DistributedCache

DistributedCache distributes application-specific, large, read-only files efficiently.

`DistributedCache` is a facility provided by the MapReduce framework to cache files (text, archives, jars and so on) needed by applications.

Applications specify the files to be cached via urls (hdfs://) in the `JobConf`.
The `DistributedCache` assumes that the files specified via hdfs:// urls are already present on the `FileSystem`.

The framework will copy the necessary files to the slave node before any tasks for the job are executed on that node. Its efficiency stems from the fact that the files are only copied once per job and the ability to cache archives which are un-archived on the slaves.

`DistributedCache` tracks the modification timestamps of the cached files. Clearly the cache files should not be modified by the application or externally while the job is executing.

`DistributedCache` can be used to distribute simple, read-only data/text files and more complex types such as archives and jars. Archives (zip, tar, tgz and tar.gz files) are *un-archived* at the slave nodes. Files have *execution permissions* set.

The files/archives can be distributed by setting the property `mapred.cache.{files|archives}`.

If more than one file/archive has to be distributed, they can be added as comma separated paths. The properties can also be set by APIs DistributedCache.addCacheFile(URI,conf)/ DistributedCache.addCacheArchive(URI,conf) and DistributedCache.setCacheFiles(URIs,conf)/DistributedCache.setCacheArchives(URIs,conf) where URI is of the form `hdfs://host:port/absolute-path#link-name`. In Streaming, the files can be distributed through command line option `-cacheFile/-cacheArchive`.

Optionally users can also direct the `DistributedCache` to *symlink* the cached file(s) into the `current working directory` of the task via the DistributedCache.createSymlink(Configuration) api. Or by setting the configuration property `mapred.create.symlink` as `yes`. The DistributedCache will use the `fragment` of the URI as the name of the symlink. For example, the URI`hdfs://namenode:port/lib.so.1#lib.so` will have the symlink name as `lib.so` in task's cwd for the file `lib.so.1` in distributed cache.

The `DistributedCache` can also be used as a rudimentary software distribution mechanism for use in the map and/or reduce tasks. It can be used to distribute both jars and native libraries. TheDistributedCache.addArchiveToClassPath(Path, Configuration) or DistributedCache.addFileToClassPath(Path, Configuration) api can be used to cache files/jars and also add them to the *classpath* of child-jvm. The same can be done by setting the configuration properties `mapred.job.classpath.{files|archives}`. Similarly the cached files that are symlinked into the working directory of the task can be used to distribute native libraries and load them.

Private and Public DistributedCache Files

DistributedCache files can be private or public, that determines how they can be shared on the slave nodes.

- "Private" DistributedCache files are cached in a local directory private to the user whose jobs need these files. These files are shared by all tasks and jobs of the specific user only and cannot be accessed by jobs of other users on the slaves. A DistributedCache file becomes private by virtue of its permissions on the file system where the files are uploaded, typically HDFS. If the file has no world readable access, or if the directory path leading to the file has no world executable access for lookup, then the file becomes private.

- "Public" DistributedCache files are cached in a global directory and the file access is setup such that they are publicly visible to all users. These files can be shared by tasks and jobs of all users on the slaves. A DistributedCache file becomes public by virtue of its permissions on the file system where the files are uploaded, typically HDFS. If the file has world readable access, AND if the directory path leading to the file has world executable access for lookup, then the file becomes public. In other words, if the user intends to make a file publicly available to all users, the file permissions must be set to be world readable, and the directory permissions on the path leading to the file must be world executable.