CS780/880 Programming Assignment 2

September 24, 2014

Due: Monday 23:59:59, October 6

This assignment is intended to provide you a vehicle for learning and experimenting with some of the basic features of the Granite system and to develop some ideas about testing applications that deal with very large data. The goal is for you to implement a file conversion utility and "verify" that it has worked.

ChunkFile.java xfdlFile n m [p]

Reorganize the input data file so that it is stored in n x m (2D) or n x m x p (3D) blocks, rather than by row and then by slice. If the input data file doesn't map exactly to full blocks, any incomplete blocks should **not** be included in the output file. Write an xfdl file to describe the output data set. The name of the output data file and the xfdl file should be the name of the input data file with "-ch-nxm" or "-ch-nxmxp" appended to the "head" part of the name. In other words, a *ChunkFile* applied to the *head* data set that is organized as 8x8x8 chunks should produce files: *head-ch-8x8x8.bin* and *head-ch-8x8x8.xfdl*.

1. Use a granite iterator to get the chunks as subblocks.
2. You need to be able to write your data blocks to a new binary file.
3. The xfdl output file described above (e.g., *head-ch-8x8x8.xfdl*) describes the new binary data file as having the same structure as the input datasource. You must also write another xfdl file that describes the same binary file as a 2D array (even if the original input is 3D), such that each row of the array is a chunk and there will be as many rows as there are chunks in the chunked file. Call this file <inputName>*-chr-nxmxp.xfdl*.

TestChunking.java inFile.xfdl inFile-chr-nxmxp.xfdl

This utility should open both datasources. You will also need to parse the name of the *chr* file to extract *n, m,* and *p*, the size of each chunk. This program will read standard input (System.in) which should have lines of the form:

```
# lines starting with a # are comments and are ignored
# "d" specifies the location of a single datum to be testedd
d 0 2 3
# "b" specifies the bounds of subblock; every position must be
tested
b 0 2 3  2 3 4
# "a" says to test every indexSpaceId in the data source
a
```

Your code should access each datum specified in the original file and compute its 2D location in the chunkrow organization and access (hopefully) the same datum using that mechanism. You will need to compute the correct chunk address/offset (row,col) from the original IndexSpaceId. You can use getFloats regardless of the data types in the original file and then compare every element of the array to make sure they all match.

Do **not** report the result of a correct comparison. For the "d" and "b" commands, you should print a single line for each IndexSpaceId that fails to match 1 or more attributes; the line should report each attribute that fails and include the correct and incorrect values. Do not report as an error the case when IndexSpaceId references a portion of the original input file that is **not** present in the chunked file because the file size/shape would have required partial chunks.

After processing each command, report the number of locations that match for all attributes, the number of locations with one or more mismatches, and the number of locations that do not exist in the chunked file. Remember, never report errors for the “a” command.

If you are not familiar with the Java *Scanner* class, you should learn how to use it. It is the easiest way to parse a simple input command language such as required here.

UnchunkFile.java file.xfdl file.xfdl-ch-nxmxp.xfdl

The file chunked by *ChunkFile.java* should be restored to linear form. The name of the *xfdl* file can be parsed to determine the chunk size. If the original file mapped perfectly to the chunk size, the result of the un-chunking should be identical to the original file. Replace the “–ch-…” portion of the filenames with “–lin”.

TestUnChunking.java inFile.xfdl inFile-lin.xfdl

The input to this program and its output is identical to that of *TestChunking*. The access to the 2nd datasource is easier, however since you do not need to compute the IndexSpaceId mapping.

Test verification

You must create test input files and a shell script to demonstrate that you have thoroughly tested your program. The input files contain commands for testing a particular data file. Use only data files that are in the “data” directory on the web site and store them all in a “data” subdirectory. Write files you create to the current working directory. **Do not submit the data files**; I have my own copies. Your script file should look something like:

```
echo "----------------- testing attr-0 3x4x2 ----------------------"
java ChunkFile data/attr-0.xfdl 3 4 2
java TestChunking data/attr-0.xfdl attr-0-ch-3x4x2.xfdl < attr-0-ch.in
java UnChunkFile data/attr-0.xfdl attr-0-ch-3x4x2.xfdl
java TestUnChunking data/attr-0.xfdl data-0-lin-3x4x2.xfdl < attr-0-un.in
echo "----------------- testing attr-0 5x5x5 ----------------------"
java ChunkFile data/attr-0.xfdl 5 5 5
java TestChunking data/attr-0.xfdl attr-0-ch-5x5x5.xfdl < attr-0-ch.in
java UnChunkFile data/attr-0.xfdl attr-0-ch-5x5x5.xfdl
java TestUnChunking data/attr-0.xfdl data-0-lin-5x5x5.xfdl < attr-0-un.in
--- much more -------
```

Submission

In addition to your java code, you must also submit a test shell script and the input data files referenced in your shell script.

Please call the test script, “run”.

Please do not put your code into a Java package. Our submission and grading tools don’t know how to deal with a package.

Assuming that 1) you are logged in to agate (or a cs workstation that mounts the agate file system), 2) your current working directory (*cwd*) contains your solution and data files, 3) all the java files in the *cwd* are part of your solution, and 4) all your test input files end with the “.in”, you can submit with a command such as:

```
~cs880/submit P2 run *.java *.in
```