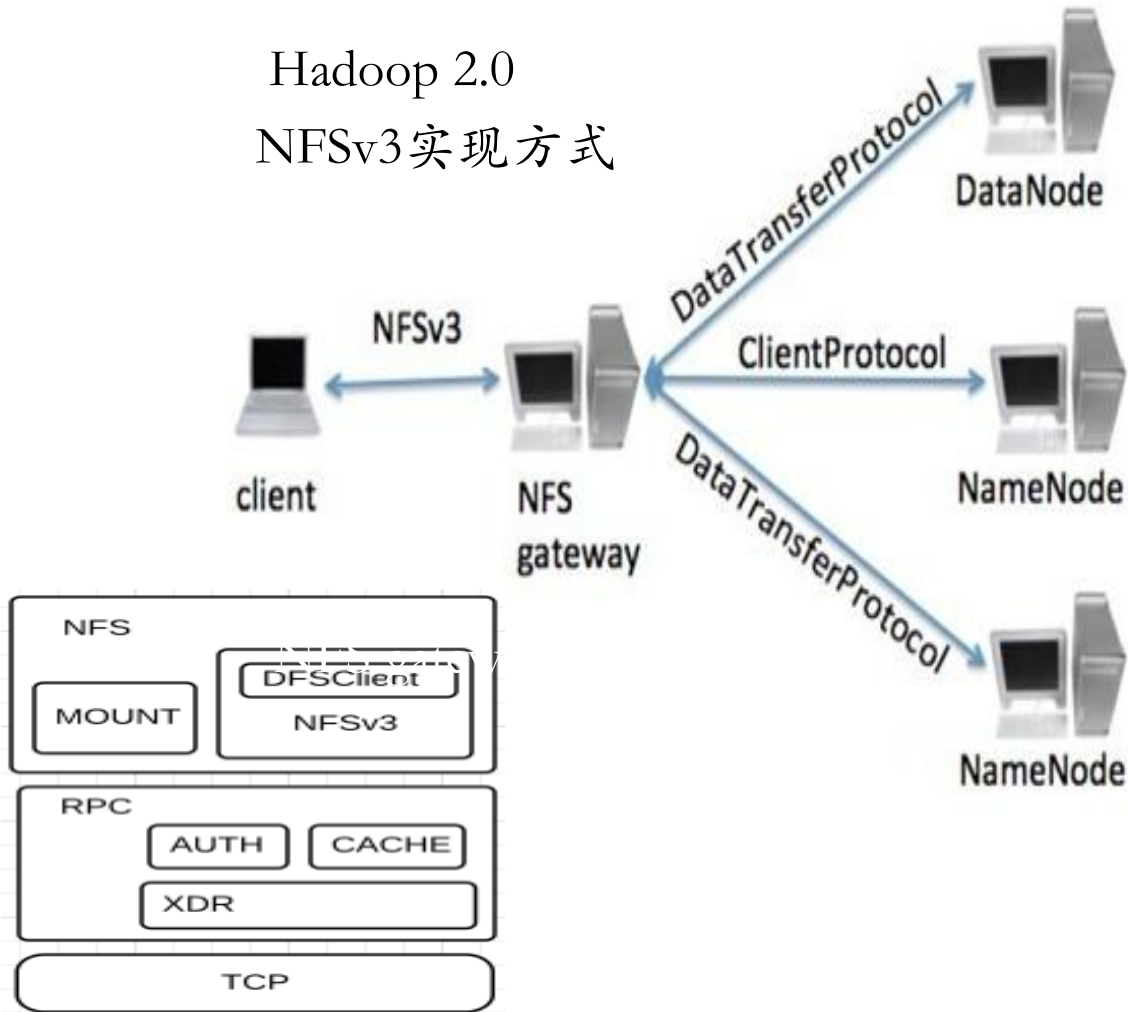


NFS访问HDFS

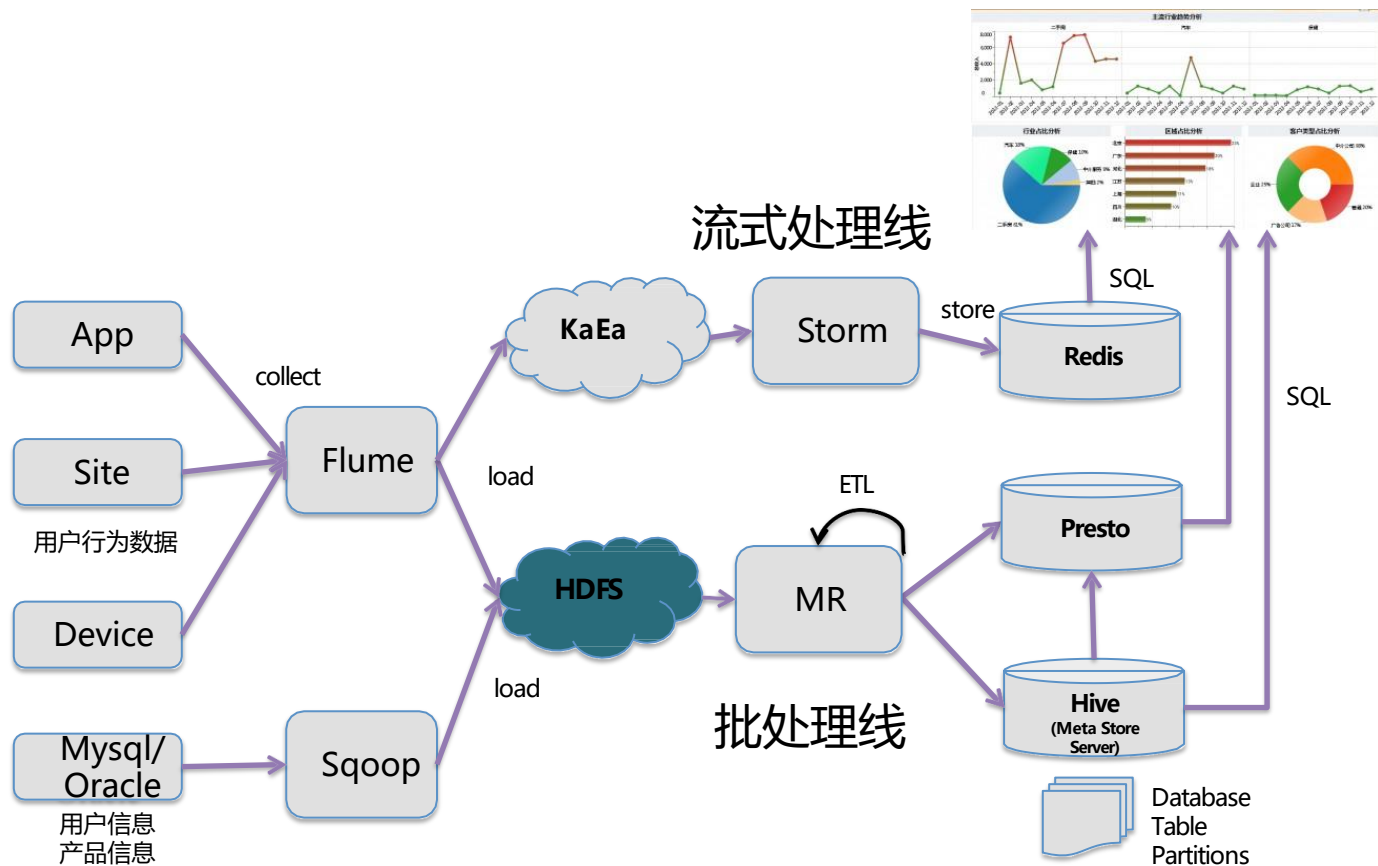
- NFS允许用户像访问本地文件系统一样访问远程文件系统，而将NFS引入HDFS后，用户可像读写本地文件一样读写HDFS上的文件，大大简化了HDFS使用
- 通过引入一个NFS gateway服务实现，该服务能将NFS协议转换为HDFS访问协议
- 用于HDFS文件浏览、文件上传、下载及大型数据流处理

Hadoop 2.0

NFSv3实现方式



日志分析系统：文件存储模块



日志分析系统：文件存储模块注意事项

➤ 数据分区

- ✓ 年/月/日

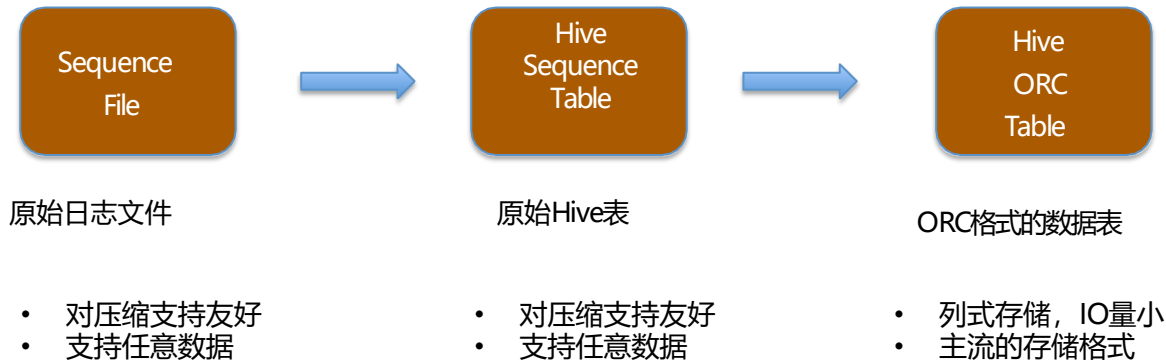
➤ 数据压缩

- ✓ 较少存储空间

➤ 数据存储格式选择

- ✓ 原始日志存储格式选择Sequence file（便于 压缩），而不是文本格式
- ✓ 原始用户信息和商品信息可采用列式存储格式
(ORC或Parquet) 保存（Hive一节会详细介绍）

日志分析系统：数据格式选择



日志分析系统：数据格式选择

➤ 文本文件

- ✓ 不便于压缩，选择合适的压缩算法很重要；
- ✓ 不建议将日志直接存成文本格式

➤ SequenceFile

- ✓ 二进制格式，便于压缩，压缩格式作为元信息 存到文件中；
- ✓ 建议采用该格式存储原始日志

SequenceFile File Layout

Data	Key	Value	Key	Value	Key	Value	Key	Value
------	-----	-------	-----	-------	-----	-------	-----	-------

小文件优化

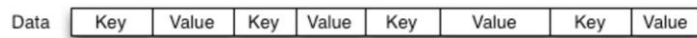
➤ 合并成大文件

- ✓ Sequence file
- ✓ Hadoop Archive

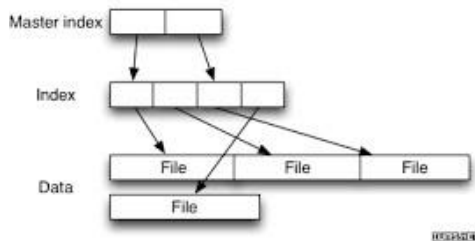
➤ 保存到key/value系统中

- ✓ HBase
- ✓ TFS (Tao Bao FileSystem)

SequenceFile File Layout



HAR File Layout



10082821

压缩与归档

4种压缩格式的特征的比较

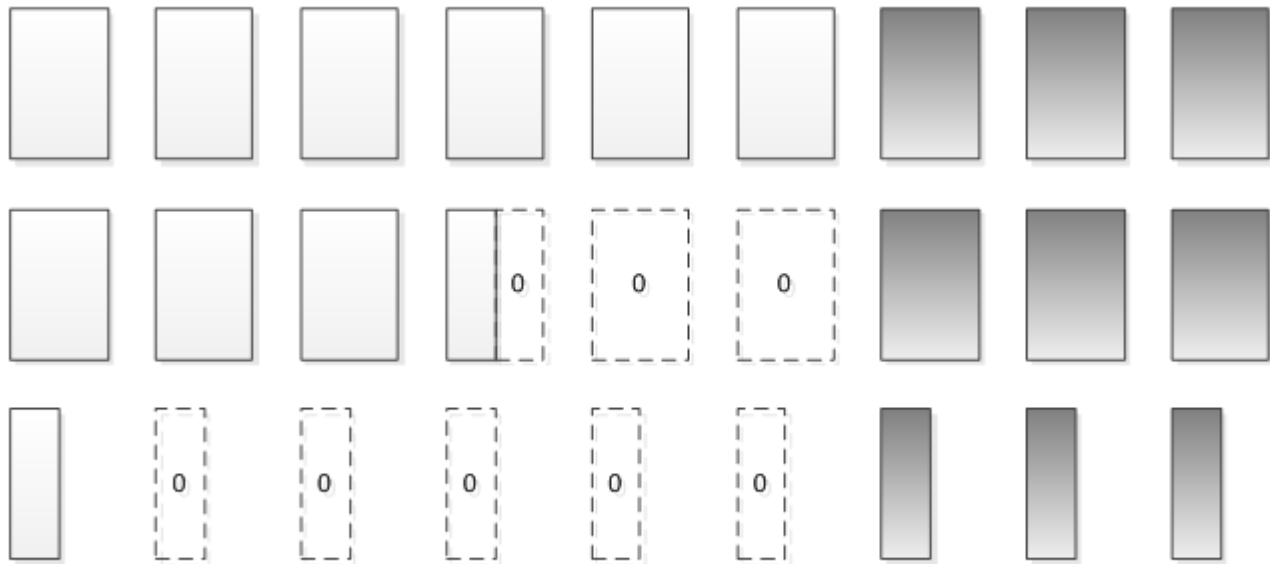
压缩格式	split	native	压缩率	速度	是否hadoop自带	linux命令	换成压缩格式后，原来的应用程序是否要修改
gzip	否	是	很高	比较快	是，直接使用	有	和文本处理一样，不需要修改
lzo	是	是	比较高	很快	否，需要安装	有	需要建索引，还需要指定输入格式
snappy	否	是	比较高	很快	否，需要安装	没有	和文本处理一样，不需要修改
bzip2	是	否	最高	慢	是，直接使用	有	和文本处理一样，不需要修改

➤ Flume

➤ MapReduce/Hive/Spark

纠删码编码

- 通过引入纠删码，节省存储空间（节省一半空间）
- Hadoop 3.0（目前为alpha版本）可用



增大热点文件副本数

➤ 通过程序API修改

- ✓ `FileSystem fs = FileSystem.get(path, conf);`
- ✓ `fs.setReplication(path, (short) 4);`

➤ 通过配置参数修改

- ✓ `dfs.replication: 1`

➤ 通过命令行

- ✓ 增加文件的副本数 `hadoop dfs -setrep -w 4 /path/to/file`
- ✓ 递归增加目录下文件的副本数
- ✓ `hadoop dfs -setrep -R -w 4 /path/to/file`

冷热数据

➤ “冷热” 数据

- ✓ 冷数据：过去半年内没访问过的数据
- ✓ 冷数据可进行特殊处理，包括高压缩，小文件合并等

➤ 找出 “冷热” 数据

- ✓ `hdfs oiv -i /home/hadoop/data/hdfs/name/current/fsimage_00000000000001619538 -p XML -o fsimage.xml`
- ✓ 找出上次访问时间为半年之前的文件
`<atime>1475047293212</atime>`

冷数据处理

- 高压缩比算法进行压缩

- ✓ Gzip或bzip2

- 合并小文件

- ✓ 使用MapReduce实现

- 异构存储

- ✓ <http://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-hdfs/ArchivalStorage.html>

异构层级存储

- 每个节点是由多种异构存储介质构成的

<property>

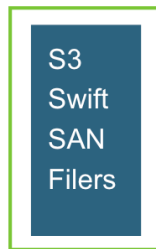
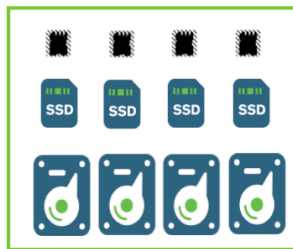
<name>dfs.datanode.data.dir</name>

<value>[disk]/dir0,[disk]/dir1,[ssd]/dir2,[ssd]/dir3</value>

</property>



All disks as a single storage



Collection of Heterogeneous Storage Media

hdfs storagepolicies -setStoragePolicy -path <path> -policy <policy>

Policy ID	Policy Name	Block Placement (n replicas)
15	Lazy_Persist	RAM_DISK: 1, DISK: n-1
12	All_SSD	SSD: n
10	One_SSD	SSD: 1, DISK: n-1
7	Hot (default)	DISK: n
5	Warm	DISK: 1, ARCHIVE: n-1
2	Cold	ARCHIVE: n

HDFS操作

- Hadoop -help
- `hadoop fs -ls /wang` 显示wang目录下的所有文件和目录
- `hadoop fs -ls -R /wang` 显示wang目录下的所有文件和目录加R
- `hadoop fs -mkdir -p /a/b/c` -p表示创建多级目录
- `hadoop fs -put libsvm.jar /wang/`
- `hadoop fs -get /wang/`
- `hadoop fs -get /wang/haha`
- `hadoop fs -rm /wang/haha`
- `hadoop fs -rmr /wang` 删除文件夹
- `hadoop fs -copyFromLocal /tmp/word.txt /wang` 复制本地文件到hdfs
- `hadoop fs -copyToLocal /wang /tmp/word.txt` 复制文件到本地系统
- `hadoop fs -cat /wang/haha`
- `hdfs fsck /wang -files -blocks -locations`显示各个文件由那些块组成，如何分布
- `hadoop fsck /wang -delete` #检查HDFS块状态，删除损坏块
- `Hadoop fs -du /wang` 显示目录中所有文件的大小
- `Hadoop fs -du -s /wang` 显示当前目录或者文件夹的大小可加选项 -s
- `hadoop fs -mv /in/test2.txt /test2.txt`
- `hadoop fs -touchz /empty.txt` 创建一个0字节的空文件

HDFS操作

- `hdfs dfsadmin -safemode get` 离开安全模式
- `Hdfs dfsadmin -safemode enter` 进入安全模式
- `hadoop dfsadmin -printTopology`
- `hadoop jar cascade.jar /wang/train1.txt /output 4`
- `Start-balancer.sh -threshold XX` 数据块重分布, XX为HDFS达到平衡状态磁盘使用率偏差值, 值越低各节点越平衡, 但消耗时间也更长
- `Hadoop dfsadmin -setSpaceQuota 1t /user/`限制一个目录最多使用磁盘空间
- `Hadoop dfsadmin -setQuota 10000 /user/`限制一个目录包含最多子目录和文件数
- 加入新的datanode
- 步骤1: 将已存在datanode上的安装包 (包括配置文件等) 拷贝到新的datanode上;
- 步骤2: `sbin/hadoop-deamon.sh.start.datanode` 启动新的datanode
- 移除旧datanode
- 步骤1: 将datanode加入黑名单, 并更新黑名单, 在NameNode上, 将datanode的host, 或者ip加入配置选项`dfs.hosts.exclude`指定的文件中
- 步骤2: `bin/hadoop dfsadmin -refreshNodes` 移除datanode

HDFS操作

```
[root@master ~]# hadoop dfsadmin -report
```

```
DEPRECATED: Use of this script to execute hdfs command is deprecated.  
Instead use the hdfs command for it.
```

```
Configured Capacity: 155312836608 (144.65 GB)  
Present Capacity: 133794054144 (124.61 GB)  
DFS Remaining: 132795457536 (123.68 GB)  
DFS Used: 998596608 (952.34 MB)  
DFS Used%: 0.75%  
Under replicated blocks: 0  
Blocks with corrupt replicas: 0  
Missing blocks: 0
```

总集群信息

```
-----  
Datanodes available: 3 (3 total, 0 dead)
```

节点信息












```
Live datanodes:
```

```
Name: 192.168.70.243:50010 (node3.hadoop)  
Hostname: node3.hadoop  
Decommission Status : Normal  
Configured Capacity: 51770945536 (48.22 GB)  
DFS Used: 332865536 (317.45 MB)  
Non DFS Used: 5849280512 (5.45 GB)  
DFS Remaining: 45588799488 (42.46 GB)  
DFS Used%: 0.64%  
DFS Remaining%: 88.06%  
Configured Cache Capacity: 0 (0 B)  
Cache Used: 0 (0 B)  
Cache Remaining: 0 (0 B)  
Cache Used%: 100.00%  
Cache Remaining%: 0.00%  
Last contact: Thu Sep 18 16:27:01 CST 2014
```

节点详细信息

HDFS常用端口

组件	节点	默认端口	配置	用途说明
HDFS	DataNode	50010	dfs.datanode.address	datanode 服务端口，用于数据传输
HDFS	DataNode	50075	dfs.datanode.http.address	http服务的端口
HDFS	DataNode	50475	dfs.datanode.https.address	https服务的端口
HDFS	DataNode	50020	dfs.datanode.ipc.address	ipc服务的端口
HDFS	NameNode	50070	dfs.namenode.http-address	http服务的端口
HDFS	NameNode	50470	dfs.namenode.https-address	https服务的端口
HDFS	NameNode	8020	fs.defaultFS	接收 Client 连接的 RPC 端口，用于获取文件系统 metadata 信息
HDFS	journalnode	8485	dfs.journalnode.rpc-address	RPC服务
HDFS	journalnode	8480	dfs.journalnode.http-address	HTTP服务
HDFS	ZKFC	8019	dfs.ha.zkfc.port	ZooKeeper FailoverController，用于NN HA

- ✓ HDFS  28
- ✓ MapReduce2 
- ✓ YARN  7
-  Tez 
- ✓ Hive
- ✓ HBase 
-  Pig
-  Sqoop
- ✓ ZooKeeper
- ✓ Storm 
-  Ambari Metrics  2
- ✓ Ranger
- ✓ Spark

Summary

Heatmaps

Configs

Quick Links ▾

Summary

28 alerts

Active NameNode  Started

ZKFailoverController Started

Standby NameNode  Started

ZKFailoverController  Started

DataNodes 41/42 Started

DataNodes Status 41 live / 1 dead / 0 decommissioning

JournalNodes 3/3 JournalNodes Live

NFSGateways 0/0 Started

NameNode Uptime 168.97 days

NameNode Heap 7.3 GB / 14.6 GB (50.3% used)

Disk Usage (DFS Used) 1.1 PB / 1.5 PB (74.51%)

Disk Usage (Non DFS Used) 88.6 TB / 1.5 PB (5.89%)

Disk Remaining 294.9 TB / 1.5 PB (19.60%)

Blocks (total) 18219526

Block Errors 0 corrupt / 32 missing / 32 under replicated

Total Files + Directories 21209218

Upgrade Status No pending upgrade

Safe Mode Status Not in safe mode

Metrics

Actions

Last 1 hour ▾

NameNode GC count

No Data Available

NameNode GC time

No Data Available

NN Connection Load

No Data Available

NameNode Heap

No Data Available

NameNode Host Load

No Data Available



logged in as: yarn

RUNNING Applications

- Cluster
- About Nodes
- Node Labels
- Applications
- NEW
- NEW SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- PAUSED
- FAILED
- Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	V-Cores Used	V-Cores Total	V-Cores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
230814	0	27	230787	308	1.24 TB	4.77 TB	0 B	308	1862	0	49	0	0	0	0












Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	MEMORY	<memory:3328,vCores:1>	<memory:102144,vCores:38>

Show 20 entries

Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	Final Status	Running Containers	Allocated CPU V-Cores	Allocated Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1516850907017_231152	sunhullang	HIVE-7024d22b-d442-4c09-9705-0fcedd05040b	TEZ	hue	0	Thu Mar 15 15:33:44 +0800 2018	N/A	RUNNING	UNDEFINED	135	135	452808	90.4	9.0	<div></div>	ApplicationMaster	0
application_1516850907017_231151	qiaoiaolong	HIVE-c9942af-co47-460f-0c93-034649618ed4	TEZ	bd_test	0	Thu Mar 15 15:32:04 +0800 2018	N/A	RUNNING	UNDEFINED	14	14	49920	110.8	1.0	<div></div>	ApplicationMaster	0
application_1516850907017_231101	bd_wh	HIVE-e210e164-1d19-4864-9687-54cc0e5a3ca4	TEZ	bd_wh	0	Thu Mar 15 15:26:36 +0800 2018	N/A	RUNNING	UNDEFINED	1	1	6666	0.2	0.1	<div></div>	ApplicationMaster	0
application_1516850907017_231086	chenchen	HIVE-d741d5c7-6d04-48fa-b1e0-e0c07c368c30	TEZ	bd_test	0	Thu Mar 15 15:25:09 +0800 2018	N/A	RUNNING	UNDEFINED	14	14	49920	110.8	1.0	<div></div>	ApplicationMaster	0
application_1516850907017_231079	haoshuaiyu	HIVE-d045f5c-e427-40c0-b7fb-ecdf1e8c769f	TEZ	bd_ais	0	Thu Mar 15 15:19:11 +0800 2018	N/A	RUNNING	UNDEFINED	66	66	222976	99.0	4.5	<div></div>	ApplicationMaster	0
application_1516850907017_230974	haoshuaiyu	HIVE-df1de517-ba26-4d27-e457-6896e8afa79d	TEZ	bd_ais	0	Thu Mar 15 14:12:35 +0800 2018	N/A	RUNNING	UNDEFINED	1	1	6666	3.0	0.1	<div></div>	ApplicationMaster	0
application_1516850907017_221915	bd_access	GomeSokStreaming	SPARK	bd_access	0	Tue Mar 13 17:49:40 +0800 2018	N/A	RUNNING	UNDEFINED	3	3	23296	3.7	0.5	<div></div>	ApplicationMaster	0
application_1516850907017_221911	bd_access	shuang11_available_aty_spark	SPARK	bd_access	0	Tue Mar 13 17:42:49 +0800 2018	N/A	RUNNING	UNDEFINED	4	4	33280	5.3	0.7	<div></div>	ApplicationMaster	0
application_1516850907017_221873	bd_access	GomeSokCurrent	SPARK	bd_access	0	Tue Mar 13 17:25:11 +0800 2018	N/A	RUNNING	UNDEFINED	3	3	26624	4.2	0.5	<div></div>	ApplicationMaster	0
application_1516850907017_221836	bd_access	realtime_rebate_spark	SPARK	bd_access	0	Tue Mar 13 17:13:36 +0800 2018	N/A	RUNNING	UNDEFINED	5	5	33280	5.3	0.7	<div></div>	ApplicationMaster	0
application_1516850907017_221713	bd_access	realtime_order_rebate_spark	SPARK	bd_access	0	Tue Mar 13 16:21:40 +0800 2018	N/A	RUNNING	UNDEFINED	3	3	9984	1.6	0.2	<div></div>	ApplicationMaster	0
application_1516850907017_207748	bd_access_wh	AppSessionExtend-2.0.jar	MAPREDUCE	bd_wh	0	Sat Mar 10 11:07:24 +0800 2018	N/A	RUNNING	UNDEFINED	4	4	23296	0.8	0.5	<div></div>	ApplicationMaster	0
application_1516850907017_68042	bd_access	product_info_realtime_new	SPARK	bd_access	0	Thu Feb 8 17:53:22 +0800 2018	N/A	RUNNING	UNDEFINED	3	3	26624	4.2	0.5	<div></div>	ApplicationMaster	0
application_1516850907017_67793	bd_access	shuang11_available_aty_spark_new	SPARK	bd_access	0	Thu Feb 8 16:31:38 +0800 2018	N/A	RUNNING	UNDEFINED	5	5	43264	6.9	0.9	<div></div>	ApplicationMaster	0
application_1516850907017_67691	bd_access	realtime_available_aty_spark	SPARK	bd_access	0	Thu Feb 8 16:14:49 +0800 2018	N/A	RUNNING	UNDEFINED	4	4	36808	5.8	0.7	<div></div>	ApplicationMaster	0
application_1516850907017_67680	bd_access	user_footprint_smv4	SPARK	bd_access	0	Thu Feb 8 16:11:31 +0800 2018	N/A	RUNNING	UNDEFINED	5	5	19968	3.2	0.4	<div></div>	ApplicationMaster	0
application_1516850907017_67677	bd_access	user_footprint_mobile	SPARK	bd_access	0	Thu Feb 8 16:09:31 +0800 2018	N/A	RUNNING	UNDEFINED	7	7	46592	7.4	0.9	<div></div>	ApplicationMaster	0
application_1516850907017_67676	bd_access	Collect3Mysql	SPARK	bd_access	0	Thu Feb 8 16:09:30 +0800 2018	N/A	RUNNING	UNDEFINED	3	3	16640	2.6	0.3	<div></div>	ApplicationMaster	0
application_1516850907017_67672	bd_access	video_realtime	SPARK	bd_access	0	Thu Feb 8 16:08:47 +0800 2018	N/A	RUNNING	UNDEFINED	3	3	13312	2.1	0.3	<div></div>	ApplicationMaster	0

- ✓ HDFS  28
- ✓ MapReduce2 
- ✓ YARN  7
-  Tez 
- ✓ Hive
- ✓ HBase 
-  Pig
-  Sqoop
- ✓ ZooKeeper
- ✓ Storm 
-  Ambari Metrics  2
- ✓ Ranger
- ✓ Spark

谢谢！

