Hortonworks Data Platform

Administration

(Nov 30, 2016)

docs.hortonworks.com

Hortonworks Data Platform: Administration

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain, free and open source.

For more information on Hortonworks technology, please visit the Hortonworks Data Platform page. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to contact us directly to discuss your specific needs.

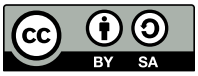

Table of Contents

1. Decommissioning Slave Nodes

Hadoop provides the decommission feature to retire a set of existing slave nodes (DataNodes, NodeManagers, or HBase RegionServers) in order to prevent data loss.

Slaves nodes are frequently decommissioned for maintainance. As a Hadoop administrator, you will decommission the slave nodes periodically in order to either reduce the cluster size or to gracefully remove dying nodes.

1.1. Prerequisites

- Ensure that the following property is defined in your `hdfs-site.xml` file.

```
<property>
 <name>dfs.hosts.exclude</name>
 <value><HADOOP_CONF_DIR>/dfs.exclude</value>
 <final>true</final>
 </property>
```

where <HADOOP_CONF_DIR> is the directory for storing the Hadoop configuration files. For example, `/etc/hadoop/conf`.

- Ensure that the following property is defined in your `yarn-site.xml` file.

```
<property>
 <name>yarn.resourcemanager.nodes.exclude-path</name>
 <value><HADOOP_CONF_DIR>/yarn.exclude</value>
 <final>true</final>
</property>
```

where <HADOOP_CONF_DIR> is the directory for storing the Hadoop configuration files. For example, `/etc/hadoop/conf`.

1.2. Decommission DataNodes or NodeManagers

Nodes normally run both a DataNode and a NodeManager, and both are typically commissioned or decommissioned together.

With the replication level set to three, HDFS is resilient to individual DataNodes failures. However, there is a high chance of data loss when you terminate DataNodes without decommissioning them first. Nodes must be decommissioned on a schedule that permits replication of blocks being decommissioned.

On the other hand, if a NodeManager is shut down, the ResourceManager will reschedule the tasks on other nodes in the cluster. However, decommissioning a NodeManager may be required in situations where you want a NodeManager to stop to accepting new tasks, or when the tasks take time to execute but you still want to be agile in your cluster management.

1.3. Decommission DataNodes

Use the following instructions to decommission DataNodes in your cluster:

- On the NameNode host machine, edit the <HADOOP_CONF_DIR>/dfs.exclude file and add the list of DataNodes hostnames (separated by a newline character).

  where <HADOOP_CONF_DIR> is the directory for storing the Hadoop configuration files. For example, /etc/hadoop/conf.

- Update the NameNode with the new set of excluded DataNodes. On the NameNode host machine, execute the following command:

```
su <HDFS_USER>
hdfs dfsadmin -refreshNodes
```

  where <HDFS_USER> is the user owning the HDFS services. For example, hdfs.

- Open the NameNode web UI (http://<NameNode_FQDN>:50070) and navigate to the DataNodes page. Check to see whether the state has changed to **Decommission In Progress** for the DataNodes being decommissioned.

- When all the DataNodes report their state as **Decommissioned** (on the DataNodes page, or on the Decommissioned Nodes page at http://<NameNode_FQDN>:8088/cluster/ nodes/decommissioned), all of the blocks have been replicated. You can then shut down the decommissioned nodes.

- If your cluster utilizes a dfs.include file, remove the decommissioned nodes from the <HADOOP_CONF_DIR>/dfs.include file on the NameNode host machine, then execute the following command:

```
su <HDFS_USER>
hdfs dfsadmin -refreshNodes
```

Note

If no dfs.include file is specified, all DataNodes are considered to be included in the cluster (unless excluded in the dfs.exclude file). The dfs.hosts and dfs.hosts.exclude properties in hdfs-site.xml are used to specify the dfs.include and dfs.exclude files.

1.4. Decommission NodeManagers

Use the following instructions to decommission NodeManagers in your cluster:

- On the NameNode host machine, edit the <HADOOP_CONF_DIR>/yarn.exclude file and add the list of NodeManager hostnames (separated by a newline character).

  where <HADOOP_CONF_DIR> is the directory for storing the Hadoop configuration files. For example, /etc/hadoop/conf.

- If your cluster utilizes a yarn.include file, remove the decommissioned nodes from the <HADOOP_CONF_DIR>/yarn.include file on the ResourceManager host machine.

Note

If no yarn.include file is specified, all NodeManagers are considered to be included in the cluster (unless excluded in the yarn.exclude

file). The `yarn.resourcemanager.nodes.include-path` and `yarn.resourcemanager.nodes.exclude-path` properties in `yarn-site.xml` are used to specify the `yarn.include` and `yarn.exclude` files.

- Update the ResourceManager with the new set of NodeManagers. On the ResourceManager host machine, execute the following command:

```
su <YARN_USER>
yarn rmadmin -refreshNodes
```

where <YARN_USER> is the user who owns the YARN services, for example, `yarn`.

1.5. Decommission HBase RegionServers

Use the following instruction to decommission HBase RegionServers in your cluster.

At the RegionServer that you want to decommission, execute:

```
su <HBASE_USER>
/usr/hdp/current/hbase-client/bin/hbase-daemon.sh stop
```

where <HBASE_USER> is the user who owns the HBase Services. For example, `hbase`.

RegionServer closes all the regions, then shuts down.

2. Manually Adding Slave Nodes to an HDP Cluster

This chapter contains the following sections:

2.1. Prerequisites

Important

The content in this section has not been updated for HDP 2.3. Please check back at a later date.

Ensure that the new slave nodes meet the following prerequisites:

- The following operating systems are supported:
  - 64-bit Red Hat Enterprise Linux (RHEL) 5 or 6
  - 64-bit CentOS 5 or 6
  - 64-bit SUSE Linux Enterprise Server (SLES) 11, SP1
- At each of your hosts:
  - yum (RHEL)
  - zypper (SLES)
  - rpm
  - scp
  - curl
  - wget
  - unzip
  - tar
  - pdsh
- Ensure that all of the ports listed in Configuring Ports are available.
- To install Hive metastore or to use an external database for Oozie metastore, ensure that you deploy either a MySQL or an Oracle database in your cluster. For instructions, see "Meet Minimum System Requirements" in the Installing HDP Manually guide.

- Your system must have the correct JDK installed on all of the nodes in the cluster. For further information, see "Meet Minimum System Requirements" in the Installing HDP Manually guide.

2.2. Add Slave Nodes

Use the following instructions to manually add a slave node:

- On each new slave node, configure the remote repository as described in "Installing ZooKeeper", in Installing HDP Manually.
- On each new slave node, install HDFS.
- On each new slave node, install compression libraries.
- On each new slave node, create the DataNode and YARN NodeManager local directories.
- Copy the Hadoop configurations to the new slave nodes and set appropriate permissions.
  - **Option I:** Copy Hadoop config files from an existing slave node.
    - On an existing slave node, make a copy of the current configurations:

      ```
      tar zcvf hadoop_conf.tgz /etc/hadoop/conf
      ```

    - Copy this file to each of the new nodes:

      ```
      rm -rf /etc/hadoop/conf
      cd /
      tar zxvf $location_of_copied_conf_tar_file/hadoop_conf.tgz
      chmod -R 755 /etc/hadoop/confa
      ```

- On each of the new slave nodes, start the NodeManager:

  ```
  su -l yarn -c "/usr/hdp/current/hadoop-yarn-nodemanager/sbin/yarn-daemon.sh
   start nodemanager"
  ```

- Optional - If you use a HDFS or YARN/ResourceManager `.include` file in your cluster, add the new slave nodes to the `.include` file, then run the applicable `refreshNodes` command.
  - To add new DataNodes to the dfs.include file:
    - On the NameNode host machine, edit the `/etc/hadoop/conf/dfs.include` file and add the list of the new slave node host names (separated by newline character).

      

      Note

      If no dfs.include file is specified, all DataNodes are considered to be included in the cluster (unless excluded in the dfs.exclude file). The `dfs.hosts` and `dfs.hosts.exlude` properties in `hdfs-site.xml` are used to specify the `dfs.include` and `dfs.exclude` files.

    - On the NameNode host machine, execute the following command:

```
su -l hdfs -c "hdfs dfsadmin -refreshNodes"
```

- To add new NodeManagers to the `yarn.include` file:
  - On the ResourceManager host machine, edit the `/etc/hadoop/conf/yarn.include` file and add the list of the slave node host names (separated by newline character).

Note

If no `yarn.include` file is specified, all NodeManagers are considered to be included in the cluster (unless excluded in the `yarn.exclude` file). The `yarn.resourcemanager.nodes.include-path` and `yarn.resourcemanager.nodes.exclude-path` properties in yarn-site.xml are used to specify the `yarn.include` and `yarn.exclude` files.

  - On the ResourceManager host machine, execute the following command:

```
su -l yarn -c "yarn rmadmin -refreshNodes"
```

2.3. Add HBase RegionServer

Use the following instructions to manually add HBase RegionServer hosts:

- On each of the new slave nodes, install HBase and ZooKeeper.
  - For RHEL/CentOS/Oracle Linux:

```
yum install zookeeper hbase
```

  - For SLES:

```
zypper install zookeeper hbase
```

- On each of the new slave nodes, add the HDP repository to yum:
  - RHEL/CentOS/Oracle Linux 6.x:

```
wget -nv http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.3.0.0/hdp.repo -O /etc/yum.repos.d/hdp.repo
```

  - RHEL/CentOS/Oracle Linux 7.x:

```
wget -nv http://public-repo-1.hortonworks.com/HDP/centos7/2.x/updates/2.3.0.0/hdp.repo -O /etc/yum.repos.d/hdp.repo
```

  - SLES SP3/SP4:

```
wget -nv http://public-repo-1.hortonworks.com/HDP/suse11sp3/2.x/updates/2.3.0.0/hdp.repo -O /etc/zypp/repos.d/hdp.repo
```

- Copy the HBase configurations to the new slave nodes and set appropriate permissions.
  - **Option I:** Copy HBase config files from an existing slave node.

- On any existing slave node, make a copy of the current configurations:

```
tar zcvf hbase_conf.tgz /etc/hbase/conf
tar zcvf zookeeper_conf.tgz /etc/zookeeper/conf
```

- Copy these files to each of the new nodes:

```
rm -rf /etc/hbase/conf
mkdir -p /etc/hbase/conf
cd /
tar zxvf $location_of_copied_conf_tar_file/hbase_conf.tgz
chmod -R 755 /etc/hbase/conf
```

```
rm -rf /etc/zookeeper/conf
mkdir -p /etc/zookeeper/conf
cd /
tar zxvf $location_of_copied_conf_tar_file/zookeeper_conf.tgz
chmod -R 755 /etc/zookeeper/conf
```

- **Option II:** Manually add Hadoop configuration files as described in "Set Up the Configuration Files", in Installing HDP Manually.

- On all of the new slave nodes, create the configuration directory, copy all of the configuration files, and set the permissions:

```
rm -r $HBASE_CONF_DIR ;
mkdir -p $HBASE_CONF_DIR ;
```

Copy all of the configuration files to $HBASE_CONF_DIR

```
chmod a+x $HBASE_CONF_DIR/;
chown -R $HBASE_USER:$HADOOP_GROUP $HBASE_CONF_DIR/../ ;
chmod -R 755 $HBASE_CONF_DIR/../
```

```
rm -r $ZOOKEEPER_CONF_DIR ;
mkdir -p $ZOOKEEPER_CONF_DIR ;
```

Copy all of the configuration files to $ZOOKEEPER_CONF_DIR

```
chmod a+x $ZOOKEEPER_CONF_DIR/;
chown -R $ZOOKEEPER_USER:$HADOOP_GROUP $ZOOKEEPER_CONF_DIR/../ ;
chmod -R 755 $ZOOKEEPER_CONF_DIR/../
```

where:

- *$HBASE_CONF_DIR* is the directory to store the HBase configuration files. For example, /etc/hbase/conf.

- *$HBASE_USER* is the user owning the HBase services. For example, hbase.

- *$HADOOP_GROUP* is a common group shared by services. For example, hadoop.

- *$ZOOKEEPER_CONF_DIR* is the directory to store the ZooKeeper configuration files. For example, /etc/zookeeper/conf

- *$ZOOKEEPER_USER* is the user owning the ZooKeeper services. For example, zookeeper.

- Start HBase RegionServer node:

```
<login as $HBASE_USER>
/usr/lib/hbase/bin/hbase-daemon.sh --config $HBASE_CONF_DIR start
 regionserver
```

- On the HBase Master host machine, edit the `/usr/lib/hbase/conf` file and add the list of slave nodes' hostnames. The hostnames must be separated by a newline character.

3. Using DistCp to Copy Files

Hadoop DistCp (distributed copy) can be used to copy data between Hadoop clusters (and also within a Hadoop cluster). DistCp uses MapReduce to implement its distribution, error handling, and reporting. It expands a list of files and directories into map tasks, each of which copies a partition of the files specified in the source list.

3.1. Using DistCp

The most common use of DistCp is an inter-cluster copy:

```
hadoop distcp hdfs://nn1:8020/source hdfs://nn2:8020/destination
```

Where `hdfs://nn1:8020/source` is the data source, and `hdfs://nn2:8020/` destination is the destination. This will expand the name space under /source on NameNode "nn1" into a temporary file, partition its contents among a set of map tasks, and start copying from "nn1" to "nn2". Note that DistCp requires absolute paths.

You can also specify multiple source directories:

```
hadoop distcp hdfs://nn1:8020/source/a hdfs://nn1:8020/source/b hdfs://
 nn2:8020/destination
```

Or specify multiple source directories from a file with the `-f` option:

```
hadoop distcp -f hdfs://nn1:8020/srclist hdfs://nn2:8020/destination
```

Where `srclist` contains:

```
hdfs://nn1:8020/source/a
hdfs://nn1:8020/source/b
```

DistCp from HDP-1.3.x to HDP-2.x

When using DistCp to copy from a HDP-1.3.x cluster to a HDP-2.x cluster, the format is:

```
hadoop distcp hftp://<hdp 1.3.x namenode host>:50070/<folder path of source>
 hdfs://<hdp 2.x namenode host>/<folder path of target>
```

Here is an example of a DistCp copy from HDP 1.3.0 to HDP-2.0:

```
hadoop distcp hftp://namenodehdp130.test.com:50070/apps/hive/warehouse/db/
 hdfs://namenodehdp20.test.com/data/raw/
```

When copying from multiple sources, DistCp will abort the copy with an error message if two sources collide, but collisions at the destination are resolved based on the options specified. By default, files already existing at the destination are skipped (i.e. not replaced by the source file). A count of skipped files is reported at the end of each job, but it may be inaccurate if a copier failed for some subset of its files, but succeeded on a later attempt.

It is important that each NodeManager is able to communicate with both the source and destination file systems. For HDFS, both the source and destination must be running the same version of the protocol, or use a backwards-compatible protocol; see "Copying Between Versions".

After a copy, you should generate and cross-check a listing of the source and destination to verify that the copy was truly successful. Since DistCp employs both Map/Reduce and the FileSystem API, issues in or between any of these three could adversely and silently affect the copy. Some have had success running with -update enabled to perform a second pass, but users should be acquainted with its semantics before attempting this.

It is also worth noting that if another client is still writing to a source file, the copy will likely fail. Attempting to overwrite a file being written at the destination should also fail on HDFS. If a source file is (re)moved before it is copied, the copy will fail with a FileNotFound exception.

3.2. Command Line Options

For a description of DistCp command line options, see DistCp Command Line Options.

3.3. Update and Overwrite

The DistCp `-update` option is used to copy files from a source that do not exist at the target, or that have different contents. The DistCp `-overwrite` option overwrites target files even if they exist at the source, or if they have the same contents.

The `-update` and `-overwrite` options warrant further discussion, since their handling of source-paths varies from the defaults in a very subtle manner.

Consider a copy from `/source/first/` and `/source/second/` to `/target/`, where the source paths have the following contents:

```
hdfs://nn1:8020/source/first/1
hdfs://nn1:8020/source/first/2
hdfs://nn1:8020/source/second/10
hdfs://nn1:8020/source/second/20
```

When DistCp is invoked without `-update` or `-overwrite`, the DistCp defaults would create directories `first/` and `second/`, under `/target`. Thus:

```
distcp hdfs://nn1:8020/source/first hdfs://nn1:8020/source/second hdfs://
nn2:8020/target
```

would yield the following contents in /target:

```
hdfs://nn2:8020/target/first/1
hdfs://nn2:8020/target/first/2
hdfs://nn2:8020/target/second/10
hdfs://nn2:8020/target/second/20
```

When either `-update` or `-overwrite` is specified, the **contents** of the source directories are copied to the target, and not the source directories themselves. Thus:

```
distcp -update hdfs://nn1:8020/source/first hdfs://nn1:8020/source/second
 hdfs://nn2:8020/target
```

would yield the following contents in `/target`:

```
hdfs://nn2:8020/target/1
```

```
hdfs://nn2:8020/target/2
hdfs://nn2:8020/target/10
hdfs://nn2:8020/target/20
```

By extension, if both source folders contained a file with the same name ("0", for example), then both sources would map an entry to `/target/0` at the destination. Rather than permit this conflict, DistCp will abort.

Now, consider the following copy operation:

```
distcp hdfs://nn1:8020/source/first hdfs://nn1:8020/source/second hdfs://
nn2:8020/target
```

With sources/sizes:

```
hdfs://nn1:8020/source/first/1 32
hdfs://nn1:8020/source/first/2 32
hdfs://nn1:8020/source/second/10 64
hdfs://nn1:8020/source/second/20 32
```

And destination/sizes:

```
hdfs://nn2:8020/target/1 32
hdfs://nn2:8020/target/10 32
hdfs://nn2:8020/target/20 64
```

Will effect:

```
hdfs://nn2:8020/target/1 32
hdfs://nn2:8020/target/2 32
hdfs://nn2:8020/target/10 64
hdfs://nn2:8020/target/20 32
```

`1` is skipped because the file-length and contents match. `2` is copied because it doesn't exist at the target. `10` and `20` are overwritten because the contents don't match the source.

If the `-update` option is used, `1` is overwritten as well.

3.4. DistCp and Security Settings

Security settings dictate whether DistCp should be run on the source cluster or the destination cluster. The general rule-of-thumb is that if one cluster is secure and the other is not secure, DistCp should be run from the secure cluster -- otherwise there may be security-related issues.

When copying data from a secure cluster to an non-secure cluster, the following configuration setting is required for the DistCp client:

```
<property>
 <name>ipc.client.fallback-to-simple-auth-allowed</name>
 <value>true</value>
</property>
```

When copying data from a secure cluster to a secure cluster, the following configuration setting is required in the `core-site.xml` file:

```
<property>
```

```
    <name>hadoop.security.auth_to_local</name>
    <value></value>
    <description>Maps kerberos principals to local user names</description>
</property>
```

3.5. Secure-to-Secure: Kerberos Principal Name

- `distcp hdfs://hdp-2.0-secure hdfs://hdp-2.0-secure` One issue here is that the SASL RPC client requires that the remote server's Kerberos principal must match the server principal in its own configuration. Therefore, the same principal name must be assigned to the applicable NameNodes in the source and the destination cluster. For example, if the Kerberos principal name of the NameNode in the source cluster is nn/host1@realm, the Kerberos principal name of the NameNode in destination cluster must be nn/host2@realm, rather than nn2/host2@realm, for example.

3.6. Secure-to-Secure: ResourceManager Mapping Rules

When copying between two HDP2 secure clusters, or from HDP1 secure to HDP2 secure, further ResourceManager (RM) configuration is required if the two clusters have different realms. In order for DistCP to succeed, the same RM mapping rule must be used in both clusters.

For example, if secure Cluster 1 has the following RM mapping rule:

```
<property>
    <name>hadoop.security.auth_to_local</name>
    <value>
      RULE:[2:$1@$0](rm@.*SEC1.SUP1.COM)s/.*/yarn/
      DEFAULT
    </value>
</property>
```

And secure Cluster 2 has the following RM mapping rule:

```
<property>
    <name>hadoop.security.auth_to_local</name>
    <value>
      RULE:[2:$1@$0](rm@.*BA.YISEC3.COM)s/.*/yarn/
      DEFAULT
    </value>
</property>
```

The DistCp job from Cluster 1 to Cluster 2 will fail because Cluster 2 cannot resolve the RM principle of Cluster 1 correctly to the yarn user, because the RM mapping rule in Cluster 2 is different than the RM mapping rule in Cluster 1.

The solution is to use the same RM mapping rule in both Cluster 1 and Cluster 2:

```
<property>
    <name>hadoop.security.auth_to_local</name>
    <value>
      RULE:[2:$1@$0](rm@.*SEC1.SUP1.COM)s/.*/yarn/
      RULE:[2:$1@$0](rm@.*BA.YISEC3.COM)s/.*/yarn/
```

```
      DEFAULT
    </value>
  </property>
```

3.7. DistCp Between HA Clusters

To copy data between HA clusters, use the `dfs.internal.nameservices` property in the `hdfs-site.xml` file to explicitly specify the name services belonging to the local cluster, while continuing to use the `dfs.nameservices` property to specify all of the name services in the local and remote clusters.

Use the following steps to copy data between HA clusters:

Modify the following properties in the `hdfs-site.xml` file for both cluster A and cluster B:

1. Add both name services to `dfs.nameservices` = HAA, HAB
2. Add the `dfs.internal.nameservices` property:
   - In cluster A:

     ```
     dfs.internal.nameservices = HAA
     ```

   - In cluster B:

     ```
     dfs.internal.nameservices = HAB
     ```

3. Add `dfs.ha.namenodes.<nameservice>` to both clusters:
   - In cluster A

     ```
     dfs.ha.namenodes.HAB = nn1,nn2
     ```

   - In cluster B

     ```
     dfs.ha.namenodes.HAA = nn1,nn2
     ```

4. Add the `dfs.namenode.rpc-address.<cluster>.<nn>` property:
   - In Cluster A:

     ```
     dfs.namenode.rpc-address.HAB.nn1 = <NN1_fqdn>:8020

     dfs.namenode.rpc-address.HAB.nn2 = <NN2_fqdn>:8020
     ```

   - In Cluster B:

     ```
     dfs.namenode.rpc-address.HAA.nn1 = <NN1_fqdn>:8020

     dfs.namenode.rpc-address.HAA.nn2 = <NN2_fqdn>:8020
     ```

5. Add the `dfs.client.failover.proxy.provider.<cluster>` property:
   - In cluster A:

```
dfs.client.failover.proxy.provider. HAB =
org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
```

- In cluster B:

```
dfs.client.failover.proxy.provider. HAA =
org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
```

6. Restart the HDFS service, then run the `distcp` command using the NameService. For example:

```
hadoop distcp hdfs://falconG/tmp/testDistcp hdfs://falconE/tmp/
```

3.8. DistCp and HDP Version

The HDP version of the source and destination clusters can determine which type of file systems should be used to read the source cluster and write to the destination cluster.

For example, when copying data from a 1.x cluster to a 2.x cluster, it is impossible to use "hdfs" for both the source and the destination, because HDP 1.x and 2.x have different RPC versions, and the client cannot understand both at the same time. In this case the WebHdfsFilesystem (webhdfs://) can be used in both the source and destination clusters, or the HftpFilesystem (hftp://) can be used to read data from the source cluster.

3.9. DistCp Data Copy Matrix: HDP1/HDP2 to HDP2

The following table provides a summary of configuration, settings and results when using DistCp to copy data from HDP1 and HDP2 clusters to HDP2 clusters.

From	To	Source Configuration	Destination Configuration	DistCp Should be Run on...	Result
HDP 1.3	HDP 2.x	insecure + hdfs	insecure + webhdfs	HDP 1.3 (source)	success
HDP 1.3	HDP 2.x	secure + hdfs	secure + webhdfs	HDP 1.3 (source)	success
HDP 1.3	HDP 2.x	secure + hftp	secure + hdfs	HDP 2.x (destination)	success
HDP 1.3	HDP 2.1	secure + hftp	secure + swebhdfs	HDP 2.1 (destination)	success
HDP 1.3	HDP 2.x	secure + hdfs	insecure + webhdfs	HDP 1.3 (source)	Possible issues are discussed here.
HDP 2.x	HDP 2.x	secure + hdfs	insecure + hdfs	secure HDP 2.x (source)	success
HDP 2.x	HDP 2.x	secure + hdfs	secure + hdfs	either HDP 2.x (source or destination)	success
HDP 2.x	HDP 2.x	secure + hdfs	secure + webhdfs	HDP 2.x (source)	success
HDP 2.x	HDP 2.x	secure + hftp	secure + hdfs	HDP 2.x (destination)	success

For the above table:

- The term "secure" means that Kerberos security is set up.
- HDP 2.x means HDP 2.0, HDP 2.1, HDP 2.2, and HDP 2.3.
- hsftp is available in both HDP-1.x and HDP-2.x. It adds https support to hftp.

3.10. Copying Data from HDP-2.x to HDP-1.x Clusters

Copying Data from HDP-1.x to HDP-2.x Clusters is also supported, however, HDP-1.x is not aware of a new checksum introduced in HDP-2.x.

To copy data from HDP-2.x to HDP-1.x:

- Skip the checksum check during source 2.x –> 1.x.
- -or-
- Ensure that the file to be copied is in CRC32 before distcp 2.x –> 1.x.

3.11. DistCp Architecture

DistCp is comprised of the following components:

- Distcp Driver
- Copy Listing Generator
- InputFormats and MapReduce Components

3.12. DistCp Driver

The DistCp Driver components are responsible for:

- Parsing the arguments passed to the DistCp command on the command-line, via:
- OptionsParser
- DistCpOptionsSwitch

Assembling the command arguments into an appropriate DistCpOptions object, and initializing DistCp. These arguments include:

- Source-paths
- Target location
- Copy options (e.g. whether to update-copy, overwrite, which file attributes to preserve, etc.)

Orchestrating the copy operation by:

- Invoking the copy-listing generator to create the list of files to be copied.
- Setting up and launching the Hadoop MapReduce job to carry out the copy.
- Based on the options, either returning a handle to the Hadoop MapReduce job immediately, or waiting until completion.

The parser elements are executed only from the command-line (or if DistCp::run() is invoked). The DistCp class may also be used programmatically, by constructing the DistCpOptions object and initializing a DistCp object appropriately.

3.13. Copy-listing Generator

The copy-listing generator classes are responsible for creating the list of files/directories to be copied from source. They examine the contents of the source paths (files/directories, including wildcards), and record all paths that need copying into a SequenceFile for consumption by the DistCp Hadoop Job. The main classes in this module include:

- **CopyListing**: The interface that should be implemented by any copy-listing generator implementation. Also provides the factory method by which the concrete CopyListing implementation is chosen.
- **SimpleCopyListing**: An implementation of CopyListing that accepts multiple source paths (files/directories), and recursively lists all of the individual files and directories under each for copy.
- **GlobbedCopyListing**: Another implementation of CopyListing that expands wildcards in the source paths.
- **FileBasedCopyListing**: An implementation of CopyListing that reads the source path list from a specified file.

Based on whether a source file list is specified in the DistCpOptions, the source listing is generated in one of the following ways:

- If there is no source file list, the GlobbedCopyListing is used. All wildcards are expanded, and all of the expansions are forwarded to the SimpleCopyListing, which in turn constructs the listing (via recursive descent of each path).
- If a source file list is specified, the FileBasedCopyListing is used. Source paths are read from the specified file, and then forwarded to the GlobbedCopyListing. The listing is then constructed as described above.

You can customize the method by which the copy-listing is constructed by providing a custom implementation of the CopyListing interface. The behaviour of DistCp differs here from the legacy DistCp, in how paths are considered for copy.

The legacy implementation only lists those paths that must definitely be copied on to the target. E.g., if a file already exists at the target (and -overwrite isn't specified), the file is not even considered in the MapReduce copy job. Determining this during setup (i.e. before

the MapReduce Job) involves file size and checksum comparisons that are potentially time consuming.

DistCp postpones such checks until the MapReduce job, thus reducing setup time. Performance is enhanced further since these checks are parallelized across multiple maps.

3.14. InputFormats and MapReduce Components

The InputFormats and MapReduce components are responsible for the actual copying of files and directories from the source to the destination path. The listing file created during copy-listing generation is consumed at this point, when the copy is carried out. The classes of interest here include:

- **UniformSizeInputFormat:** This implementation of org.apache.hadoop.mapreduce.InputFormat provides equivalence with Legacy DistCp in balancing load across maps. The aim of the UniformSizeInputFormat is to make each map copy roughly the same number of bytes. Therefore, the listing file is split into groups of paths, such that the sum of file sizes in each InputSplit is nearly equal to every other map. The splitting is not always perfect, but its trivial implementation keeps the setup time low.

- **DynamicInputFormat and DynamicRecordReader:** The DynamicInputFormat implements org.apache.hadoop.mapreduce.InputFormat, and is new to DistCp. The listing file is split into several "chunk files", the exact number of chunk files being a multiple of the number of maps requested for in the Hadoop Job. Each map task is "assigned" one of the chunk files (by renaming the chunk to the task's id), before the Job is launched. Paths are read from each chunk using the DynamicRecordReader, and processed in the CopyMapper. After all of the paths in a chunk are processed, the current chunk is deleted and a new chunk is acquired. The process continues until no more chunks are available. This "dynamic" approach allows faster map tasks to consume more paths than slower ones, thus speeding up the DistCp job overall.

- **CopyMapper:** This class implements the physical file copy. The input paths are checked against the input options (specified in the job configuration), to determine whether a file needs to be copied. A file will be copied only if at least one of the following is true:

  - A file with the same name does not exist at target.

  - A file with the same name exists at target, but has a different file size.

  - A file with the same name exists at target, but has a different checksum, and -skipcrccheck is not mentioned.

  - A file with the same name exists at target, but -overwrite is specified.

  - A file with the same name exists at target, but differs in block-size (and block-size needs to be preserved).

- **CopyCommitter:** This class is responsible for the commit phase of the DistCp job, including:

  - Preservation of directory permissions (if specified in the options)

- Clean up of temporary files, work directories, etc.

3.15. DistCp Frequently Asked Questions

- **Why does -update not create the parent source directory under a pre-existing target directory?** The behavior of -update and -overwrite is described in detail in the Using DistCp section of this document. In short, if either option is used with a pre-existing destination directory, the contents of each source directory are copied over, rather than the source directory itself. This behavior is consistent with the legacy DistCp implementation.
- **How does the new DistCp (version 2) differ in semantics from the legacy DistCp?**
  - Files that are skipped during copy previously also had their file-attributes (permissions, owner/group info, etc.) unchanged, when copied with Legacy DistCp. These are now updated, even if the file copy is skipped.
  - In Legacy DistCp, empty root directories among the source path inputs were not created at the target. These are now created.
- **Why does the new DistCp (version 2) use more maps than legacy DistCp?** Legacy DistCp works by figuring out what files need to be actually copied to target before the copy job is launched, and then launching as many maps as required for copy. So if a majority of the files need to be skipped (because they already exist, for example), fewer maps will be needed. As a consequence, the time spent in setup (i.e. before the MapReduce job) is higher. The new DistCp calculates only the contents of the source paths. It doesn't try to filter out what files can be skipped. That decision is put off until the MapReduce job runs. This is much faster (vis-a-vis execution-time), but the number of maps launched will be as specified in the -m option, or 20 (the default) if unspecified.
- **Why does DistCp not run faster when more maps are specified?** At present, the smallest unit of work for DistCp is a file. i.e., a file is processed by only one map. Increasing the number of maps to a value exceeding the number of files would yield no performance benefit. The number of maps launched would equal the number of files.
- **Why does DistCp run out of memory?** If the number of individual files/directories being copied from the source path(s) is extremely large (e.g. 1,000,000 paths), DistCp might run out of memory while determining the list of paths for copy. This is not unique to the new DistCp implementation. To get around this, consider changing the -Xmx JVM heap-size parameters, as follows:

```
bash$ export HADOOP_CLIENT_OPTS="-Xms64m -Xmx1024m"
 bash$ hadoop distcp /source /target
```

3.16. Appendix

Map Sizing

By default, DistCp makes an attempt to size each map comparably so that each copies roughly the same number of bytes. Note that files are the finest level of granularity, so increasing the number of simultaneous copiers (i.e. maps) may not always increase the number of simultaneous copies nor the overall throughput.

DistCp also provides a strategy to "dynamically" size maps, allowing faster DataNodes to copy more bytes than slower nodes. Using the dynamic strategy (explained in the Architecture), rather than assigning a fixed set of source files to each map task, files are instead split into several sets. The number of sets exceeds the number of maps, usually by a factor of 2-3. Each map picks up and c opies all files listed in a chunk. When a chunk is exhausted, a new chunk is acquired and processed, until no more chunks remain.

By not assigning a source path to a fixed map, faster map tasks (i.e. DataNodes) are able to consume more chunks – and thus copy more data – than slower nodes. While this distribution isn't uniform, it is fair with regard to each mapper's capacity.

The dynamic strategy is implemented by the DynamicInputFormat. It provides superior performance under most conditions.

Tuning the number of maps to the size of the source and destination clusters, the size of the copy, and the available bandwidth is recommended for long-running and regularly run jobs.

Copying Between Versions of HDFS

For copying between two different versions of Hadoop, you will usually use HftpFileSystem. This is a read-only FileSystem, so DistCp must be run on the destination cluster (more specifically, on NodeManagers that can write to the destination cluster). Each source is specified as `hftp://<dfs.http.address>/<path>` (the default `dfs.http.address` is `<namenode>:50070`).

MapReduce and Other Side-Effects

As mentioned previously, should a map fail to copy one of its inputs, there will be several side-effects.

- Unless -overwrite is specified, files successfully copied by a previous map will be marked as "skipped" on a re-execution.
- If a map fails mapreduce.map.maxattempts times, the remaining map tasks will be killed (unless -i is set).
- If mapreduce.map.speculative is set final and true, the result of the copy is undefined.

SSL Configurations for HSFTP Sources

To use an HSFTP source (i.e. using the HSFTP protocol), a SSL configuration file needs to be specified (via the -mapredSslConf option). This must specify 3 parameters:

- ssl.client.truststore.location: The local file system location of the trust-store file, containing the certificate for the NameNode.
- ssl.client.truststore.type: (Optional) The format of the trust-store file.
- ssl.client.truststore.password: (Optional) Password for the trust-store file.

The following is an example of the contents of a SSL Configuration file:

```
<configuration>
 <property>
```

```
    <name>ssl.client.truststore.location</name>
    <value>/work/keystore.jks</value>
    <description>Truststore to be used by clients like distcp. Must be
 specified.</description>
 </property>

 <property>
    <name>ssl.client.truststore.password</name>
    <value>changeme</value>
    <description>Optional. Default value is "".</description>
 </property>

 <property>
    <name>ssl.client.truststore.type</name>
    <value>jks</value>
    <description>Optional. Default value is "jks".</description>
 </property>
</configuration>
```

The SSL configuration file must be in the classpath of the DistCp program.