Hortonworks Data Platform

System Administration

(May 9, 2016)

docs.hortonworks.com

Hortonworks Data Platform: System 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.

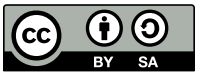


Table of Contents

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. HBase Cluster Capacity and Region Sizing

This section describes how to plan the capacity of an HBase cluster and the size of its RegionServers.

The following table provides information about HBase concepts:

HBase Concept	Description
Region	A group of contiguous HBase table rows. Tables start with one region and additional regions are dynamically added as the table grows. Regions can be spread across multiple hosts to provide load balancing and quick recovery from failure. There are two types of region: primary and secondary. A secondary region is a replicated primary region located on a different RegionServer.
RegionServer	Serves data requests for one or more regions. A single region is serviced by only one RegionServer, but a RegionServer may serve multiple regions.
Column family	A group of semantically related columns stored together.
Memstore	In-memory storage for a RegionServer. RegionServers write files to HDFS after the memstore reaches a configurable maximum value specified with the `hbase.hregion.memstore.flush.size` property in the `hbase-site.xml` configuration file.
Write Ahead Log (WAL)	In-memory log where operations are recorded before they are stored in the memstore.
Compaction storm	When the operations stored in the memstore are flushed to disk, HBase consolidates and merges many smaller files into fewer large files. This consolidation is called compaction, and it is usually very fast. However, if many RegionServers hit the data limit specified by the memstore at the same time, HBase performance may degrade from the large number of simultaneous major compactions. Administrators can avoid this by manually splitting tables over time.

2.1. Node Count and JVM Configuration

The number of nodes in an HBase cluster is typically driven by the following considerations:

- Physical size of the data
- Read/Write Throughput

Physical Size of the Data

The physical size of data on disk is affected by the following factors:

Factor Affecting Size of Physical Data	Description
HBase Overhead	The default amount of disk space required for a single HBase table cell. Smaller table cells require less overhead. The minimum cell size is 24 bytes and the default maximum is 10485760 bytes. Administrators can customize the maximum cell size with the `hbase.client.keyvalue.maxsize` property in the `hbase-site.xml` configuration file. HBase table cells are aggregated into blocks, and the block size is also configurable for each column family with the `hbase.mapreduce.hfileoutputformat.blocksize` property. The default value is 65536 bytes. Administrators may reduce this value for tables with highly random data access patterns to improve query latency.
Compression	Choose a data compression tool that makes sense for your data to reduce the physical size of data on disk. Unfortunately, HBase cannot ship with LZO due to licensing issues. However, HBase administrators may install LZO after installing

Factor Affecting Size of Physical Data	Description
	HBase. GZIP provides better compression than LZO but is slower. HBase also supports Snappy.
HDFS Replication	HBase uses HDFS for storage, so replicating HBase data stored in HDFS affects the total physical size of data. A typical replication factor of 3 for all HBase tables in a cluster would triple the physical size of the stored data.
RegionServer Write Ahead Log (WAL)	The size of the Write Ahead Log, or WAL, for each RegionServer has minimal impact on the physical size of data. The size of the WAL is typically fixed at less than half of the memory for the RegionServer. Although this factor is included here for completeness, the impact of the WAL on data size is negligible and its size is usually not configured.

Read/Write Throughput

The number of nodes in an HBase cluster may also be driven by required throughput for disk reads and/or writes. The throughput per node greatly depends on table cell size, data request patterns, as well as node and cluster configuration. Use YCSB tools to test the throughput of a single node or a cluster to determine if read/write throughput should drive your the number of nodes in your HBase cluster. A typical throughput for write operations for one RegionServer is 5-15 Mb/s. Unfortunately, there is no good estimate for read throughput; it varies greatly depending on physical data size, request patterns, and hit rate for the block cache.

2.2. Region Count and Size

In general, an HBase cluster runs smoother with fewer regions. However, administrators cannot directly configure the number of regions for a RegionServer. However, administrators can indirectly increase the number of regions in the following ways: In addition, administrators can indirectly affect the number of regions for a RegionServer in the following ways:

- Increase the size of the memstore for a RegionServer
- Increase the size of a region

In addition, administrators can increase the number of regions for a RegionServer by pre-splitting large regions to spread data and the request load across the cluster. HBase allows administrators to individually configure each HBase table, which is useful when tables have different workloads and use cases. Most region settings can be set on a per-table basis with HTableDescriptor class, as well as the HBase CLI. These methods override the properties in the hbase-site.xml configuration file. For further information, see Configure Compactions.

Note

The HDFS replication factor affects only disk usage and should not be considered when planning the size of regions. The other factors described in the table above are applicable.

2.3. Increase Memstore Size for RegionServer

Usage of the RegionServer's memstore largely determines the maximum number of regions for the RegionServer. Each region has its own memstores, one for each column family,

which grow to a configurable size, usually between 128 and 256 Mb. Administrators specify this size with the hbase.hregion.memstore.flush.size property in the hbase-site.xml configuration file. The RegionServer dedicates some fraction of total memory to region memstores based on the value of the hbase.regionserver.global.memstore.size configuration property. If usage exceeds this configurable size, HBase may become unresponsive or compaction storms might occur.

Use the following formula to estimate the number of regions for a RegionServer:

```
(regionserver_memory_size) * (memstore_fraction) /
((memstore_size) * (num_column_families))
```

For example, assume the following configuration:

- RegionServer with 16 Gb RAM (or 16384 Mb)
- Memstore fraction of .4
- Memstore with 128 Mb RAM
- 1 column family in table

The formula for this configuration would look as follows:

```
(16384 Mb * .4) / ((128 Mb * 1) = approximately 51 regions
```

The easiest way to decrease the number of regions for this example RegionServer is increase the RAM of the memstore to 256 Mb. The reconfigured RegionServer would then have approximately 25 regions, and the HBase cluster will run more smoothly if the reconfiguration is applied to all RegionServers in the cluster. The formula can be used for multiple tables with the same configuration by using the total number of column families in all the tables.

Note

The formula assumes all regions are filled at approximately the same rate. If a fraction of the cluster's regions are written to, divide the result by this fraction.

If the data request pattern is dominated by write operations rather than read operations, increase the memstore fraction. However, this increase negatively impacts the block cache.

2.4. Increase Size of the Region

The other way to indirectly increase the number of regions for a RegionServer is to increase the size of the region with the `hbase.hregion.max.filesize` property in the `hbase-site.xml` configuration file. Administrators increase the number of regions for a RegionServer by increasing the specified size at which new regions are dynamically allocated.

Maximum region size is primarily limited by compactions. Very large compactions can degrade cluster performance. The recommended maximum region size is 10 - 20 Gb. For HBase clusters running version 0.90.x, the maximum recommended region size is 4 Gb

and the default is 256 Mb. If you are unable to estimate the size of your tables, retain the default value. Increase the region size only if your table cells tend to be 100 Kb or larger.

Note

HBase 0.98 introduces stripe compactions as an experimental feature that also allows administrators to increase the size of regions. For more information, see Experimental:Stripe Compactions at the Apache HBase site.

2.5. Initial Configuration and Tuning

HBase administrators typically use the following methods to initially configure the cluster:

- Increase the request handler thread count
- Configure the size and number of WAL files
- Configure compactions
- Pre-split tables
- Tune JVM garbage collection

Increase the Request Handler Thread Count

Administrators who expect their HBase cluster to experience a high volume request pattern should increase the number of listeners generated by the RegionServers. Use the `hbase.regionserver.handler.count` property in the `hbase-site.xml` configuration file to set the number higher than the default value of `30`.

Configure the Size and Number of WAL Files

HBase uses the Write Ahead Log, or WAL, to recover memstore data not yet flushed to disk if a RegionServer crashes. Administrators should configure these WAL files to be slightly smaller than the HDFS block size. By default, an HDFS block is 64 Mb and a WAL is approximately 60 Mb. Hortonworks recommends that administrators ensure that enough WAL files are allocated to contain the total capacity of the memstores. Use the following formula to determine the number of WAL files needed:

```
(regionserver_heap_size * memstore fraction) / (default_WAL_size)
```

For example, assume the following HBase cluster configuration:

- 16 GB RegionServer heap
- 0.4 memstore fraction
- 60 MB default WAL size

The formula for this configuration looks as follows:

```
(16384 MB * 0.4 / 60 MB = approximately 109 WAL files
```

Use the following properties in the `hbase-site.xml` configuration file to configure the size and number of WAL files:

Configuration Property	Description	Default
hbase.regionserver.maxlogs	Sets the maximum number of WAL files.	32
hbase.regionserver.logroll.multiplier	Multiplier of HDFS block size.	0.95
hbase.regionserver.hlog.blocksize	Optional override of HDFS block size.	Value assigned to actual HDFS block size.

Note

If recovery from failure takes longer than expected, try reducing the number of WAL files to improve performance.

2.6. Configure Compactions

Administrators who expect their HBase clusters to host large amounts of data should consider the affect that compactions have on write throughput. For write-intensive data request patterns, administrators should consider less frequent compactions and more store files per region. Use the hbase.hstore.compaction.min property in the hbase-site.xml configuration file to increase the minimum number of files required to trigger a compaction. Administrators opting to increase this value should also increase the value assigned to the hbase.hstore.blockingStoreFiles property since more files will accumulate.

Pre-split Tables

Administrators can pre-split tables during table creation based on the target number of regions per RegionServer to avoid costly dynamic splitting as the table starts to fill up. In addition, it ensures that the regions in the pre-split table are distributed across many host machines. Pre-splitting a table avoids the cost of compactions required to rewrite the data into separate physical files during automatic splitting. If a table is expected to grow very large, administrators should create at least one region per RegionServer. However, do not immediately split the table into the total number of desired regions. Rather, choose a low to intermediate value. For multiple tables, do not create more than one region per RegionServer, especially if you are uncertain how large the table will grow. Creating too many regions for a table that will never exceed 100 Mb in size isn't useful; a single region can adequately services a table of this size.

Configure the JVM Garbage Collector

A RegionServer cannot utilize a very large heap due to the cost of garbage collection. Administrators should specify no more than 24 GB for one RegionServer.

3. Hive Authorization

Authorization determines whether a user has the required permissions to perform select operations, such as creating, reading, and writing data, as well as editing table metadata. Apache Ranger provides centralized authorization for all HDP components, and Hive also provides three authorization models. Administrators should consider the specific use case when choosing an authorization model.

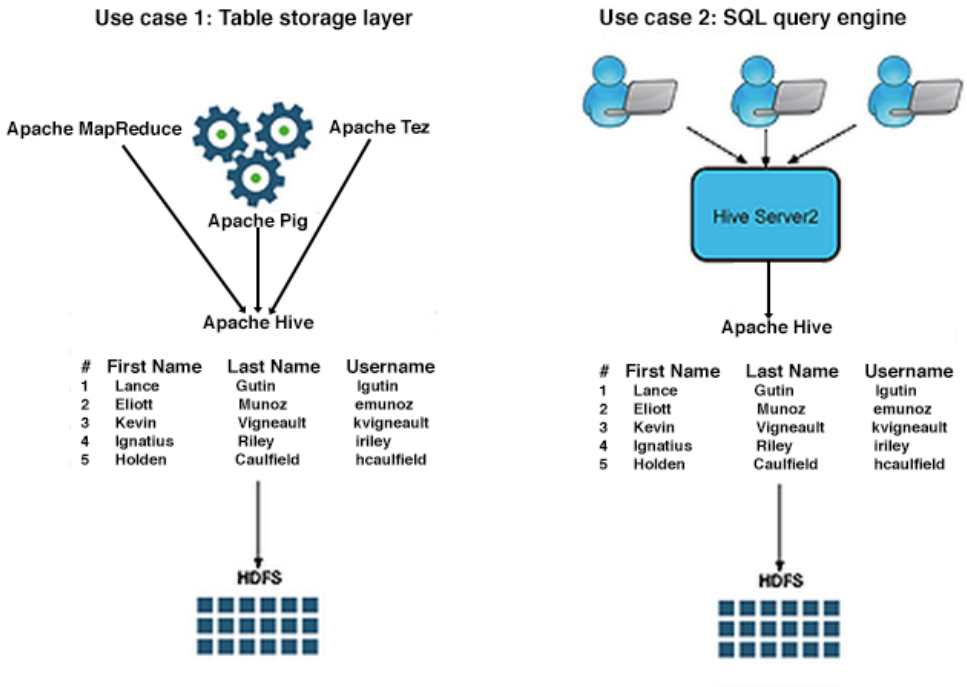
There are two primary use cases for Hive:

- **Table storage layer**

  Many HDP components and underlying technologies, such as Apache Hive, Apache HBase, Apache Pig, Apache MapReduce, and Apache Tez rely on Hive as a table storage layer.

- **SQL query engine**

  Hadoop administrators, business analysts, and data scientists use Hive to run SQL queries, both from the Hive CLI and remotely through a client connecting to Hive through HiveServer2. These users often configure a data analysis tool, such as Tableau, to connect to Hive through HiveServer2.



When using a JDBC or ODBC driver, the value of the hive.server2.enable.doAs configuration property in hive.site.xml determines the user account that runs a Hive query. The value assigned to this property depends on the desired Hive authorization model and, in the case of storage-based authorization, on the desired use case.

In addition to the centralized authorization provided by Apache Ranger, Hive provides three authorization models:

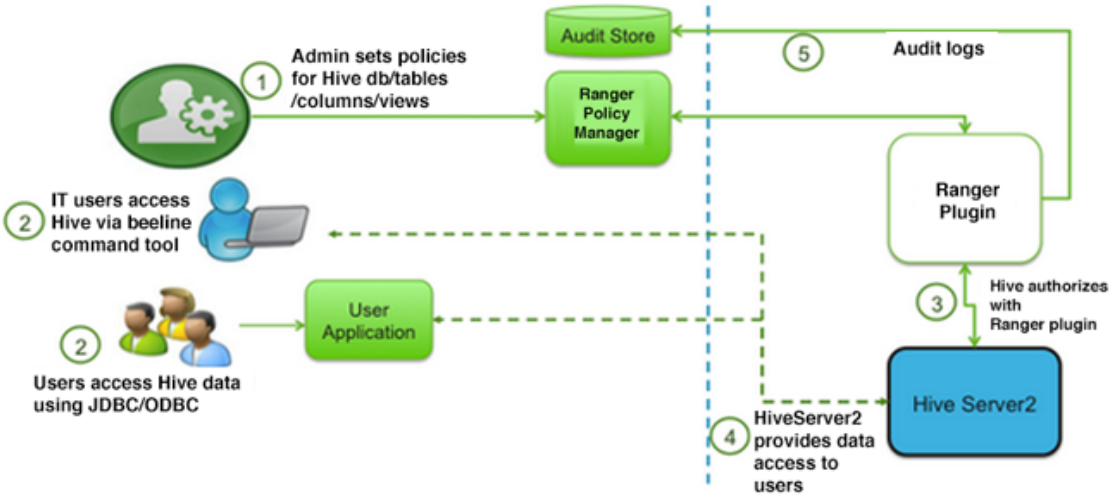
Authorization model	Secure?	Fine-grained authorization (column, row level)	Privilege management using GRANT/REVOKE statements	Centralized management GUI
Apache Ranger	Secure	Yes	Yes	Yes
SQL standard-based	Secure	Yes, through privileges on table views	Yes	No
Storage-based	Secure	No. Authorization at the level of databases, tables and partitions	No. Table privilege based on HDFS permission	No
Hive default	Not secure. No restriction on which users can run GRANT statements	Yes	Yes	No

Note

Administrators can secure the Hive CLI with Kerberos and by setting permisssions on the HDFS directories where tables reside. The exception to this is storage-based authorization, which does not require managing HDFS permissions and is the most secure authorization model for the Hive CLI.

3.1. Ranger-Hive Integration

Apache Ranger provides centralized policy management for authentication and auditing of all HDP components, including Hive. All HDP components are installed with an Ranger plugin used to intercept authorization requests for that component, as shown in the following illustration.



Apache Ranger-Hive integration

Administrators who are responsible for managing access to multiple components are strongly encouraged to use the Ranger Policy Manager to configure authentication for Hive rather than using storage-based or SQL standard-based authorization to take advantage of the ease-of-use provided by the Policy Manager. However, there are two primary use cases where administrators might choose to integrate Ranger with SQL standard-based authorization provided by Hive:

- An administrator is responsible for Hive authentication but not authentication for other HDP components
- An administrator wants row-level authentication for one or more table views

In the first use case, an administrator could choose any of the authorization models provided by Hive. The second use case is possible by integrating Ranger with SQL standard-based authorization provided by Hive. Hortonworks recommends that administrators who use both Ranger and SQL standard-based authorization use either White Policies in the Policy Manager or GRANT and REVOKE statements in Hive, but not both. Authentication changes made with GRANT and REVOKE statements appear as updates the corresponding White Policy; there is no need to configure authorization both ways. Ranger also provides an option to disable the use of GRANT and REVOKE statements.

There are two notable differences between Ranger authorization and SQL standard-based authorization:

- Ranger does not have the concept of a role. Instead, Ranger translates roles into users and groups.
- The ADMIN permission in Ranger is the equivalent to the WITH GRANT OPTION in SQL standard-based authorization. However, the ADMIN permission gives the grantee the ability to grant all permissions rather than just the permissions possessed by the grantor. With SQL standard-based authorization, the WITH GRANT OPTION applies only to permissions possessed by the grantor.

For more information about using ranger to configure Hive authorization, see the Apache Ranger User Guide. For more information about SQL standard-based authorization, see the following section.

3.2. SQL Standard-Based Authorization in Hive

SQL standard-based authorization provides fine-grained control using GRANT and REVOKE statements and supports row and column-level access with table views. Granting access to a table view is safer than granting access to the underlying table. This authorization model is disabled for the Hive command line. Secure access from the Hive CLI is not possible because users have direct access to HDFS and can bypass SQL standard-based authorization checks and even disable the authorization model. As the name suggests, this authorization model mimics traditional SQL compliant authorization in relational database systems with the GRANT and REVOKE commands. A user's privileges are checked when she runs a Hive query or command.

For more information about the ongoing work to fully support the SQL-2011 standard, see "SQL Compliance" in the HDP Data Services Guide.

Administrators can grant roles as well as privileges. Users can belong to one or more roles. Two roles have special meaning:

- public
- admin

All users belong to the public role. Administrators should use this role in GRANT statements intended to grant a privilege to all users. Administrators should add users who do the work of a database administrator to the admin role. These users have privileges to run additional commands such as CREATE ROLE and DROP ROLE, and they can access objects to which they haven't been given explicit access. However, users who belong to the admin role need to run the SET ROLE command before using the privileges of the admin role because this role is not included with the current roles by default.

The ownership of a table, view, or database determines who is authorized to perform certain actions. For example, the user who creates a table, view, or database becomes its owner. In the case of tables and views, the owner gets all the privileges with the GRANT option. Administrators can also use the ALTER DATABASE command to specify a role as the owner of a database.

SQL standard-based authorization models considers users with access to the following functionality as privileged:

- Hive CLI
- HDFS commands
- Pig CLI
- hadoop jar command
- MapReduce

These tools and commands do not access data through HiveServer2, so SQL standard-based authorization cannot authorize their access. Hortonworks recommends that administrators configure storage-based authorization on the Hive Metastore server to control access to data in Hive tables for these users. The two authorization models are compatible.

Note

Currently, SQL standard-based authorization does not poll groups from LDAP.

3.3. Required Privileges for Hive Operations

Privileges apply to tables and views, but not databases. The following privileges may be granted and revoked:

- Y = required privilege
- Y + G = required privilege and the ability to grant the privilege to other users

The following privileges are required for the specified Hive operations:

- Y = required privilege
- Y + G = required privilege and the ability to grant the privilege to other users

Hive Operation	SELECT	INSERT	DELETE	Update	Ownership	Admin	URI privilege (POSIX + ownership)
GRANT						Y	
REVOKE						Y	
SHOW GRANT						Y	
SHOW ROLE GRANT						Y	
CREATE ROLE						Y	
SET ROLE						Y	
DROP ROLE						Y	
CREATE TABLE					Y (of database)		
DROP TABLE					Y		
DESCRIBE TABLE	Y						
SHOW PARTITIONS	Y						
ALTER TABLE LOCATION					Y		Y (for new location)
ALTER PARTITION LOCATION					Y		Y (for new partition location
ALTER TABLE ADD PARTITION		Y					Y (for partition location)
ALTER TABLE DROP PARTITION			Y				
all other ALTER TABLE commands					Y		
TRUNCATE TABLE					Y		
CREATE VIEW	Y + G						
ALTER VIEW PROPERTIES					Y		
ALTER VIEW RENAME					Y		
DROP VIEW PROPERTIES					Y		
DROP VIEW					Y		
ANALYZE TABLE	Y	Y					

SHOW COLUMNS	Y						
SHOW TABLE STATUS	Y						
SHOW TABLE PROPERTIES	Y						
CREATE TABLE AS SELECT	Y (of input)				Y	Y (of database)	
UPDATE TABLE	Y						
CREATE INDEX						Y (of table)	
DROP INDEX						Y	
ALTER INDEX REBUILD						Y	
ALTER INDEX PROPERTIES						Y	
QUERY (INSERT, SELECT queries)	Y (input)	Y (output)	Y (output)				
LOAD		Y (output)	Y (output)				Y (input location)
SHOW CREATE TABLE	Y + G						
CREATE FUNCTION						Y	
DROP FUNCTION						Y	
CREATE MACRO						Y	
DROP MACRO						Y	
MSCK (metastore check)						Y	
ALTER DATABASE						Y	
CREATE DATABASE							Y (for custom location)
EXPLAIN	Y						
DROP DATABASE				Y			

3.4. Configuring SQL Standard-Based Authorization

Use the following procedure to configure SQL standard-based authorization for Hive:

- Set the following configuration properties in hive-site.xml to enable SQL standard-based authorization.
  - `hive.server2.enable.doAs`

    Allows Hive queries to be run by the user who submits the query, rather than by the `hive` user. Must be set to FALSE for SQL standard-based authorization.
  - `hive.users.in.admin.role`

    Comma-separated list of users assigned to the ADMIN role.
- Hive administrator must grant herself the ADMIN privilege:

  ```
  GRANT admin TO USER hiveadmin;
  ```
- Administrators must start HiveServer2 with the following command-line options:

Command line option	Required value
`hive.security.authorization.manager`	org.apache.hadoop.hive.ql.security.authorization.plugin.sql
`hive.security.authorization.enabled`	true
`hive.security.authenticator.manager`	org.apache.hadoop.hive.ql.security.SessionStateUserAuthenticator
`hive.metastore.uris`	"" (Quotation marks surrounding a single empty space)

These properties appear in the following snippet of hive-site.xml:

```
<property>
 <name>hive.security.authorization.manager</name>
 <value>org.apache.hadoop.hive.ql.security.authorization.plugin.sql</ value>
</property>

<property>
 <name>hive.security.authorization.enabled</name>
 <value>true</value>
</property>

<property>
 <name>hive.security.authenticator.manager</name>
 <value>org.apache.hadoop.hive.ql.security.SessionStateUserAuthenticator</
value>
</property>

<property>
 <name>hive.metastore.uris</name>
 <value>""</value>
</property>
```

3.5. Storage-Based Authorization in Hive

As the name implies, storage-based authorization relies on the authorization provided by the storage layer. In the case of an HDP cluster, the storage layer is HDFS, which provides both POSIX and ACL permissions. Hive is one of many HDP components that share storage on HDFS. HCatalog provides all of these components with a single consistent view metadata, and this is why storage-based authorization is enabled in the Hive Metastore server. By enabling this model on the Hive Metastore Server, Hadoop administrators

can provide consistent data and metadata authorization. The model controls access to metadata and checks permissions on the corresponding directories of the HDFS file system. Traditional POSIX permissions for the HDFS directories where tables reside determine access to those tables. For example, to alter table metadata for a table stored in HDFS at /user/hive/warehouse/mytable, a user must have write permissions on that directory. However, this authorization model doesn't support column-level security.

In addition to the traditional POSIX permissions model, HDFS also provides ACLs, or access control lists, as described in *ACLs on HDFS*. An ACL consists of a set of ACL entries, and each entry names a specific user or group and grants or denies read, write, and execute permissions for the specified user or group. These ACLs are also based on POSIX specifications, and they are compatible with the traditional POSIX permissions model.

HDFS ACL permissions provide administrators with authentication control over databases, tables, and table partitions on the HDFS file system. For example, an administrator can create a role with a set of grants on specific HDFS tables, then grant the role to a group of users. Roles allow administrators to easily reuse permission grants. Hortonworks recommends relying on POSIX permissions and a small number of ACLs to augment the POSIX permissions for exceptions and edge cases.

Note

A file with an ACL incurs additional memory cost to the NameNode due to the alternate algorithm used for permission checks on such files.

3.6. Required Permissions for Hive Operations

The following table shows the minimum permissions required for Hive operations using storage-based authorization:

Operation	Database Read Access	Database Write Access	Table Read Access	Table Write Access
LOAD				X
EXPORT			X	
IMPORT				X
CREATE TABLE		X		
CREATE TABLE AS SELECT		X	X (source table)	
DROP TABLE		X		
SELECT			X	
ALTER TABLE				X
SHOW TABLES	X			

3.7. Configuring Storage-based Authorization

Set the following configuration properties in hive-site.xml to enable storage-based authorization:

Configuration Property	Description
`hive.security.authorization.enabled`	Enables or disables Hive client authorization. This property must be set to false in hive-site.xml, but set to true for HiveServer2. Administrators can do this in one of two

Configuration Property	Description
	ways: as a command-line option when starting HiveServer2 or in a separate hiveserver2-site.xml configuration file.
hive.security.authorization.manager	The class name of the Hive client authorization manager. Specify the following value for storage-based authorization: org.apache.hadoop.hive.ql.security. authorization.StorageBasedAuthorizationProvider.
hive.server2.enable.doAs	Allows Hive queries to be run by the user who submits the query rather than the Hive user. Must be set to true for storage-based access.
hive.metastore.pre.event.listeners	Enables Metastore security. Specify the following value: org.apache.hadoop.ql.security .authorization.AuthorizationPre EventListener.
hive.security.metastore.authorization.manager	The class name of the Hive Metastore authorization manager. Specify the following value for storage-based authorization: org.apache.hadoop.hive.ql.security. authorization. StorageBasedAuthorizationProvider.

These properties appear in the following snippet of hive-site.xml:

```
<property>
   <name>hive.security.authorization.enabled</name>
   <value>false</value>
</property>

<property>
   <name>hive.security.authorization.manager</name>
   <value>org.apache.hadoop.hive.ql.security.authorization.
StorageBasedAuthorizationProvider</value>
</property>

<property>
   <name>hive.server2.enable.doAs</name>
   <value>true</value>
</property>

<property>
   <name>hive.metastore.pre.event.listeners</name>
   <name>org.apache.hadoop.ql.security.authorization.
AuthorizationPreEventListener</name>
</property>

<property>
   <name>hive.security.metastore.authorization.manager</name>
   <value>org.apache.hadoop.hive.ql.security.authorization.
StorageBasedAuthorizationProvider</value>
</property>
```

Administrators can use either of the following methods to create new tables and databases with appropriate storage-based permissions:

- Use the Hive CLI to create the table or database, then manually modify the POSIX permissions using the HDFS file system commands.
- Use the HCatalog CLI

The HCatalog command line tool uses the same syntax as Hive, but creates the table or database with a corresponding directory owned by the user creating it. Assigning a group permission is also supported. However, there are known issues with the HCatalog CLI:

- Some metadata operations do not check for authorization. See Apache JIRA HIVE_3009 for more information.
- Hive currently performs authorization checks on the client, rather than the server. This allows malicious users to circumvent these checks.
- DDL statements for managing permissions have no effect on storage-based authorization, but they do not return error messages. See Apache JIRA HIVE-3010 for more information.

4. Manually Adding Slave Nodes to an HDP Cluster

This chapter contains the following sections:

4.1. 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.

4.2. 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"
```

4.3. 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
```

  - For Ubuntu:

```
apt-get 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/2.4.2.0/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/2.4.2.0/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/2.4.2.0/hdp.repo -O /etc/zypp/repos.d/hdp.repo
```

- Ubuntu 12:

```
wget http://public-repo-1.hortonworks.com/HDP/ubuntu12/2.x/updates/2.4.2.
0/hdp.list -O /etc/apt/sources.list.d/hdp.list
```

- Ubuntu 14:

```
wget http://public-repo-1.hortonworks.com/HDP/ubuntu14/2.x/updates/2.4.2.
0/hdp.list -O /etc/apt/sources.list.d/hdp.list
```

- Debian 6 (Deprecated):

  HDP support for Debian 6 is deprecated with HDP 2.4.2. Future versions of HDP will no longer be supported on Debian 6.

```
wget http://public-repo-1.hortonworks.com/HDP/debian6/2.x/updates/2.4.2.0/
hdp.list -O /etc/apt/sources.list.d/hdp.list
```

- Debian 7:

```
wget http://public-repo-1.hortonworks.com/HDP/debian7/2.x/updates/2.4.2.0/
hdp.list -O /etc/apt/sources.list.d/hdp.list
```

- 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.

5. Optimizing HBase I/O

This chapter describes several ways to optimize HBase I/O. We start with an overview of HBase I/O, followed by a discussion of configuration options.

The information in this section is oriented toward basic BlockCache and MemStore tuning. As such, it describes a subset of cache configuration options. HDP supports additional BlockCache and MemStore properties, as well as other configurable performance optimizations such as RPC, HFile block size settings, and HFile compaction. For a complete list of configurable properties, see the hbase-default.xml source file in GitHub.

An Overview of HBase I/O

The following table describes several concepts related to HBase file operations and memory (RAM) caching.

HBase Component	Description
HFile	An HFile contains table data, indexes over that data, and metadata about the data.
Block	An HBase block is the smallest unit of data that can be read from an HFile. Each HFile consists of a series of blocks. (Note: an HBase block is different than an HDFS block or other underlying file system blocks.)
BlockCache	BlockCache is the main HBase mechanism for low-latency random read operations. BlockCache is one of two memory cache structures maintained by HBase. When a block is read from HDFS, it is cached in BlockCache. Frequent access to rows in a block cause the block to be kept in cache, improving read performance.
MemStore	MemStore ("memory store") is the second of two cache structures maintained by HBase. MemStore improves write performance. It accumulates data until it is full, and then writes ("flushes") the data to a new HFile on disk. MemStore serves two purposes: it increases the total amount of data written to disk in a single operation, and it retains recently-written data in memory for subsequent low-latency reads.
Write Ahead Log (WAL)	The WAL is a log file that records all changes to data until the data is successfully written to disk (MemStore is flushed). This protects against data loss in the event of a failure before MemStore contents are written to disk.

BlockCache and MemStore reside in random-access memory (RAM); HFiles and the Write Ahead Log are persisted to HDFS.

Figure 1 shows write and read paths (simplified):

- During write operations, HBase writes to WAL and MemStore. Data is flushed from MemStore to disk according to size limits and flush interval.
- During read operations, HBase reads the block from BlockCache or MemStore if it is available in those caches. Otherwise it reads from disk and stores a copy in BlockCache.

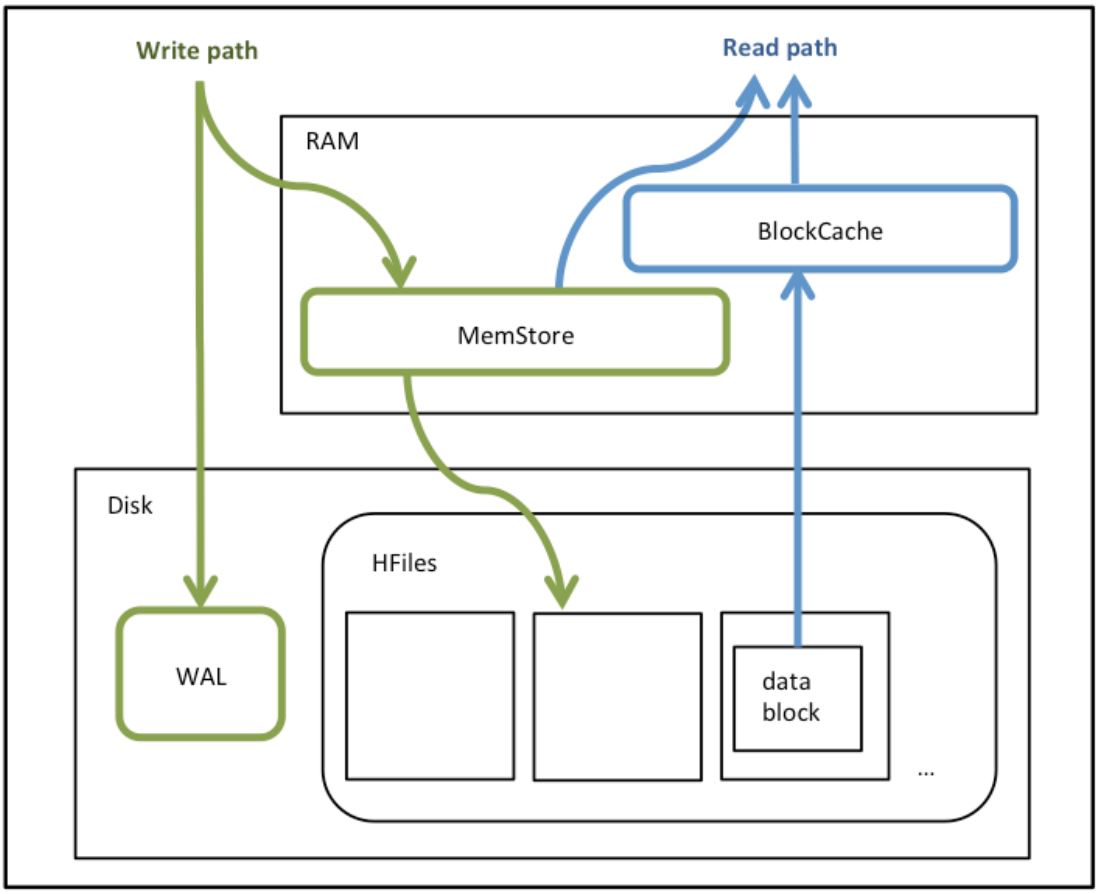


Figure 1. HBase read and write operations

By default, BlockCache resides in an area of RAM that is managed by the Java Virtual Machine ("JVM") Garbage Collector; this area of memory is known as “on-heap" memory or the "Java heap." The BlockCache implementation that manages on-heap cache is called LruBlockCache.

If you have stringent read latency requirements and you have more than 20 GB of RAM available on your servers for use by HBase RegionServers, consider configuring BlockCache to use both on-heap and off-heap ("direct") memory, as shown below. The associated BlockCache implementation is called BucketCache. Read latencies for BucketCache tend to be less erratic than LruBlockCache for large cache loads, because BucketCache (not JVM Garbage Collection) manages block cache allocation.

Figure 2 illustrates the two BlockCache implementations and MemCache, which always resides on the JVM heap.

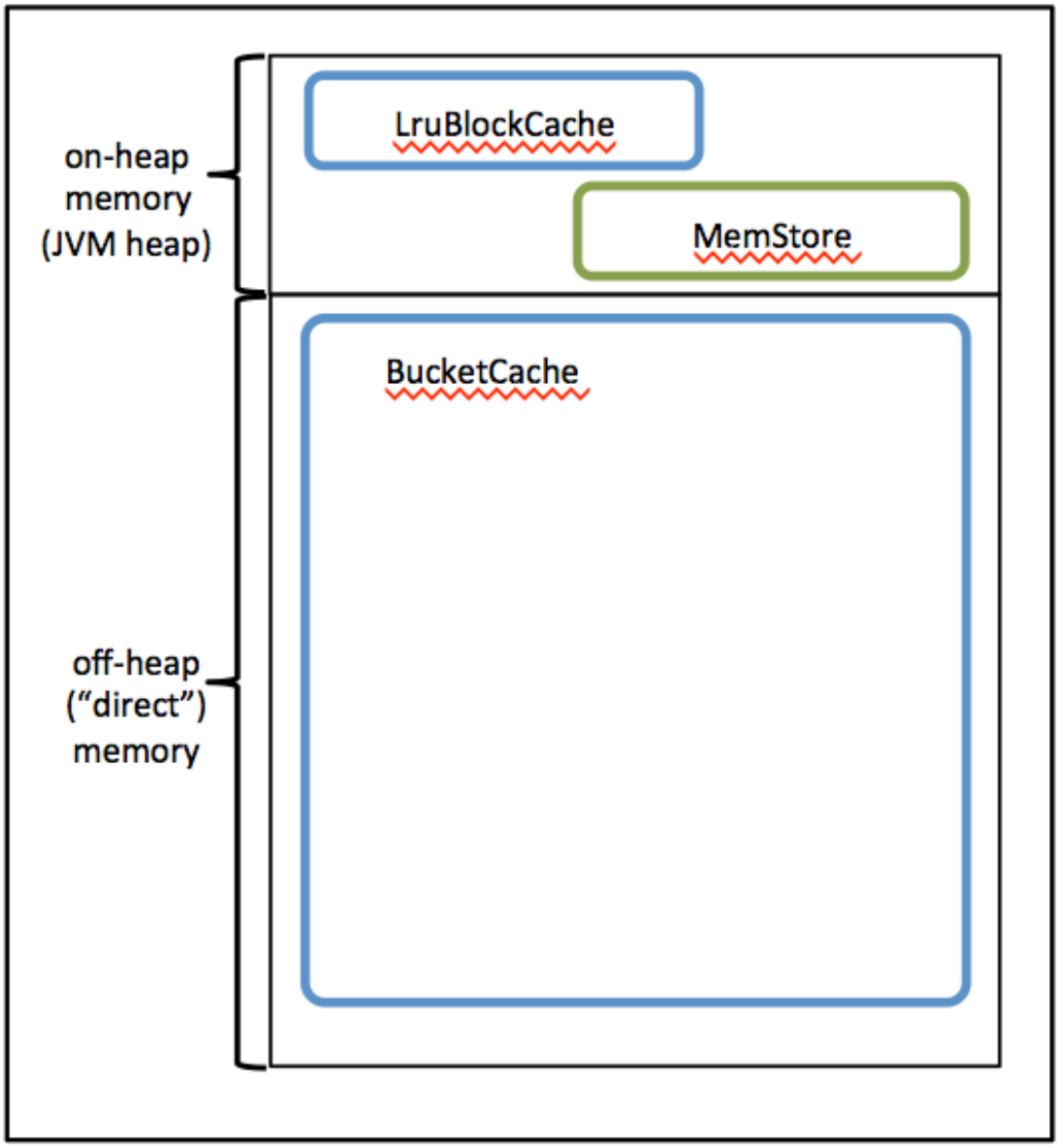


Figure 2. BucketCache implementation

- **Additional notes:**
- BlockCache is enabled by default for all HBase tables.
- BlockCache is beneficial for all read operations – random and sequential – though it is of primary consideration for random reads.
- All regions hosted by a RegionServer share the same BlockCache.
- You can turn BlockCache caching on or off per column family.

5.1. Configuring BlockCache

If you have less than 20 GB of RAM available for use by HBase, consider tailoring the default on-heap BlockCache implementation (LruBlockCache) for your cluster.

If you have more than 20 GB of RAM available, consider adding off-heap BlockCache (BucketCache).

The first few steps are the same for both options:

- Specify the maximum amount of on-heap RAM to allocate to the HBase RegionServer on each node. The default is 1 GB, which is too small for production.

  To alter the default allocation, set the "RegionServers maximum Java heap size" value (Ambari), or set the HBASE_HEAPSIZE environment variable in `hbase-env.sh` (manual installation). Specify the value in megabytes. The HBase startup script uses `$HBASE_HEAPSIZE` to override the default maximum JVM heap size (`-Xmx`).

  The following example sets the maximum on-heap memory allocation to 20 GB in `hbase-env.sh`:

  ```
  export HBASE_HEAPSIZE=20480
  ```

- Determine (or estimate) the proportions of reads and writes in your workload, and use these proportions to specify on-heap memory for BlockCache and MemStore. The sum of the two allocations must be less than or equal to 0.8. The following table describes the two properties.

Property	Default Value	Description
`hfile.block.cache.size`	0.4	Proportion of maximum JVM heap size (Java `-Xmx` setting) to allocate to BlockCache. A value of 0.4 allocates 40% of the maximum heap size.
`hbase.regionserver.global.memstore.upperLimit`	0.4	Proportion of maximum JVM heap size (Java `-Xmx` setting) to allocate to MemStore. A value of 0.4 allocates 40% of the maximum heap size.

  Use the following guidelines to determine the two proportions:

  - The default configuration for each property is 0.4, which configures BlockCache for a mixed workload with roughly equal proportions of random reads and writes.
  - If your workload is read-heavy and you do not plan to configure off-heap cache – your amount of available RAM is less than 20 GB – increase `hfile.block.cache.size` and decrease `hbase.regionserver.global.memstore.upperLimit` so that the values reflect your workload proportions. This will optimize read performance.
  - If your workload is write-heavy, decrease `hfile.block.cache.size` and increase `hbase.regionserver.global.memstore.upperLimit` proportionally.
  - As noted earlier, the sum of `hfile.block.cache.size` and `hbase.regionserver.global.memstore.upperLimit` must be less than or equal to 0.8 (80%) of the maximum Java heap size specified by HBASE_HEAPSIZE (`-Xmx`). If you allocate more than 0.8 across both caches, the HBase RegionServer process will return an error and will not start.

- Do not set `hfile.block.cache.size` to zero. At a minimum, specify a proportion that allocates enough space for HFile index blocks. To review index block sizes, use the RegionServer Web GUI for each server.

- Edit the corresponding values in your `hbase-site.xml` file(s). Here are the default definitions:

```
<property>
     <name>hfile.block.cache.size</name>
     <value>0.4</value>
     <description>Percentage of maximum heap (-Xmx setting) to allocate to
 block
       cache used by HFile/StoreFile. Default of 0.4 allocates 40%.
     </description>
 </property>

 <property>
     <name>hbase.regionserver.global.memstore.upperLimit</name>
     <value>0.4</value>
     <description>Maximum size of all memstores in a region server before
 new
       updates are blocked and flushes are forced. Defaults to 40% of heap.
     </description>
 </property>
```

- If you have less than 20 GB of RAM for use by HBase, you are done with the configuration process. Restart (or rolling restart) your cluster. Check log files for error messages. If you have more than 20 GB of RAM for use by HBase, consider configuring the variables and properties in the next subsection.

5.2. (Optional) Configuring Off-heap Memory (BucketCache)

At this point, you have determined that you have enough physical memory on your servers to extend BlockCache beyond the JVM Heap. You have configured values for the JVM heap, LruBlockCache, and MemStore (shown in Figure 2, copied here for reference).

Next, configure and enable BucketCache. To do this, note the memory specifications and values in *Configuring BlockCache*, plus a few additional values shown in Figure 3 and in the following table.

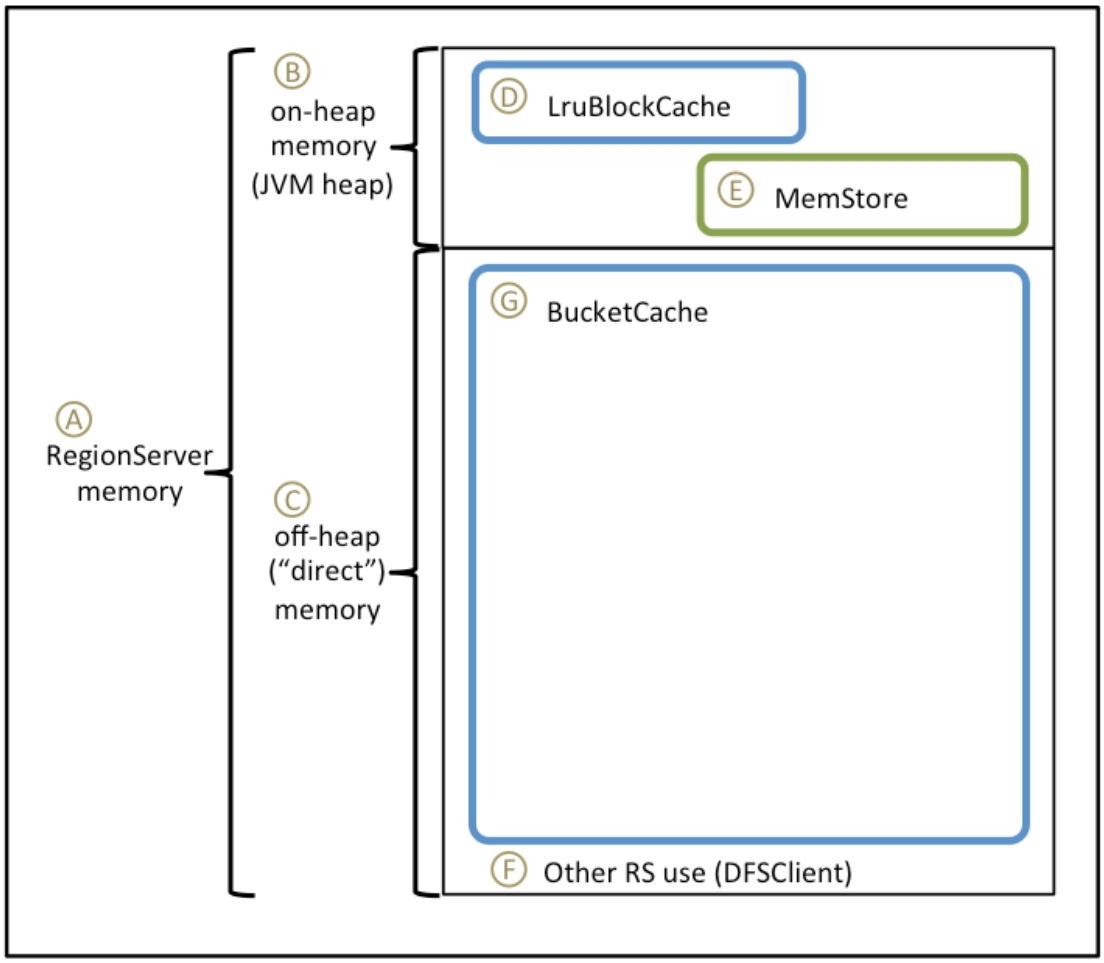


Figure 3. Configuring BucketCache

In the following table, the first column refers to the elements in Figure 3. The second column describes each element and, if applicable, its associated variable or property name. The third column contains values and formulas.

The fourth column computes values based on the following sample configuration parameters:

- 128 GB for the RegionServer process (there is additional memory available for other HDP processes)
- Workload consists of 75% reads, 25% writes.
- HBASE_HEAPSIZE = 20 GB (20480 MB)

Note

Most of the following values are specified in megabytes; three are proportions.

Item	Description	Value or Formula	Example
A	Total physical memory for RegionServer operations:	(hardware dependent)	131072

	on-heap plus off-heap ("direct") memory (MB)		
B*	HBASE_HEAPSIZE (-Xmx) Maximum size of JVM heap (MB)	Recommendation: 20480	20480
C	-XX:MaxDirectMemorySize Amount of off-heap ("direct") memory to allocate to HBase (MB)	A - B	131072 - 20480 = 110592
Dp*	hfile.block.cache.size Proportion of maximum JVM heap size (HBASE_HEAPSIZE, -Xmx) to allocate to BlockCache. The sum of this value plus hbase.regionserver.global.memstore.size must not exceed 0.8.	(proportion of reads) * 0.8	0.75 * 0.8 = 0.6
Dm	Maximum amount of JVM heap to allocate to BlockCache (MB)	B * Dp	20480 * 0.6 = 12288
Ep*	hbase.regionserver.global.memstore.size Proportion of maximum JVM heap size (HBASE_HEAPSIZE, -Xmx) to allocate to MemStore. The sum of this value plus hfile.block.cache.size must be less than or equal to 0.8.	0.8 - Dp	0.8 - 0.6 = 0.2
F	Amount of off-heap memory to reserve for other uses (DFSClient; MB)	Recommendation: 1024 to 2048	2048
G	Amount of off-heap memory to allocate to BucketCache (MB)	C - F	110592 - 2048 = 108544
	hbase.bucketcache.size Total amount of memory to allocate to BucketCache, on- and off-heap (MB)	Dm + G	12288 + 108544 = 120832

* Specified in *Configuring BlockCache*

After completing the steps in *Configuring BlockCache*, follow these steps to configure BucketCache.

- In the hbase-env.sh file for each RegionServer, or in the hbase-env.sh file supplied to Ambari, set the -XX:MaxDirectMemorySize argument for HBASE_REGIONSERVER_OPTS to the amount of direct memory you wish to allocate to HBase. In the sample configuration, the value would be 110592m. (-XX:MaxDirectMemorySize accepts a number followed by a unit indicator; m indicates megabytes.)

  ```
  HBASE_OPTS="$HBASE_OPTS -XX:MaxDirectMemorySize=110592m"
  ```

- In the hbase-site.xml file, specify BucketCache size and percentage. For the sample configuration, the values would be 120832 and 0.89830508474576, respectively.

If you choose to round the proportion, round up. This will allocate space related to rounding error to the (larger) off-heap memory area.

```
<property>
   <name>hbase.bucketcache.size</name>
   <value>120832</value>
</property>
```

- In the `hbase-site.xml` file, set `hbase.bucketcache.ioengine` to `offheap`. This enables BucketCache.

```
<property>
    <name>hbase.bucketcache.ioengine</name>
    <value>offheap</value>
</property>
```

- Restart (or rolling restart) the cluster. It can take a minute or more to allocate BucketCache, depending on how much memory you are allocating. Check logs for error messages.

5.3. Compressing BlockCache

BlockCache compression caches data and encoded data blocks in their on-disk format, rather than decompressing and decrypting them before caching. When compression is enabled on a column family, this allows more data to fit into the amount of memory dedicated to BlockCache. Decompression is repeated every time a block is accessed, but the increase in available cache space can have a positive impact on throughput and mean latency.

BlockCache compression is particularly useful when you have more data than RAM allocated to BlockCache, but your compressed data would fit into BlockCache. (The savings must be worth the increased garbage collection overhead and overall CPU load).

If your data can fit into block cache without compression, or if your workload is sensitive to extra CPU or garbage collection overhead, we recommend against enabling BlockCache compression.

Block cache compression is disabled by default. To enable BlockCache compression:

- Set `hbase.block.data.cachecompressed` to `true` in the `hbase-site.xml` file on each RegionServer.
- Restart or rolling restart your cluster. Check logs for error messages.

Important

This feature requires compression to be enabled on the table. For more information, see Enable Compression on a ColumnFamily.

6. 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.

6.1. 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.

6.2. Command Line Options

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

6.3. 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.

6.4. 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>
```

6.5. 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.

6.6. 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>
```

6.7. 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.

6.8. 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.

6.9. 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.

6.10. DistCp Architecture

DistCp is comprised of the following components:

- DistCp driver
- Copy-listing generator classes
- InputFormats and MapReduce components

6.11. 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.

6.12. 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.

6.13. 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.

6.14. 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
```

6.15. 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.