3 Ensuring  Data  Integrity

What is Apache Hadoop’s Track Record for Data Integrity?

* + 98 blocks were blocks explicitly created with a single replica. Applications sometime create files with replication factor of 1 to gain addition performance advantage when the data is not critical or easily reproduced.

533 blocks were temporary blocks abandoned by a failed client. Due to a bug in release 0.20’s append, when a client abandons a file, its temporary blocks are not correctly deleted and unfortunately reported as corrupt.

Wednesday, June 04, 2014

10:48 AM

•Understand  block-­‐scanning  report

•Running  file-­‐system  check

•Understand  replication  factor,  under  &  over  replication

•Setting  up  NFS  Gateway  for  easy  access  to  HD

**Data Integrity**

Users of Hadoop rightly expect that no data will be lost or corrupted during storage or processing. However, since every I/O operation on the disk or network carries with it a small chance of introducing errors into the data that it is reading or writing, when the volumes of data flowing through the system are as large as the ones Hadoop is capable of handling, the chance of data corruption occurring is high.

The usual way of detecting corrupted data is by computing a *checksum* for the data when it first enters the system, and again whenever it is transmitted across a channel that is unreliable and hence capable of corrupting the data. The data is deemed to be corrupt if the newly generated checksum doesn’t exactly match the original. This technique doesn’t offer any way to fix the data—merely error detection. (And this is a reason for not using low-end hardware; in particular, be sure to use ECC memory.) Note that it is possible that it’s the checksum that is corrupt, not the data, but this is very unlikely, since the checksum is much smaller than the data.

**Datanode block scanner**

Every datanode runs a block scanner, which periodically verifies all the blocks stored

on the datanode. This allows bad blocks to be detected and fixed before they are read

by clients. The DataBlockScanner maintains a list of blocks to verify and scans them one

by one for checksum errors. The scanner employs a throttling mechanism to preserve

disk bandwidth on the datanode.

Blocks are periodically verified every three weeks to guard against disk errors over time

(this is controlled by the dfs.datanode.scan.period.hours property, which defaults to

504 hours). Corrupt blocks are reported to the namenode to be fixed.

You can get a block verification report for a datanode by visiting the datanode’s web

interface at <http://datanode:50075/blockScannerReport>.

* <http://datanode:50075/blockScannerReport>

?listblocks, the report is preceded by a list of all the blocks on the datanode along with their latest verification status.

**Audit Logging**

Audit logging is implemented using log4j logging

at the INFO level, and in the default configuration it is disabled, as the log threshold is

set to WARN in log4j.properties:

log4j.logger.org.apache.hadoop.fs.FSNamesystem.audit=WARN

You can enable audit logging by replacing WARN with INFO, and the result will be a log line written to the namenode’s log for every HDFS event.

**Tools**

1. **Dfsadmin**

dfsadmin tool is a multipurpose tool for finding information about the state of

HDFS, as well as performing administration operations on HDFS.

Commands that alter HDFS state typically require superuser

privileges.

Machine generated alternative text: Table 10-2. dfsadmin commands
Command Description
- help Shows help for a given command, or all commands if no command is specified.
-report Shows filesystem statistics (similar to those shown in the web UI) and information on connected
data nodes.
-rnetasave Dumps information to a file ¡n Hadoop’s log directory about blocks that are being replicated or
deleted, and a list of connected data nodes.
— safernode Changes or query the state of safe mode. See Safe Mode on page 298.
- saveNatnespace Saves the current in-memory filesystem image to a new fsimoge file and resets the edits file. This
Operation may be performed only in safe mode.
-refreshNodes Updatesthe set ofdatanodesthatare permitted toconnecttothe namenode. See Commissioning
and Decommissioning Nodes on page 313.
- upgradeProgres s Gets information on the progress ofan HOFS upgrade or forces an upgrade to proceed. See
Upgrades on page 316.
- flnallzeupgrade Removes the previous version of the datanodes’ and namenode’s storage directories. Used after
an upgrade has been applied and the duster is running successfully on the new version. See
Upgrades on page 316.
- setC$.iota Sets directory quotas. Directory quotas set a limit on the number of names (files or directories) in
the directory tree. Directory quotas are useful for preventing users from creating large numbers
of small files, a measure that helps preserve the namenode’s memory (recall that accounting
information for every file, directory, and block in the filesystem is stored in memory).
-cirQuota Clears specified directory quotas.
- set SpaceQuota Sets space quotas on directories. Space quotas set a limit on the size of files that may be stored in
a directory tree. They are useful for giving users a limited amount of storage.
-cirSpaceQuota Clears specified space quotas.
-ref reshServlceAcl Refreshes the namenode’s service-level authorization polky file.

1. **Filesystem check (fsck)**

Hadoop provides an fsck utility for checking the health of files in HDFS. The tool looks for blocks that are missing from all datanodes, as well as under- or over-replicated blocks.

**hadoop fsck /**

To check a file, fsck retrieves the metadata for the file’s blocks and looks for problems or inconsistencies. Note that fsck retrieves all of its information from the namenode; it does not communicate with any datanodes to actually retrieve any block data.

output from fsck is self-explanatory

**Over-replicated blocks**

These are blocks that exceed their target replication for the file they belong to.

Over-replication is not normally a problem, and HDFS will automatically delete

excess replicas.

**Under-replicated blocks**

These are blocks that do not meet their target replication for the file they belong

to. HDFS will automatically create new replicas of under-replicated blocks until

they meet the target replication. You can get information about the blocks being

replicated (or waiting to be replicated) using hadoop dfsadmin -metasave.

**Misreplicated blocks**

These are blocks that do not satisfy the block replica placement policy (see “Replica Placement” in HDFS). For example, for a replication level of three in a multirack cluster, if all three replicas of a block are on the same rack, then the block is misreplicated since the replicas should be spread across at least two racks for resilience.

A misreplicated block is not fixed automatically by HDFS (at the time of this writing). As a workaround, you can fix the problem manually by increasing the replication of the file the block belongs to (using hadoop fs -setrep), waiting until the block gets replicated, then decreasing the replication of the file back to its original value.

**Corrupt blocks**

These are blocks whose replicas are all corrupt. Blocks with at least one noncorrupt replica are not reported as corrupt; the namenode will replicate the noncorrupt replica until the target replication is met.

**Missing replicas**

These are blocks with no replicas anywhere in the cluster.

Corrupt or missing blocks are the biggest cause for concern

perform one of the following actions on them:

• Move the affected files to the /lost+found directory in HDFS, using the -move option. Files are broken into chains of contiguous blocks to aid any salvaging efforts youmay attempt.

• Delete the affected files, using the -delete option. Files cannot be recovered after being deleted.

**Finding the blocks for a file.**

The fsck tool provides an easy way to find out which blocks are

in any particular file.

For example: **% hadoop fsck /user/tom/part-00007 -files -blocks -racks**

/user/tom/part-00007 25582428 bytes, 1 block(s): OK

0. blk\_-3724870485760122836\_1035 len=25582428 repl=3 [/default-rack/10.251.43.2:50010, /default-rack/10.251.27.178:50010, /default-rack/10.251.123.163:50010]

This says that the file /user/tom/part-00007 is made up of one block and shows the datanodes where the blocks are located. The fsck options used are as follows:

• The -files option shows the line with the filename, size, number of blocks, and

its health (whether there are any missing blocks).

• The -blocks option shows information about each block in the file, one line per

block.

• The -racks option displays the rack location and the datanode addresses for each block.

**Balancer**

Over time, the distribution of blocks across datanodes can become unbalanced. An unbalanced cluster can affect locality for MapReduce, and it puts a greater strain on the highly utilized datanodes, so it’s best avoided.

The balancer program is a Hadoop daemon that re-distributes blocks by moving them from over-utilized datanodes to under-utilized datanodes, while adhering to the block replica placement policy that makes data loss unlikely by placing block replicas on different racks.

You can start the balancer with:

% start-balancer.sh

The balancer runs until the cluster is balanced; it cannot move any more blocks, or it loses contact with the namenode. It produces a logfile in the standard log directory,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rebalancing Threshold** | | 10.0 % | | The percentage deviation from average utilization, after which a node will be rebalanced. (for example, '10.0' for 10%) |
| **Rebalancing Policy** | DataNode | | The policy that should be used to rebalance HDFS storage. The default DataNode policy balances the storage at the DataNode level. This is similar to the balancing policy from prior releases. The BlockPool policy balances the storage at the block pool level as well as at the Datanode level. The BlockPool policy is relevant only to a Federated HDFS service | | |

It limits the bandwidth that it uses to copy a block from one node to another. The default is a modest 1 MB/s, but this can be changed by setting the dfs.balance.bandwidthPerSec property in hdfs-site.xml, specified

in bytes.

**HDFS NFS Gateway**

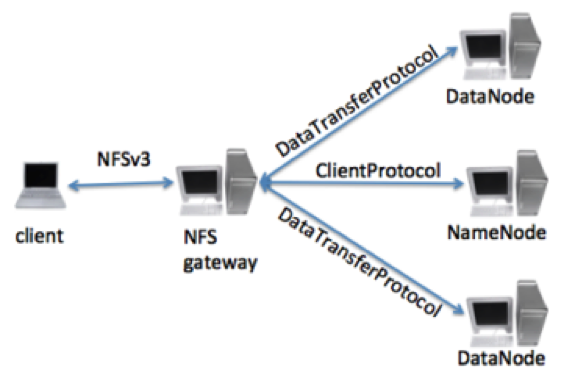
With NFS access to HDFS, you can mount the HDFS cluster as a volume on client machines and have native command line, scripts or file explorer UI to view HDFS files and load data into HDFS.  NFS thus enables file-based applications to perform file read and write operations directly to Hadoop. This greatly simplifies data management in Hadoop and expands the integration of Hadoop into existing toolsets.

Network File System (NFS) is a distributed file system protocol that allows access to files on a remote computer in a manner similar to how local file system is accessed.  With a NFS gateway for Hadoop, files can now be browsed, downloaded and written to and from HDFS as if it is local file system.

We are excited to work with the community to enable a robust roadmap for NFS functionality, focussing on the following capabilities:

* NFSv4 and other protocols for access to HDFS
* Highly Available NFS Gateway
* Secure Hadoop (Kerberos) integration

NFS gateway uses DFSClient to access HDFS. Recently there is some performance improvement and comparison with FUSE



**Configure settings for the HDFS NFS gateway:**

NFS gateway uses the same configurations as used by the NameNode and DataNode. Configure the following three properties based on your application's requirement:

Edit the hdfs-default.xml file on your NFS gateway machine and modify the following property:

<property>  
 <name>dfs.access.time.precision</name>  
 <value>3600000</value>  
 <description>The access time for HDFS file is precise upto this value.   
 The default value is 1 hour. Setting a value of 0 disables  
 access times for HDFS.  
 </description>  
</property>

Update the following property to hdfs-site.xml:

<property>      
 <name>dfs.datanode.max.xcievers</name>      
 <value>1024</value>   
</property>

Add the following property to hdfs-site.xml:

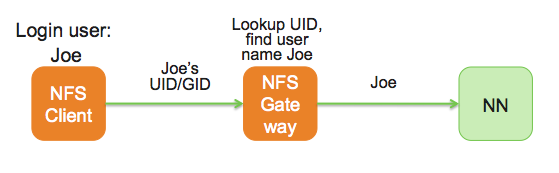
<property>      
 <name>dfs.nfs3.dump.dir</name>      
 <value>/tmp/.hdfs-nfs</value>   
</property>

**Start NFS gateway service.**

Three daemons are required to provide NFS service: rpcbind (or portmap), mountd and nfsd. The NFS gateway process has both nfsd and mountd.

It shares the HDFS root "/" as the only export. It is recommended to use the portmap included in NFS gateway package

The following illustrates how the UID and name are communicated between the NFS client, NFS gateway, and NameNode.



**Hadoop archives**

Hadoop archives are special format archives. A Hadoop archive maps to a file system directory.

A Hadoop archive always has a \*.har extension.

A Hadoop archive directory contains metadata (in the form of \_index and \_masterindex) and data (part-\*) files. The \_index file contains the name of the files that are part of the archive and the location within the part files.

Usage: hadoop archive -archiveName name <src>\* <dest>

-archiveName is the name of the archive you would like to create.

An example would be foo.har. The name should have a \*.har extension. The inputs are file system pathnames which work as usual with regular expressions. The destination directories (multiple directories) would contain the archive. Note that this is a Map/Reduce job that creates the archives. You would need a map reduce cluster to run this. The following is an example:

hadoop archive -archiveName foo.har /user/hadoop/dir1 /user/hadoop/dir2 /user/zoo/

In the above example /user/hadoop/dir1 and /user/hadoop/dir2 will be archived in the following file system directory -- /user/zoo/foo.har. The sources are not changed or removed when an archive is created.

Also, note that archives are immutable. So, rename's, deletes and creates return an error.

Many files can be archived in one file