Making Hadoop TAR Aware

Joydip Datta
Aug 31, 2013

Executive Summary

TAR is a widely used format for storing and distributing large collections of files such as backup images, large datasets *etc.* This documents details the use of FileSystem abstraction to access a TAR file from Hadoop without extracting it.

Motivation

TAR is a very popular tool to combine many small files and directories into one single large file. Together with compression methods such as gunzip, TAR provides a nice way to archive, store and distribute large numbers of files together. TAR is also very popular format for storing backup images, distributing large datasets etc. Many of those files could be used as an input to analytic jobs.

Apache Hadoop, as of now, is not TAR aware. That is, it can not directly read a file *inside* a TAR. Neither it can run map-reduce on those files. To run analytic jobs on a TAR, one needs to first copy it to local disk, un-TAR it, then copy back to Hadoop file system. Or convert it to sequence file/other Hadoop aware format using custom (java) program. This procedure is time consuming and the user ends up having two copies of data.

By making Hadoop TAR aware, Hadoop can directly read files *inside* a TAR and run analytic jobs on those file. This way, no conversion/extraction is required.

How to Get the Tar FileSystem

Get it from GitHub: The source codes are available in GitHub repository under Apache License 2.0: https://github.com/JDatta/TarFileSystem. Download or clone the source code form GitHub and use “mvn package” to build it.

Design overview

We use the Hadoop FileSystem abstraction to expose the TAR file. We implement a TarFileSystem class that extends FileSystem. All storage access in Hadoop happens through an well defined interface called FileSystem. By providing a FileSystem for TAR, virtually all Hadoop applications (including FS-Shell and Mapreduce) becomes Hadoop Aware. The implementation is mostly modeled after Hadoop Archives and employs similar access mechanisms.

An TAR file is simply a sequence of file records where each record contain a header section and a data section. The header section contains metadata of the file including file system

stats such as size, permission, modification date etc. The file itself is stored in the data section.

On the first run, the TarFileSystem, while initializing does a single scan of the whole tar file and creates an index with filename and offset where they start in the tar archive. The rest of the TarFileSystem code uses this index to access individual files within the tar archive. TarFileSystem also stores this index in a .index file so that in subsequent runs the TarFileSystem can load the index directly from the .index file and does not to scan the whole tar file. This can specially boost the performance of mapreduce jobs on a tar file as the index can be created and stored by the client and can be reused by the individual mappers.

Distribution and Configuration

TAR File System binary for Hadoop is distributed as a JAR library (TarFileSystem.jar). This JAR contains the main `TarFileSystem` class and other supporting classes. The user needs to copy this JAR to the `HADOOP_HOME/lib` directory (HDFS_HOME/lib for Hadoop 2.0) or add the jar to `HADOOP_CLASSPATH` environment variable.

Next you need to expose `tar://` uri schema to Hadoop by adding the following property in `HADOOP_CONF_DIR/core-site.xml`

```
<property>
  <name>fs.tar.impl</name>
  <value>org.apache.hadoop.fs.tar.TarFileSystem</value>
</property>
```

Optional Configuration:
By default, TarFileSystem creates an `.index` file in the same directory where the tar file resides. Index writing may fail if you do not have sufficient permission in that directory. In that case you may specify a temporary directory where you have write permission and tell TarFileSystem to use that directory instead. You may specify the following property in `core-site.xml` for this:

```
<property>
  <name>tarfs.tmp.dir</name>
  <value>/a/directory/with/write/permission</value>
</property>
```

Note that, **TarFileSystem will still prefer the same directory where the tar file exists for writing the .index file**. Only if writing to the same directory fails it will use the `tarfs.tmp.dir`. In that case, if `tarfs.tmp.dir` is not specified or writing to that directory also fail, it will skip writing the .index file with a warning message.

Using TAR File System

Hadoop can access a TAR archive using TAR URI SCHEMA (URI starting with `tar://`). The following examples shows this:

Following is a TAR inside Hadoop file System

```
[jd@morpheus hadoop-1.0.3]$ bin/hadoop fs -ls /tardemo/archive.tar ↵
Found 1 items
-rw-r--r--   1 jd supergroup    1751040 2013-07-15 20:30 /tardemo/archive.tar
```

To access files inside this tar, simply prepone this with tar:// to make it a TAR File System URI

```
[jd@morpheus hadoop-1.0.3]$ bin/hadoop fs -ls tar:///tardemo/archive.tar ↵
13/07/15 20:33:04 INFO tar.TarFileSystem: *** Using Tar file system ***
Found 3 items
-rw-rw-r--   1 jd jd     502760 2013-07-15 20:27 /tardemo/archive.tar+data/file2.txt
-rw-rw-r--   1 jd jd     594933 2013-07-15 20:26 /tardemo/archive.tar+data/file1.txt
-rw-rw-r--   1 jd jd     641720 2013-07-15 20:27 /tardemo/archive.tar+data/file3.txt
```

To access a file inside a TAR archive, append the name of the file after the TAR URI using a '+' sign

```
[jd@morpheus hadoop-1.0.3]$ bin/hadoop fs -cat
tar://hdfs-localhost:54310/tardemo/archive.tar+data/file1.txt ↵
13/07/15 20:38:35 INFO tar.TarFileSystem: *** Using Tar file system ***
This is the file content.
[...]
```

In TAR File System, the TAR archive is modeled like a directory and all the files inside a TAR are modeled like files within a directory. One can run mapreduce jobs on files within a TAR archive just like they do it on normal files.

```
[jd@morpheus hadoop-1.0.3]$ bin/hadoop jar hadoop*examples*.jar wordcount ↵
tar:///tardemo/archive.tar wc_out
13/07/15 20:43:05 INFO tar.TarFileSystem: *** Using Tar file system ***
13/07/15 20:43:05 INFO input.FileInputFormat: Total input paths to process : 3
13/07/15 20:43:05 INFO mapred.JobClient: Running job: job_201307151954_0001
13/07/15 20:43:06 INFO mapred.JobClient:  map 0% reduce 0%
 [...]
```

To access files in a different Hadoop cluster, you may mention the underlying FS schema (the FS that hosts the TAR file), NameNode address and port number in the URI

```
[jd@morpheus hadoop-1.0.3]$ bin/hadoop fs -ls
tar://hdfs-localhost:54310/tardemo/archive.tar
13/07/15 20:35:10 INFO tar.TarFileSystem: *** Using Tar file system ***
Found 3 items
-rw-rw-r--   1 jd jd     502760 2013-07-15 20:27 /tardemo/archive.tar+data/file2.txt
-rw-rw-r--   1 jd jd     594933 2013-07-15 20:26 /tardemo/archive.tar+data/file1.txt
-rw-rw-r--   1 jd jd     641720 2013-07-15 20:27 /tardemo/archive.tar+data/file3.txt
```

Conclusion

TAR is a widely used format for storing and distributing large collections of files such as backup images, large datasets *etc.* Using TarFileSystem plugin for Hadoop, one can list, read and run analytic jobs on TAR images without the need of extraction or conversion.

Hadoop also performs badly for large number of small files. To make get around this and to get the performance gain from sequential reads, one needs to create Hadoop Archives or Sequence files. But those formats are not understood outside Hadoop. As a second benifit of TAR File System, we can create one split for multiple consecutive small files to speed up mapreduce jobs.