Apache Hadoop Distributed Copy

v. 0.23.1

User Guide

Table of Contents

1 Introduction

1.1 Overview

DistCp (distributed copy) is a tool used for large inter/intra-cluster copying. It uses Map/Reduce to effect its distribution, error handling and recovery, and reporting. It expands a list of files and directories into input to map tasks, each of which will copy a partition of the files specified in the source list.

The erstwhile implementation of DistCp has its share of quirks and drawbacks, both in its usage, as well as its extensibility and performance. The purpose of the DistCp refactor was to fix these shortcomings, enabling it to be used and extended programmatically. New paradigms have been introduced to improve runtime and setup performance, while simultaneously retaining the legacy behaviour as default.

This document aims to describe the design of the new DistCp, its spanking new features, their optimal use, and any deviance from the legacy implementation.

2 Usage

2.1 Basic Usage

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

```
bash$ hadoop jar hadoop-distcp.jar hdfs://nn1:8020/foo/bar \
hdfs://nn2:8020/bar/foo
```

This will expand the namespace under `/foo/bar` on nn1 into a temporary file, partition its contents among a set of map tasks, and start a copy on each TaskTracker from nn1 to nn2.

One can also specify multiple source directories on the command line:

```
bash$ hadoop jar hadoop-distcp.jar hdfs://nn1:8020/foo/a \
hdfs://nn1:8020/foo/b \
hdfs://nn2:8020/bar/foo
```

Or, equivalently, from a file using the `-f` option:

```
bash$ hadoop jar hadoop-distcp.jar -f hdfs://nn1:8020/srclist \
hdfs://nn2:8020/bar/foo
```

Where `srclist` contains

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

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 per 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 TaskTracker can reach and 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, it is recommended that one generates and cross-checks 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 the 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's 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 FileNotFoundException.

Please refer to the detailed Command Line Reference for information on all the options available in DistCp.

2.2 Update and Overwrite

`-update` is used to copy files from source that don't exist at the target, or have different contents. `-overwrite` overwrites target-files even if they exist at the source, or have the same contents.

Update and Overwrite options warrant special attention, 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 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 (say, `0`), then both sources would map an entry to `/target/0` at the destination. Rather than to 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 since the contents don't match the source.

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

3 Command Line Reference

3.1 Options Index

Flag	Description	Notes
`-p[rbugp]`	Preserve r: replication number b: block size u: user g: group p: permission	Modification times are not preserved. Also, when `-update` is specified, status updates will **not** be synchronized unless the file sizes also differ (i.e. unless the file is re-created).
`-i`	Ignore failures	As explained in the Appendix, this option will keep more accurate statistics about the copy than the default case. It also preserves logs from failed copies, which can be valuable for debugging. Finally, a failing map will not cause the job to fail before all splits are attempted.
`-log <logdir>`	Write logs to <logdir>	DistCp keeps logs of each file it attempts to copy as map output. If a map fails, the log output will not be retained if it is re-executed.
`-m <num_maps>`	Maximum number of simultaneous copies	Specify the number of maps to copy data. Note that more maps may not necessarily improve throughput.
`-overwrite`	Overwrite destination	If a map fails and `-i` is not specified, all the files in the split, not only those that failed, will be recopied. As discussed in the Usage documentation, it also changes the semantics for generating destination paths, so users should use this carefully.
`-update`	Overwrite if src size different from dst size	As noted in the preceding, this is not a "sync" operation. The only criterion examined is the source and destination file sizes; if they differ, the source file replaces the destination file. As discussed in the Usage documentation, it also changes the semantics for generating destination paths, so users should use this carefully.
`-f <urilist_uri>`	Use list at <urilist_uri> as src list	This is equivalent to listing each source on the command line. The `urilist_uri` list should be a fully qualified URI.
`-filelimit <n>`	Limit the total number of files to be <= n	**Deprecated!** Ignored in the new DistCp.

`-sizelimit <n>`	Limit the total size to be <= n bytes	**Deprecated!** Ignored in the new DistCp.
`-delete`	Delete the files existing in the dst but not in src	The deletion is done by FS Shell. So the trash will be used, if it is enable.
`-strategy {dynamic| uniformsize}`	Choose the copy-strategy to be used in DistCp.	By default, uniformsize is used. (i.e. Maps are balanced on the total size of files copied by each map. Similar to legacy.) If "dynamic" is specified, `DynamicInputFormat` is used instead. (This is described in the Architecture section, under InputFormats.)
`-bandwidth`	Specify bandwidth per map, in MB/ second.	Each map will be restricted to consume only the specified bandwidth. This is not always exact. The map throttles back its bandwidth consumption during a copy, such that the **net** bandwidth used tends towards the specified value.
`-atomic {-tmp <tmp_dir>}`	Specify atomic commit, with optional tmp directory.	`-atomic` instructs DistCp to copy the source data to a temporary target location, and then move the temporary target to the final-location atomically. Data will either be available at final target in a complete and consistent form, or not at all. Optionally, `-tmp` may be used to specify the location of the tmp-target. If not specified, a default is chosen. **Note:** tmp_dir must be on the final target cluster.
`-mapredSslConf <ssl_conf_file>`	Specify SSL Config file, to be used with HSFTP source	When using the hsftp protocol with a source, the security- related properties may be specified in a config-file and passed to DistCp. <ssl_conf_file> needs to be in the classpath.
`-async`	Run DistCp asynchronously. Quits as soon as the Hadoop Job is launched.	The Hadoop Job-id is logged, for tracking.

4 Architecture

4.1 Architecture

The components of the new DistCp may be classified into the following categories:

- DistCp Driver
- Copy-listing generator
- Input-formats and Map-Reduce components

4.1.1 DistCp Driver

The DistCp Driver components are responsible for:

- Parsing the arguments passed to the DistCp command on the command-line, via:
  - OptionsParser, and
  - 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 Map-Reduce Job to carry out the copy.
  - Based on the options, either returning a handle to the Hadoop MR Job immediately, or waiting till completion.

The parser-elements are exercised 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.

4.1.2 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 wild-cards), and record all paths that need copy into a sequence- file, for consumption by the DistCp Hadoop Job. The main classes in this module include:

1. 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.
2. SimpleCopyListing: An implementation of CopyListing that accepts multiple source paths (files/directories), and recursively lists all the individual files and directories under each, for copy.
3. GlobbedCopyListing: Another implementation of CopyListing that expands wild-cards in the source paths.
4. 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:

1. If there's no source-file-list, the GlobbedCopyListing is used. All wild-cards are expanded, and all the expansions are forwarded to the SimpleCopyListing, which in turn constructs the listing (via recursive descent of each path).
2. 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.

One may 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 target. E.g. if a file already exists at the target (and -overwrite isn't specified), the file isn't even considered in the Map-Reduce Copy Job. Determining this during setup (i.e. before the Map-Reduce Job) involves file-size and checksum-comparisons that are potentially time-consuming.

The new DistCp postpones such checks until the Map-Reduce Job, thus reducing setup time. Performance is enhanced further since these checks are parallelized across multiple maps.

4.1.3 Input-formats and Map-Reduce components

The Input-formats and Map-Reduce components are responsible for the actual copy 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. Apropos, 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 isn't 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 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's Configuration), to determine whether a file needs copy. A file will be copied only if at least one of the following is true:
  - A file with the same name doesn't 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 isn't 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.

5 Appendix

5.1 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.

The new DistCp also provides a strategy to "dynamically" size maps, allowing faster data-nodes to copy more bytes than slower nodes. Using `-strategy dynamic` (explained in the Architecture), rather than to assign 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 copies 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. data-nodes) 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.

5.2 Copying between versions of HDFS

For copying between two different versions of Hadoop, one will usually use HftpFileSystem. This is a read-only FileSystem, so DistCp must be run on the destination cluster (more specifically, on TaskTrackers 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).

5.3 Map/Reduce and other side-effects

As has been mentioned in the preceding, 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 on a re-execution will be marked as "skipped".
- If a map fails `mapred.map.max.attempts` times, the remaining map tasks will be killed (unless `-i` is set).
- If `mapred.speculative.execution` is set set `final` and `true`, the result of the copy is undefined.

5.4 SSL Configurations for HSFTP sources:

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

- `ssl.client.truststore.location`: The local-filesystem 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 the contents of a Map-Red 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 class-path of the DistCp program.

6 FAQ

6.1 Frequently Asked Questions

General

6.2 General

Why does -update not create the parent source-directory under a pre-existing target directory?

The behaviour of `-update` and `-overwrite` is described in detail in the Usage section of this document. In short, if either option is used with a pre-existing destination directory, the **contents** of each source directory is copied over, rather than the source-directory itself. This behaviour is consistent with the legacy DistCp implementation as well.

[top]

How does the new DistCp differ in semantics from the Legacy DistCp?

- Files that are skipped during copy used to also have 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.
- Empty root directories among the source-path inputs were not created at the target, in Legacy DistCp. These are now created.

[top]

Why does the new DistCp 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 M/R 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 till the M/R 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 (default) if unspecified.

[top]

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 lauched would equal the number of files.

[top]

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
```

[top]