Offline Image Viewer Guide

Table of contents

1 Overview

The Offline Image Viewer is a tool to dump the contents of hdfs fsimage files to human-readable formats in order to allow offline analysis and examination of an Hadoop cluster's namespace. The tool is able to process very large image files relatively quickly, converting them to one of several output formats. The tool handles the layout formats that were included with Hadoop versions 16 and up. If the tool is not able to process an image file, it will exit cleanly. The Offline Image Viewer does not require an Hadoop cluster to be running; it is entirely offline in its operation.

The Offline Image Viewer provides several output processors:

1. **Ls** is the default output processor. It closely mimics the format of the `lsr` command. It includes the same fields, in the same order, as `lsr` : directory or file flag, permissions, replication, owner, group, file size, modification date, and full path. Unlike the `lsr` command, the root path is included. One important difference between the output of the `lsr` command this processor, is that this output is not sorted by directory name and contents. Rather, the files are listed in the order in which they are stored in the fsimage file. Therefore, it is not possible to directly compare the output of the `lsr` command this this tool. The Ls processor uses information contained within the Inode blocks to calculate file sizes and ignores the `-skipBlocks` option.
2. **Indented** provides a more complete view of the fsimage's contents, including all of the information included in the image, such as image version, generation stamp and inode- and block-specific listings. This processor uses indentation to organize the output into a hierarchal manner. The `lsr` format is suitable for easy human comprehension.
3. **Delimited** provides one file per line consisting of the path, replication, modification time, access time, block size, number of blocks, file size, namespace quota, diskspace quota, permissions, username and group name. If run against an fsimage that does not contain any of these fields, the field's column will be included, but no data recorded. The default record delimiter is a tab, but this may be changed via the `-delimiter` command line argument. This processor is designed to create output that is easily analyzed by other tools, such as Apache Pig. See the Analyzing Results section for further information on using this processor to analyze the contents of fsimage files.
4. **XML** creates an XML document of the fsimage and includes all of the information within the fsimage, similar to the `lsr` processor. The output of this processor is amenable to automated processing and analysis with XML tools. Due to the verbosity of the XML syntax, this processor will also generate the largest amount of output.
5. **FileDistribution** is the tool for analyzing file sizes in the namespace image. In order to run the tool one should define a range of integers `[0, maxSize]` by specifying `maxSize` and a `step`. The range of integers is divided into segments of size `step`: `[0, s1, ..., sn-1, maxSize]`, and the processor calculates how many files

in the system fall into each segment $[s_{i-1}, s_i)$. Note that files larger than `maxSize` always fall into the very last segment. The output file is formatted as a tab separated two column table: Size and NumFiles. Where Size represents the start of the segment, and numFiles is the number of files form the image which size falls in this segment.

2 Usage

2.1 Basic

The simplest usage of the Offline Image Viewer is to provide just an input and output file, via the `-i` and `-o` command-line switches:

```
bash$ bin/hadoop oiv -i fsimage -o fsimage.txt
```

This will create a file named fsimage.txt in the current directory using the Ls output processor. For very large image files, this process may take several minutes.

One can specify which output processor via the command-line switch `-p`. For instance:

```
bash$ bin/hadoop oiv -i fsimage -o fsimage.xml -p XML
```

or

```
bash$ bin/hadoop oiv -i fsimage -o fsimage.txt -p Indented
```

This will run the tool using either the XML or Indented output processor, respectively.

One command-line option worth considering is `-skipBlocks`, which prevents the tool from explicitly enumerating all of the blocks that make up a file in the namespace. This is useful for file systems that have very large files. Enabling this option can significantly decrease the size of the resulting output, as individual blocks are not included. Note, however, that the Ls processor needs to enumerate the blocks and so overrides this option.

2.2 Example

Consider the following contrived namespace:

```
drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:17 /anotherDir

-rw-r--r--   3 theuser supergroup  286631664 2009-03-16 21:15 /anotherDir/biggerfile

-rw-r--r--   3 theuser supergroup       8754 2009-03-16 21:17 /anotherDir/smallFile

drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:11 /mapredsystem

drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:11 /mapredsystem/theuser

drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:11 /mapredsystem/theuser/
mapredsystem
```

```
drwx-wx-wx   - theuser supergroup          0 2009-03-16 21:11 /mapredsystem/theuser/
mapredsystem/ip.redacted.com

drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:12 /one

drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:12 /one/two

drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:16 /user

drwxr-xr-x   - theuser supergroup          0 2009-03-16 21:19 /user/theuser
```

Applying the Offline Image Processor against this file with default options would result in the following output:

```
machine:hadoop-0.21.0-dev theuser$ bin/hadoop oiv -i fsimagedemo -o fsimage.txt

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:16 /

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:17 /anotherDir

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:11 /mapredsystem

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:12 /one

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:16 /user

-rw-r--r--  3   theuser supergroup    286631664 2009-03-16 14:15 /anotherDir/biggerfile

-rw-r--r--  3   theuser supergroup         8754 2009-03-16 14:17 /anotherDir/smallFile

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:11 /mapredsystem/theuser

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:11 /mapredsystem/theuser/
mapredsystem

drwx-wx-wx  -   theuser supergroup            0 2009-03-16 14:11 /mapredsystem/theuser/
mapredsystem/ip.redacted.com

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:12 /one/two

drwxr-xr-x  -   theuser supergroup            0 2009-03-16 14:19 /user/theuser
```

Similarly, applying the Indented processor would generate output that begins with:

```
machine:hadoop-0.21.0-dev theuser$ bin/hadoop oiv -i fsimagedemo -p Indented -o
 fsimage.txt

FSImage

  ImageVersion = -19

  NamespaceID = 2109123098

  GenerationStamp = 1003

  INodes [NumInodes = 12]
```

```
  Inode
    INodePath =
    Replication = 0
    ModificationTime = 2009-03-16 14:16
    AccessTime = 1969-12-31 16:00
    BlockSize = 0
    Blocks [NumBlocks = -1]
    NSQuota = 2147483647
    DSQuota = -1
    Permissions
      Username = theuser
      GroupName = supergroup
      PermString = rwxr-xr-x
remaining output omitted
```

3 Options

3.1 Option Index

Flag	Description
`[-i|--inputFile] <input file>`	Specify the input fsimage file to process. Required.
`[-o|--outputFile] <output file>`	Specify the output filename, if the specified output processor generates one. If the specified file already exists, it is silently overwritten. Required.
`[-p|--processor] <processor>`	Specify the image processor to apply against the image file. Currently valid options are Ls (default), XML and Indented..
`-skipBlocks`	Do not enumerate individual blocks within files. This may save processing time and outfile file space on namespaces with very large files. The `Ls` processor reads the blocks to correctly determine file sizes and ignores this option.
`-printToScreen`	Pipe output of processor to console as well as specified file. On extremely large namespaces,

Flag	Description
	this may increase processing time by an order of magnitude.
`-delimiter <arg>`	When used in conjunction with the Delimited processor, replaces the default tab delimiter with the string specified by `arg`.
`[-h|--help]`	Display the tool usage and help information and exit.

4 Analyzing Results

The Offline Image Viewer makes it easy to gather large amounts of data about the hdfs namespace. This information can then be used to explore file system usage patterns or find specific files that match arbitrary criteria, along with other types of namespace analysis. The Delimited image processor in particular creates output that is amenable to further processing by tools such as Apache Pig. Pig provides a particularly good choice for analyzing these data as it is able to deal with the output generated from a small fsimage but also scales up to consume data from extremely large file systems.

The Delimited image processor generates lines of text separated, by default, by tabs and includes all of the fields that are common between constructed files and files that were still under constructed when the fsimage was generated. Examples scripts are provided demonstrating how to use this output to accomplish three tasks: determine the number of files each user has created on the file system, find files were created but have not accessed, and find probable duplicates of large files by comparing the size of each file.

Each of the following scripts assumes you have generated an output file using the Delimited processor named `foo` and will be storing the results of the Pig analysis in a file named `results`.

4.1 Total Number of Files for Each User

This script processes each path within the namespace, groups them by the file owner and determines the total number of files each user owns.

numFilesOfEachUser.pig:

```
-- This script determines the total number of files each user has in
-- the namespace. Its output is of the form:
--   username, totalNumFiles

-- Load all of the fields from the file
A = LOAD '$inputFile' USING PigStorage('\t') AS (path:chararray,
                                                 replication:int,
                                                 modTime:chararray,
                                                 accessTime:chararray,
```

```
                                                blockSize:long,
                                                numBlocks:int,
                                                fileSize:long,
                                                NamespaceQuota:int,
                                                DiskspaceQuota:int,
                                                perms:chararray,
                                                username:chararray,
                                                groupname:chararray);


-- Grab just the path and username
B = FOREACH A GENERATE path, username;

-- Generate the sum of the number of paths for each user
C = FOREACH (GROUP B BY username) GENERATE group, COUNT(B.path);

-- Save results
STORE C INTO '$outputFile';
```

This script can be run against pig with the following command:

```
bin/pig -x local -param inputFile=../foo -param outputFile=../
results ../numFilesOfEachUser.pig
```

The output file's content will be similar to that below:

```
bart 1
lisa 16
homer 28
marge 2456
```

4.2 Files That Have Never Been Accessed

This script finds files that were created but whose access times were never changed, meaning they were never opened or viewed.

neverAccessed.pig:

```

-- This script generates a list of files that were created but never
-- accessed, based on their AccessTime

-- Load all of the fields from the file
A = LOAD '$inputFile' USING PigStorage('\t') AS (path:chararray,
                                                replication:int,
                                                modTime:chararray,
                                                accessTime:chararray,
                                                blockSize:long,
                                                numBlocks:int,
                                                fileSize:long,
                                                NamespaceQuota:int,
                                                DiskspaceQuota:int,
                                                perms:chararray,
                                                username:chararray,
                                                groupname:chararray);
```

```
-- Grab just the path and last time the file was accessed
B = FOREACH A GENERATE path, accessTime;

-- Drop all the paths that don't have the default assigned last-access time
C = FILTER B BY accessTime == '1969-12-31 16:00';

-- Drop the accessTimes, since they're all the same
D = FOREACH C GENERATE path;

-- Save results
STORE D INTO '$outputFile';
```

This script can be run against pig with the following command and its output file's content will be a list of files that were created but never viewed afterwards.

```
bin/pig -x local -param inputFile=../foo -param outputFile=../
results ../neverAccessed.pig
```

4.3 Probable Duplicated Files Based on File Size

This script groups files together based on their size, drops any that are of less than 100mb and returns a list of the file size, number of files found and a tuple of the file paths. This can be used to find likely duplicates within the filesystem namespace.

probableDuplicates.pig:

```
-- This script finds probable duplicate files greater than 100 MB by
-- grouping together files based on their byte size. Files of this size
-- with exactly the same number of bytes can be considered probable
-- duplicates, but should be checked further, either by comparing the
-- contents directly or by another proxy, such as a hash of the contents.
-- The scripts output is of the type:
--    fileSize numProbableDuplicates {(probableDup1), (probableDup2)}

-- Load all of the fields from the file
A = LOAD '$inputFile' USING PigStorage('\t') AS (path:chararray,
                                                 replication:int,
                                                 modTime:chararray,
                                                 accessTime:chararray,
                                                 blockSize:long,
                                                 numBlocks:int,
                                                 fileSize:long,
                                                 NamespaceQuota:int,
                                                 DiskspaceQuota:int,
                                                 perms:chararray,
                                                 username:chararray,
                                                 groupname:chararray);

-- Grab the pathname and filesize
B = FOREACH A generate path, fileSize;

-- Drop files smaller than 100 MB
C = FILTER B by fileSize > 100L  * 1024L * 1024L;
```

```
-- Gather all the files of the same byte size
D = GROUP C by fileSize;

-- Generate path, num of duplicates, list of duplicates
E = FOREACH D generate group AS fileSize, COUNT(C) as numDupes, C.path AS files;

-- Drop all the files where there are only one of them
F = FILTER E by numDupes > 1L;

-- Sort by the size of the files
G = ORDER F by fileSize;

-- Save results
STORE G INTO '$outputFile';
```

This script can be run against pig with the following command:

```
bin/pig -x local -param inputFile=../foo -param outputFile=../
results ../probableDuplicates.pig
```

The output file's content will be similar to that below:

```
1077288632 2 {(/user/tennant/work1/part-00501),(/user/tennant/work1/part-00993)}
1077288664 4 {(/user/tennant/work0/part-00567),(/user/tennant/work0/part-03980),(/user/
tennant/work1/part-00725),(/user/eccelston/output/part-03395)}
1077288668 3 {(/user/tennant/work0/part-03705),(/user/tennant/work0/part-04242),(/user/
tennant/work1/part-03839)}
1077288698 2 {(/user/tennant/work0/part-00435),(/user/eccelston/output/part-01382)}
1077288702 2 {(/user/tennant/work0/part-03864),(/user/eccelston/output/part-03234)}
```

Each line includes the file size in bytes that was found to be duplicated, the number of duplicates found, and a list of the duplicated paths. Files less than 100MB are ignored, providing a reasonable likelihood that files of these exact sizes may be duplicates.