Hawk Weisman

Computer Science 112

Laboratory #5 Lab Report

PLEDGE:

Empirical Analysis of Compression Algorithms

**Methods**

The goal of this laboratory assignment was to determine which of three compression algorithms was the most effective, both in terms of compression and in terms of processing time. The compression algorithms tested were ZIP compression, Pack200, and Pack200 + GZIP. In order to extend the assignment, two additional compression algorithms, LZ4 and 7Z, were also tested. Implementations of these sorting algorithms were run from the command line on a number of different jar files from various sources, while CPU elapsed time (in ms) was recorded using the Unix time command. Sizes (in kB) of the files were then recorded using the “Properties” window of the Ubuntu Linux graphical user interface. Each command was ran five times for each test file, and the elapsed times were averaged. The percent reduction in file size was then calculated using the formula

*r* = 100•(F - F′ / F), where F is the size of the input file and F′ is the size of the output file.

The implementation of ZIP compression tested was the Java jar tool, run using the command /usr/bin/time jar cf <output file name> <input file name> **.** The Pack200 and Pack200 and GZIP compression algorithms were tested using the Unix pack200 tool, using the commands /usr/bin/time pack200 <output file name> <input file name> (for PACK200 and GZIP), and /usr/bin/time pack200 -g <output file name> <input file name>, for Pack200 without GZIP. 7z compression was tested using the package p7zip-full using the command /usr/bin/time 7z a <output file name> <input file name>, and LZ4 compression was tested using an implementation of LZ4 checked out from the Subversion version control repository found at <http://lz4.googlecode.com/svn/trunk/> and executed using the command /usr/bin/time ./lz4demo <input file name> <output file name>. The files tested were bailey.jar, a structures library written by Duane Bailey, profiler.jar, a library containing execution timing code, twitter4j-core-3.0.3.jar, a file containing the Twitter4j library for interfacing with Twitter, StatOMat.jar, a component of a statistics program written as a personal project, and psae.jar, a simple executable jarfile downloaded as a component of a Java tutorial on creating executable jarfiles.

**Results**

Table 1: CPU Elapsed Time (ms)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **StatOMat.jar** | **bailey.jar** | **psae.jar** | **profiler.jar** | **twitter4j-core-3.0.3.jar** |
| **ZIP** | 0.006 | 0.63 | 0.012 | 0.012 | 0.334 |
| **PACK200** | 0.11 | 0.914 | 0.12 | 0.214 | 1.44 |
| **PACK+GZIP** | 0.114 | 0.928 | 0.136 | 0.476 | 1.446 |
| **7Z** | 0.022 | 0.074 | 0.018 | 0.024 | 0.09 |
| **LZ4** | 0 | 0.02 | 0 | 0 | 0.032 |

Table 2: Percent Reduction

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **StatOMat.jar** | **bailey.jar** | **psae.jar** | **profiler.jar** | **twitter4j-core-3.0.3.jar** |
| **ZIP** | 51.28% | 46.48% | 81.72% | 67.23% | 57.76% |
| **PACK200** | -110.53% | 41.81% | 41.18% | 38.96% | 31.30% |
| **PACK+GZIP** | 31.58% | 71.23% | 59.24% | 58.44% | 70.44% |
| **7Z** | -52.63% | 7.38% | 35.29% | 12.99% | 10.44% |
| **LZ4** | 15.79% | 5.71% | 35.29% | 12.99% | 8.67% |

**Analysis**

In general, LZ4 was the fastest compression algorithm, followed by 7z and ZIP. As seen in Table 1, LZ4 was able to compress all files other than twitter4j-core-3.0.3.jar and bailey.jar, the two largest jarfiles, in less than one millisecond. The Unix time command was not capable of greater granularity, so variation in the times taken to compress StatOMat.jar, psae.jar, and profiler.jar is unknown. Figure 1, a graph of the time taken by each algorithm to compress each test file, illustrates the effectiveness of LZ4 and 7Z compression. This performance is not surprising, since LZ4 is referred to as its authors as an “extremely fast compression algorithm”, and its effectiveness has been frequently noted.

In comparison, Pack200 and Pack200 and GZIP both performed extremely poorly in terms of CPU elapsed time. Interestingly, as seen in Figure 1, in some cases, Pack200 and Pack200 + GZIP were very close in performance, but in other cases (such as profiler.jar), the addition of GZIP caused the compression to take as much as twice as long. The fact that Pack200 + GZIP performs so slowly is interesting, given its extensive use in HTTP compression.

File size and contents also influences compression time for all algorithms. All the compression algorithms compressed StatOMat.jar and psae.jar, both of which contained only a very small amount of Java bytecode, very quickly. Interestingly, however, ZIP, 7z, and LZ4 exhibited similar performance on the larger profiler.jar, while Pack200 and Pack200 + GZIP took significantly longer to compress that file. These discrepancies might have something to do with Pack200 being optimized for compressing Java bytecode , as StatOMat.jar and psae.jar contained only Java bytecode, while profiler.jar contained Java bytecode as well as other files, including source code and HTML documentation. Bailey.jar contains only Java bytecode, but is a much larger file containing a great deal more bytecode. Interestingly, twitter4j-core-3.0.3.jar, the file that took the longest for Pack200 and Pack200 + GZIP to compress, contained only bytecode. This extreme compression time might be due simply to the fact that twitter4k-core-3.0.3.jar is a very large file, and all tested algorithms took the most time to compress it.

Percent reduction provides another metric by which the performance of compression algorithms may be assessed. As seen in Figure 2, ZIP compression (the jar tool) tended to provide the greatest percent reduction, except in the case of twitter4j-3.0.3-jar and bailey.jar, where Pack200 + GZIP provided the greatest percent reduction. Pack200 + GZIP tended in general to provide the second-best percent reduction. Interestingly, in the case of StatOMat.jar, Pack200 and 7z both produced a compressed file that was significantly larger than the input file. Why this happened is unknown, but I suspect it has something to do with interaction between those compression algorithms and the content of StatOMat.jar. This seems very unusual, however, because StatOMat.jar is very similar to the other jarfiles tested – it contains only three Java bytecode files and three Java source code files.

In cases where smaller file sizes are more important than execution time, algorithms should be selected based on their percent reduction. ZIP and Pack + GZIP are both strong choices in such a case. In cases where speed of compression is more important than file size, such as one where a great deal of files need to be compressed rapidly, compression algorithms should be selected based on CPU elapsed time. LZ4 is probably the best algorithm in such a case, with 7z a close second.