6-20-2000

The Performance of Distance Education Video in Java

Mark Claypool
Worcester Polytechnic Institute, claypool@wpi.edu

Tom Coates
Worcester Polytechnic Institute

Shawn Hooley
Worcester Polytechnic Institute

Eric Shea
Worcester Polytechnic Institute

Chris Spellacy
Worcester Polytechnic Institute

WPI-CS-TR-00-17 June 2000

The Performance of Distance Education Video in Java

by

Mark Claypool
Tom Coates
Shawn Hooley
Eric Shea
Chris Spellacy

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

The Performance of Distance Education Video in Java

Mark Claypool, Tom Coates, Shawn Hooley, Eric Shea and Chris Spellacy
{claypool}@cs.wpi.edu

Computer Science Department
Worcester Polytechnic Institute
100 Institute Road, Worcester MA 01609

June 20, 2000

Abstract

The tremendous growth in both Java and multimedia present an opportunity for cross-platform distance education systems exploiting the power of audio and video to enhance the education process. However, little research has been done on evaluating Java multimedia performance nor on assessing its viability as a platform for distance education. In this paper, we present experiments that measure the multimedia performance of an MPEG-1 client in Java, and evaluate its potential as a distance education platform considering both video frame rate and jitter. We find Just-In-Time compilation, local media access and processor choice significantly affect multimedia performance, while choice of operating system, Java virtual machine and garbage collection have a negligible effect on multimedia performance. While overall Java still lags considerably behind multimedia performance in C++, suitable video performance can be achieved in Java, which, if carefully deployed, promises to enhance distance education systems.

1 Introduction

Traditional education systems are increasingly being challenged with providing education to students that are not able to travel to the schools themselves. At their heart, these *distance education* systems have a teacher and students separated by physical distance and relying upon technology to support the educational process. While early distance education efforts relied upon closed-circuit television and telephones to facilitate instruction, the power of today's computers and the connectivity of today's networks present the opportunity for multimedia from an online teacher, over a network to the desktop of students [19]. These new streaming multimedia applications promise to enhance the effectiveness of distance education

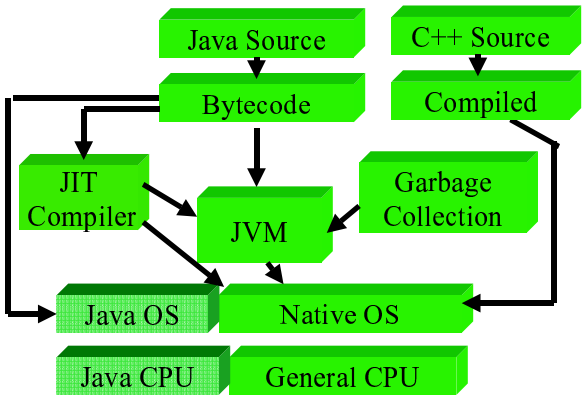

Figure 1: **Java Runtime Environments.** This figure depicts the possible runtime environments for a video application, as well as a C++ application.

programs by allowing interactive sessions between teachers and distant students coupled with the ability to synchronize online voice and video with the lecture notes and slides.

Java is an equally promising technology for distance education with the potential to transform application development as we know it. The "write once, run anywhere" nature of Java bytecode allows distance education providers to deliver a uniform interface, and hence a uniform education, to each student regardless of computing platform. This becomes especially true as end-host computing environments that students may be using are becoming more diversified, from Web TVs and palmtops to PCs and full-featured workstations. The Java Media APIs are designed to meet the increasing demand for multimedia, supporting audio, video, animations and telephony [23]. Distance education systems such as SHLL [4] and CUseeMe [32] that use video along with other presentation media, or distance education approaches such as through adventure games [3] that support interactive distance, or even entire distance education systems such as at the Open University [30], will inevitably depend upon multimedia applications developed in Java.

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 bytecodes into machine code one by one, over and over again; a Just in Time (JIT) compiler translates some the bytecode into machine code just before they are to be used and caches them in memory for reuse; and a static native compiler translates all the bytecode operations into native machine code, taking full advantage of traditional compiler optimizations. Figure 1 shows these run-time options along-side the option of compilation of native.

Related work on Java performance has concentrated on the performance of traditional benchmarks such as Spec95 and the jBYTEmark in Java environments [20, 15]. CaffeineMark seeks to provide an indicator of Java Applet performance in a Java runtime environment [27]. Other research has concentrated on achieve optimum perform in Java environments

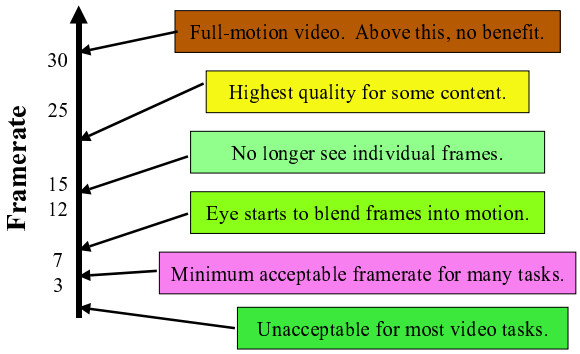

Figure 2: **Frame Rate and Application Quality.** The above figure depicts a range of video frame rates and their affects on the quality of a multimedia application.

[10]. 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 traditional applications. Unlike traditional applications in which response time, such as the time to complete a compilation, is the basic unit of performance, the basic unit of multimedia performance is the rate at which frames (either audio or video) are played. Frame rate has a direct impact on how understandable a distance education lecture will be, how satisfied students will be with the quality of the educational process and even how attractive the educational subject will seem [1, 22, 29, 12, 14]. Figure 2 depicts the effects of frame rate on multimedia quality.

Starting from the bottom, rates of 0-2 frames per second are typically unacceptable for video-based tasks. While the video images may still be useful, the media is perceived more as a series of still images than as a sequence of video frames. 3 frames per second has been found to be the minimum acceptable frame rate for many video-based tasks. At about 7 frames per second, the eye starts to blend the individual frames into smooth motion. From 12-15 frames per second the eye stops being able to distinguish the individual frames. At 20 frames per second, maximum video quality is achieved for low-motion or low-resolution video. 30 frames per second achieves maximum video quality for all videos, as the eye does not perceive any differences for video sequences played at frame rates higher than 30 frames per second.

The performance of a distance education multimedia is also influenced by the successful delivery and timing of the these frames. Although we often think of multimedia as a 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 [5]: *delay*, the time it takes information to move from the server through the client to the user; *jitter*, the variation

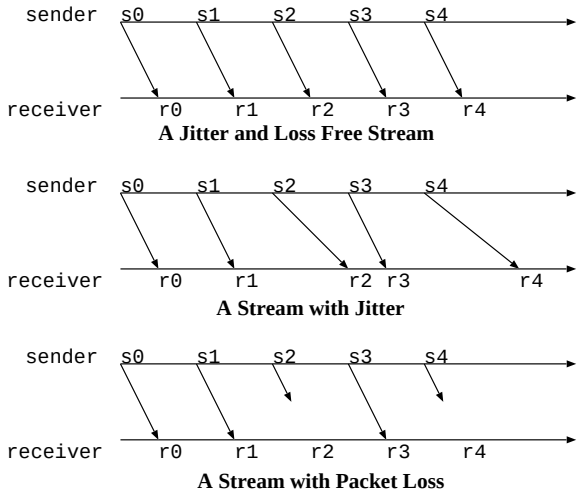

Figure 3: **Jitter and Loss in a Multimedia Stream.** The above figures model packet video between sender and receiver. Each si is the time at which the sender transmits video frame i. Each ri is the time at which the receiver receives frame i.

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.

In the absence of jitter and packet loss, video frames can be played as they are received, resulting in a smooth playout, as depicted in Figure 3-top. However, in the presence of jitter, interarrival times will vary, as depicted in Figure 3-middle. In Figure 3-middle, the third frame arrives late at r*2*. In this scenario, the user would see the frozen image of the most recently delivered frame (frame two) until the tardy frame (frame three) arrived. The tardy frame (frame three) would then be played only briefly in order to preserve the timing for the subsequent frame (frame four). In the presence of packet loss, some frames will not even arrive at the receiver, as depicted in Figure 3-bottom. In Figure 3-bottom the third and fifth frames do not arrive at the receiver. In the case of the loss of frame three, the viewer would see a frozen image of the most recently delivered frame (frame two), and the video stream would then jump to the next frame that arrived (frame four).

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 [7]. Since many network performance studies concentrate on the effects of loss on multimedia quality [24, 17, 16, 13, 12], we concentrate on jitter, and of course frame rate, as measures of Java video performance.

In order to understand how Java might impact the performance of distance-based education using multimedia, it is important to examine some of the overhead associated with Java run-time environments. Figure 4 depicts the run-time overheads associated with running

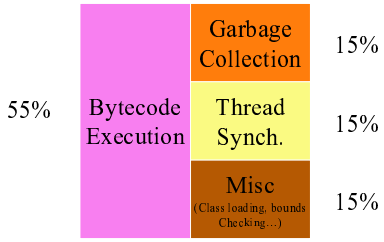

Figure 4: **Java Run-time Execution.** The above figure depicts the relative run-time spent executing Java programs.

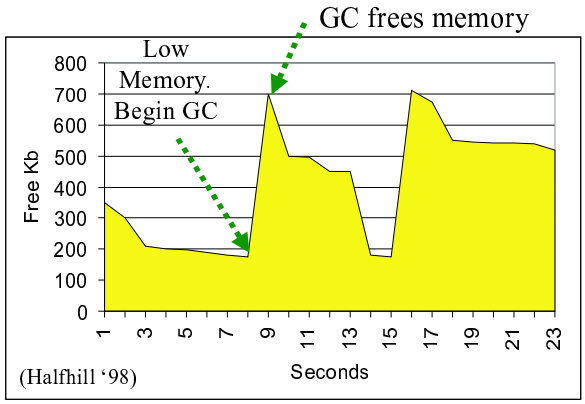

Figure 5: **Garbage Collection.** Available memory decreases as objects are created. Low memory triggers garbage collection which frees up objects that are no longer being used.

Java applications. Roughly half of the time is spent executing bytecode [15]. The other half of the time is used in doing garbage collection to free up memory that is no longer used, synchronizing threads and miscellaneous tasks such as class loading and bounds checking.

As to garbage collection, Java, like other object-oriented languages, makes 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 [15]. Most significantly, a chart of the memory usage of a JVM shows a jagged sawtooth pattern (see Figure 5, from [15]), indicating that garbage collection is intermittent and likely increases jitter. Moreover, our previous work has Java servers do suffer from increased jitter versus native-code servers [6].

In addition to the Java runtime options and choice of JVM, JIT compilation and garbage collection, client applications may be configured in a variety of other ways, as depicted in Figure 1. The video file can be delivered by a remote server or accessed from the local disk,

the client processor can be upgraded, or the client can choose an application developed in C or C++.

In this work we investigate how effective a distance education platform Java can be by measuring 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. We examine both frame rate and jitter in determining Java runtime performance. Our results may be useful for distance education providers in order to decide on appropriate technologies for their students and computer science researchers in order to focus their energies on the most critical bottlenecks in Java performance.

The rest of this paper is laid out as follows: Section 2 describes our experiments used to measure the performance of video in Java; Section 3 analyzes the results from the experiments, including subsections on frame rate and jitter; and Section 4 summarizes our conclusions as to the effectiveness of Java for distance education video and lists possible future work.

2 Experiments

In order to measure the performance of a distance education multimedia using Java, we built a client-server video system designed to simulate a student watching a real-time streaming lecture. As Java has its greatest potential impact as a client-side architecture, our experiments were designed to isolate the effects of Java on video performance at the client. Thus, the server was written in C++ in order to achieve optimum performance. The server streams MPEG-1 frames across a network to the client. The video frames can be read from a file, in the case of a pre-recorded lecture, or from a video codec if the lecture "live" and, in particular, if it is interactive. The client, written in Java, receives the video from the server, renders the frames and plays them on the screen. In the client, we varied the hardware platform, Java virtual machines, JIT compilation, and garbage collection in order to better understand the impact of Java on video performance. Since many distance education programs do, in fact, use a pre-recorded lecture, the client has the option of writing some or all of the video file to a local disk before beginning playout. Thus, we compare the performance of video from across the network from a server to that of video from a local hard drive. Lastly, an additional goal of our experiments is to determine how close video performance in Java comes to that of native code. Thus, we compare the performance of our client coded in Java to that of a client coded in C++.

For accessing the video from the server, our client connected to our server with a TCP connection over a socket. Our server was written in C++ as a Win32 console application 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 socket 1362, and transmits the file. The 64 byte MPEG

header is sent first, followed by the MPEG data which is broken up into separate frames. This is done by the use of a sliding window which 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, which is the last frame, is sent.

Our client built upon a Java Applet developed by Carlos Hasan of the University of Chili [18], and was written in Java using Sun's Java Development Kit (JDK) version 1.2. The 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.

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 Java VM version 1.2 (called Java) and the Microsoft Java VM version 5.0 (called Jview), and running the VMs as JIT enabled or disabled. We ran tests with garbage collection enabled and disabled. Also, data was collected with the local disk version for a comparison of local vs. network performance.

We tested several MPEG-1 videos, but report the results from an animation of a lighter falling through the sky and being lit (`lighter.mpg`):

```
670 320x240 frames, 30 frames/sec, 22 sec
GOP: IPBBPBBPBBPBB, Mean Frame Size: 4554 bytes
Total Compression Rate:  1.98 % of uncompressed 24 bit images =
   0.47 bits per pixel, 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 frame rate and jitter. Each experiment was run 5 times and the average frame rate and jitter for the 5 runs was recorded.

Figure 6 presents the frame rate results and Figure 7 presents the jitter results. The next section analyzes the results, including the metrics themselves and the impact of each video technology on distance education.

3 Analysis

In this section, we analyze the results from our experiments. Our two primary measures of performance are frame rate and jitter. As described in Section 1, the frame rate has a direct impact on the effectiveness of distance education video. A frame rate of 3 frames per second is considered the minimal acceptable frame rate and a frame rate of 20-30 frames per second is a the maximum target. Jitter will determine how smooth the video frames appear to the student. Jitter shown to the student may be perceived as is loss [7] and video that is buffered by the client, a popular technique for reducing jitter, will result in increased latency making

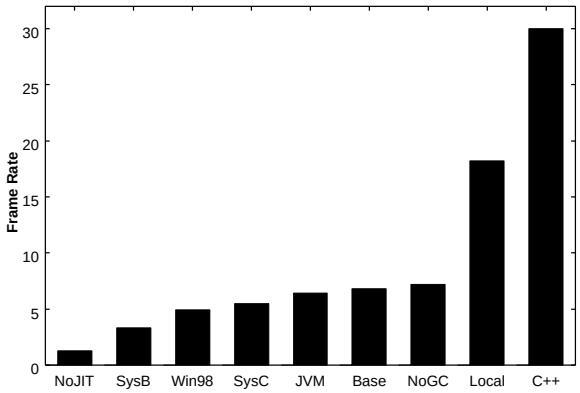

Figure 6: **Video Frame Rate.** This figure depicts the frame rate results from all experiments. The vertical axis is in frames per second. The horizontal axis depicts the runtime environments for the video client. *Base* is a Pentium II 300 MHz Intel PC running Windows NT and Microsoft's `Jview` with JIT and garbage collection enabled, receiving the video from the server over a network. *NoJit* is the Base with Just-In-Time compilation disabled. *SysB* is the Base with a Pentium 233 MHz processor. *Win98* is the Base running Windows 98. *SysC* is the Base with a Pentium Pro 200 MHz processor. *JVM* is the Base running Sun's Java Virtual Machine. *NoGC* is the Base with garbage collection disabled. *Local* is the Base reading video data from the local IDE hard-drive. *C++* is the Base with a compiled C++ player.

interactive lectures more difficult and making the distance education system less effective for student devices with low memory such as palmtops or Web TV's.

3.1 Frame Rate

The frame rate was determined by counting the number of frames in the video sequence and dividing that by the total time required to play the entire video.

3.1.1 Java Runtime

We tested Sun's `JVM` version 1.2 and Microsoft's `Jview` version 5.0. As seen in Figure 6-*JVM*, 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 Java runs better under Windows 98. This suggests that the choice of the Java Virtual Machine does not have a critical bearing on the video performance.

Figure 6-*NoJIT* shows Java performance using Microsoft's `Jview` with JIT disabled, instead of enabled as in the Base system. 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 frame rate drops to slightly over 1 frame per second. We found similar results for Sun's `JVM`. This enormous impact on performance suggests that student devices,

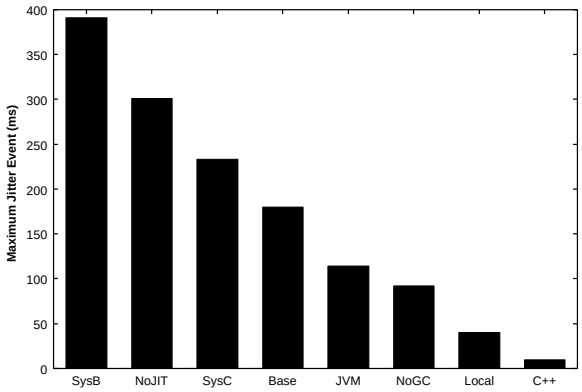

Figure 7: **Video Jitter.** This figure depicts the jitter, measured as the maximum time between consecutive frame playouts, from all experiments. The vertical axis is in milliseconds. The horizontal axis depicts the runtime environments for the video client. *Base* is a Pentium II 300 MHz Intel PC running Windows NT and Microsoft's `Jview` with JIT and garbage collection enabled, receiving the video from the server over a network. *SysB* is the Base with a Pentium 233 MHz processor. *NoJit* is the Base with Just-In-Time compilation disabled. *SysC* is the Base with a Pentium Pro 200 MHz processor. *JVM* is the Base running Sun's Java Virtual Machine. *NoGC* is the Base with garbage collection disabled. *Local* is the Base reading video data from the local IDE hard-drive. *C++* is the Base with a compiled C++ player.

whether a palmtop, laptop or desktop, must be JIT enabled in order to achieve acceptable video performance. Fortunately, the latest versions of Netscape Navigator and Microsoft's Internet Explorer all come with built in JIT compilers.

Figure 6-*NoGC* depicts Java performance with garbage collection disabled instead of enabled as is the usual case. 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 much of a difference in performance. This is fortunate since disabling garbage collection is rarely an option in Java programs, but it also indicates that efforts spent improving the performance of garbage collection routines will not enhance video performance significantly.

3.1.2 Operating System

Figure 6-*Win98* depicts the Java performance under Windows 98 instead of Windows NT (v.4.0 service pack 4) in the Base system. 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 systems. Thus, the choice of operating system on a student device, from a palmtop to a desktop computer, may have a have significant effect on the performance, and hence the effectiveness, of the educational video.

3.1.3 Processor

Figure 6 depicts the Java performance under three different processor environments, the Base system, a Pentium II 300MHz (SysA), a Pentium 233 MHz (SysB) and a Pentium Pro 200 MHz (SysC). 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 shown by Figure 6-*Base*, has twice as high a frame rate as the slowest system, a Pentium 233MHz shown by Figure 6-*SysB*. The benefits from the hardware on the student's machine will definitely affect the impact of the distance education material. Distance education providers will want to explicitly provide minimum client-side hardware requirements in order to ensure that their students are receiving effective educational video.

3.1.4 Video Location

Figure 6-*Local* shows the impact on framerate of the location of the video to be played out. The client doing local playback read the file from an IDE hard drive. The client doing remote playback connected to a server on a different workstation on the same LAN. Surprisingly, local playback is over 2.5 times faster than remote playback. This suggests that Java will be more effective for lectures that are pre-recorded since, as they do not have an interactive component where the instructor talks to the students in real-time, they can be downloaded ahead of time and played out by the students.

3.1.5 Native

We ran our test video on a MPEG player written in C++ and compiled into native code. The C++ player was able to play the video at full-motion video speed of 30 frames per second, as depicted in Figure 6-*C++*. Moreover, the C++ player used approximately 15% of the CPU, allowing a possible maximum playback of 200 frames per second. This enormous increase in performance means that if high-quality, full-motion video is required for the education, only a C++ compiler will suffice. Fortunately, very few distance education systems require this level of performance.

3.2 Jitter

In playing out our video, we recorded the playout time of each frame at the client. The "raw" jitter data, then, is the interarrival time between each frame playout. Figure 8 depicts the raw jitter from one experimental run. A jitter-free playout would appear as a flat-line. Notice that the playout in Figure 8 has enormous "spikes" that are as much as 200 milliseconds throughout the sequence.

Quantitative measurements of jitter used by past researchers have included *co-variance* (ratio of the standard deviation over the mean) [9], *absolute deviation* (does not square the distance from the mean) [26], and *number of gaps* (that occur in the video playout after buffering) [28, 11], and *variance* or *standard deviation* [8, 25, 21, 2].

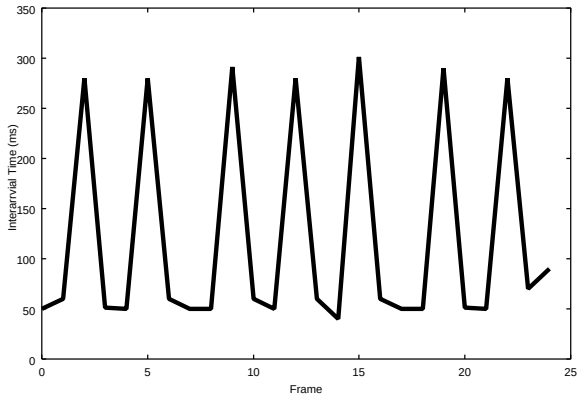

Figure 8: **Java Jitters.** This figure depicts the interarrival times between playout of video frames at the client. The vertical axis is the time between frame playout in milliseconds. The horizontal axis is the frame number. The system is an Intel PC running Windows NT and Microsoft's `Jview` with JIT and garbage collection enabled, receiving the video from the server over a network.

In addition, one useful measure of jitter is *range* (the maximum delay between any two consecutive frames) [31] since it provides the maximum delay needed for complete buffering of jitter events and also represents the largest jitter event that will be seen by the user in the event of unbuffered video. For example, the maximum jitter event in Figure 8 is about 250 milliseconds (between frames 15 and 16).

If all jitter events are removed due to client-side buffering, it is important to understand the implications of end-to-end delay on interactive multimedia. Human perception of delay is around 100 milliseconds, so jitter events less than 100 milliseconds are too small to be noticed. The maximum acceptable delay for interactive multimedia, such as in an interactive classroom experience, is around 250 milliseconds. Delays of 500 milliseconds and above are generally too large for an interactive session.

3.2.1 Java Runtime

As seen in Figure 7-*JVM*, Sun's JVM had a slight effect on jitter, requiring about 75 fewer milliseconds of delay buffering to remove the jitter events. This difference is less than would normally perceived by students watching a video lecture, even if it was interactive.

Figure 7-*NoJIT* shows that JIT compilation makes a significant difference in jitter performance, as the non-JIT video player required over 100 milliseconds more buffering to remove the jitter events. The delay buffering required by video players without JIT compilation is large enough that any interactive distance classroom session will likely be severely degraded.

We hypothesized that the periodic nature of garbage collection were causing the large "spikes" in the interarrival time, seen in Figure 8. Thus, disabling garbage collection should greatly reduce jitter. Figure 9 depicts video jitter with garbage collection disabled compared with video jitter with garbage collection enabled. The mean interarrival time is a bit higher

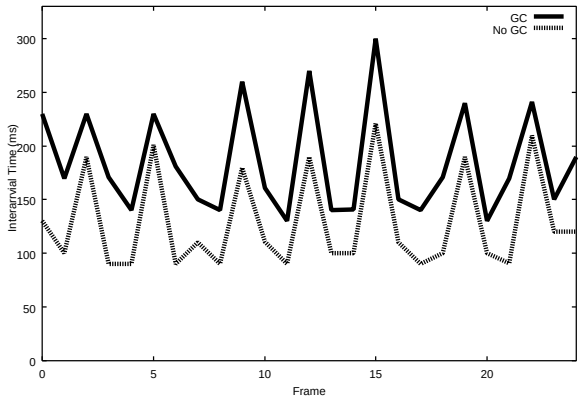

Figure 9: **Garbage Collection Jitter.** This figure depicts jitter, seen by the interarrival times, due to garbage collection. The vertical axis is the time between frame playout in milliseconds. The horizontal axis is the frame number. The top line depicts jitter with garbage collection enabled. The bottom line depicts jitter with garbage collection disabled.

when garbage collection is disabled (seen also as a higher frame rate in Section 3.1.1), but the "spikes" occur at the same locations. The spike size is a bit less with garbage collection disabled, but not to the extent that we expected.

3.2.2 Processor

Figure 7 indicates that the processor speed made an enormous difference on the amount of jitter. The slowest processor (SysB) had unacceptable jitter performance, nearly such that an interactive sessions was unusable. SysC had marginal, but acceptable, performance. This again emphasizes that the hardware on the student's machine will definitely affect the impact of the distance education material. Distance education providers will want to explicitly provide minimum client-side hardware, both for acceptable frame rate performance and jitter performance, in order to ensure that their students are receiving effective educational video.

3.2.3 Video Location

Upon seeing the periodic nature of the jitter "spikes" in Figure 8, we formed an additional hypothesis that the jitter may be due to the network effects, notably the TCP/IP stack. Figure 10 depicts jitter when the video is received over the network compared to the jitter when the video is read from the local hard drive. The local video has a maximum jitter event of around 40 milliseconds, which is much less than the nearly 250 milliseconds of jitter from the network video. However, the periodic nature of the jitter is similar, as both video streams have their "spikes" at about the same places. Still, local video would require less buffering than is perceivable by human perception, while the buffering required for the

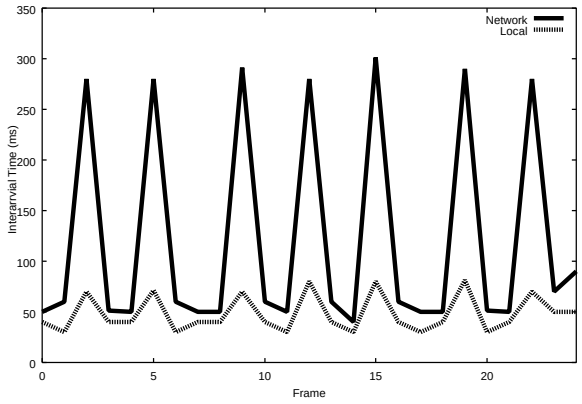

Figure 10: **Network Jitter.** This figure depicts jitter, seen by the interarrival times, due to effects in the network. The vertical axis is the time between frame playout in milliseconds. The horizontal axis is the frame number. The top line depicts jitter when the video is received over the network. The bottom line depicts jitter when the video is read from the local hard drive.

networked video is borderline unacceptable.

3.2.4 Native

Once again the native video player written in C++ performed much better than the Java video player, as seen in Figure 7-*C++*. The C++ player had jitter events on the order of 10 milliseconds and even this jitter was likely do to the clock granularity of around 10 milliseconds and the actual jitter could be less than that if the system supported a higher resolution clock. A native C++ player should definitely be considered for any interactive distance education video as it will immensely reduce the delay requirement to achieve a smooth video stream.

4 Conclusions

The use of distance education to expand the reach of education providers promises to provide opportunities to many who may be otherwise unable to attend school. Java, with its portable nature and toolkits for building user-interfaces, promises to help the reach of distance education institutions by enabling students on a variety of end-user devices that support a variety of operating systems to partake in the education system. Java can provide a mechanism for students to receive multimedia lectures, support interactive on-line help sessions and provide interactive working environments.

Despite this potential, any detailed understanding of multimedia performance in Java has not been undertaken. To the best of our knowledge, we are the first to provide experiment-based Java performance for MPEG-1 video. Moreover, our study provides performance num-

bers using a typical multimedia server and client under a number of runtime configurations, allowing us to detect the bottlenecks in Java performance.

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 operating system, the Java Virtual Machine being used, and disabling garbage collection.

After identifying which variables had the greatest impact, we then measured the level of Java performance that we could achieve under ideal circumstances. The best frame rate that we found with our streaming Java MPEG-1 player was 7.45 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.

Our identification of performance bottlenecks is useful for distance education providers as they can decide upon the necessary Java runtime configurations required to support their instructional systems. Computer science researchers can also use this information to focus their energies on improving Java based multimedia in the areas that will have the greatest impact on performance.

5 Future Work

A great deal of work still remains to be done in this area of study. First most, a Java-based education system should be deployed, following the configuration guidelines mentioned above, to verify that the effects on users are as indicated by video used in prior user studies. Often, actual limits of acceptable performance, such as for frame rate and jitter, are task-specific. Distance education video may well carry its own set of acceptable performance criteria.

As the network appears to be a significant bottleneck to smooth video performance, and most distance education based systems will require a networking component, future work should do an in-depth study of the effects of different kinds networking on video performance, especially jitter. For example, a client-server combination can use Remote Method Invocation (RMI) or CORBA to have "pull" technology, possibly achieving better throughput. Or, the Java-to-Native Interface (JNI) can be used to access compiled code that handles networking possibly achieving better networking performance. UDP, rather than TCP, is often the preferred network protocol of choice for multimedia applications. Therefore, further studies detailing the effects of UDP, RMI or JNI may yield better networking performance for Java based distance education.

Java technology continues to evolve. Detailed experiments on the performance of new technologies such as Sun's Hotspot, the JavaCPU and JavaOS may provide further insight as to the most effective ways of providing Java-based distance education tools. Similarly, new multimedia technologies such as MPEG-4 may provide a better forum for distance education instruction.

6 Notes

The complete source code used in this research can be downloaded from the Perform Web page at:

`http://perform.wpi.edu/`

References

[1] Ronnie T. Apteker, James A. Fisher, Valentin S. Kisimov, and Hanoch Neishlos. Video Acceptability and Frame Rate. *IEEE Multimedia*, pages 32 – 40, Fall 1995.

[2] Barberis and Pazzaglia. Analysis and Optimal Design of a Packet Voice Receiver. *IEEE Transactions on Communication*, February 1980.

[3] L. Carswell and D. Benyon. An Adventure Game Approach to Multimedia Distance Education. In *Integrating Technology into Computer Supported Education*, pages 122 – 124, June 1996.

[4] H. Chen, Y. Chia, G. Chen, and J. Hong. An RTP-based Syncrhonized Hypermedia Live Lecture System for Distance Education. In *Proceedings of the ACM Multimedia Conference*, volume 1, pages 91 – 99, November 1999.

[5] Mark Claypool and John Riedl. *End-to-End Quality in Multimedia Applications (Ch 40)*. CRC Press, Boca Raton, FL, 1999.

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

[7] Mark Claypool and Jonathan Tanner. The Effects of Jitter on the Perceptual Quality of Video. In *Proceedings of the ACM Multimedia Conference*, volume 2, November 1999.

[8] Domenico Ferrari. Delay Jitter Control Scheme for Packet-Switching Internetworks. *Computer Communications*, 15(6):367 – 373, July 1992.

[9] Sally Floyd, Mark Handley, Jitendra Padhye, and Jorg Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proceedings of ACM SIGCOMM Conference*, 2000. To appear.

[10] 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.

[11] Daniel Frankowski and John Riedl. Hiding Jitter in an Audio Stream. Technical Report TR-93-50, University of Minnesota Department of Computer Science, 1993.

[12] G. Ghineas and J.P. Thomas. QoS Impact on User Perception and Understanding of Multimedia Video Clips. In *Proceedings of ACM Multimedia Conference*, Bristol, UK, September 1998.

[13] Steven Gringeri, Ghumip Khasnabish, Arianne Lewis, Khaled Shuaib, Roman Egorov, and Bert Basch. Transmission of MPEG-2 Video Streams over ATM. *IEEE Multimedia*, 5(1):58 – 71, Jan-Mar 1998.

[14] X. Guo and C. Pattinson. Quality of Service Requirements for Multimedia Communications. In *Proceedings of Time and the Web Workshop*, June 1997.

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

[16] V. Hardman, M. A. Sasse, and I. Kouvelas. Successful Multi-party Audio Communication over the Internet. *Communications of the ACM*, 41(5):74 – 80, 1998.

[17] Vicky Hardman, Martina Angela Sasse, Mark Handley, and Anna Watson. Reliable Audio for Use over the Internet. In *Proceedings of Internet Society's International Networking Conference (INET)*, 1995.

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

[19] Starr Roxanne Hiltz. Teaching in a Virtual Classroom. In *Proceedings of the International Conference on Computer Assisted Instruction*, 1995. URL: http://www.njit.edu/CCCC/VC/Papers/Teaching.html.

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

[21] Saimin Jin, Dhadesugoor R. Vaman, and Divyendu Sina. A Performance Mangement Framework to Provide Bounded Packet Delay and Variance in Packet Switched Networks. *Computer Networks and ISDN Sytems*, pages 249 – 264, September 1991.

[22] Michael J. Massimino and Thomas B. Sheridan. Teleoperator performance with varying force and visual feedback. In *Human Factors*, pages 145 – 157, March 1994.

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

[24] C. Perkins, O. Hodson, and V. Hardman. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Network Magazine*, Sep/Oct 1998.

[25] Ramachandran Ramjee, Jim Kurose, Don Towsley, and Henning Schulzrinne. Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks. In *Proceedings of the 13th Annual Joint Conference of the IEEE Computer and Communications Societies on Networking for Global Communciation*, volume 2, pages 680 – 688, June 1994.

[26] Henning Schulzrinne. Voice Communications across the Internet: A Network Voice Terminal. Technical report, University of Massachussetts Department of Electrical Engineering, August 1992.

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

[28] D. Stone and K. Jeffay. An Empirical Study of Delay Jitter Management Policies. *ACM Multimedia Systems*, 2(6):267 – 279, January 1995.

[29] Merryanna Swartz and Daniel Wallace. Effects of Frame Rate and Resolution Reduction on Human Performance. In *Proceedings of IS&T's 46th Annual Conference*, Munich, Germany, 1993.

[30] P. Thomas, L. Carswell, J. Emms, M. Petre, B. Poniatowska, and B. Price. Distance Education over the Internet. In *Integrating Technology into Computer Supported Education*, pages 147 – 149, June 1996.

[31] Dinesh C. Verma, Hui Zhang, and Domenico Ferrari. Delay Jitter Control for Real-Time Communication in a Packet Switching Network. *IEEE Computer*, pages 35 – 43, 1991.

[32] Individual.com via NewsEdge Corporation. OneNet and CUseeMe Networks Show First Web-Based E-Learning Demo With Live Interactive Video and Integrated Content, June 2000.