Cloudera Runtime 7.0.2

Managing Data Storage

Date published: 2019-11-07
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Optimizing data storage

You can consider the following options to optimize data storage in HDFS clusters: increasing storage space through erasure coding, using codecs for compressing data, and balancing data across an HDFS cluster.

Erasure coding overview

Data durability describes how resilient data is to loss. When data is stored in HDFS, CDP provides two options for data durability. You can use replication, which HDFS was originally built on, or Erasure Coding (EC).

Note: The comparisons between EC and replication use a replication factor of 3 (three copies of data are maintained) since that is the default.

Replication

HDFS creates two copies of data, resulting in three total instances of data. These copies are stored on separate DataNodes to guard against data loss when a node is unreachable. When the data stored on a node is lost or inaccessible, it is replicated from one of the other nodes to a new node so that there are always multiple copies. The number of replications is configurable, but the default is three. Cloudera recommends keeping the replication factor to at least three when you have three or more DataNodes. A lower replication factor leads to a situation where the data is more vulnerable to DataNode failures since there are fewer copies of data spread out across fewer DataNodes..

When data is written to an HDFS cluster that uses replication, additional copies of the data are automatically created. No additional steps are required.

Replication supports all data processing engines that CDP supports.

Erasure Coding (EC)

EC is an alternative to replication. When an HDFS cluster uses EC, no additional direct copies of the data are generated. Instead, data is striped into blocks and encoded to generate parity blocks. If there are any missing or corrupt blocks, HDFS uses the remaining data and parity blocks to reconstruct the missing pieces in the background. This process provides a similar level of data durability to 3x replication but at a lower storage cost.

Additionally, EC is applied when data is written. This means that to use EC, you must first create a directory and configure it for EC. Then, you can either replicate existing data or write new data into this directory.

EC supports the following data processing engines:

- Hive
- MapReduce
- Spark

With both data durability schemes, replication and EC, recovery happens in the background and requires no direct input from a user.

HDFS clusters can be configured with a single data durability scheme (3x replication or EC), or with a hybrid data durability scheme where EC enabled directories co-exist on a cluster with other directories that are protected with the traditional 3x replication model. This decision should be based on the temperature of the data (how often the data is accessed) stored in HDFS. Hot data, data that is accessed frequently, should use replication. Cold data, data that is accessed less frequently, can take advantage of EC's storage savings.

Understanding erasure coding policies

The EC policy determines how data is encoded and decoded. An EC policy is made up of the following parts: codec-number of data blocks-number of parity blocks-cell size.

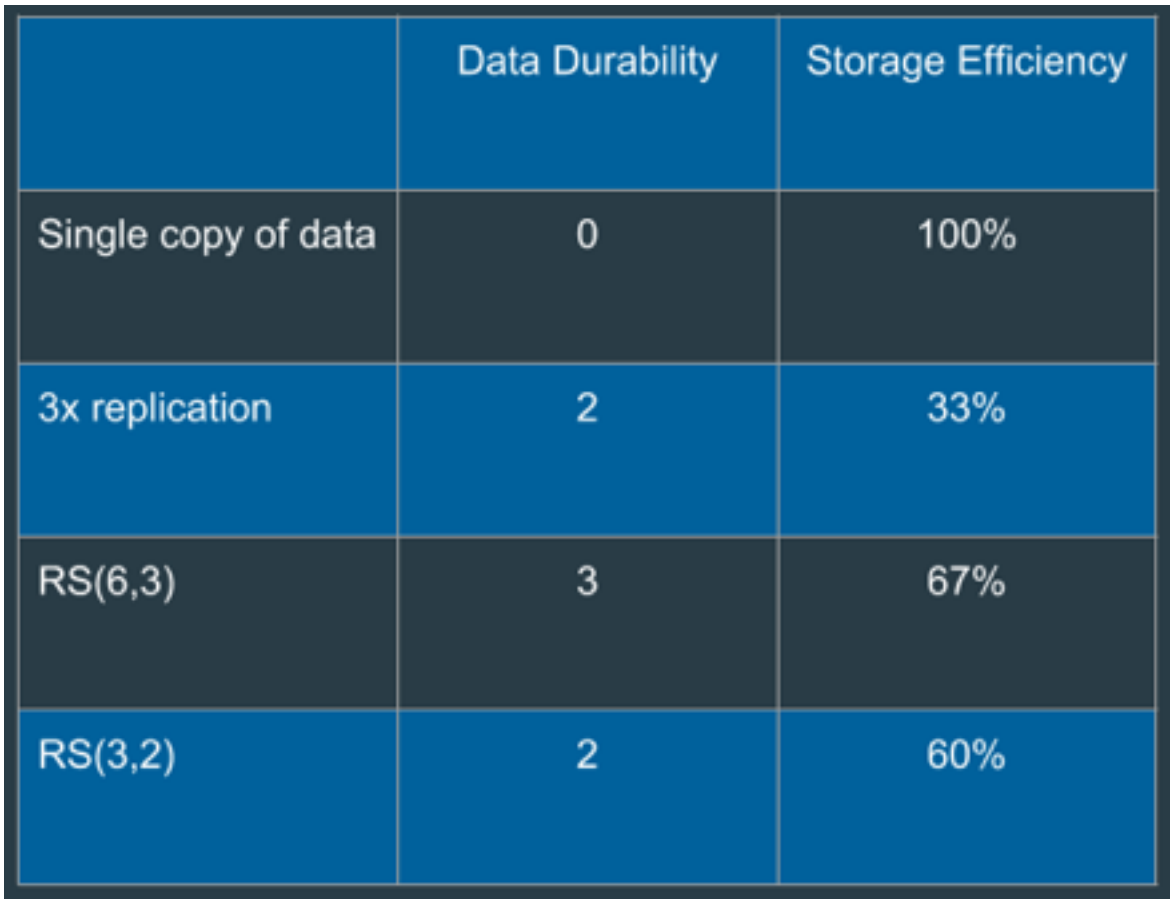
- Codec: The erasure codec that the policy uses. CDP currently supports Reed-Solomon (RS).
- Number of Data Blocks: The number of data blocks per stripe. The higher this number, the more nodes that need to be read when accessing data because HDFS attempts to distribute the blocks evenly across DataNodes.
- Number of Parity Blocks: The number of parity blocks per stripe. Even if a file does not use up all the data blocks available to it, the number of parity blocks will always be the total number listed in the policy.
- Cell Size: The size of one basic unit of striped data.

For example, a RS-6-3-1024k policy has the following attributes:

- Codec: Reed-Solomon
- Number of Data Blocks: 6
- Number of Parity Blocks: 3
- Cell Size: 1024k

The sum of the number of data blocks and parity blocks is the data-stripe width. When you make hardware plans for your cluster, the number of racks should at least equal the stripe width in order for the data to be resistant to rack failures.

The following image compares the data durability and storage efficiency of different RS codecs and replication:



	Data Durability	Storage Efficiency
Single copy of data	0	100%
3x replication	2	33%
RS(6,3)	3	67%
RS(3,2)	2	60%

Storage efficiency is the ratio of data blocks to total blocks as represented by the following formula: data blocks / (data blocks + parity blocks)

Comparing replication and erasure coding

You must consider factors such as data temperature, i/o cost, storage cost, and file size when comparing replication and erasure coding.

Data Temperature

Data temperature refers to how often data is accessed. EC works best with cold data that is accessed and modified infrequently. There is no data locality, and all reads are remote. Replication is more suitable for hot data, data that is accessed and modified frequently because data locality is a part of replication.

I/O Cost

EC has higher I/O costs than replication for the following reasons:

- EC spreads data across nodes and racks, which means reading and writing data comes at a higher network cost.
- A parity block is generated when data is written, thus impacting write speed. This can be slower than writing to a file when the replication factor is one but is faster than writing two or more replicas.
- If data is missing or corrupt, a DataNode reads the remaining data and parity blocks in order to reconstruct the data. This process requires CPU and network resources.

Cloudera recommends at least a 10 GB network connection if you want to use EC.

Storage Cost

EC has a lower storage overhead than replication because multiple copies of data are not maintained. Instead, a number of parity blocks are generated based on the EC policy. For the same amount of data, EC will store fewer blocks than 3x replication in most cases. For example with a Reed-Solomon (6,3), HDFS stores three parity blocks for each set of 6 data blocks. With replication, HDFS stores 12 replica blocks for every six data blocks, the original block and three replicas. The case where 3x replication requires fewer blocks is when data is stored in small files.

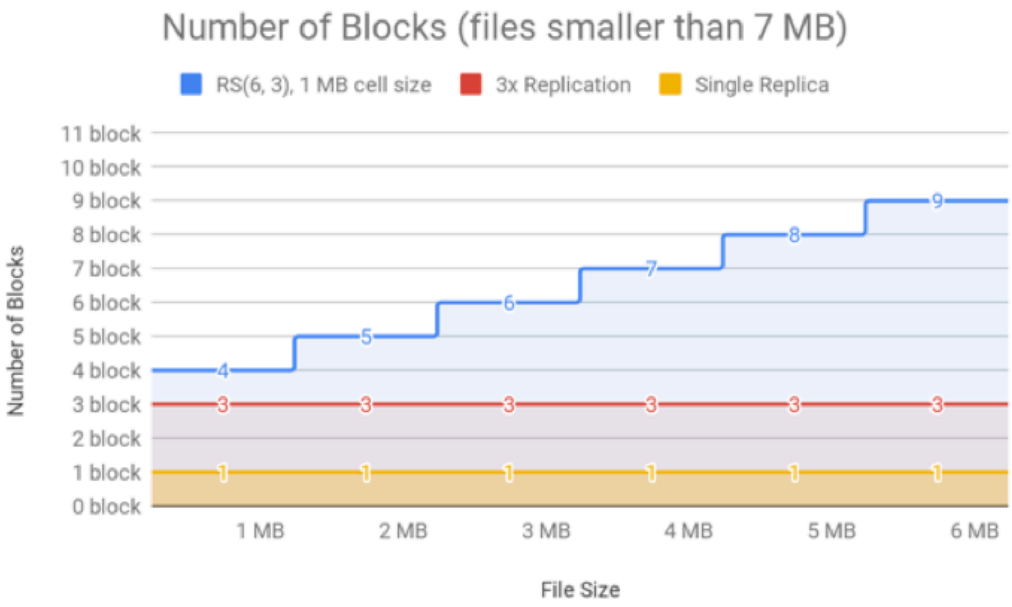
File Size

Erasure coding works best with larger files. The total number of blocks is determined by data blocks + parity blocks, which is the data-stripe width discussed earlier.

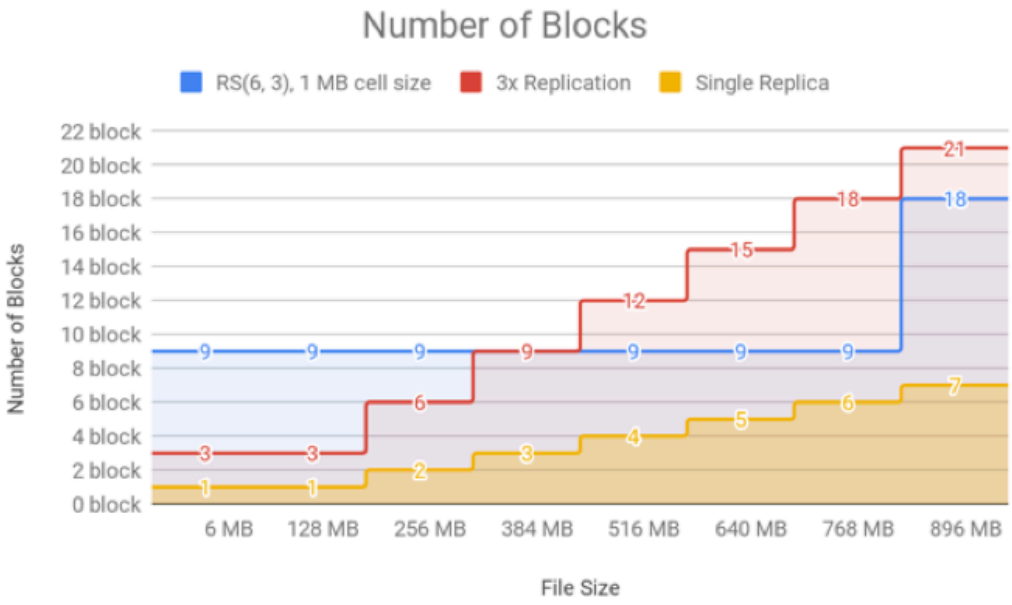
128 MB is the default block size. With RS (6,3), each block group can hold up to (128 MB * 6) = 768 MB of data. Inside each block group, there will be 9 total blocks, 6 data blocks, each holding up to 128 MB, and 3 parity blocks. This is why EC works best with larger files. For a chunk of data less than the block size, replication writes one data block to three DataNodes; EC, on the other hand, still needs to stripe the data to data blocks and calculate parity blocks. This leads to a situation where erasure coded files will generate more blocks than replication because of the parity blocks required. The bytes/blocks ratio is worse for small files which increases the memory usage of the NameNode.

The following two figures show how replication (with a replication factor of 3) compares to EC based on the number of blocks relative to file size.

For very small files of sizes smaller than the value of data blocks * cell size (in case of RS(6, 3) it is (6 * 1) MB), the number of actual data blocks is less than the data blocks defined by the erasure coding policy, though the number of parity blocks is always the same. For example, in the case of RS(6,3) with a cell size of 1 MB, a 1MB file consists of one data block, rather than six, but it still has three parity blocks. A 1MB file would therefore require four block objects in total. If 3x replication were used, the same file would require only three block objects.



As shown in the following figure, the RS(6,3) EC policy ensures that the number of data blocks continues to remain less than that used by 3x replication for larger file sizes.



Supported Engines

Replication supports all data processing engines that CDP supports.

EC supports the following data processing engines: Hive, MapReduce, and Spark.

Unsupported Features

The XOR codec for EC is not supported. Additionally, certain HDFS functions are not supported with EC: hflush, hsync, concat, setReplication, truncate and append. For more information, see Erasure Coding Limitations. and HDFS Erasure Coding Limitations.

Prerequisites for enabling erasure coding

Before enabling erasure coding on your data, you must consider various factors such as the type of policy to use, the type of data, and the rack or node requirements.

- Note the limitations for EC.
- Determine which EC policy you want to use:
- Determine if you want to use EC for existing data or new data. If you want to use EC for existing data, you need to replicate that data with distcp or BDR.
- Verify that your cluster setup meets the rack and node requirements.

Limitations of erasure coding

The limitations of erasure coding include non-support of XOR codecs and certain HDFS functions.

EC does not support the following:

- XOR codecs
- Certain HDFS functions: hflush, hsync, concat, setReplication, truncate and append. For more information, see Erasure Coding Limitations.

Using erasure coding for existing data

You must set a supported EC policy for a directory and copy the existing data to the directory.

Procedure

1. Create a new directory or choose an existing directory.
2. View the supported EC policies.

```
hdfs ec -listPolicies
```

3. Enable a supported EC policy.

```
hdfs ec -enablePolicy -policy <policy>
```

4. Set the EC policy for the directory you want to use.

```
hdfs ec -setPolicy -path <directory> [-policy <policyName>]
```

   - path. Required. Specify the HDFS directory you want to apply the EC policy to.
   - policy. Optional. The EC policy you want to use for the directory you specified. If you do not provide this parameter, the EC policy you specified in the Default Policy when Setting Erasure Coding setting from Cloudera Manager is used.

5. Copy the data to the directory you set an EC policy for.

   You can use the distcp tool or Cloudera Manager's Backup and Disaster Recovery process.

Using erasure coding for new data

You must create a new directory and then set a supported EC policy for the directory.

Procedure

1. Create a new directory or choose an existing directory.
2. View the supported EC policies.

```
hdfs ec -listPolicies
```

3. Enable a supported EC policy.

```
hdfs ec -enablePolicy -policy <policy>
```

4. Set the EC policy for the directory you want to use.

```
hdfs ec -setPolicy -path <directory> [-policy <policyName>]
```

- path. Required. Specify the HDFS directory you want to apply the EC policy to.
- policy. Optional. The EC policy you want to use for the directory you specified. If you do not provide this parameter, the EC policy you specified in the Fallback Erasure Coding Policy setting from Cloudera Manager is used.

5. Set the destination for the data to the directory you enabled EC for. No action beyond that is required.

   When data is written to the directory, it will be erasure coded based on the policy you set.

Advanced erasure coding configuration

You can customize the behavior of EC through a combination of the hdfs ec subcommand and the Cloudera Manager Admin Console.

About this task

This procedure provides information about configuring certain properties for EC with the Cloudera Manager Admin Console.

For information about the hdfs ec see Erasure coding CLI command on page 10.

Procedure

1. Select Clusters and choose the HDFS cluster you want to configure.
2. Navigate to the Configuration tab and select the Erasure Coding category.
3. Configure the advanced EC properties.

   - DataNode Striped Read Timeout: The timeout for striped reads during background data reconstruction in milliseconds.
   - DataNode Striped Read Threads: The number of threads that a DataNode can use during background data reconstruction.
   - Erasure Coding Reconstruction Weight: The relative weight of resources used by EC for data recovery. The number of blocks that must be read is based on the EC policy used. For example, RS-6-3-1024k requires six blocks to be read. Replication only requires one block to be read. Higher values result in fewer reconstruction tasks being able to run concurrently. The number of blocks required to be read to recover data is multiplied by this weight to determine the total weight of the recovery task. The total weight of the recovery task counts against the limit set with the dfs.namenode.replication.max-streams property.
   - Fallback Erasure Coding Policy: The fallback Erasure Coding policy that HDFS uses if no policy is specified when you run the -setPolicy command.

     Note: You must disable erasure coding in small clusters to prevent potential data loss. If the cluster falls back to an Erasure Coding policy value that requires a greater number of DataNodes or racks than available, this situation could result in a potential data loss. As a result, the Erasure Coding Policy Verification Test returns "Concerning" health. To prevent the issue, set the value of the fallback Erasure Coding policy to No Default Erasure Coding Policy. In addition, you must disable all the erasure coding policies that are enabled on the cluster.

Erasure coding CLI command

Use the hdfs ec command to set erasure coding policies on directories.

```
hdfs ec [generic options]
```

```
      [-setPolicy -path <path> [-policy <policyName>] [-replicate]]
      [-getPolicy -path <path>]
      [-unsetPolicy -path <path>]
      [-listPolicies]
      [-addPolicies -policyFile <file>]
      [-listCodecs]
      [-removePolicy -policy <policyName>]
      [-enablePolicy -policy <policyName>]
      [-disablePolicy -policy <policyName>]
      [-help [cmd ...]]
```

Options:

- [-setPolicy [-p <policyName>] <path>]: Sets an EC policy on a directory at the specified path. The following EC policies are supported: RS-3-2-1024k (Reed-Solomon with 3 data blocks, 2 parity blocks and 1024 KB cell size), RS-6-3-1024k, RS-LEGACY-6-3-1024k, and XOR-2-1-1024k.

  <path>: A directory in HDFS. This is a mandatory parameter. Setting a policy only affects newly created files, and does not affect existing files.

  <policyName>: The EC policy to be used for files under the specified directory. This is an optional parameter, specified using the -p flag. If no policy is specified, the system default Erasure Coding policy is used. The default policy is RS-6-3-1024k.
- -replicate: Forces a directory to use the default 3x replication scheme.

  Note: You cannot specify -replicate and -policy <policyName> at the same time. Both the arguments are optional.
- -getPolicy -path <path>: Gets details of the EC policy of a file or directory for the specified path.
- [-unsetPolicy -path <path>]: Removes an EC policy already set by a setPolicy on a directory. This option does not work on a directory that inherits the EC policy from a parent directory. If you run this option on a directory that does not have an explicit policy set, no error is returned.
- [-addPolicies -policyFile <file>]: Adds a list of EC policies. HDFS allows you to add 64 policies in total, with the policy ID in the range of 64 to 127. Adding policies fails if there are already 64 policies.
- [-listCodecs]: Lists all the supported EC erasure coding codecs and coders in the system.
- [-removePolicy -policy <policyName>]: Removes an EC policy.
- [-enablePolicy -policy <policyName>]: Enables an EC policy.
- [-disablePolicy -policy <policyName>]: Disables an EC policy.
- [-help]: Displays help for a given command, or for all commands if none is specified.

Erasure coding background recovery work on the DataNodes can be tuned using the following configuration parameters in hdfs-site.xml.

- dfs.datanode.ec.reconstruction.stripedread.timeout.millis: Timeout for striped reads. Default value is 5000 ms.
- dfs.datanode.ec.reconstruction.threads: Number of threads used by the DataNode for the background recovery task. The default value is 8 threads.
- dfs.datanode.ec.reconstruction.stripedread.buffer.size: Buffer size for reader service. Default value is 64 KB.
- dfs.datanode.ec.reconstruction.xmits.weight: The relative weight of xmits used by the EC background recovery task when compared to replicated block recovery. The default value is 0.5.

  If the parameter is set to 0, the EC task always has one xmit. The xmits of an erasure coding recovery task are calculated as the maximum value between the number of read streams and the number of write streams. For example, if an EC recovery task needs to read from six nodes and write to two nodes, the xmit value is max(6, 2) * 0.5 = 3.

Erasure coding examples

You can use the hdfs ec command with its various options to set erasure coding policies on directories.

Viewing the list of erasure coding policies

The following example shows how you can view the list of available erasure coding policies:

```
hdfs ec -listPolicies
Erasure Coding Policies:
ErasureCodingPolicy=[Name=RS-10-4-1024k, Schema=[ECSchema=[Codec=rs,
 numDataUnits=10, numParityUnits=4]], CellSize=1048576, Id=5], State=DISABLED
ErasureCodingPolicy=[Name=RS-3-2-1024k, Schema=[ECSchema=[Codec=rs,
 numDataUnits=3, numParityUnits=2]], CellSize=1048576, Id=2], State=DISABLED
ErasureCodingPolicy=[Name=RS-6-3-1024k, Schema=[ECSchema=[Codec=rs,
 numDataUnits=6, numParityUnits=3]], CellSize=1048576, Id=1], State=ENABLED
ErasureCodingPolicy=[Name=RS-LEGACY-6-3-1024k, Schema=[ECSchema=[Codec=rs-
legacy, numDataUnits=6, numParityUnits=3]], CellSize=1048576, Id=3],
 State=DISABLED
ErasureCodingPolicy=[Name=XOR-2-1-1024k, Schema=[ECSchema=[Codec=xor,
 numDataUnits=2, numParityUnits=1]], CellSize=1048576, Id=4], State=DISABLED
```

Enabling an erasure coding policy

In the previous example, the list of erasure coding policies indicates that RS-6-3-1024k is already enabled. If required, you can enable additional policies as mentioned in the following example:

```
hdfs ec -enablePolicy -policy RS-3-2-1024k
Erasure coding policy RS-3-2-1024k is enabled
```

Setting an erasure coding policy

The following example shows how you can set the erasure coding policy RS-6-3-1024k on a particular directory:

```
hdfs ec -setPolicy -path /data/dir1 -policy RS-6-3-1024k
Set erasure coding policy RS-6-3-1024k on /data/dir1
```

To confirm whether the specified directory has the erasure coding policy applied, run the hdfs ec -getPolicy command:

```
hdfs ec -getPolicy -path /data/dir1
RS-6-3-1024k
```

Checking the block status on an erasure-coded directory

After enabling erasure coding on a directory, you can check the block status by running the hdfs fsck command. The following example output shows the status of the erasure-coded blocks:

```
hdfs fsck /data/dir1
.
.
.
Erasure Coded Block Groups:
Total size: 434424 B
Total files: 1
Total block groups (validated): 1 (avg. block group size 434424 B)
Minimally erasure-coded block groups: 1 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Average block group size: 4.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Fri Mar 21 19:39:11 UTC 2018 in 1 milliseconds
```

```
The filesystem under path '/data/dir1' is HEALTHY
```

Changing the erasure coding policy

You can use the hdfs ec setPolicy command to change the erasure coding policy applied on a particular directory.

```
hdfs ec -setPolicy -path /data/dir1 -policy RS-3-2-1024k
Set erasure coding policy RS-3-2-1024k on /data/dir1
```

You can check the check the block status after applying the new policy. The following example output shows the status of the erasure-coded blocks for a directory that has the RS-3-2-1024k policy:

```
hdfs fsck /data/dir1
.
.
.
Erasure Coded Block Groups:
Total size: 68644 B
Total files: 2
Total block groups (validated): 2 (avg. block group size 34322 B)
Minimally erasure-coded block groups: 2 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Average block group size: 2.5
Missing block groups:  0
Corrupt block groups:  0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Mon Apr 09 10:11:06 UTC 2018 in 3 milliseconds


The filesystem under path '/data/dir1' is HEALTHY
```

You can apply the default 3x replication policy and check the block status as specified in the following examples:

```
hdfs ec -setPolicy -path /data/dir1 -replicate
Set erasure coding policy replication on /tmp/data1/
Warning: setting erasure coding policy on a non-empty directory will not
 automatically convert existing files to replication
```

```
hdfs fsck /data/dir1
.
.
.
Erasure Coded Block Groups:
Total size: 34322 B
Total files: 1
Total block groups (validated): 1 (avg. block group size 34322 B)
Minimally erasure-coded block groups: 1 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Average block group size: 2.0
Missing block groups:  0
Corrupt block groups:  0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Tue Apr 10 04:34:14 UTC 2018 in 2 milliseconds

The filesystem under path '/data/dir1' is HEALTHY
```

Increasing storage capacity with HDFS compression

Linux supports GzipCodec, DefaultCodec, BZip2Codec, LzoCodec, and SnappyCodec. Typically, GzipCodec is used for HDFS compression.

To configure data compression, you can either enable a data compression codec, for example, GZipCodec, as the default or use the codec from the command line with a one-time job.

Enable GZipCodec as the default compression codec

For the MapReduce framework, update relevant properties in core-site.xml and mapred-site.xml to enable GZipCodec as the default compression codec.

Procedure

1. Edit the core-site.xml file on the NameNode host machine.

```
<property>
  <name>io.compression.codecs</name>
  <value>org.apache.hadoop.io.compress.GzipCodec,
    org.apache.hadoop.io.compress.DefaultCodec,com.hadoop.compression.lzo.
    LzoCodec,org.apache.hadoop.io.compress.SnappyCodec</value>
  <description>A list of the compression codec classes that can be used
    for compression/decompression.</description>
</property>
```

2. Edit the mapred-site.xml file on the JobTracker host machine.

```
<property>
  <name>mapreduce.map.output.compress</name>
  <value>true</value>
</property>

<property>
  <name>mapreduce.map.output.compress.codec</name>
  <value>org.apache.hadoop.io.compress.GzipCodec</value>
</property>

<property>
  <name>mapreduce.output.fileoutputformat.compress.type</name>
  <value>BLOCK</value>
</property>
```

3. Enable the following two configuration parameters to enable job output compression. Edit the mapred-site.xml file on the Resource Manager host machine.

```
<property>
  <name>mapreduce.output.fileoutputformat.compress</name>
  <value>true</value>
</property>

<property>
  <name>mapreduce.output.fileoutputformat.compress.codec</name>
  <value>org.apache.hadoop.io.compress.GzipCodec</value>
</property>
```

4. Restart the cluster.

Use GZipCodec with a one-time job

You can configure GZipcodec to compress the output of a MapReduce job.

Procedure

To use GzipCodec with a one-time only job, add the options to configure compression for the MapReduce job and configure GZipCodec for the output of the job.

```
hadoop jar hadoop-examples-1.1.0-SNAPSHOT.jar sort sbr"-Dmapred.compress.map
.output=true"
sbr"-Dmapred.map.output.compression.codec=org.apache.hadoop.io.compress.G
zipCodec"
sbr "-Dmapred.output.compress=true"
sbr"-Dmapred.output.compression.codec=org.apache.hadoop.io.compress.GzipC
odec"sbr -outKey org.apache.hadoop.io.Textsbr
-outValue org.apache.hadoop.io.Text input output
```

Setting HDFS quotas

As an administrator, you can set up HDFS quotas for the number of file and directory names used and the amount of space used by directories.

Considerations for working with HDFS quotas

- The quotas for names and the quotas for space are independent of each other.
- You cannot create more files and directories if their creation would cause the quotas to exceed.
- Block allocation fails if the quota prevents a full block from being written.
- If you are using replication, each replica of a block counts against the quota.

File count limits

- The file count quota is a limit on the number of file and directory names in the directory configured.
- A directory counts against its own quota. Therefore, a quota value of 1 forces the directory to remain empty.
- File counts are based on the intended replication factor for the files. Changing the replication factor for a file will increase or decrease the corresponding quota values.

Disk space limits

- The space quota is a hard limit on the number of bytes used by files in the tree rooted at the directory being configured.
- Each replica of a block counts against the quota.
- The disk space quota calculation takes replication into account. Therefore, the calculation uses the replicated size of each file and not the user-facing size.
- The disk space quota calculation includes open files (that are being written) and files that already written.
- Block allocations for files being written fail if the quota does not allow a full block to be written.

Related Information

HDFS Quotas Guide

Set up a storage policy for HDFS

Set quotas using Cloudera Manager

You can use set file count or space quotas using Cloudera Manager. You must have administrator privileges to set the quotas.

Procedure

1. From the HDFS service page, select the File Browser tab.
2. Browse the file system to find the directory for which you want to set quotas.
3. Click the directory name so that it appears in the gray panel above the listing of its contents and in the detail section to the right of the File Browser table.
4. Click the Edit Quota button for the directory.

   A Manage Quota pop-up displays, where you can set file count or disk space limits for the directory you have selected.
5. When you have set the limits you want, click OK.

Configuring heterogeneous storage in HDFS

A variety of storage types are supported with HDFS. You can choose which storage type to assign to each DataNode Data Directory. Specifying a storage type allows you to optimize your data usage and lower your costs, based on your data usage frequency.

Each DataNode in a cluster is configured with a set of data directories. You can configure each data directory with a storage type. The storage policy dictates which storage types to use when storing the file or directory.

Some reasons to consider using different types of storage are as follows:

- You have datasets with temporal locality (for example, time-series data). The latest data can be loaded initially into SSD for improved performance, then migrated out to disk as it ages.
- You need to move cold data to denser archival storage because the data will rarely be accessed and archival storage is much cheaper. This could be done with simple age-out policies: for example, moving data older than six months to archival storage.

Set up a storage policy for HDFS

You can use Cloudera Manager to set up storage policy on a DataNode data directory.

Procedure

1. Check the HDFS Service Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml to be sure that dfs .storage.policy.enabled has not been changed from its default value of true.
2. Specify the storage types for each DataNode Data Directory that is not a standard disk, by adding the storage type in brackets at the beginning of the directory path.

   For example:

   ```
   [SSD]/dfs/dn1
   [DISK]/dfs/dn2
   [ARCHIVE]/dfs/dn3
   ```
3. Open a terminal session on any HDFS host and run the `hdfs storagepolicies -setStoragePolicy` command for each path on which you want to set a storage policy.

   ```
   hdfs storagepolicies -setStoragePolicy -path <path> -policy <policy>
   path_to_file_or_directory -policy policy_name
   ```

4. To move the data to the appropriate storage based on the current storage policy, use the `mover` utility, from any HDFS host.

   Use `mover -h` to get a list of available options. To migrate all data at once (this may take a long time), you can set the path to /.

   ```
   hdfs mover -p <path>
   ```

   Note: Quotas are enforced at the time you set the storage policy or when writing the file, not when quotas are changed. The Mover tool does not recognize quota violations. It only verifies that a file is stored on the storage types specified in its policy.

Related Information

Setting HDFS quotas

Set up SSD storage using Cloudera Manager

You can use Cloudera Manager to set up SSD storage for your data directories.

Procedure

1. Set up your cluster normally, but customize your DataNodes with the [ssd] prefix for data directories.

   Adding [ssd] can also be done after initial setup (which requires an extra HDFS restart).
2. Stop HBase.
3. Using the HDFS client, move /hbase to /hbase_backup.
4. Re-create /hbase using the Cloudera Manager command in the HBase service (this ensures that proper permissions are used).
5. Using the HDFS client, set the storage policy for /hbase to be SSD only.
6. Use the DistCp to copy /hbase_backup to /hbase.

   ```
   hadoop distcp /hbase_backup /hbase
   ```
7. Start HBase.

Balancing data across an HDFS cluster

The HDFS Balancer is a tool for balancing the data across the storage devices of a HDFS cluster.

You can also specify the source DataNodes, to free up the spaces in particular DataNodes. You can use a block distribution application to pin its block replicas to particular DataNodes so that the pinned replicas are not moved for cluster balancing.

Why HDFS data becomes unbalanced

Factors such as addition of DataNodes, block allocation in HDFS, and behavior of the client application can lead to the data stored in HDFS clusters becoming unbalanced.

Addition of DataNodes

When new DataNodes are added to a cluster, newly created blocks are written to these DataNodes from time to time. The existing blocks are not moved to them without using the HDFS Balancer.

Behavior of the client application

In some cases, a client application might not write data uniformly across the DataNode machines. A client application might be skewed in writing data, and might always write to some particular machines but not others. HBase is an example of such a client application. In other cases, the client application is not skewed by design, for example, MapReduce or YARN jobs.

The data is skewed so that some of the jobs write significantly more data than others. When a Datanode receives the data directly from the client, it stores a copy to its local storage for preserving data locality. The DataNodes receiving more data generally have higher storage utilization.

Block allocation in HDFS

HDFS uses a constraint satisfaction algorithm to allocate file blocks. Once the constraints are satisfied, HDFS allocates a block by randomly selecting a storage device from the candidate set uniformly. For large clusters, the blocks are essentially allocated randomly in a uniform distribution, provided that the client applications write data to HDFS uniformly across the DataNode machines. Uniform random allocation might not result in a uniform data distribution because of randomness. This is generally not a problem when the cluster has sufficient space. The problem becomes serious when the cluster is nearly full.

Configurations and CLI options for the HDFS Balancer

You can configure the HDFS Balancer by changing various configuration options or by using the command line.

Properties for configuring the Balancer

Depending on your requirements, you can configure various properties for the HDFS Balancer.

dfs.datanode.balance.max.concurrent.moves

Limits the maximum number of concurrent block moves that a DataNode is allowed for balancing the cluster. If you set this configuration in a DataNode, the DataNode throws an exception when the limit is exceeded. If you set this configuration in the HDFS Balancer, the HDFS Balancer schedules concurrent block movements within the specified limit. The DataNode setting and the HDFS Balancer setting can be different. As both settings impose a restriction, an effective setting is the minimum of them.

It is recommended that you set this to the highest possible value in DataNodes and adjust the runtime value in the HDFS Balancer to gain the flexibility. The default value is 100.

You can reconfigure without DataNode restart. Follow these steps to reconfigure a DataNode:

1. Change the value of *dfs.datanode.balance.max.concurrent.moves* from the Configuration tab of the HDFS service from Cloudera Manager.
2. Refresh the cluster.

You can use the default value of 100 as the maximum number of concurrent block moves in most of the situations. If you want to set it to a lower value, you can consider a value between 25 and 50. The recommended maximum value for this parameter is 200.

dfs.datanode.balance.bandwidthPerSec

Limits the bandwidth in each DataNode using for balancing the cluster. Changing this configuration does not require restarting DataNodes.

The default is 100 MB/s.

dfs.balancer.moverThreads

Limits the number of total concurrent moves for balancing in the entire cluster. Set this property to the number of threads in the HDFS Balancer for moving blocks. Each block move requires a thread.

The default is 1000.

dfs.balancer.max-size-to-move

With each iteration, the HDFS Balancer chooses DataNodes in pairs and moves data between the DataNode pairs. Limits the maximum size of data that the HDFS Balancer moves between a chosen DataNode pair. If you increase this configuration when the network and disk are not saturated, increases the data transfer between the DataNode pair in each iteration while the duration of an iteration remains about the same.

The default is 10GB.

dfs.balancer.getBlocks.size

Specifies the total data size of the block list returned by a getBlocks(..).

When the HDFS Balancer moves a certain amount of data between source and destination DataNodes, it repeatedly invokes the getBlocks(..) rpc to the Namenode to get lists of blocks from the source DataNode until the required amount of data is scheduled.

The default is 2GB.

dfs.balancer.getBlocks.min-block-size

Specifies the minimum block size for the blocks used to balance the cluster.

The default is 10MB.

Note: If the majority of files in the cluster have block size smaller than 10MB, then set a lower value for this parameter, because the HDFS Balancer might not be able to find enough blocks greater than this size to move.

dfs.datanode.block-pinning.enabled

Specifies if block-pinning is enabled. When you create a file, a user application can specify a list of favorable DataNodes by way of the file creation API in DistributedFileSystem. The NameNode uses its best effort, allocating blocks to the favorable DataNodes. If dfs.datanode.block-pinning.enabled is set to true, if a block replica is written to a favorable DataNode, it is “pinned” to that DataNode. The pinned replicas are not moved for cluster balancing to keep them stored in the specified favorable DataNodes. This feature is useful for block distribution aware user applications such as HBase.

The default is false.

Balancer commands

You can use various command line options with the `hdfs balancer` command to work with the HDFS Balancer.

Balancing policy, threshold, and blockpools

[-policy <policy>]

Specifies which policy to use to determine if a cluster is balanced.

The two supported policies are blockpool and datanode. Setting the policy to blockpool means that the cluster is balanced if each pool in each node is balanced while datanode means that a cluster is balanced if each DataNode is balanced.

The default policy is datanode.

[-threshold <threshold>]

Specifies a number in [1.0, 100.0] representing the acceptable threshold of the percentage of storage capacity so that storage utilization outside the average +/- the threshold is considered as over/under utilized.

The default threshold is 10.0.

[-blockpools <comma-separated list of blockpool ids>]

Specifies a list of block pools on which the HDFS Balancer runs. If the list is empty, the HDFS Balancer runs on all existing block pools.

The default value is an empty list.

Include and exclude lists

[-include [-f <hosts-file> | <comma-separated list of hosts>]]

When the include list is non-empty, only the DataNodes specified in the list are balanced by the HDFS Balancer. An empty include list means including all the DataNodes in the cluster. The default value is an empty list.

[-exclude [-f <hosts-file> | <comma-separated list of hosts>]]

The DataNodes specified in the exclude list are excluded so that the HDFS Balancer does not balance those DataNodes. An empty exclude list means that no DataNodes are excluded. When a DataNode is specified in both in the include list and the exclude list, the DataNode is excluded. The default value is an empty list.

Idle-iterations and run during upgrade

[-idleiterations <idleiterations>]

Specifies the number of consecutive iterations in which no blocks have been moved before the HDFS Balancer terminates with the NO_MOVE_PROGRESS exit status.

Specify -1 for infinite iterations. The default is 5.

[-runDuringUpgrade]

If specified, the HDFS Balancer runs even if there is an ongoing HDFS upgrade. If not specified, the HDFS Balancer terminates with the UNFINALIZED_UPGRADE exit status.

When there is no ongoing upgrade, this option has no effect. It is usually not desirable to run HDFS Balancer during upgrade. To support rollback, blocks being deleted from HDFS are moved to the internal trash directory in DataNodes and not actually deleted. Running the HDFS Balancer during upgrading cannot reduce the usage of any DataNode storage.

Source DataNodes

[-source [-f <hosts-file> | <comma-separated list of hosts>]]

Specifies the source DataNode list. The HDFS Balancer selects blocks to move from only the specified DataNodes. When the list is empty, all the DataNodes are chosen as a source. The option can be used to free up the space of some particular DataNodes in the cluster. Without the -source option, the HDFS Balancer can be inefficient in some cases.

The default value is an empty list.

The following table shows an example, where the average utilization is 25% so that D2 is within the 10% threshold. It is unnecessary to move any blocks from or to D2. Without specifying the source nodes, HDFS Balancer first moves blocks from D2 to D3, D4 and D5, since they are under the same rack, and then moves blocks from D1 to D2, D3, D4 and D5. By specifying D1 as the source node, HDFS Balancer directly moves blocks from D1 to D3, D4 and D5.

Table 1: Example of utilization movement

Datanodes (with the same capacity)	Utilization	Rack
D1	95%	A
D2	30%	B
D3, D4, and D5	0%	B

Recommended configurations for the Balancer

The HDFS Balancer can run in either Background or Fast modes. Depending on the mode in which you want the Balancer to run, you can set various properties to recommended values.

Background and fast modes

HDFS Balancer runs as a background process. The cluster serves other jobs and applications at the same time.

Fast Mode

HDFS Balancer runs at maximum (fast) speed.

Table 2: DataNode configuration properties

Property	Default	Background Mode	Fast Mode
dfs.datanode.balance.-max.concurrent.moves	5	4 x (# of disks)	4 x (# of disks)
dfs.datanode.balance.-max.bandwidthPerSec	1048576 (1 MB)	use default	10737418240 (10 GB)

Table 3: Balancer configuration properties

Property	Default	Background Mode	Fast Mode
dfs.datanode.balance.-max.concurrent.moves	5	# of disks	4 x (# of disks)
dfs.balancer.-moverThreads	1000	use default	20,000
dfs.balancer.-max-size-to-move	10737418240 (10 GB)	1073741824 (1GB)	107374182400 (100 GB)
dfs.balancer.-getBlocks.min-block-size	10485760 (10 MB)	use default	104857600 (100 MB)

Troubleshooting tip

When the block report length exceeds the configured limit, you might receive the following exception in the NameNode logs:

```
java.io.IOException: Requested data length 141557760 is longer than maximum
configured RPC length 134217728
```

Causes

When there are too many blocks in one or more Datanodes, for example in the range of several millions or 10M+, it leads to a large size of block report from those particular Datanodes, and the block report length exceeds the default value of 128M. In that case the NameNode only receives a part of block report instead of the entire bock report as the block report gets truncated at the default configured length of 128M. Since the HDFS Balancer receives block information from NameNode, with wrong information of block number the balancer might not move the blocks from the Datanode with the above issue.

Solutions

Increase the value of ipc.maximum.data.length to 256M or above the reported data length from the above exception, so that the NameNode can receive a large size of block report. The balancer would then be able to progress further.

Note: If the above mentioned exception is sustained in the NameNode logs for a prolonged period of time, then the property modification would be required regardless of whether the balancer needs to be run or not and even for normal and correct HDFS operations.

Configuring and running the HDFS balancer using Cloudera Manager

Learn how to run HDFS Balancer by using Cloudera Manager. You configure the balancer threshold, concurrent moves, and block size. Then you run the HDFS Balancer. You also learn about recommended values for the HDFS Balancer.

In Cloudera Manager, the HDFS balancer utility is implemented by the Balancer role. The Balancer role usually shows a health of None on the HDFS Instances tab because it does not run continuously.

The Balancer role is normally added (by default) when the HDFS service is installed. If it has not been added, you must add a Balancer role to rebalance HDFS and to see the Rebalance action.

Configuring the balancer threshold

Learn how to configure the Balancer Threshold property, before using HDFS Balancer feature.

About this task

The Balancer has a default threshold of 10%, which ensures that disk usage on each DataNode differs from the overall usage in the cluster by no more than 10%. For example, if overall usage across all the DataNodes in the cluster is 40% of the cluster's total disk-storage capacity, the script ensures that DataNode disk usage is between 30% and 50% of the DataNode disk-storage capacity. To change the threshold, perform the following steps:

Procedure

1. Got to the HDFS service in Cloudera Manager.
2. Select the Configuration tab.
3. Select Scope Balancer .
4. Select Category Main .
5. Set the Rebalancing Threshold property.

   To apply this configuration property to other role groups as needed, edit the value for the appropriate role group. For more information, see Modifying Configuration Properties Using Cloudera Manager.
6. Enter a reason for change, and then click Save Changes.

Configuring concurrent moves

Learn how to configure concurrent moves for the DataNode and Balancer to prevent the balancer from taking too many resources from the DataNode and interfering with normal cluster operations.

About this task

The property dfs.datanode.balance.max.concurrent.moves sets the maximum number of threads used by the DataNode balancer for pending moves. It is a throttling mechanism to prevent the balancer from taking too many resources from the DataNode and interfering with normal cluster operations. Increasing the value allows the balancing process to complete more quickly. Decreasing the value allows rebalancing to complete more slowly, but is less likely to compete for resources with other tasks on the DataNode. To use this property, you need to set the value on both the DataNode and the Balancer.

Procedure

1. Configure concurrent moves on the DataNode.
   a) Go to the HDFS service.
   b) Select the Configuration tab.
   c) Search for DataNode Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml.
   d) Add the following code to the configuration field, for example, setting the value to 50.

```
<property>
  <name>dfs.datanode.balance.max.concurrent.moves</name>
  <value>50</value>
```

```
</property>
```

e) Click Save Changes.
f) Restart the DataNode.

2. Configure concurrent moves on the Balancer.
   a) Go to the HDFS service.
   b) Select the Configuration tab.
   c) Search for Balancer Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml.
   d) Add the following code to the configuration field, for example, setting the value to 50.

```
<property>
  <name>dfs.datanode.balance.max.concurrent.moves</name>
  <value>50</value>
</property>
```

   e) Click Save Changes.

Recommended configurations for the balancer

Depending on whether you want the HDFS Balancer to run in the background or at maximum speed, you can configure the values of certain properties through safety valves.

Property	Values for Running the Balancer in the Background	Value for Running the Balancer at Maximum Speed
DataNode		
dfs.datanode.balance.bandwidthPerSec	10 MB	10 GB
Balancer		
dfs.balancer.moverThreads	1000	20000
dfs.balancer.max-size-to-move	10 GB	100 GB
dfs.balancer.getBlocks.min-block-size	10 MB	100 MB

Running the balancer

Learn how to run the HDFS Balancer.

Procedure

1. Go to the HDFS service.
2. Ensure that the service has a Balancer role.
3. Select Actions Rebalance .
4. Click Rebalance to confirm.

   If you see a Finished status, the Balancer ran successfully.

Configuring block size

Learn how to configure the block metadata batch size and minimum block size for HDFS.

About this task

The Block Metadata Batch Size property configures the amount of block metadata that gets retrieved. The Minimum Block Size property configures the smallest block to consider for moving. Tuning these properties can improve performance during balancing.

Procedure

1. In the Cloudera Manager Admin Console, select Clusters <HDFS cluster> .
2. Select the Configuration tab.

3. Search for the following properties:
   - Block Metadata Batch Size (dfs.balancer.getBlocks.size)
   - Minimum Block Size (dfs.balancer.getBlocks.min-block-size)
4. Set the values for each, and click Save Changes.

Cluster balancing algorithm

The HDFS Balancer runs in iterations. Each iteration contains the following four steps: storage group classification, storage group pairing, block move scheduling, and block move execution.

Storage group classification

The HDFS Balancer first invokes the `getLiveDatanodeStorageReport` rpc to the Namenode to the storage report for all the storages in all Datanodes. The storage report contains storage utilization information such as capacity, dfs used space, remaining space, and so forth, for each storage in each DataNode.

A Datanode can contain multiple storages and the storages can have different storage types. A storage group Gi,T is defined to be the group of all the storages with the same storage type T in Datanode i. For example, Gi,DISK is the storage group of all the DISK storages in Datanode i. For each storage type T in each DataNode i, HDFS Balancer computes Storage Group Utilization (%)

Ui,T = 100% (storage group used space)/(storage group capacity),

and Average Utilization (%)

Uavg,T = 100% * (sum of all used spaces)/(sum of all capacities).

Let # be the threshold parameter (default is 10%) and GI,T be the storage group with storage type T in DataNode I.

```
                    Over-Utilized:       {Gi,T :  Uavg,T + # < Ui,T},
Average + Threshold ------------------------------------------------------
                    Above-Average: {Gi,T :  Uavg,T < Ui,T <= Uavg,T + #},
Average -------------------------------------------------------------------
                    Below-Average: {Gi,T :  Uavg,T - # <= Ui,T <= Uavg,T},
Average - Threshold ------------------------------------------------------
                    Under-Utilized:      {Gi,T :  Ui,T < Uavg,T - # }.
```

A storage group is over-utilized or under-utilized if its utilization is larger or smaller than the difference between the average and the threshold. A storage group is above-average or below-average if its utilization is larger or smaller than average but within the threshold.

If there are no over-utilized storages and no under-utilized storages, the cluster is said to be balanced. The HDFS Balancer terminates with a SUCCESS state. Otherwise, it continues with storage group pairing.

Storage group pairing

The HDFS Balancer selects over-utilized or above-average storage as source storage, and under-utilized or below-average storage as target storage. It pairs a source storage group with a target storage group (source # target) in a priority order depending on whether or not the source and the target storage reside in the same rack.

The Balancer uses the following priority orders for pairing storage groups from the source and the target.

- Same-Rack (where the source and the target storage reside in the same rack)

  Over-Utilized # Under-Utilize

  Over-Utilized # Below-Average

  Above-Average # Under-Utilized

- Any (where the source and target storage do not reside in the same rack)

  Over-Utilized # Under-Utilized

  Over-Utilized # Below-Average

  Above-Average # Under-Utilized

Block move scheduling

For each source-target pair, the HDFS Balancer chooses block replicas from the source storage groups and schedules block moves.

A block replica in a source DataNode is a good candidate if it satisfies all of the following conditions:

- The storage type of the block replica in the source DataNode is the same as the target storage type.
- The storage type of the block replica is not already scheduled.
- The target does not already have the same block replica.
- The number of racks of the block is not reduced after the move.

Logically, the HDFS Balancer schedules a block replica to be “moved” from a source storage group to a target storage group. In practice, a block usually has multiple replicas. The block move can be done by first copying the replica from a proxy, which can be any storage group containing one of the replicas of the block, to the target storage group, and then deleting the replica in the source storage group.

After a candidate block in the source DataNode is specified, the HDFS Balancer selects a storage group containing the same replica as the proxy. The HDFS Balancer selects the closest storage group as the proxy in order to minimize the network traffic.

When it is impossible to schedule a move, the HDFS Balancer terminates with a NO_MOVE_BLOCK exit status.

Block move execution

The HDFS Balancer dispatches a scheduled block move by invoking the DataTransferProtocol.replaceBlock(..) method to the target DataNode.

The Balancer specifies the proxy, and the source as delete-hint in the method call. The target DataNode copies the replica directly from the proxy to its local storage. When the copying process has been completed, the target DataNode reports the new replica to the NameNode with the delete-hint. NameNode uses delete-hint to delete the extra replica, that is, delete the replica stored in the source.

After all block moves are dispatched, the HDFS Balancer waits until all the moves are completed. Then, the HDFS Balancer continues running a new iteration and repeats all of the steps. If the scheduled moves fail for 5 consecutive iterations, the HDFS Balancer terminates with a NO_MOVE_PROGRESS exit status.

Exit statuses for the HDFS Balancer

The HDFS Balancer concludes a cluster balancing operation with a specific exit status that indicates whether the operation succeeded or failed, with supporting reasons.

Table 4: Exit statuses for the HDFS balancer

Status	Value	Description
SUCCESS	0	The cluster is balanced. There are no over or under-utilized storages, with regard to the specified threshold.
ALREADY_RUNNING	-1	Another HDFS Balancer is running.
NO_MOVE_BLOCK	-2	The HDFS Balancer is not able to schedule a move.
NO_MOVE_PROGRESS	-3	All of the scheduled moves have failed for 5 consecutive iterations.
IO_EXCEPTION	-4	An IOException occurred.

Status	Value	Description
ILLEGAL_ARGUMENTS	-5	An illegal argument in the command or configuration occurred.
INTERUPTED	-6	The HDFS Balancer process was interrupted.
UNFINALIZED_UPGRADE	-7	The cluster is being upgraded.

Optimizing performance

You can consider the following options to optimize the performance of an HDFS cluster: swapping disk drives on a DataNode, caching data, configuring rack awareness, customizing HDFS, optimizing NameNode disk space with Hadoop archives, identifying slow DataNodes and improving them, optimizing small write operations by using DataNode memory as storage, and implementing short-circuit reads.

Improving performance with centralized cache management

Centralized cache management enables you to specify paths to directories that are cached by HDFS, thereby improving performance for applications that repeatedly access the same data.

Centralized cache management in HDFS is an explicit caching mechanism. The NameNode communicates with DataNodes that have the required data blocks available on disk, and instructs the DataNodes to cache the blocks in off-heap caches.

Benefits of centralized cache management in HDFS

Centralized cache management in HDFS offers many significant advantages such as explicit pinning, querying cached blocks for task placement, and improving cluster memory utilization.

- Explicit pinning prevents frequently used data from being evicted from memory. This is particularly important when the size of the working set exceeds the size of main memory, which is common for many HDFS workloads.
- Because DataNode caches are managed by the NameNode, applications can query the set of cached block locations when making task placement decisions. Co-locating a task with a cached block replica improves read performance.
- When a block has been cached by a DataNode, clients can use a more efficient zero-copy read API. Since checksum verification of cached data is done once by the DataNode, clients can incur essentially zero overhead when using this new API.
- Centralized caching can improve overall cluster memory utilization. When relying on the operating system buffer cache on each DataNode, repeated reads of a block will result in all n replicas of the block being pulled into the buffer cache. With centralized cache management, you can explicitly pin only m of the n replicas, thereby saving n-m memory.

Use cases for centralized cache management

Centralized cache management is useful for files that are accessed repeatedly and for mixed workloads that have performance SLAs.

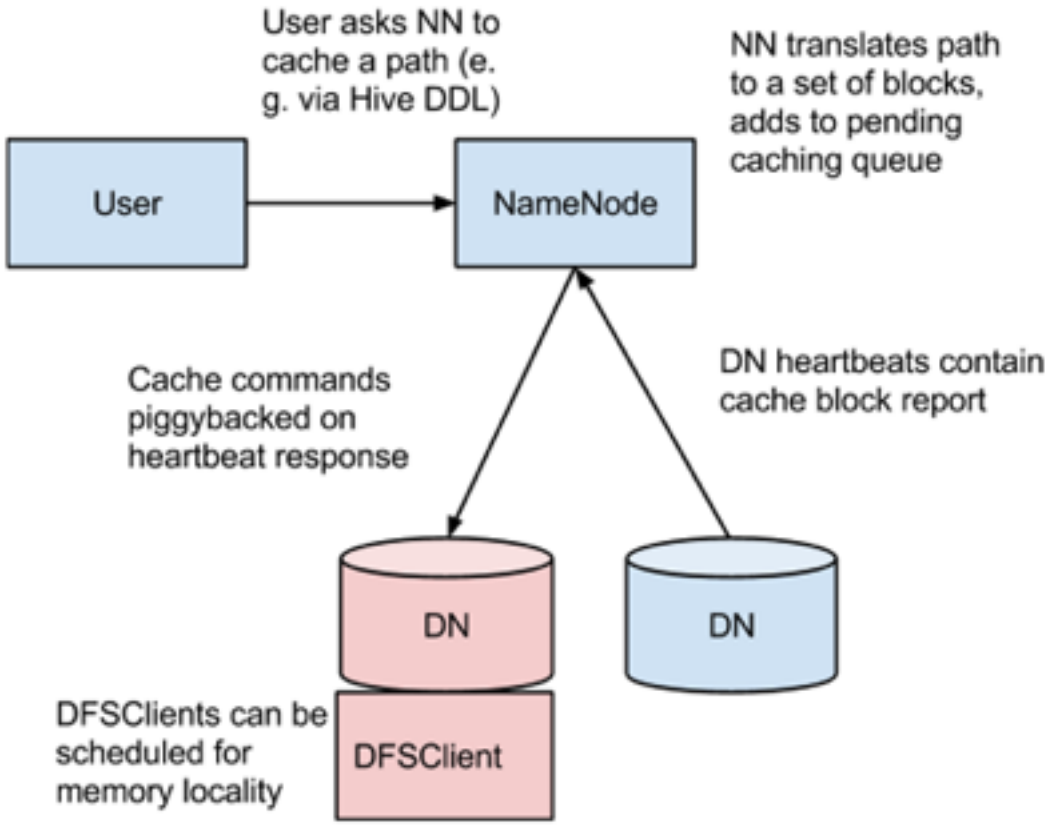
- Files that are accessed repeatedly: For example, a small fact table in Hive that is often used for joins is a good candidate for caching. Conversely, caching the input of a once-yearly reporting query is probably less useful, since the historical data might only be read once.
- Mixed workloads with performance SLAs: Caching the working set of a high priority workload ensures that it does not compete with low priority workloads for disk I/O.

Centralized cache management architecture

In a centralized cache management, the NameNode is responsible for coordinating all of the DataNode off-heap caches in the cluster. The NameNode periodically receives a cache report from each DataNode. The cache report

describes all of the blocks cached on the DataNode. The NameNode manages DataNode caches by piggy-backing cache and uncache commands on the DataNode heartbeat.

The following figure illustrates the centralized cached management architecture.



The NameNode queries its set of cache directives to determine which paths should be cached. Cache directives are persistently stored in the fsimage and edit logs, and can be added, removed, and modified through Java and command-line APIs. The NameNode also stores a set of cache pools, which are administrative entities used to group cache directives together for resource management, and to enforce permissions.

The NameNode periodically re-scans the namespace and active cache directives to determine which blocks need to be cached or uncached, and assigns caching work to DataNodes. Re-scans can also be triggered by user actions such as adding or removing a cache directive or removing a cache pool.

Cache blocks that are under construction, corrupt, or otherwise incomplete are not cached. If a Cache directive covers a symlink, the symlink target is not cached.

Caching can only be applied to directories and files.

Related Information

Caching terminology

Caching terminology

A cache directive defines the path to cache while a cache pool manages groups of cache directives.

Cache directive

Defines the path to be cached. Paths can point either directories or files. Directories are cached non-recursively, meaning only files in the first-level listing of the directory will be cached.

Cache directives also specify additional parameters, such as the cache replication factor and expiration time. The replication factor specifies the number of block replicas to cache. If multiple cache directives refer to the same file, the maximum cache replication factor is applied.

The expiration time is specified on the command line as a time-to-live (TTL), which represents a relative expiration time in the future. After a cache directive expires, it is no longer taken into consideration by the NameNode when making caching decisions.

Cache pool

An administrative entity that manages groups of cache cirectives. Cache pools have UNIX-like permissions that restrict which users and groups have access to the pool. Write permissions allow users to add and remove cache directives to the pool. Read permissions allow users to list the Cache Directives in a pool, as well as additional metadata. Execute permissions are unused.

Cache pools are also used for resource management. Cache pools can enforce a maximum memory limit, which restricts the aggregate number of bytes that can be cached by directives in the pool. Normally, the sum of the pool limits will approximately equal the amount of aggregate memory reserved for HDFS caching on the cluster. Cache pools also track a number of statistics to help cluster users track what is currently cached, and to determine what else should be cached.

Cache pools can also enforce a maximum time-to-live. This restricts the maximum expiration time of directives being added to the pool.

Related Information

Centralized cache management architecture

Properties for configuring centralized caching

You must enable JNI to use centralized caching. In addition, you must configure various properties and consider the locked memory limit for configuring centralized caching.

Native libraries

In order to lock block files into memory, the DataNode relies on native JNI code found in libhadoop.so. Be sure to enable JNI if you are using HDFS centralized cache management.

Configuration properties

Configuration properties for centralized caching are specified in the hdfs-site.xml file.

Required properties

Only the following property is required:

- dfs.datanode.max.locked.memory This property determines the maximum amount of memory (in bytes) that a DataNode will use for caching. The "locked-in-memory size" ulimit (ulimit -l) of the DataNode user also needs to be increased to exceed this parameter (for more details, see the following section on). When setting this value, remember that you will need space in memory for other things as well, such as the DataNode and application JVM heaps, and the operating system page cache. Example:

```
<property>
  <name>dfs.datanode.max.locked.memory</name>
  <value>268435456</value>
</property>
```

Optional properties

The following properties are not required, but can be specified for tuning.

- dfs.namenode.path.based.cache.refresh.interval.ms The NameNode will use this value as the number of milliseconds between subsequent cache path re-scans. By default, this parameter is set to 300000, which is five minutes. Example:

```
<property>
  <name>dfs.namenode.path.based.cache.refresh.interval.ms</name>
  <value>300000</value>
</property>
```

- dfs.time.between.resending.caching.directives.ms The NameNode will use this value as the number of milliseconds between resending caching directives. Example:

```
<property>
  <name>dfs.time.between.resending.caching.directives.ms</name>
  <value>300000</value>
</property>
```

- dfs.datanode.fsdatasetcache.max.threads.per.volume The DataNode will use this value as the maximum number of threads per volume to use for caching new data. By default, this parameter is set to 4. Example:

```
<property>
  <name>dfs.datanode.fsdatasetcache.max.threads.per.volume</name>
  <value>4</value>
</property>
```

- dfs.cachereport.intervalMsec The DataNode will use this value as the number of milliseconds between sending a full report of its cache state to the NameNode. By default, this parameter is set to 10000, which is 10 seconds. Example:

```
<property>
  <name>dfs.cachereport.intervalMsec</name>
  <value>10000</value>
</property>
```

- dfs.namenode.path.based.cache.block.map.allocation.percent The percentage of the Java heap that will be allocated to the cached blocks map. The cached blocks map is a hash map that uses chained hashing. Smaller maps may be accessed more slowly if the number of cached blocks is large. Larger maps will consume more memory. The default value is 0.25 percent. Example:

```
<property>
  <name>dfs.namenode.path.based.cache.block.map.allocation.percent</name>
  <value>0.25</value>
</property>
```

OS limits

If you get the error "Cannot start datanode because the configured max locked memory size...is more than the datanode's available RLIMIT_MEMLOCK ulimit," this means that the operating system is imposing a lower limit on the amount of memory that you can lock than what you have configured. To fix this, you must adjust the ulimit -l value that the DataNode runs with. This value is usually configured in /etc/security/limits.conf, but this may vary depending on what operating system and distribution you are using.

You have correctly configured this value when you can run ulimit - l from the shell and get back either a higher value than what you have configured or the string "unlimited", which indicates that there is no limit. Typically, ulimit -l returns the memory lock limit in kilobytes (KB), but dfs.datanode.max.locked.memory must be specified in bytes.

For example, if the value of dfs.datanode.max.locked.memory is set to 128000 bytes:

```
<property>
```

```
  <name>dfs.datanode.max.locked.memory</name>
  <value>128000</value>
</property>
```

Set the memlock (max locked-in-memory address space) to a slightly higher value. For example, to set memlock to 130 KB (130,000 bytes) for the hdfs user, you would add the following line to /etc/security/limits.conf.

```
hdfs - memlock 130
```

Commands for using cache pools and directives

You can use the Command-Line Interface (CLI) to create, modify, and list cache pools and cache directives using the hdfs cacheadmin subcommand.

Cache Directives are identified by a unique, non-repeating, 64-bit integer ID. IDs will not be reused even if a Cache Directive is removed.

Cache Pools are identified by a unique string name.

You must first create a Cache Pool, and then add Cache Directives to the Cache Pool.

Cache Pool Commands

- addPool -- Adds a new Cache Pool.

  Usage:

  ```
  hdfs cacheadmin -addPool <name> [-owner <owner>] [-group <group>]
  [-mode <mode>] [-limit <limit>] [-maxTtl <maxTtl>]
  ```

  Options:

  Table 5: Cache Pool Add Options

Option	Description
<name>	The name of the pool.
<owner>	The user name of the owner of the pool. Defaults to the current user.
<group>	The group that the pool is assigned to. Defaults to the primary group name of the current user.
<mode>	The UNIX-style permissions assigned to the pool. Permissions are specified in octal (e.g. 0755). Pool permissions are set to 0755 by default.
<limit>	The maximum number of bytes that can be cached by directives in the pool, in aggregate. By default, no limit is set.
<maxTtl>	The maximum allowed time-to-live for directives being added to the pool. This can be specified in seconds, minutes, hours, and days (e.g. 120s, 30m, 4h, 2d). Valid units are [smhd]. By default, no maximum is set. A value of "never" specifies that there is no limit.

- modifyPool -- Modifies the metadata of an existing Cache Pool.

  Usage:

  ```
  hdfs cacheadmin -modifyPool <name> [-owner <owner>] [-group <group>]
  ```

```
[-mode <mode>] [-limit <limit>] [-maxTtl <maxTtl>]
```

Options:

Table 6: Cache Pool Modify Options

Option	Description
<name>	The name of the pool to modify.
<owner>	The user name of the owner of the pool.
<group>	The group that the pool is assigned to.
<mode>	The UNIX-style permissions assigned to the pool. Permissions are specified in octal (e.g. 0755).
<limit>	The maximum number of bytes that can be cached by directives in the pool, in aggregate.
<maxTtl>	The maximum allowed time-to-live for directives being added to the pool. This can be specified in seconds, minutes, hours, and days (e.g. 120s, 30m, 4h, 2d). Valid units are [smdh]. By default, no maximum is set. A value of "never" specifies that there is no limit.

- removePool -- Removes a Cache Pool. This command also "un-caches" paths that are associated with the pool.

  Usage:

  ```
  hdfs cacheadmin -removePool <name>
  ```

  Options:

  Table 7: Cache Pool Remove Options

Option	Description
<name>	The name of the Cache Pool to remove.

- listPools -- Displays information about one or more Cache Pools, such as name, owner, group, permissions, and so on.

  Usage:

  ```
  hdfs cacheadmin -listPools [-stats] [<name>]
  ```

  Options:

  Table 8: Cache Pools List Options

Option	Description
-stats	Displays additional Cache Pool statistics.
<name>	If specified, lists only the named Cache Pool.

- help -- Displays detailed information about a command.

  Usage:

  ```
  hdfs cacheadmin -help <command-name>
  ```

  Options:

  Table 9: Cache Pool Help Options

Option	Description
<command-name	Displays detailed information for the specified command name. If no command name is specified, detailed help is displayed for all commands.

Cache Directive Commands

- addDirective -- Adds a new Cache Directive.

  Usage:

  ```
  hdfs cacheadmin -addDirective -path <path> -pool <pool-name> [-force]
  [-replication <replication>] [-ttl <time-to-live>]
  ```

  Options:

  Table 10: Cache Pool Add Directive Options

Option	Description
<path>	The path to the cache directory or file.
<pool-name>	The Cache Pool to which the Cache Directive will be added. You must have Write permission for the Cache Pool in order to add new directives.
-force	Skips checking of the Cache Pool resource limits.
<replication>	The cache replication factor to use. Default setting is 1.
<time-to-live>	How long the directive is valid. This can be specified in minutes, hours and days (e.g. 30m, 4h, 2d). Valid units are [smdh]. A value of "never" indicates a directive that never expires. If unspecified, the directive never expires.

- removeDirective -- Removes a Cache Directive.

  Usage:

  ```
  hdfs cacheadmin -removeDirective <id>
  ```

  Options:

  Table 11: Cache Pool Remove Directive Options

Option	Description
<id>	The ID of the Cache Directive to remove. You must have Write permission for the pool that the directive belongs to in order to remove it. You can use the -listDirectives command to display a list of Cache Directive IDs.

- removeDirectives -- Removes all of the Cache Directives in a specified path.

  Usage:

  ```
  hdfs cacheadmin -removeDirectives <path>
  ```

  Options:

  Table 12: Cache Pool Remove Directives Options

Option	Description
<path>	The path of the Cache Directives to remove. You must have Write permission for the pool that the directives belong to in order to remove them. You can use the -listDirectives command to display a list of Cache Directives.

- listDirectives -- Returns a list of Cache Directives.

  Usage:

  ```
  hdfs cacheadmin -listDirectives [-stats] [-path <path>] [-pool <pool>]
  ```

  Options:

  Table 13: Cache Pools List Directives Options

Option	Description
<path>	Lists only the Cache Directives in the specified path. If there is a Cache Directive in the <path> that belongs to a Cache Pool for which you do not have Read access, it will not be listed.
<pool>	Lists on the Cache Directives in the specified Cache Pool.
-stats	Lists path-based Cache Directive statistics.

Customizing HDFS

You can use the dfs.user.home.base.dir property to customize the HDFS home directory. In addition, you can configure properties to control the size of the directory that holds the NameNode edits directory.

Customize the HDFS home directory

By default, the HDFS home directory is set to /user/<user_name>. Use the dfs.user.home.base.dir property to customize the HDFS home directory.

Procedure

In hdfs-site.xml file, set the value of the dfs.user.home.base.dir property.

```
<property>
  <name>dfs.user.home.dir.prefix</name>
  <value>/user</value>
  <description>Base directory of user home.</description>
</property>
```

In the example, <value> is the path to the new home directory.

Properties to set the size of the NameNode edits directory

You can configure the dfs.namenode.num.checkpoints.retained and dfs.namenode.num.extra.edits.retained properties to control the size of the directory that holds the NameNode edits directory.

- dfs.namenode.num.checkpoints.retained: The number of image checkpoint files that are retained in storage directories. All edit logs necessary to recover an up-to-date namespace from the oldest retained checkpoint are also retained.
- dfs.namenode.num.extra.edits.retained: The number of extra transactions that should be retained beyond what is minimally necessary for a NameNode restart. This can be useful for audit purposes, or for an HA setup where a remote Standby Node may have been offline for some time and require a longer backlog of retained edits in order to start again.

Optimizing NameNode disk space with Hadoop archives

Hadoop Archives (HAR) are special format archives that efficiently pack small files into HDFS blocks.

The Hadoop Distributed File System (HDFS) is designed to store and process large data sets, but HDFS can be less efficient when storing a large number of small files. When there are many small files stored in HDFS, these small files occupy a large portion of the namespace. As a result, disk space is under-utilized because of the namespace limitation.

Hadoop Archives (HAR) can be used to address the namespace limitations associated with storing many small files. A Hadoop Archive packs small files into HDFS blocks more efficiently, thereby reducing NameNode memory usage while still allowing transparent access to files. Hadoop Archives are also compatible with MapReduce, allowing transparent access to the original files by MapReduce jobs.

Overview of Hadoop archives

Storing a large number of small files in HDFS leads to inefficient utilization of space – the namespace is overutilized while the disk space might be underutilized. Hadoop Archives (HAR) address this limitation by efficiently packing small files into large files without impacting the file access.

The Hadoop Distributed File System (HDFS) is designed to store and process large (terabytes) data sets. For example, a large production cluster may have 14 PB of disk space and store 60 million files.

However, storing a large number of small files in HDFS is inefficient. A file is generally considered to be "small" when its size is substantially less than the HDFS block size. Files and blocks are name objects in HDFS, meaning that they occupy namespace (space on the NameNode). The namespace capacity of the system is therefore limited by the physical memory of the NameNode.

When there are many small files stored in the system, these small files occupy a large portion of the namespace. As a consequence, the disk space is underutilized because of the namespace limitation. In one real-world example, a production cluster had 57 million files less than 256 MB in size, with each of these files taking up one block on the NameNode. These small files used up 95% of the namespace but occupied only 30% of the cluster disk space.

Hadoop Archives (HAR) can be used to address the namespace limitations associated with storing many small files. HAR packs a number of small files into large files so that the original files can be accessed transparently (without expanding the files).

HAR increases the scalability of the system by reducing the namespace usage and decreasing the operation load in the NameNode. This improvement is orthogonal to memory optimization in the NameNode and distributing namespace management across multiple NameNodes.

Hadoop Archive is also compatible with MapReduce — it allows parallel access to the original files by MapReduce jobs.

Hadoop archive components

You can use the Hadoop archiving tool to create Hadoop Archives (HAR). The Hadoop Archive is integrated with the Hadoop file system interface. Files in a HAR are exposed transparently to users. File data in a HAR is stored in multipart files, which are indexed to retain the original separation of data.

Hadoop archiving tool

Hadoop Archives can be created using the Hadoop archiving tool. The archiving tool uses MapReduce to efficiently create Hadoop Archives in parallel. The tool can be invoked using the command:

```
hadoop archive -archiveName name -p <parent> <src>* <dest>
```

A list of files is generated by traversing the source directories recursively, and then the list is split into map task inputs. Each map task creates a part file (about 2 GB, configurable) from a subset of the source files and outputs the metadata. Finally, a reduce task collects metadata and generates the index files.

HAR file system

Most archival systems, such as tar, are tools for archiving and de-archiving. Generally, they do not fit into the actual file system layer and hence are not transparent to the application writer in that the archives must be expanded before use.

The Hadoop Archive is integrated with the Hadoop file system interface. The HarFileSystem implements the FileSyst em interface and provides access via the har:// scheme. This exposes the archived files and directory tree structures transparently to users. Files in a HAR can be accessed directly without expanding them.

For example, if we have the following command to copy an HDFS file to a local directory:

```
hdfs dfs –get hdfs://namenode/foo/file-1 localdir
```

Suppose a Hadoop Archive bar.har is created from the foo directory. With the HAR, the command to copy the original file becomes:

```
hdfs dfs –get har://namenode/bar.har/foo/file-1 localdir
```

Users only need to change the URI paths. Alternatively, users may choose to create a symbolic link (from hdfs ://namenode/foo to har://namenode/bar.har/foo in the example above), and then even the URIs do not need to be changed. In either case, HarFileSystem will be invoked automatically to provide access to the files in the HAR. Because of this transparent layer, HAR is compatible with the Hadoop APIs, MapReduce, the FS shell command-line interface, and higher-level applications such as Pig, Zebra, Streaming, Pipes, and DistCp.

HAR format data model

The Hadoop Archive data format has the following layout:

```
foo.har/_masterindex //stores hashes and offsets
foo.har/_index //stores file statuses
foo.har/part-[1..n] //stores actual file data
```

The file data is stored in multipart files, which are indexed in order to retain the original separation of data. Moreover, the file parts can be accessed in parallel by MapReduce programs. The index files also record the original directory tree structures and file status.

Create a Hadoop archive

Use the `hadoop archive` command to invoke the Hadoop archiving tool.

Procedure

Run the hadoop archive command by specifying the archive name to create, the parent directory relative to the archive location, the source files to archive, and the destination archive location.

```
hadoop archive -archiveName name -p <parent> <src>* <dest>
```

The archive name must have a .har extension

Note:

- Archiving does not delete the source files. If you want to delete the input files after creating an archive to reduce namespace, you must manually delete the source files.
- Although the `hadoop archive` command can be run from the host file system, the archive file is created in the HDFS file system from directories that exist in HDFS. If you reference a directory on the host file system and not HDFS, the system displays the following error:

```
The resolved paths set is empty. Please check whether the srcPaths e
xist, where srcPaths
 = [</directory/path>]
```

Example

Consider the following example of archiving two files:

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

This example creates an archive using /user/hadoop as the relative archive directory. The directories /user/hadoop/dir1 and /user/hadoop/dir2 will be archived in the /user/zoo/foo.har archive.

List files in Hadoop archives

Use the hdfs dfs -ls command to list files in Hadoop archives.

Procedure

Run the `hdfs dfs -ls` command by specifying the archive directory location.
To specify the directories in an archive directory foo.har located in /usr/zoo, run the following command:

```
hdfs dfs -ls har:///user/zoo/foo.har/
```

Assuming the archive directory foo.har contains two directories dir1 and dir2, the command returns the following

```
har:///user/zoo/foo.har/dir1
har:///user/zoo/foo.har/dir2
```

Note:

Consider an archive created using the following command:

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

If you list the files of the archive created in the preceding command, the command returns the following:

```
har:///user/zoo/foo.har/hadoop
har:///user/zoo/foo.har/hadoop/dir1
har:///user/zoo/foo.har/hadoop/dir2
```

Note that the modified parent argument causes the files to be archived relative to /user/.

Format for using Hadoop archives with MapReduce

To use Hadoop Archives with MapReduce, you must reference files differently than you would with the default file system. If you have a Hadoop Archive stored in HDFS in /user/zoo/foo.har, you must specify the input directory as har:///user/zoo/foo.har to use it as a MapReduce input.

Because Hadoop Archives are exposed as a file system, MapReduce can use all of the logical input files in Hadoop Archives as input.

Detecting slow DataNodes

Slow DataNodes in an HDFS cluster can negatively impact the cluster performance. Therefore, HDFS provides a mechanism to detect and report slow DataNodes that have a negative impact on the performance of the cluster.

HDFS is designed to detect and recover from complete failure of DataNodes:

- There is no single point of failure.
- Automatic NameNode failover takes only a few seconds.
- Because data replication can be massively parallelized in large clusters, recovery from DataNode loss occurs within minutes.
- Most jobs are not affected by DataNode failures.

However, partial failures can negatively affect the performance of running DataNodes:

- Slow network connection due to a failing or misconfigured adapter.
- Bad OS or JVM settings that affect service performance.
- Slow hard disk.
- Bad disk controller.

Slow DataNodes can have a significant impact on cluster performance. A slow DataNode may continue sending heartbeats successfully, and the NameNode will keep redirecting clients to slow DataNodes. HDFS DataNode monitoring provides detection and reporting of slow DataNodes that negatively affect cluster performance.

Enable detection of slow DataNodes

When slow DataNode detection is enabled, DataNodes collect latency statistics on their peers during write pipelines, and use periodic outlier detection to determine slow peers. The NameNode aggregates reports from all DataNodes and flags potentially slow nodes. Slow DataNode detection is disabled by default.

Procedure

1. To enable slow DataNode detection, set the value of the dfs.datanode.peer.stats.enabled property to true in hdfs-site.xml.

```
<property>
  <name>dfs.datanode.peer.stats.enabled</name>
  <value>true</value>
</property>
```

2. Access the slow DataNode statistics either from the NameNode JMX page at http://<namenode_host>:50070/jmx or from the DataNode JMX page at http://<datanode_host>:50075/jmx.
   In the following JMX output example, the time unit is milliseconds, and the peer DataNodes are healthy because the latencies are in milliseconds:

```
"name" : "Hadoop:service=DataNode,name=DataNodeInfo",
"modelerType" : "org.apache.hadoop.hdfs.server.datanode.DataNode",    "Sen
dPacketDownstreamAvgInfo" : "{
        \"[192.168.7.202:50075]RollingAvgTime\" : 1.4476967370441458,
        \"[192.168.7.201:50075]RollingAvgTime\" : 1.5569170444798432
}"
```

Allocating DataNode memory as storage

HDFS supports efficient writes of large data sets to durable storage, and also provides reliable access to the data. This works well for batch jobs that write large amounts of persistent data. Emerging classes of applications are driving use

cases for writing smaller amounts of temporary data. Using DataNode memory as storage addresses the use case of applications that want to write relatively small amounts of intermediate data sets with low latency.

Writing block data to memory reduces durability, as data can be lost due to process restart before it is saved to disk. HDFS attempts to save replica data to disk in a timely manner to reduce the window of possible data loss.

DataNode memory is referenced using the RAM_DISK storage type and the LAZY_PERSIST storage policy.

HDFS storage types

The storage type identifies the underlying storage media.

HDFS supports the following storage types:

- ARCHIVE - Archival storage is for very dense storage and is useful for rarely accessed data. This storage type is typically cheaper per TB than normal hard disks.
- DISK - Hard disk drives are relatively inexpensive and provide sequential I/O performance. This is the default storage type.
- SSD - Solid state drives are useful for storing hot data and I/O-intensive applications.
- RAM_DISK - This special in-memory storage type is used to accelerate low-durability, single-replica writes.

When you add the DataNode Data Directory, you can specify which type of storage it uses, by prefixing the path with the storage type, in brackets. If you do not specify a storage type, it is assumed to be DISK.

LAZY_PERSIST memory storage policy

Use the LAZY_PERSIST storage policy to store data blocks on the configured DataNode memory.

For LAZY_PERSIST, the first replica is stored on RAM_DISK (DataNode memory), and the remaining replicas are stored on DISK. The fallback storage for both creation and replication is DISK.

The following table summarizes these replication policies:

Policy ID	Policy Name	Block Placement (for n replicas)	Fallback storage for creation	Fallback storage for replication
15	LAZY_PERSIST	RAM_DISK: 1, DISK:n-1	DISK	DISK

Configure DataNode memory as storage

Configuring memory on a DataNode as storage requires you to shut down the particular DataNode, set RAM_DISK as the storage type, set the LAZY_PERSIST storage policy to store data, and then start the DataNode.

Procedure

1. Shut down the DataNode.
2. Use required mount commands to allocate a certain portion of the DataNode memory as storage.
   The following example shows how you can allocate 2GB memory for use by HDFS.

   ```
   sudo mkdir -p /mnt/hdfsramdisk
   sudo mount -t tmpfs -o size=2048m tmpfs /mnt/hdfsramdisk
   sudo mkdir -p /usr/lib/hadoop-hdfs
   ```

3. Assign the RAM_DISK storage type to ensure that HDFS can assign data to the DataNode memory configured as storage.

   To specify the DataNode as RAM_DISK storage, insert [RAM_DISK] at the beginning of the local file system mount path and add it to the dfs.name.dir property in hdfs-default.xml.

   The following example shows the updated mount path values for dfs.datanode.data.dir

   ```
   <property>
   ```

```
  <name>dfs.datanode.data.dir</name>
  <value>file:///grid/3/aa/hdfs/data/,[RAM_DISK]file:///mnt/hdfsramdisk/</
value>
</property>
```

4. Set the LAZY_PERSIST storage policy to store data on the configured DataNode memory.
   The following example shows how you can use the `hdfs dfsadmin -getStoragepolicy` command to configure the LAZY_PERSIST storage policy:

   ```
   hdfs dfsadmin -getStoragePolicy /memory1 LAZY_PERSIST
   ```

   Note: When you update a storage policy setting on a file or directory, the new policy is *not* automatically enforced. You must use the HDFS mover data migration tool to actually move blocks as specified by the new storage policy.
5. Start the DataNode.
6. Use the HDFS mover tool to move data blocks according to the specified storage policy.

   The HDFS mover data migration tool scans the specified files in HDFS and verifies if the block placement satisfies the storage policy. For the blocks that violate the storage policy, the tool moves the replicas to a different storage type in order to fulfill the storage policy requirements.

Improving performance with short-circuit local reads

In HDFS, reads normally go through the DataNode. Thus, when a client asks the DataNode to read a file, the DataNode reads that file off of the disk and sends the data to the client over a TCP socket. "Short-circuit" reads bypass the DataNode, allowing the client to read the file directly. This is only possible in cases where the client is co-located with the data. Short-circuit reads provide a substantial performance boost to many applications.

Prerequisites for configuring short-ciruit local reads

To configure short-circuit local reads, you must enable libhadoop.so.

See the Native Libraries Guide for details on enabling this library.

Related Information

Native Libraries Guide

Properties for configuring short-circuit local reads on HDFS

To configure short-circuit local reads, you must add various properties to the hdfs-site.xml file. Short-circuit local reads must be configured on both the DataNode and the client.

Property Name	Property Value	Description
dfs.client.read.shortcircuit	true	Set this to true to enable short-circuit local reads.

Property Name	Property Value	Description
dfs.domain.socket.path	/var/lib/hadoop-hdfs/ dn_socket	The path to the domain socket. Short-circuit reads make use of a UNIX domain socket. This is a special path in the file system that allows the client and the DataNodes to communicate. You will need to set a path to this socket. The DataNode needs to be able to create this path. On the other hand, it should not be possible for any user except the hdfs user or root to create this path. For this reason, paths under /var/run or /var/lib are often used. In the file system that allows the client and the DataNodes to communicate. You will need to set a path to this socket. The DataNode needs to be able to create this path. On the other hand, it should not be possible for any user except the hdfs user or root to create this path. For this reason, paths under /var/run or /var/lib are often used.
dfs.client.domain.socket.data.traffic	false	This property controls whether or not normal data traffic will be passed through the UNIX domain socket. It is recommended that you set the value of this property to false. Abnormal data traffic will be passed through the UNIX domain socket.
dfs.client.use.legacy.blockreader.local	false	Setting this value to false specifies that the new version (based on HDFS-347) of the short-circuit reader is used. Setting this value to true would mean that the legacy short-circuit reader would be used.
dfs.datanode.hdfs-blocks-metadata.enabled	true	Boolean which enables back-end DataNode-side support for the experimental DistributedFileSystem#getFile VBlockStorageLocationsAPI.
dfs.client.file-block-storage-locations.timeout	60	Timeout (in seconds) for the parallel RPCs made in DistributedFileSystem #getFileBlockStorageLocations(). This property is deprecated but is still supported for backward compatibility
dfs.client.file-block-storage-locations.timeout.millis	60000	Timeout (in milliseconds) for the parallel RPCs made in DistributedFileSystem #getFileBlockStorageLocations(). This property replaces dfs.client.file-block-storage-locations.timeout, and offers a finer level of granularity.
dfs.client.read.shortcircuit.skip.checksum	false	If this configuration parameter is set, short-circuit local reads will skip checksums. This is normally not recommended, but it may be useful for special setups. You might consider using this if you are doing your own checksumming outside of HDFS.

Property Name	Property Value	Description
dfs.client.read.shortcircuit.streams.cache.size	256	The DFSClient maintains a cache of recently opened file descriptors. This parameter controls the size of that cache. Setting this higher will use more file descriptors, but potentially provide better performance on workloads involving many seeks.
dfs.client.read.shortcircuit.streams.cache.expiry.ms	300000	This controls the minimum amount of time (in milliseconds) file descriptors need to sit in the client cache context before they can be closed for being inactive for too long.

The XML for these entries:

```
<configuration>
<property>
  <name>dfs.client.read.shortcircuit</name>
  <value>true</value>
</property>

<property>
  <name>dfs.domain.socket.path</name>
  <value>/var/lib/hadoop-hdfs/dn_socket</value>
</property>

<property>
  <name>dfs.client.domain.socket.data.traffic</name>
  <value>false</value>
</property>

<property>
  <name>dfs.client.use.legacy.blockreader.local</name>
  <value>false</value>
</property>

<property>
  <name>dfs.datanode.hdfs-blocks-metadata.enabled</name>
  <value>true</value>
</property>

<property>
  <name>dfs.client.file-block-storage-locations.timeout.millis</name>
  <value>60000</value>
</property>

<property>
  <name>dfs.client.read.shortcircuit.skip.checksum</name>
  <value>false</value>
</property>

<property>
  <name>dfs.client.read.shortcircuit.streams.cache.size</name>
  <value>256</value>
</property>

<property>
  <name>dfs.client.read.shortcircuit.streams.cache.expiry.ms</name>
  <value>300000</value>
</property>

</configuration>
```

Using DistCp to copy files

Hadoop DistCp (distributed copy) can be used to copy data between CDP clusters (and also within a CDP 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.

Using DistCp

Use DistCp to copy files between various clusters.

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 with HFTP

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.

Update and overwrite

Use the -update option to copy files from a source when they do not exist at the target. Use the -overwrite function to overwrite the target files even if the content is the same.

The DistCp -update option is used to copy files from a source that does not exist at the target, or that has 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 seco nd/, 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 hd
fs://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 does not 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.

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>
```

Secure-to-secure: Kerberos principal name

Assign the same principle name to applicable NameNodes in the source and destination clusters.

distcp hdfs://cdp-secure hdfs://cdp-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.

Secure-to-secure: ResourceManager mapping rules

When copying between two CDP secure clusters, 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>
```

DistCp between HA clusters

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

About this task

Use the following steps to copy data between HA clusters:

Edit the HDFS Client Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml for both cluster A and cluster B:

Procedure

1. Open the Cloudera Manager Admin Console.
2. Go to the HDFS service.
3. Click the Configuration tab.
4. Select Scope Gateway .
5. Select Category Advanced .

6. Search for HDFS Client Advanced Configuration Snippet (Safety Valve) for hdfs-site.xml, and add the various properties as specified:
   a) Add both name services to dfs.nameservices = HAA, HAB
   b) Add the dfs.internal.nameservices property:
      - In cluster A:

        dfs.internal.nameservices = HAA
      - In cluster B:

        dfs.internal.nameservices = HAB
   c) Add dfs.ha.namenodes.<nameservice> to both clusters:
      - In cluster A

        dfs.ha.namenodes.HAB = nn1,nn2
      - In cluster B

        dfs.ha.namenodes.HAA = nn1,nn2
   d) Add the dfs.namenode.rpc-address.<cluster>.<nn> property:
      - In Cluster A:

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

        dfs.namenode.rpc-address.HAB.nn2 = <NN2_fqdn>:8020
      - In Cluster B:

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

        dfs.namenode.rpc-address.HAA.nn2 = <NN2_fqdn>:8020
   e) Add the following properties to enable distcp over WebHDFS and secure WebHDFS:
      - In Cluster A:

        dfs.namenode.http-address.HAB.nn1 = <NN1_fqdn>:50070

        dfs.namenode.http-address.HAB.nn2 = <NN2_fqdn>:50070

        dfs.namenode.https-address.HAB.nn1 = <NN1_fqdn>:50470

        dfs.namenode.https-address.HAB.nn2 = <NN2_fqdn>:50470
      - In Cluster B:

        dfs.namenode.http-address.HAA.nn1 = <NN1_fqdn>:50070

        dfs.namenode.http-address.HAA.nn2 = <NN2_fqdn>:50070

        dfs.namenode.https-address.HAA.nn1 = <NN1_fqdn>:50470

        dfs.namenode.https-address.HAA.nn2 = <NN2_fqdn>:50470
   f) Add the dfs.client.failover.proxy.provider.<cluster> property:
      - In cluster A:

        dfs.client.failover.proxy.provider. HAB = org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
      - In cluster B:

        dfs.client.failover.proxy.provider. HAA = org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
   g) Restart the HDFS service, then run the distcp command using the NameService. For example:

      hadoop distcp hdfs://HAA/tmp/testDistcp hdfs://HAB/tmp/

Using DistCp with Amazon S3

You can copy HDFS files to and from an Amazon S3 instance. You must provision an S3 bucket using Amazon Web Services and obtain the access key and secret key.

You can pass these credentials on the distcp command line, or you can reference a credential store to "hide" sensitive credentials so that they do not appear in the console output, configuration files, or log files.

Amazon S3 block and native filesystems are supported with the s3a:// protocol.

Example of an Amazon S3 Block Filesystem URI: s3a://*bucket_name*/path/to/file

S3 credentials can be provided in a configuration file (for example, core-site.xml):

```
<property>
    <name>fs.s3a.access.key</name>
    <value>...</value>
</property>
<property>
    <name>fs.s3a.secret.key</name>
    <value>...</value>
</property>
```

You can also enter the configurations in the Advanced Configuration Snippet for core-site.xml, which allows Cloudera Manager to manage this configuration.

You can also provide the credentials on the command line:

```
hadoop distcp -Dfs.s3a.access.key=... -Dfs.s3a.secret.key=... s3a://
```

For example:

```
hadoop distcp -Dfs.s3a.access.key=myAccessKey -Dfs.s3a.secret.key=mySecretKe
y /user/hdfs/mydata s3a://myBucket/mydata_backup
```

Important: Entering secrets on the command line is inherently insecure. These secrets may be accessed in log files and other artifacts. Cloudera recommends that you use a credential provider to store secrets.

Note: Using the -diff option with the distcp command requires a Distributed FileSystem on both the source and destination and is not supported when using distcp to copy data to or from Amazon S3.

Using a credential provider to secure S3 credentials

You can run the distcp command without having to enter the access key and secret key on the command line. This prevents these credentials from being exposed in console output, log files, configuration files, and other artifacts.

About this task

Running the distcp command in this way requires that you provision a credential store to securely store the access key and secret key. The credential store file is saved in HDFS.

Note: Using a Credential Provider does not work with MapReduce v1 (MRV1).

Procedure

1. Provision the credentials by running the following commands:

   ```
   hadoop credential create fs.s3a.access.key -value access_key -provider j
   ceks://hdfs/path_to_credential_store_file
   ```

```
hadoop credential create fs.s3a.secret.key -value secret_key -provider j
ceks://hdfs/path_to_credential_store_file
```

For example:

```
hadoop credential create fs.s3a.access.key -value foobar -provider jceks
://hdfs/user/alice/home/keystores/aws.jceks
hadoop credential create fs.s3a.secret.key -value barfoo -provider jceks
://hdfs/user/alice/home/keystores/aws.jceks
```

You can omit the -value option and its value and the command will prompt the user to enter the value.

For more details on the hadoop credential command, see Credential Management (Apache Software Foundation).

Note: If you configure the hadoop.security.credential.provider.path property to jceks://hdfs/path_to_credential_store_file, ZKFC might fail to start and NameNodes might remain in standby state. So, Cloudera recommends not to add or change this property in Cloudera Manager.

2. Copy the contents of the /etc/hadoop/conf directory to a working directory.
3. Add the following to the core-site.xml file in the working directory:

```
<property>
<name>hadoop.security.credential.provider.path</name>
<value>jceks://hdfs/path_to_credential_store_file</value>
</property>
```

4. Set the HADOOP_CONF_DIR environment variable to the location of the working directory:

```
export HADOOP_CONF_DIR=path_to_working_directory
```

What to do next

After completing these steps, you can run the distcp command using the following syntax:

```
hadoop distcp source_path s3a://destination_path
```

You can also reference the credential store on the command line, without having to enter it in a copy of the core-site.xml file. You also do not need to set a value for HADOOP_CONF_DIR. Use the following syntax:

```
hadoop distcp source_path s3a://bucket_name/destination_path
-Dhadoop.security.credential.provider.path=jceks://
hdfspath_to_credential_store_file
```

Examples of DistCp commands using the S3 protocol and hidden credentials

You can various distcp command options to copy files between your CDP clusters and Amazon S3.

Copying files to Amazon S3

```
hadoop distcp /user/hdfs/mydata s3a://myBucket/mydata_backup
```

Copying files from Amazon S3

```
hadoop distcp s3a://myBucket/mydata_backup //user/hdfs/mydata
```

Copying files to Amazon S3 using the -filters option to exclude specified source files

You specify a file name with the -filters option. The referenced file contains regular expressions, one per line, that define file name patterns to exclude from the distcp job. The pattern specified in the regular expression should match the fully-qualified path of the intended files, including the

scheme (hdfs, webhdfs, s3a, etc.). For example, the following are valid expressions for excluding files:

```
hdfs://x.y.z:8020/a/b/c
webhdfs://x.y.z:50070/a/b/c
s3a://bucket/a/b/c
```

Reference the file containing the filter expressions using -filters option. For example:

```
hadoop distcp -filters /user/joe/myFilters /user/hdfs/mydata s
3a://myBucket/mydata_backup
```

Contents of the sample myFilters file:

```
.*foo.*
.*/bar/.*
hdfs://x.y.z:8020/tmp/.*
hdfs://x.y.z:8020/tmp1/file1
```

The regular expressions in the myFilters exclude the following files:

- .*foo.* – excludes paths that contain the string "foo".
- .*/bar/.* – excludes paths that include a directory named bar.
- hdfs://x.y.z:8020/tmp/.* – excludes all files in the /tmp directory.
- hdfs://x.y.z:8020/tmp1/file1 – excludes the file /tmp1/file1.

Copying files to Amazon S3 with the -overwrite option.

The -overwrite option overwrites destination files that already exist.

```
hadoop distcp -overwrite /user/hdfs/mydata  s3a://user/mydata_ba
ckup
```

For more information about the -filters, -overwrite, and other options, see DistCp Guide: Command Line Options (Apache Software Foundation).

DistCp additional considerations

DistCp provides a strategy to “dynamically” size maps, allowing faster DataNodes to copy more bytes than slower nodes.

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.

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 is not uniform, it is fair with regard to each mapper’s capacity.

The dynamic strategy 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 Major Versions of HDFS

For copying between two different versions of Hadoop, you can usually use WebHDFS.

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.

APIs for accessing HDFS

Use the WebHDFS REST API to access an HDFS cluster from applications external to the cluster. WebHDFS supports all HDFS user operations including reading files, writing to files, making directories, changing permissions and renaming. In addition, WebHDFS uses Kerberos and delegation tokens for authenticating users.

Set up WebHDFS on a secure cluster

Setting up WebHDFS on a secure cluster requires you to update certain properties on hdfs-site.xml, create an HTTP service user principal, create a keytab for the principal, and verify the association of the principal and keytab with the correct HTTP service.

Procedure

1. Create an HTTP service user principal.

   ```
   kadmin: addprinc -randkey HTTP/$<Fully_Qualified_Domain_Name>@$<Realm_Na
   me>.COM
   ```

   where:

   - Fully_Qualified_Domain_Name: Host where the NameNode is deployed.
   - Realm_Name: Name of your Kerberos realm.
2. Create a keytab file for the HTTP principal.

   ```
   kadmin: xst -norandkey -k /etc/security/spnego.service.keytab HTTP/$<Ful
   ly_Qualified_Domain_Name>
   ```
3. Verify that the keytab file and the principal are associated with the correct service.

   ```
   klist –k -t /etc/security/spnego.service.keytab
   ```
4. Add the dfs.web.authentication.kerberos.principal and dfs.web.authentication.kerberos.keytab properties to hdfs-site.xml.

   ```
   <property>
     <name>dfs.web.authentication.kerberos.principal</name>
     <value>HTTP/$<Fully_Qualified_Domain_Name>@$<Realm_Name>.COM</value>
   </property>
   <property>
     <name>dfs.web.authentication.kerberos.keytab</name>
     <value>/etc/security/spnego.service.keytab</value>
   </property>
   ```

5. Restart the NameNode and the DataNodes.

Using HttpFS to provide access to HDFS

Apache Hadoop HttpFS is a service that provides HTTP access to HDFS. HttpFS has a REST HTTP API supporting all HDFS filesystem operations (both read and write).

Common HttpFS use cases are:

- Read and write data in HDFS using HTTP utilities (such as curl or wget) and HTTP libraries from languages other than Java (such as Perl).
- Transfer data between HDFS clusters running different versions of Hadoop (overcoming RPC versioning issues), for example using Hadoop DistCp.
- Accessing WebHDFS using the NameNode Web UI port (default port 9870). Access to all data hosts in the cluster is required, because WebHDFS redirects clients to the DataNode port (default port 9864). If the cluster is behind a firewall, and you use WebHDFS to read and write data to HDFS, then Cloudera recommends you use the HttpFS server. The HttpFS server acts as a gateway. It is the only system that is allowed to send and receive data through the firewall.

HttpFS supports Hadoop pseudo-authentication, HTTP SPNEGO Kerberos, and additional authentication mechanisms using a plugin API. HttpFS also supports Hadoop proxy user functionality.

The webhdfs client file system implementation can access HttpFS using the Hadoop filesystem command (hadoop fs), by using Hadoop DistCp, and from Java applications using the Hadoop file system Java API.

The HttpFS HTTP REST API is interoperable with the WebHDFS REST HTTP API.

Add the HttpFS role

You can use Cloudera Manager to add the HttpFS role.

1. Go to the HDFS service.
2. Click the Instances tab.
3. Click Add Role Instances.
4. Click the text box below the HttpFS field. The Select Hosts dialog box displays.
5. Select the host on which to run the role and click OK.
6. Click Continue.
7. Check the checkbox next to the HttpFS role and select Actions for Selected Start .

Using Load Balancer with HttpFS

Configure the HttpFS Service to work with the load balancer that you configured for the service.

1. In the Cloudera Manager Admin Console, navigate to Cluster <HDFS service> .
2. On the Configuration tab, search for the following property:

```
HttpFS Load Balancer
```

3. Enter the hostname and port for the load balancer in the following format:

```
<hostname>:<port>
```

4. Save the changes.

Note: When you set this property, Cloudera Manager regenerates the keytabs for HttpFS roles. The principal in these keytabs contains the load balancer hostname. If there is a Hue service that depends on this HDFS service, the Hue service has the option to use the load balancer as its HDFS Web Interface Role.

HttpFS authentication

To enable HttpFS to work with Kerberos security on your CDP cluster, ensure that you have configured Kerberos for the cluster.

Important: If the NameNode, Secondary NameNode, DataNode, JobTracker, TaskTrackers, ResourceManager, NodeManagers, HttpFS, or Oozie services are configured to use Kerberos HTTP SPNEGO authentication, and two or more of these services are running on the same host, then all of the running services must use the same HTTP principal and keytab file used for their HTTP endpoints.

Use curl to access a URL protected by Kerberos HTTP SPNEGO

To access a URL protected by Kerberos HTTP SPNEGO, ensure that your version of curl supports GSS and is capable of running curl -V.

Procedure

1. Run curl -V.

```
$ curl -V
curl 7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7 OpenSSL/0.9.8l
zlib/1.2.3
Protocols: tftp ftp telnet dict ldap http file https ftps
Features: GSS-Negotiate IPv6 Largefile NTLM SSL libz
```

2. Log on to the KDC using kinit.

```
$ kinit
Please enter the password for username@LOCALHOST:
```

3. Use curl to fetch the protected URL.

```
$ curl --cacert
             /path/to/truststore.pem --negotiate -u : -b ~/cookiejar.txt
 -c ~/cookiejar.txt https://localhost:14000/webhdfs/v1/?op=liststatus
```

where:

- The --cacert option is required if you are using TLS/SSL certificates that curl does not recognize by default.
- The --negotiate option enables SPNEGO in curl.
- The -u: option is required but the username is ignored (the principal that has been specified for kinit is used).
- The -b and -c options are used to store and send HTTP cookies.
- Cloudera does not recommend using the -k or --insecure option as it turns off curl's ability to verify the certificate.

Data storage metrics

Use Java Management Extensions (JMX) APIs to collect the metrics exposed by the various HDFS daemons.

You can use various HDFS metrics to understand the state of your cluster.

Using JMX for accessing HDFS metrics

You can access HDFS metrics over Java Management Extensions (JMX) through either the web interface of an HDFS daemon or by directly accessing the JMX remote agent.

Using the HDFS daemon web interface

You can access JMX metrics through the web interface of an HDFS daemon. This is the recommended method.

For example, use the following command format to access the NameNode JMX:

```
curl -i http://localhost:50070/jmx
```

You can use the qry parameter to fetch only a particular key:

```
curl -i http://localhost:50070/jmx?qry=Hadoop:service=NameNode,name=NameNode
Info
```

Directly accessing the JMX remote agent

This method requires that the JMX remote agent is enabled with a JVM option when starting HDFS services.

For example, the following JVM options in hadoop-env.sh are used to enable the JMX remote agent for the NameNode. It listens on port 8004 with SSL disabled. The user name and password are saved in the mxremote.password file.

```
export HADOOP_NAMENODE_OPTS="-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.password.file=$HADOOP_CONF_DIR/jmxremote.pass
word
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.port=8004 $HADOOP_NAMENODE_OPTS"
```

See the Oracle Java SE documentation for more information about the related settings.

You can also use the jmxquery tool to retrieve information through JMX.

Hadoop has a built-in JMX query tool, jmxget. For example:

```
hdfs jmxget -server localhost -port 8004 -service NameNode
```

Note: jmxget requires that authentication be disabled, as it does not accept a user name and password.

Using JMX can be challenging for operations personnel who are not familiar with JMX setup, especially JMX with SSL and firewall tunnelling. Therefore, we recommend that you collect JMX information through the web interface of HDFS daemons rather than directly accessing the JMX remote agent.

Related Information

JMX Query

Monitoring and Management Using JMX Query

Configure the G1GC garbage collector

You must follow certain recommendations when switching from the currently used Concurrent Mark Sweep (CMS) GC to G1GC.

Recommended settings for G1GC

The recommended settings for configuring Garbage First Garbage Collector (G1GC) include allocating more Java heap space when compared to the Concurrent Mark Sweep (CMS) GC, and setting specific values for properties such as MaxGCPauseMillis and ParallelGCThreads.

The following NameNode settings are recommended for G1GC in a large cluster:

- Approximately 10% more Java heap space (-XX:Xms and -XX:Xmx) should be allocated to the NameNode, as compared to CMS setup.

  Note: Do not configure G1GC if the NameNode heap size is greater than 64 GB.

- For large clusters (>50M files), MaxGCPauseMillis should be set to 4000.
- You should set ParallelGCThreads to 20 (default for a 32-core machine), as opposed to 8 for CMS.
- Other G1GC parameters should be left set to their default values.

Related Information

Switching from CMS to G1GC

Switching from CMS to G1GC

To move from Concurrent Mark Sweep (CMS) GC to Garbage First (G1) GC, you must update the HADOOP_NAMENODE_OPTS settings in hadoop-env.sh.

Procedure

On the Ambari dashboard, select HDFS > Configs > Advanced > Advanced hadoop-env.

Make the following changes to the HADOOP_NAMENODE_OPTS settings:

- Replace -XX:+UseConcMarkSweepGC with -XX:+UseG1GC
- Remove -XX:+UseCMSInitiatingOccupancyOnly and -XX:CMSInitiatingOccupancyFraction=####
- Remove -XX:NewSize=#### and -XX:MaxNewSize=####
- (Optional) Add -XX:MaxGCPauseMillis=####
- (Optional) Add -XX:InitiatingHeapOccupancyPercent=####
- (Optional) Add -XX:ParallelGCThreads=####, if not present.

  The default value of this parameter is set to the number of logical processors (up to a value of 8). For more than eight logical processors, the default value is set to 5/8th the number of logical processors.

Related Information

Recommended settings for G1GC

HDFS Metrics

Many aggregate metrics are available in addition to base metrics. If an entity type has parents defined, you can formulate all possible aggregate metrics using the formula base_metric_across_parents.

In addition, metrics for aggregate totals can be formed by adding the prefix total_ to the front of the metric name.

Use the type-ahead feature in the Cloudera Manager chart browser to find the exact aggregate metric name, in case the plural form does not end in "s".

For example, the following metric names may be valid for HDFS:

- alerts_rate_across_clusters
- total_alerts_rate_across_clusters

Some metrics, such as alerts_rate, apply to nearly every metric context. Others only apply to a certain service or role.

Metric Name	Description	Unit	Parents
alerts_rate	The number of alerts.	events per second	cluster
block_capacity	The block capacity of the NameNode	blocks	cluster
blocks_total	Blocks total	blocks	cluster
blocks_with_corrupt_replicas	Blocks with corrupt replicas	blocks	cluster
canary_duration	Duration of the last or currently running canary job	ms	cluster
cm_time_since_last_fsimage_fetch	Time since last FsImage was fetched by Cloudera Reports Manager	seconds	cluster
cm_time_since_last_fsimage_index	Time since last FsImage was indexed By Cloudera Reports Manager	seconds	cluster
dfs_capacity	Total configured HDFS storage capacity	bytes	cluster
dfs_capacity_used	Storage space used by HDFS files	bytes	cluster
dfs_capacity_used_non_hdfs	Storage space used by non-HDFS files	bytes	cluster
events_critical_rate	The number of critical events.	events per second	cluster
events_important_rate	The number of important events.	events per second	cluster
events_informational_rate	The number of informational events.	events per second	cluster
excess_blocks	The total number of excess blocks	blocks	cluster
expired_heartbeats	The number of expired heartbeats	heartbeats	cluster
files_total	The number of files and directories in the HDFS	files	cluster
health_bad_rate	Percentage of Time with Bad Health	seconds per second	cluster
health_concerning_rate	Percentage of Time with Concerning Health	seconds per second	cluster
health_disabled_rate	Percentage of Time with Disabled Health	seconds per second	cluster
health_good_rate	Percentage of Time with Good Health	seconds per second	cluster
health_unknown_rate	Percentage of Time with Unknown Health	seconds per second	cluster
missing_blocks	Missing blocks	blocks	cluster
pending_deletion_blocks	The number of replicas pending deletion.	replicas	cluster
pending_replication_blocks	The number of blocks with replication pending	blocks	cluster
scheduled_replication_blocks	The number of blocks with replication currently scheduled	blocks	cluster
under_replicated_blocks	Under-replicated blocks	blocks	cluster
xceivers	Transceivers	transceivers	cluster