Chapter 17

Video Performance in Java

Mark Claypool, Tom Coates, Shawn Hooley, Eric Shea and Chris Spellacy
Worcester Polytechnic Institute, MA

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 chapter, 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++.

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 (SunMicrosystems, 1999). 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 (Hsieh, Conte, Johnson, Gyllenhaal and Hwu, 1997; Halfhill and Gallant, 1998; Standard Performance Evaluation Corporation). CaffeineMark seeks to provide an indicator of Java Applet performance in a Java runtime environment (Pendragon Software, 1999). Other research has concentrated on achieving optimum performance in Java environments (Fraenkel, Nguyen, Nguyen, Redpath and Singhal, 1997). 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 (Claypool and Riedl, 1999): 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 (Claypool and Riedl, 1999). 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 (Claypool and Tanner, 1999).

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 (Halfhill et al., 1998). Most significantly, a chart of the memory usage of a JVM shows a jagged sawtooth pattern (see Figure 1, from Halfhill et al., 1998), indicating that garbage collection is intermittent and likely increases jitter. More-
Related Content

Using Web 2.0 Features on Social Networks for Word-of-Mouth Effects
www.irma-international.org/chapter/using-web-features-social-networks/60472/

A Top-Down Framework for Modeling Routing Design Complexity
www.irma-international.org/chapter/a-top-down-framework-for-modeling-routing-design-complexity/131367/

Studying Web 2.0 Interactivity: A Research Framework and Two Case Studies
www.irma-international.org/article/studying-web-interactivity/75310/

Beauty in the Background: A Content Analysis of Females in Interactive Digital Games
www.irma-international.org/article/beauty-background-content-analysis-females/75313/

A Diacritical Study in Web Design: Communicability Versus Statistical Manipulation
Francisco Vicente Cipolla Ficarra and Maria Valeria Ficarra (2010). Quality and Communicability for Interactive Hypermedia Systems: Concepts and Practices for Design (pp. 91-120).
www.irma-international.org/chapter/diacritical-study-web-design/41085/