
cloudera

Hadoop Deployment

Installation, Configuration, Maintenance

Cloudera Inc.
October 2, 2009

Overview

- Installing Hadoop
- Configuration Parameters
- Maintenance Tasks
- Upgrading Software

Installing Hadoop

1. Setup an automated installation method
   - RedHat Linux's Kickstart
   - Debian's Fully Automatic Installation
2. Download the Cloudera distribution into your package manager:
   - Instructions on http://archive.cloudera.com for yum and apt-get.
3. Install Sun Java 6 or later
4. V18:
   - Run the Cloudera Configurator (https://my.cloudera.com)
   - Install the configured distribution

   `# yum install hadoop-0.18 -y` or `# apt-get -y install hadoop-0.18`
5. V20:
   - Install the configured distribution
   - Make required configuration changes
6. Test the installation. Sample data and job:

   `http://archive.cloudera.com/docs/_installing_hadoop_standalone_mode.html`
7. Create an HDFS home directory for each user
8. Configure SSH to allow Hadoop user to login without a password

Deploying the Configuration

- To deploy a custom configuration to your entire cluster:
  - Push the `/etc/hadoop-0.20/conf.my_cluster` directory to all nodes in your cluster
  - Set **`alternatives`** rules on all nodes to activate your config
  - Restart all the services on your cluster
- (Later: what you need to set/edit in the configuration)

Format the Namenode

- Make sure to format the namenode as user **`hadoop`**.
- If you install the **`hadoop-conf-pseudo`** package, it will format the namenode as user **`hadoop`** for you.
- To format the namenode manually:
  - **`su -s /bin/bash - hadoop -c 'hadoop namenode –format'`**

Start Daemons

- Start Daemons (e.g., **`service hadoop-0.20-namenode start)`**
- Look for signs of life
  - Do a DFS benchmark (Copy files into/out of HDFS)
  - Run an example job
    - Copy files into Hadoop for input:

```
$ hadoop fs –put /etc/hadoop-0.20/conf/*.xml input
```

    - Run the example ‘grep’ job:

```
$ hadoop jar /usr/lib/hadoop-0.20/hadoop-*-examples.jar \
  grep input output 'dfs[a-z.]+'
```

    - View the output:

```
$ hadoop fs -cat output/part-00000 | head
```

- Make sure secondary namenode is working
  - Check that **`${fs.checkpoint.dir}`** fills with data after an hour or so.

The Configuration Files

- Live in /etc/hadoop-0.20/conf
  - Primary configuration files are written in XML
  - Separate configuration files for MapReduce, HDFS, and “core”
    - mapred-site.xml, hdfs-site.xml, core-site.xml
- Hadoop 0.18 combined all of these into “hadoop-site.xml”

Example Configuration File

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl“
    href="configuration.xsl"?>
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:8021</value>
  </property>
</configuration>
```

Parameter Priority Order

- MapReduce job-specific properties are set by (in priority order):
  - Values explicitly set in the JobConf object for a MapReduce job
  - values from the hadoop-site.xml on the client machine
  - values from the hadoop-site.xml on the tasktracker node
- Values at the end of this list can be forced to override values ahead of them if they're marked as "final" in the configuration file:

```
<property>
  <name>some.final.argument</name>
  <value>somevalue</value>
  <final>true</final>
</property>
```

Recommended Parameters (HDFS)

dfs.block.size	The block size for new files. Set in bytes. Default: 67108864 (64MB); Recommended: 134217728 bytes (128 MB)
dfs.data.dir	Where on the local filesystem a datanode stores its blocks. If this is a comma-delimited list of directories (no spaces between the comma and the path), data will be stored in all named directories.
dfs.datanode.du.reserved	The amount of free space (within `du.pct`) which HDFS should refuse to occupy (in bytes). Recommended: 10 GB
dfs.datanode.handler.count	Number of server threads on the datanode. (Default: 3; Recommended: 3—5)
dfs.name.dir	Where on the local filesystem a namenode stores its data.
dfs.namenode.handler.count	Number of server threads on the namenode.
dfs.permissions	If yes use the hadoop permissions system. If no, permission *checking* is turned off, but all other behavior is unchanged. Regardless of whether permissions are on or off, chmod, chgrp and chown *always* check permissions. This value is read once by the namenode at startup.
dfs.replication	How many machines a single file should be replicated to before it becomes available. (Default: 3)

Recommended Parameters (HDFS)

fs.checkpoint.dir	The location of the storage directory created by the secondary namenode.
fs.default.name	URI of namenode. (Example: hdfs://someserver.foo.com:8020/)
fs.trash.interval	Number of minutes between trash checkpoints. If zero, the trash feature is disabled. At a “checkpoint,” Hadoop moves files to subdirectory of their home directory named "`.Trash`". Files are initially moved to a `current` sub-directory of the trash directory. At the interval set in this property, the system makes a checkpoint of the current trash and removes older checkpoints.

Recommended Parameters (Core)

hadoop.tmp.dir	A base for other temporary directories.
hadoop.rpc.socket.factory.class.default	Default SocketFactory to use. This parameter is expected to be formatted as "`package.FactoryClassName`". If users connect through a SOCKS proxy, you don't want their SocketFactory settings interfering with the socket factory associated with the actual daemons. Set this to empty; also, mark as **`<final>true</final>`**
hadoop.rpc.socket.factory.class. ClientProtocol	SocketFactory to use to connect to a DFS. If null or empty, the system uses `hadoop.rpc.socket.class.default`. This socket factory is also used by DFS Client to create sockets to datanodes. Set to empty and mark final.
hadoop.rpc.socket.factory.class. JobSubmissionProtocol	SocketFactory to use to connect to a Map/Reduce master (jobtracker). If null or empty, then use `hadoop.rpc.socket.class.default`. Set to empty and mark final.

Recommended Parameters (Core)

io.file.buffer.size	Determines how much data is buffered during read and write operations. Use a multiple of hardware page size, in bytes. (Default: 4096; Recommend more like 65536 (64 KB))
io.sort.mb	The total amount of buffer memory to use while sorting files, in megabytes. By default, gives each merge stream 1MB. We recommend a higher memory-limit (Default: 100; Recommended: 256 MB)
io.sort.factor	The number of streams to merge at once while sorting files. This determines the number of open file handles. We recommend more streams merged at once while sorting files (Default: 10; Recommended: 64)
io.compression.codecs	A list of the compression codec classes that can be used for compression/decompression. (Recommended: `org.apache.hadoop.io.compress.DefaultCodec`, `org.apache.hadoop.io.compress.GzipCodec`)
Also need to set fs.default.name in this file as well; should match value in hdfs-site.xml	

Recommended Parameters (MapReduce)

mapred.child.java.opts	Java options for the tasktracker child processes. This value is passed to Java and indicates the size of the memory allocation pool in bytes or the unit indicated. (Default: “-Xmx200m” (200MB); Recommended: based on your cluster).
mapred.child.ulimit	Enforces a per-process memory cap by specifying the maximum virtual memory, in KB, that is afforded to the MapReduce child program. This must be strictly greater than the heap size request in `mapred.child.java.opts`, because it must accommodate both the user heap and the memory requirements of the JVM itself. (No Default; Recommended: based on mapred.child.java.opts)
mapred.job.tracker	Host or IP and port of jobtracker. (`foo.com:8021`)
mapred.job.tracker.handler.count	The number of server threads for the jobtracker. This should be roughly 4% of the number of tasktracker nodes. Increasing this number (over the default) gives you more jobtracker server threads to handle RPCs from large number of tasktrackers. (Default: 10)
mapred.local.dir	The local directory where MapReduce stores intermediate data files. May be a comma-separated list of directories on different devices in order to spread disk I/O.
mapred.reduce.parallel.copies	The number of parallel transfers run by reduce during its copy phase. (Default 5; Recommended: roughly 1/2 of the number of nodes * number of mappers/node.)

Recommended Parameters (MapReduce)

mapred.reduce.tasks	The number of reduce tasks per job. typically set to the number of available nodes * reduce slots/node. (Default: 1)
mapred.map.tasks.speculative.execution	Allow multiple instances of some map tasks to be executed in parallel. (Default: true; Recommended: true)
mapred.reduce.tasks.speculative.execution	Allow multiple instances of some reduce tasks to be executed in parallel. (Default: true; Recommended: false)
mapred.tasktracker.map.tasks.maximum	The maximum number of map tasks that can run simultaneously by a tasktracker. The value should be a function of the number of cores and amount of RAM. (Default: 2)
mapred.tasktracker.reduce.tasks.maximum	The maximum number of reduce tasks that will be run simultaneously by a task tracker. The value should be a function of the number of cores and amount of RAM. (Default: 2)
mapred.jobtracker.taskScheduler	The class responsible for scheduling the tasks. (Recommended: org.apache.hadoop.mapred.FairScheduler)
mapred.fairscheduler.allocation.file	Location of the allocation file for the FairScheduler.

Recommended Parameters (MapReduce)

mapred.output.compression.type	How job outputs are to compressed as SequenceFiles. (Default: RECORD; Recommended: BLOCK)
tasktracker.http.threads	The number of worker threads allowed for the http server. The http server is used by reduces to fetch intermediate map-outputs. (Default: 40; Recommended: if your cluster is 20+ nodes, set this to 60 or 80)

Additional Configuration Files

- Several more configuration files in /etc/hadoop-0.20/conf:
  - hadoop-env.sh – Environment variables for Hadoop daemons
  - dfs/mapred include/exclude scripts – Controls who can connect to the namenode and jobtracker
  - masters, slaves – Host name lists for ssh control
  - hadoop-policy.xml – Access control policies
  - log4j.properties – Logging (covered in “Logging”)
  - fair-scheduler.xml – Scheduler (covered in “The Fair Scheduler”)
  - hadoop-metrics.properties – (covered in “Monitoring”)

Environment Setup (hadoop-env.sh)

- Sets environment variables necessary for Hadoop:
  - HADOOP_CLASSPATH
  - HADOOP_HEAPSIZE
  - HADOOP_LOG_DIR
  - HADOOP_PID_DIR
  - JAVA_HOME
- Sourced into all Hadoop control scripts

Host “Include” Files

- A fixed set of datanodes are allowed to connect to the namenode; a fixed set of tasktrackers are allowed to connect to the jobtracker
- The namenode and jobtracker each should have a whitelist of hosts allowed to connect, filename set in configuration:
  - mapred-site.xml: **`mapred.hosts`**
  - hdfs-site.xml: **`dfs.hosts`**
- Whitelist file contains a line for each datanode’s network address
- There are also **`mapred.hosts.exclude`** and **`dfs.hosts.exclude`** properties that are used for blacklists; these should be set to empty files.
  - Use is explained later

Note on Include File vs. Slaves File

- The "slaves" file is used by a script that starts daemons on other machines. It runs a loop over that file to find hosts on which it should start servers.
  - Less important in CDH, but useful for restarting services without rebooting nodes
- The host include file is a whitelist of machines that are allowed to act as datanodes. These are logically distinct concepts, though you may find them symlinked together in practice.

Finalizing Configuration Changes

- Stop and restart the cluster daemons to pull in changes

```
$ cd /usr/lib/hadoop-0.20
$ bin/stop-all.sh
$ bin/start-all.sh
```

- No need to restart datanodes if all you changed was namenode configuration
- stop-all, start-all need to be run from master node
  - Use start-mapred.sh, start-dfs.sh for separate masters

Other Configuration Steps?

- Firewall
- Network

Rack Topology Script

- Arguments are IP addresses / hostnames of slave nodes
- Must return hierarchical “rack id” for each, in the order of arguments
  - e.g., 10.2.40.5 → “/datactr1/rack40”
- How to access this information left to sysadmin
  - Flat file, database, hardcoded into script…
- Script filename controlled by **`topology.script.file.name`** in core-site.xml

Example Rack Script

```
#!/bin/bash


while [ ! -z "$1" ]; do
  addr=$1
  echo "SELECT rack FROM HOST_RACK WHERE host='$addr'" \
    | mysql rackmapping | tail -1
  shift
done
```

Maintaining Your Cluster

- Get familiar with the tools
- Monitor the right stuff
- Manage jobs
- Manage log files
- Balance the data load
- Add (or remove) nodes
- Run file system integrity checks
- Ensure proper backup
- Restoring from backups

Sysadmin Tools

- **HDFS**
  - dfsadmin — retrieves HDFS statistics, initiates safe mode, performs upgrades, and sets filesystem usage quotas
  - fsck — checks the health of HDFS
  - Datanode block scanner —webpage that lists errors found in blocks stored on a datanode

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

  - balancer — balances data blocks across datanodes
  - distcp — copies large amounts of data between distributed filesystems
- **MapReduce**
  - mradmin — change the service-level authorization configuration for the jobtracker without restarting the daemon:

    ```
    $ bin/hadoop mradmin -refreshServiceAcl
    ```

Hadoop Ports to Know: Web UIs for Users

One of your best tools is the **web app**, but you have to know where to look!

	Daemon	**Default Port**	**Configuration Parameter**
HDFS	Namenode	50070	dfs.http.address
	Datanodes	50075	dfs.datanode.http.address
	Secondary namenode	50090	dfs.secondary.http.address
	Backup/Checkpoint node	50105	dfs.backup.http.address
MR	Jobtracker	50030	mapred.job.tracker.http.address
	Tasktrackers	50060	mapred.task.tracker.http.address

Hadoop Ports to Know: Com Ports for Admin

Daemon	Default Port	Configuration Parameter	Protocol	Used for
Namenode	8020	fs.default.name	IPC: ClientProtocol	Filesystem metadata operations.
Datanode	50010	dfs.datanode.address	Custom Hadoop Xceiver: DataNode and DFSClient	DFS data transfer
Datanode	50075	dfs.datanode.ipc.address	IPC: InterDatanodeProtocol, ClientDatanodeProtocol, ClientProtocol	Block metadata operations and recovery
Backupnode	50100	dfs.backup.address	Same as namenode	HDFS Metadata Operations
Jobtracker	Ill-defined. (Common values are 8021, 9001, or 8012.)	mapred.job.tracker	IPC: JobSubmissionProtocol, InterTrackerProtocol	Job submission, task tracker heartbeats.
Tasktracker	127.0.0.1:0 (Binds to an unused local port.)	mapred.task.tracker.report.address	IPC: TaskUmbilicalProtocol	Communicating with child jobs.

Monitor the right stuff

- This is important enough that there's a whole lesson on monitoring

Manage Jobs

- Balance cluster resources across jobs, make sure one job doesn't monopolize cluster resources
- Another whole lesson later, on Fair Scheduler

Manage Log files

- Clearing these logs does not affect Hadoop's run time.
- Consider archiving the logs if they are of interest in the job-development process.
- Make sure you do not move or delete a file that is currently being written to by a running job.
- Hadoop daemon logs
  - Rotated daily (default) by log4j
- Job Configuration XML
  - not rotated. About the same size for each job submitted
- Job Statistics
  - not rotated. Size/growth depends on the number of tasks running in a job
- Standard Error / Standard Out
  - not rotated. Produced by calls in the application code
- log4j
  - not rotated. Produced by calls in the application code

Balance the Data Load

- Rebalancer utility reviews data block placement and adjusts blocks to ensure all disks are within X% utilization of each other
- Set the threshold utilization when you call the command
- When to rebalance?
  - Immediately after adding new nodes to the cluster
  - Regular maintenance every week or so
- Doesn't interfere with ongoing processing by default.
  - Control the bandwidth: **`dfs.balance.bandwidthPerSec`** in hdfs-site.xml
- You can start the balancer in the background by running:

```
/usr/lib/hadoop-0.20/bin/start-balancer.sh
```

Adding Nodes to the Cluster

1. Add the network addresses of the new nodes to the "include" file.
   - The "include" file is the file pointed to by the **`dfs.hosts`** and **`mapred.hosts`** properties.
2. Update the namenode with the new set of datanodes:

   ```
   $ hadoop dfsadmin -refreshNodes
   ```

3. Update the "slaves" file with the new nodes, so that they are included in future operations performed by the Hadoop control scripts.
4. Start the new datanode and tasktracker instances.
5. Restart the jobtracker
   - Currently, there is no command to refresh the set of permitted nodes in the jobtracker.
   - Consider setting mapred.jobtracker.restart.recover to "true" to make the jobtracker recover running jobs after a restart.
   - ... Or maybe don’t use mapred.hosts in the first place
6. Check that the new datanodes and tasktrackers appear in the web UI.
7. Balance the data across the cluster, including the new nodes.

Removing Nodes from the Cluster

1. Add the network addresses of the new nodes to the “exclude” file.
   - The “exclude” file is the file pointed to by the **`dfs.hosts.exclude`** and **`mapred.hosts.exclude`** properties.
2. Update the namenode with the new set of datanodes:

   ```
   $ hadoop dfsadmin -refreshNodes
   ```

3. Jobtracker UI shows the admin state change to "Decommission In Progress" for the datanodes being removed.
   - Datanodes start copying their blocks to other datanodes in the cluster.
   - When all the datanodes report their state as "Decommissioned, " then all the blocks have been replicated.
4. Shut down the decommissioned nodes.
5. Remove the nodes from the “include” file, and run:

   ```
   $ hadoop dfsadmin -refreshNodes
   ```

6. Remove the nodes from the "slaves" file.

Run file system integrity checks

- The **`fsck`** command
  - Best way to see missing or corrupt data blocks:

    ```
    $ hadoop fsck /
    ```

  - Set this command as a daily CRON job that emails the output to administrators.
  - Pick a low-usage time to run the check.
- dfsadmin –metasave
  - find specific missing blocks
  - lists replica information for each block
  - the line for a missing blocks contains the word "MISSING."

    ```
    $ hadoop dfsadmin -metasave
    ```

Ensure Proper Backup

- Hadoop Safeguards: Data vs metadata
- Namenode
- Datanodes

Hadoop Safeguards

- Inherent data storage redundancy
  - By default, Hadoop data redundancy is 3-fold
- Reduces the likelihood of losing data from individual nodes failing.
- Back up critical data should the redundancy not be enough
- Back up metadata in the cluster that is critical for restarting the cluster should the namenode or jobtracker nodes fail.

Backups for the Namenode

- The namenode stores the HDFS metadata in memory for fast lookup, including:
  - File names
  - Directory structures
  - Block locations
- The namenode also maintains on-disk data structures to ensure that metadata persists:
  - A snapshot of the in-memory metadata
  - Edit log of changes that have been made since the snapshot was last taken

Backups for the Namenode

- Metadata backup strategies:
  - Ensure that no metadata is lost in the event of a namenode failure:
    - configure `dfs.name.dir` such that it writes to several local disks and at least one NFS mount (or other remote file system).
  - Allow recovery from accidental data loss due to user error (such as a careless `hadoop fs -rmr /*`).
    - Use data from the secondary namenode
- To perform regular backups of the namenode metadata, query the following URLs:
  - Snapshot: http://<nn.domain.com>:50070/getimage?getimage=1
  - Edits log: http://<nn.domain.com>:50070/getimage?getedit=1
- LVM snapshots allow for more reliable backups of the snapshot and edit log.

Backups for Datanodes

- Metadata for each datanode
  - Namenode generates a unique namespace id for the instance.
  - Datanodes bind to this namespace id at first connection and establish a unique "storage ID"
  - If lost, the datanode cannot participate in an HDFS instance and the data blocks it stores cannot be reached.
  - You'll see "**`InconsistentFSStateException`**" error
- VERSION file in **`${dfs.data.dir}/current`** stores:
  - storage ID
  - version of Hadoop used to create the block files
- VERSION file
  - unique for each node, same across all disks on the node.
- To avoid missing or corrupted VERSION files:
  - For each Datanode, store a copy of **`${dfs.data.dir}/current/VERSION`** in a separate directory, possibly off-machine.
  - The VERSION files won't change once they're created.
  - If needed, restore the backed-up VERSION files and restart HDFS.

Recovering from a Namenode Failure

- Assuming you have backups of the edits log and snapshot of in-memory metadata, then...
- Stop all Hadoop daemons
- Place edits log and snapshot in the one of the values of **<dfs.name.dir>**:
  - Snapshot: **<dfs.name.dir>/current/fsimage**
  - Edits log: **<dfs.name.dir>/current/edits**
- Make sure **current/VERSION** exists
- Make sure all of the new files are owned by **hadoop:hadoop**

```
$ chown -R hadoop:hadoop /hdd1/name
```

- Restart Hadoop
- Verify that HDFS has been restored:

```
$ hadoop fs -ls
```

Recovery: Datanode Failure

- If nodes fail one-by-one
  - Hadoop data replication ensures that data blocks will never become inaccessible
- If a rack fails
  - if the cluster is rack-aware, blocks are replicated across multiple racks and you shouldn't have data loss
- If all the nodes in your cluster fail simultaneously
  - no amount of replication within the cluster will help
- If you have more failures than your **`dfs.replication`** setting
  - it is possible that you will have missing data blocks.
  - How do you know? Run **`fsck`** and look for "corrupt" blocks
  - If you can recover at least one, then the blocks become available at replication count 1. The namenode will trigger re-replication

Upgrading Software: When to upgrade?

Cloudera's "stable" and "testing" releases

- stable repository packages
  - are deemed ready for production clusters.
  - passed all unit tests, functional tests
  - a few months of "soak time" in production environments.
  - To ensure stability, we can't always put the latest features into our current stable release.
- testing repository packages
  - recommended for people who want more features.
  - pass unit and functional tests
  - Do not have the same "soak time" as our stable packages.
  - A testing release represents a work in progress that will eventually be promoted to "stable."
- CDH2 is our current testing release.
- You can expect a new stable and testing release about every quarter.
- Cloudera supports a stable release for at least one year after an alternative stable release is available.

Upgrading Software: Current Repositories

Repository	CDH Release	Released	Patched Source	Apt Repository	Yum Repository
Stable	CDH1	March 2009	/cdh/stable	/debian	/redhat/cdh/stable
Testing	CDH2	August 2009	/cdh/testing	/debian	/redhat/cdh/testing

Upgrading Software: Version Syntax

Full Package Version	Component	Branch	Base Version	Patch Level
hadoop-0.18-0.18.3+76	Hadoop	0.18	0.18.3	76
hadoop-0.20-0.20.0+45	hadoop	0.20	0.20.0	45
hadoop-0.18-0.18.3+76.3	hadoop	0.18	0.18.3	76.3
pig-0.4.0+14	pig	-	0.4.0	14

- The dot releases in the patch level ensure software continuity as packages are promoted from testing to stable.
- After a package is promoted, the major patch level will never change.

Migration to New Hardware

- If you need to move your working cluster to a new location such as a new data center
- Reserve IP addresses in the destination datacenter for the nodes to be moved.
  - These IP addresses can be different than the original IP addresses.
  - Bring up the namenode with the same namespace id in its file, the datanodes will connect to it.
  - If you change the hostname of the namenode, update **`fs.default.name`** on each datanode.
- Prepare for hardware failures.
  - Consider increasing the number of replicas of your important data before the move (default is 3):

```
$ hadoop fs -setrep -R 6 /
```

  - Change it back when the migration is complete.
  - Increasing the levels of replication may take several hours (or days)
- Backup critical data sets
  - Consider using distcp to copy essential datasets to a spare cluster before moving the nodes.

Using distcp

- Usage: **`$ hadoop distcp src dest`**
  - **`hadoop distcp hdfs://foo/d1 hdfs://bar/d1`**
- hdfs:// requires versions match
  - For cross-version transfers, use hftp:// on one end-point.
  - (Note: hftp is **not** “ftp for Hadoop”)
- Useful arguments:
  - -i – ignore errors
  - -update – update existing directory

Conclusions

- Sysadmins have important functions throughout cluster lifecycle
- Installation, physical migration most challenging parts
- Lots of data that needs backups; periodic checkups a good idea