Video Performance in Java

Mark Claypool, Tom Coates, Shawn Hooley, Eric Shea and Chris Spellacy

Computer Science Department
100 Institute Road
Worcester Polytechnic Institute
Worcester, MA 01609
Phone: (508) 831-5409
Email: `claypool@cs.wpi.edu`

ABSTRACT

The tremendous growth in both Java and multimedia present an opportunity for cross-platform multimedia applications. However, little research has been done on Java multimedia performance. In this paper, we present experiments that measure the multimedia performance of an MPEG-1 client in Java. We find Just-In-Time compilation, local media access and processor type significantly affect multimedia performance, while choice of operating system, Java virtual machine and garbage collection have a negligible effect on multimedia performance. Overall, Java still lags considerably behind multimedia performance in C++.

1 INTRODUCTION

The power of today's computers and the connectivity of today's networks present the opportunity for multimedia from a server, over a network to the desktop. These new streaming multimedia applications promise to enrich our interactions with the power and flexibility of computers. Java is equally promising with the potential to transform application development as we know it. The "write once, run anywhere" nature of Java bytecode continues to score major implementation wins, especially at large organizations whose need for cross-platform solutions overrides other factors. The Java Media APIs are designed to meet the increasing demand for multimedia, supporting audio, video, animations and telephony [Mic99]. The use of Java for continuous media applications is inevitable.

Before Java can be executed, it must first be compiled from source code into what is known as *bytecode*. There are several different ways of executing bytecode as native machine code: a Java Virtual Machine (JVM) is an interpreter that translates the bytecode into machine code one by one, over and over again; a Just in Time (JIT) compiler translates some of the bytecode into machine code just before it is to be used and caches it in memory for reuse; and a static native compiler translates all the bytecode operations into native machine code ahead of time, taking full advantage of traditional compiler optimizations.

Related work on Java performance has concentrated on the performance of traditional benchmarks such as SPEC JVM98 and jBYTEmark in Java environments [HCJ+97, HG98, spec]. CaffeineMark seeks to provide an indicator of Java Applet performance in a Java runtime environment [Sof99]. Other research has concentrated on achieving optimum performance in Java environments [FNN+97]. Such research has shown that JIT and static native compilation can provide impressive performance improvements over purely interpreted Java.

However, traditional benchmarks tend to model traditional application performance. Multimedia applications have very different performance requirements than do traditional applications. Although we often think of multimedia as a continuous stream of data, computer systems handle multimedia in discrete events. An event may be receiving an update packet or displaying a rendered video frame on the screen. The quantity and timing of these events give us measures that affect application quality. There are three measures that determine quality for most multimedia applications [CR99b]: *delay*, the time it takes information to move from the server through the client to the user; *jitter*, the variation in delay, can cause gaps in the playout of a stream such as in an audioconference, or a choppy appearance to a video display; and *loss* which can take many forms such as reduced bits of color, pixel groups, smaller images, dropped frames and lossy compression [CR99a]. In a distributed application, jitter can be caused by disk devices or media codecs, operating system, workstation load and network load. Delay and loss are the primary concerns for traditional text-based applications, while jitter has been shown to be a fundamental concern for multimedia applications [CT99].

In addition, object-oriented languages such as Java make heavy use of memory. Java removes the burden of memory management from the programmer through runtime garbage collection. This freedom comes at a performance price, however, as JVMs often spend 15 percent to 20 percent of their time on garbage collection [HG98]. Most significantly, a chart of the memory usage of a JVM shows a jagged sawtooth pattern (see Figure 1, from [HG98]), indicating that garbage collection is intermittent and likely increases jitter. Moreover, our previous work shows Java servers do suffer from increased jitter versus native-code servers [CT98].

In addition to the Java runtime options of JVM, JIT compilation and garbage collection, client applications may be configured in a variety of other ways, as depicted in Figure 2. A remote server can deliver the video file or the client can access the video file from the local disk, the client processor can be upgraded, or the client application can be developed in C or C++.

In this work we investigate the performance of a client Java MPEG-1 player under two different JVMs, using combinations of JIT compilation and garbage collection. We compare these performance differences across three different processors, local disk access and two operating systems. From this data we determine the greatest bottlenecks to high-quality Java multimedia performance, and how best to improve the overall quality.

2 EXPERIMENTS

In order to measure the performance of Java multimedia, we built a client-server video system. The server, written in C++, streams MPEG-1 frames across the LAN to the client. The client, written in Java, decompresses and displays the video frames on the screen. In the client, we varied the hardware platform, Java virtual machine, JIT compilation, garbage collection, and local versus remote video file.

For accessing the remote file, our client connected to our server with a TCP connection over a socket. Our C++ server is a Win32 console application in order to be as fast as possible and minimize the effects of the server on the performance of the client. The server accepts the name of the MPEG file to transmit as a command line argument. It then listens for a connection on a socket, and transmits the file. The 64 byte MPEG header is sent first, followed by the MPEG data that is broken up into separate frames. This is done using a sliding window that scans the file as it is read from the disk for the MPEG flag signifying the end of a frame. At the end of the file, the remaining data, the last frame, is sent and the socket is closed.

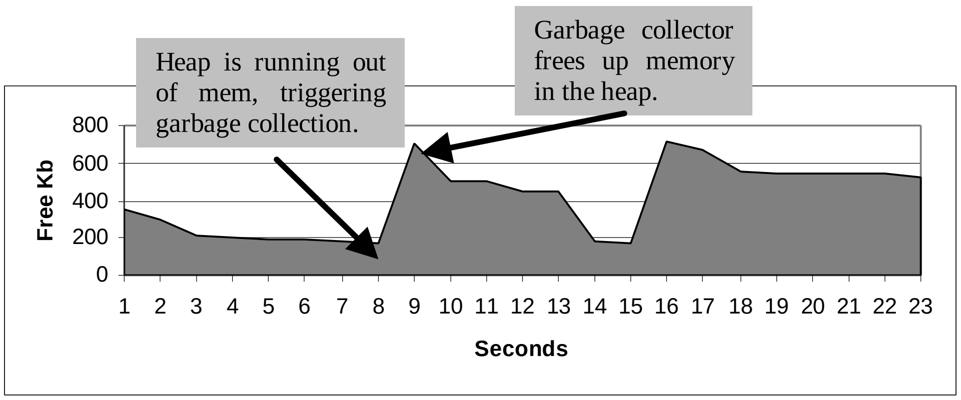
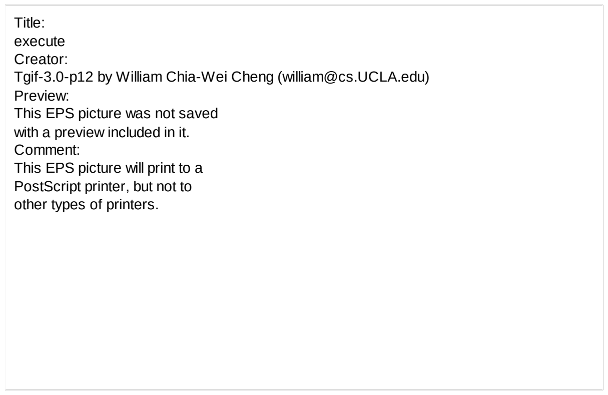


Figure 1. The Pattern of Garbage Collection. This jagged pattern is typical of memory availability when the garbage collector disposes of groups of objects.



Title:
execute
Creator:
Tgif-3.0-p12 by William Chia-Wei Cheng (william@cs.UCLA.edu)
Preview:
This EPS picture was not saved
with a preview included in it.
Comment:
This EPS picture will print to a
PostScript printer, but not to
other types of printers.

Figure 2. Runtime Environments for Video Players. This figure depicts the possible runtime environments for a Java video player and a C++ video player. The options we investigate in this research are in parentheses.

Our client extends a Java Applet developed by Carlos Hasan of the University of Chili [Has98] and is written in Java using Sun's Java Development Kit (JDK) version 1.2. Our client is a multithreaded application instead of an Applet to control the file transmission, with the MPEG decompression and display running in a different thread started by the play button. The client has timing hooks to record performance data to a file.

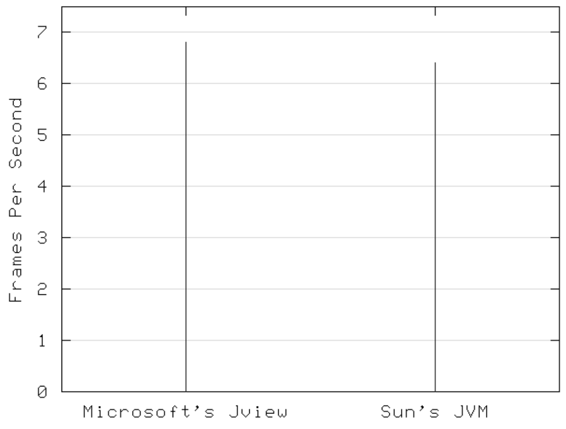


Figure 3. Java Virtual Machine. This figure depicts the impact of the choice of Java Virtual Machine on video performance. The vertical axis is in frames per second. The horizontal axis depicts Microsoft's `Jview` and Sun's `JVM`. Both tests were run on Windows NT.

All tests were run on a dedicated 10 Mbps Ethernet network. The server ran on a separate system. The test cases included running the client on Windows 98 and Windows NT 4.0 Workstation with Service Pack 4 installed, running the Sun's Java virtual machine version 1.2 (called `JVM`) and the Microsoft's Java virtual machine version 5.0 (called `Jview`) with JIT compilation enabled or disabled. We did experimental runs with garbage collection enabled and disabled, and by accessing the MPEG video file from the local disk and from the server over the network.

We tested an MPEG video of a lighter falling through the sky and being lit (`lighter.mpg`). The MPEG statistics are given below:

```
lighter.mpg:
      670 320x240 frames
      Rate: 30 frames/sec
      Length: 22 sec
      GOP: IPBBPBBPBBPBB
      Mean Frame Size: 4554 bytes
      Compression Rate:  1.98%
      Bandwidth: 1.09 MBits/sec
```

During each experimental run, four data points were collected per frame: start decompression time, stop decompression time, start display time and stop display time. Also, the client start and stop times were recorded when the first data packet was received and when the final frame finished displayed, respectively. This allowed us to measure framerate and jitter. Each experiment was run 5 times and the average framerate and jitter for the 5 runs was recorded.

3 RESULTS

3.1 Java Runtime

Figure 3 compares the performance of two Java virtual machines on Windows NT. We tested Sun's `JVM` version 1.2 and Microsoft's `Jview` version 5.0. Sun's `JVM` was only 7% faster. However, we found that there were subtle differences between the two. For instance, `Jview` performs slightly better under Windows NT, whereas `JVM` performs better under Windows 98. Since the performance differences due to the VM are slight, for all subsequent tests we used Microsoft's `Jview`, unless explicitly noted.

Figure 4 shows Java performance using Microsoft's `Jview` with JIT compilation enabled and without JIT compilation enabled on Windows NT. The use of JIT compilation makes a huge difference in performance. When JIT is enabled, the frame rate is almost 7 frames per second. However, with JIT disabled the framerate drops to slightly over 1 frame per second. We found similar results for Sun's `JVM`.

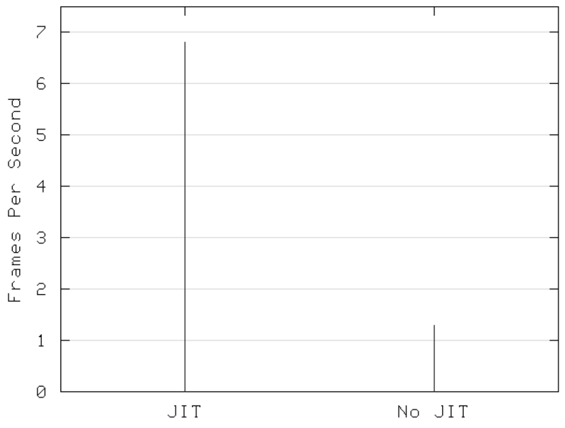


Figure 4. This figure depicts the impact of Just-In-Time (JIT) compilation on video performance. The vertical axis is in frames per second. The horizontal axis depicts Microsoft's `Jview` with and without JIT compilation. Both tests were run on Windows NT.

Figure 5 depicts Java performance with garbage collection enabled and with garbage collection disabled. With garbage collection disabled the frame rate is 7.2 frames per second. When garbage collection is enabled the frame rate is 6.8 frames per second. As shown, garbage collection does not make a significant difference in performance.

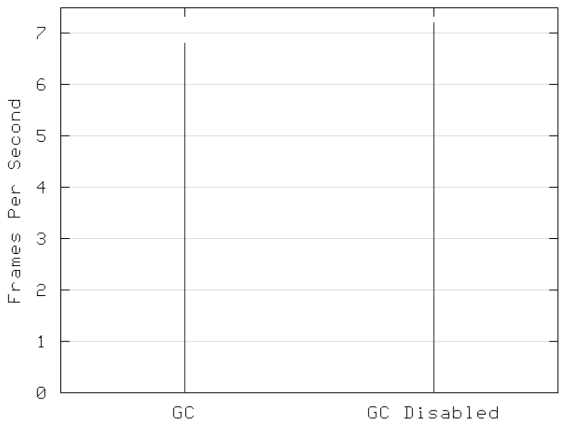


Figure 5. Garbage Collection. This figure depicts the impact of garbage collection on video performance. The vertical axis is in frames per second. The horizontal axis depicts Microsoft's `Jview` with and without garbage collection. Both tests were run on Windows NT.

3.2 Operating System

Figure 6 depicts Java performance under two different operating systems, Windows 98 and Windows NT (v.4.0 service pack 4). Windows 98 provides 4.9 frames per second while Windows NT provides 6.8 frames per second. The performance of Windows NT was also better than that of Windows 98 on the other two systems tested.

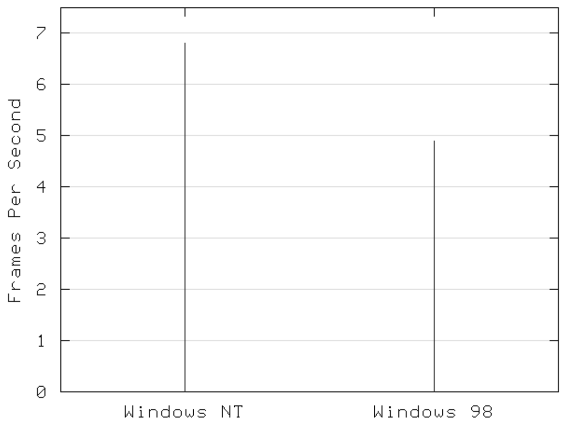


Figure 6. Operating System. This figure depicts the impact of Microsoft operating system choice on video performance. The vertical axis is in frames per second. The horizontal axis depicts Microsoft's Windows 98 and Windows NT operating systems running Microsoft's `Jview`.

3.3 Processor

Figure 7 depicts the Java performance under systems with different processors. It shows the average frame rate for three systems running Windows NT and using Microsoft's `Jview`. The processor that the tests were run on made a large difference on frame rate. We find that the fastest system, a Pentium II 300MHz, has twice as high a frame rate as the slowest system, a Pentium 233MHz.

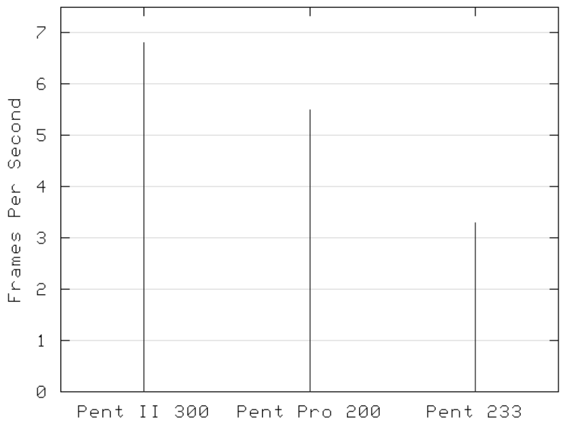


Figure 7. Processor. This figure depicts the impact of processor on video performance. The vertical axis is in frames per second. The horizontal axis depicts three platforms: a Pentium 233 MHz, a Pentium Pro 200 MHz and a Pentium II 300 MHz, all running Microsoft's `Jview`. All tests were run on Windows NT.

3.4 Video Location

Figure 8 shows the impact on framerate from the location of the video to be played out. The client doing local playback read the file from the hard drive. The client doing remote playback connected to a server on a different workstation on the same LAN. Both tests were run on Windows NT. Surprisingly, local playback is over 2.5 times faster than remote playback.

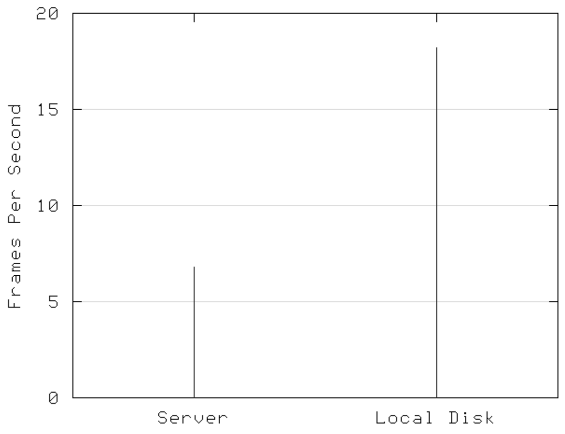


Figure 8. Video Location. This figure depicts the impact that the location of the video file has on video performance. The vertical axis is in frames per second. The horizontal axis depicts accessing a file from the local hard drive or from a server across the network. Both tests were run on Windows NT.

3.5 Interpreted vs. Native

We ran our test video on a MPEG player written in C++ and compiled into native code on Windows NT. The C++ player was able to play the video at full-motion video speed of 30 frames per second, as depicted in Figure 8. Moreover, the C++ player used approximately 15% of the CPU, suggesting a possible maximum playback of 200 frames per second.

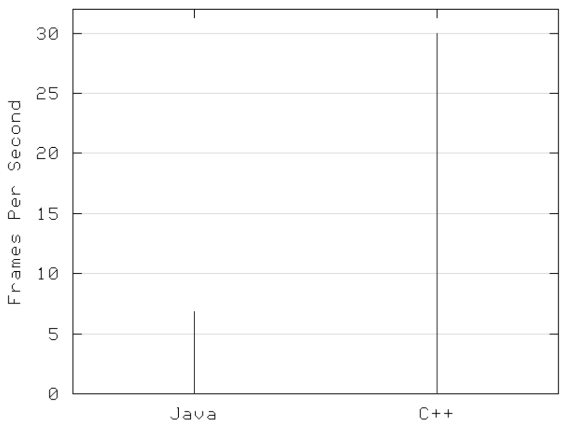


Figure 9. Interpreted vs. Native. This figure depicts a comparison of the multimedia performance of interpreted Java code to native compiled C++ code. The vertical axis is in frames per second. The horizontal axis depicts Java or C++. Both tests were run on Windows NT.

4 CONCLUSIONS

To the best of our knowledge, we are the first to provide experiment-based Java performance for MPEG-1 players. In addition, we provide performance using the multimedia server and under a number of system and Java runtime configurations. Our MPEG client and server allow us to benchmark Java runtime systems and compare performance to C++ runtime systems.

Of the variables that we tested we found that Just-In-Time compilation, local access to the MPEG-1 video, and the client workstation processor type influence multimedia performance the most. Other variables that we tested extensively and found to make a minimal difference were the operating system, the Java Virtual Machine being used, and disabling garbage collection.

After identifying which variables had the greatest impact, we then measured level of performance that we could achieve under ideal circumstances. The best frame rate that we found with our streaming Java MPEG-1 player was 7.5 frames per second, using JIT on the Pentium II 300MHz, Windows NT system. This performance falls far below the full-motion video 30 frames per second. Moreover, native compiled C++ code could theoretically achieve over 200 frames per second on the same system.

Future continuations of this project include in-depth exploration of the effect of Java on jitter. Other research that may prove useful is to investigate how some other video standards perform through Java. Future work also includes performance of new technologies such as Sun's Hotspot, the JavaCPU and JavaOS.

REFERENCES

[CR99a] Mark Claypool and John Riedl, The Effects of High-Speed Networks on Multimedia Jitter, In *Proceedings of IASTED* Euromedia *Conference*, Munich, Germany, April 25-28, 1999.

[CR99b] M. Claypool and J. Riedl. End-to-End Quality in Multimedia Applications. *Chapter 40 in Handbook on Multimedia Computing*, 1999.

[CT98] M. Claypool and J. Tanner. The Effects of Java on Jitter in a Continuous Media Stream. In *Proceedings of IEEE Multimedia Technology and Applications (MTAC) Conference*, September 1998.

[CT99] M. Claypool and J. Tanner. The Effects of Jitter on the Perceptual Quality of Video. In *Proceedings of ACM Multimedia Conference*, November 1999.

[FNN+97] M. Fraenkel, B. Nguyen, J. Nguyen, R. Redpath, and S. Singhal. Building High-Performance Applications and Services in Java: An Experimental Study. In *Object-oriented Programming, Systems, Languages and Applications (Addendum) (OOPSLA)*, pages 16 - 20, 1997.

[Has98] C. Hasan. MPEG-1 Video Stream Decoder Applet, 1998. [Online] at `http://www.dcc.uchile.cl/~chasan/MPEGPlayer.zip`.

[HCJ+97] C. Hsieh, M. Conte, T. Johnson, J. Gyllenhaal, and W. Hwu. Optimizing NET Compilers for Improved Java Performance. *IEEE Computer*, June 1997.

[HG98] T. Halfhill and A. Gallant. How to Soup Up Java. *Byte Magazine*, May 1998.

[Mic99] Sun Microsystems. Java Media Application Programming Interfaces (APIs), May 1999. [Online] at `http://java.sun.com/products/java-media`.

[Sof99] Pendragon Software. CaffeineMark 3.0: The Industry Standard Java Benchmark, 1999. [Online] at `http://www.webfayre.com/pendragon/cm3/index.html`.

[spec] Standard Performance Evaluation Corporation. [Online] at `http://www.spec.org/`