Finding parameters which influence the performance of Java for streaming video over the network

Vikram Chhabra, Akshay Kothare and Mark Claypool
Computer Science Department
Worcester Polytechnic Institute
100 Institute Road, Worcester, MA 01609
{vikram,akshay,claypool}@cs.wpi.edu

Abstract

Java has been growing tremendously as a language and as a platform over the past few years. Multimedia applications written in Java are gaining popularity because of the platform independence of Java. Basically, the write once - run anywhere nature of Java applications is what makes Java a potential contender for developing software compared to other high level languages. However, there is still a lot of work that needs to be done to determine how multimedia performance is affected in Java using its various features such as Garbage collection, Just in Time (JIT) compilation. In this paper, we present experiments that measure the multimedia performance of an MPEG-1 client in Java, considering the frame rate and jitter. We find that Just in Time compilation, different MPEG files, local media access as well as access to media over a network, have implications on multimedia performance. Local media access shows less jitter than the network access and different MPEG files show different jitter amounts. Although, overall Java lags behind C++, as far as multimedia performance is concerned, we believe that improvement in performance can be achieved in Java through the use of its inherent features like JIT.

Keywords: Frame Rate, Delay, Jitter.

1. Introduction

Java, being an interpreted language, lacks in performance as compared to C++, however it manages to score over C++ in the area of platform independence and security. The write once - run anywhere nature of Java bytecode is what makes Java a very serious contender for developing multimedia-streaming applications. Before a Java application can be executed, its source code needs to be compiled into bytecode. *Bytecode* is the object code that is processed by the Java Virtual Machine. The *virtual machine* acts as an interface between the compiled Java code and the underlying hardware platform. This bytecode can then be executed in a number of ways such as Just in Time compiler, Static Native Compiler or a Java CPU. A *Just in Time compiler* (JIT) translates the bytecode into machine code just before it is to be used and caches the machine code in memory for reuse. On the other hand, a *Static Native Compiler* compiles the bytecode into optimized machine specific code for the target platform and makes full use of all the optimizations that a traditional compiler has to offer. This resulting code is generally faster than the bytecode executed by the Java Virtual Machine even with its JIT compilation option being turned on. Java doesn't use pointers, which are a major source of bugs and security leaks in C or C++.

Multimedia means more than one medium for presentation. It could be combination of text, sound and images. We restrict our definition of multimedia to video only. *Frame Rate*, *Delay* and *Jitter* are critical parameters when it comes to delivering real-time video over the Internet. *Frame Rate* is the number of frames displayed per second. *Delay* is the time taken for a frame to travel from the server to the client and *Jitter* is the variance in this delay. These parameters influence the quality of video output that the client receives. A substantial amount of delay between each frame can cause the video to play at a slower frame rate than its actual frame rate and hence appear to be disturbing. The actual frame rate is one at which the MPEG file is encoded. On the other hand, in the presence of jitter the client would see pictures either having a few frozen frames or see jumping of frames, in order to preserve the timing for its subsequent frame.

Since the client and server are at remote locations, hence we use the reflection of delay i.e. the frame *inter-arrival time* to measure the delay. So for all practical purposes, *Jitter* is the variance in this inter-arrival time. *Loss* is another parameter which can be experienced in different ways in terms of lost bits, lost frames, corrupt frames etc. Text applications such as chat sessions are affected by parameters such as delay and loss. As far as video applications are concerned, it is passable to have a few lost frames and have a constant delay but the presence of jitter is not tolerable for multimedia applications.

There are a number of parameters that need to be tested to evaluate the performance of Java for streaming multimedia. We have tested four such parameters. Our first parameter is different network setups in which the distance between the client and server is varied. The second parameter is different types of movie files, static to dynamic. The third parameter is different versions of Java, and finally the fourth parameter is toggling the JIT on and off. This paper is divided into the following sections: the previous work done for evaluating Java's performance, our approach that talks about our test strategy, results of our tests, conclusions we draw from these results and finally future work for areas which we did not cover.

2. Previous Work

To evaluate the performance of Java for streaming video, Claypool *et al.* designed a client-server using the TCP/IP protocol [1]. Although the client was written in Java, the server was written in C++ to avoid any performance degradation on the server side. The task of this server is to break an MPEG-1 layer file into frames and then send these frames over the network using the TCP/IP protocol. The server keeps reading the MPEG file until it reaches a frame delimiter and then sends it to the client. The client receives this TCP stream and displays it on the local terminal. The client was originally written by Carlos Hasan as an applet and then modified by Claypool *et al.* [1] to add timing hooks which would note: the frame start decompress time, stop decompress time, start display time and stop display time, in milliseconds, and store them into a text file. A number of experiments were performed to evaluate the performance of Java for streaming video. These included toggling JIT and garbage collection features of JVM, playback of the MPEG file using different setups such as different processors, comparison with C++ etc. Their results, as stated in [1], point out that JIT, local access of the file and the processing power of the client mostly influence the performance. We shall continue their work and test other possible variables involved that could affect the performance of streaming video using Java.

3. Our Approach

The work done until now used a server and client made for the Microsoft based operating systems. Our first task was to port the code for Linux operating system. The client did not require any change, which was expected given multi-platform feature of Java, but the server required some changes. These included, removing calls to the Winsock library of windows, such as removing the WSAstartup function etc. To perform our experiment we used two dedicated Intel based machines and installed SUSE Linux 6.3 kernel version 2.2.13 on both of them, one to run the server and the other for the client. The server was a Pentium MMX @ 233 MHz PC with 64MB of RAM and the client was a Pentium II @ 300 MHz PC with 128MB of RAM. The two machines had Etherlink Network Cards, which were connected to the network using a broadband Coaxial Cable at 100Mbps. Our performance measuring tools were the *frame rate* and *jitter*. The following subsections underline the parameters we tested:

3.1 Underlying Network Setup

Our first test was to find out the effect of network on jitter. So we designed different network setups in the following hierarchical manner :

1.1 " Local Playback "
Local playback of the MPEG Video file from IDE hard disk using our client (written in Java) on our Linux box

1.2 " Dedicated Server-Client Setup "
Run the server and the client on our two Linux boxes connected to each other but disconnected from the rest of the world

1.3 " Normal Setup "
Run the client and server on our two Linux boxes which are connected to the rest of the world

1.4 " Different Switch Setup "
Run the client on our Linux box but run the server on a machine which falls under a different switch (Different departments of WPI)

1.5 " Different Gateway Setup "
Run the client on our Linux box but run the server from outside WPI gateway

So for each test the client was run on the same Linux machine and at every step the server was moved further away from the client. These tests were performed with two different MPEG files. We shall discuss the results of these tests in the next section.

3.2 Type of MPEG File

The next goal was to test the effect of the type of MPEG file on jitter. Video clips could vary from static to dynamic. Static videos are like news clips or interviews, where there is less movement between successive frames. On the other hand, dynamic videos are fast changing video sequences, such as action or sports scenes. To perform this test, we used the "normal setup" i.e. client and server running on two different Linux machines connected to the rest of the world.

3.3 Versions of Java

The next variable we tested was different versions of Java, older ones, in which JIT was provided by default, and the new ones, in which JIT needs to be added separately. We used Sun's JDK 1.2.2, JDK 1.1.8 and JDK 1.1.7. According to the documentation, JDK 1.1.7 and 1.1.8 have JIT on by default. We could not find any difference in performance after switching JIT off. On the other hand, JDK 1.2.2 doesn't have JIT on by default. We tried using Borland's JBuilder 3.5 with it, but could not see any performance gain. In our tests we have used Shudo Kazuyuki's ShuJIT [http://www.shudo.net/jit].

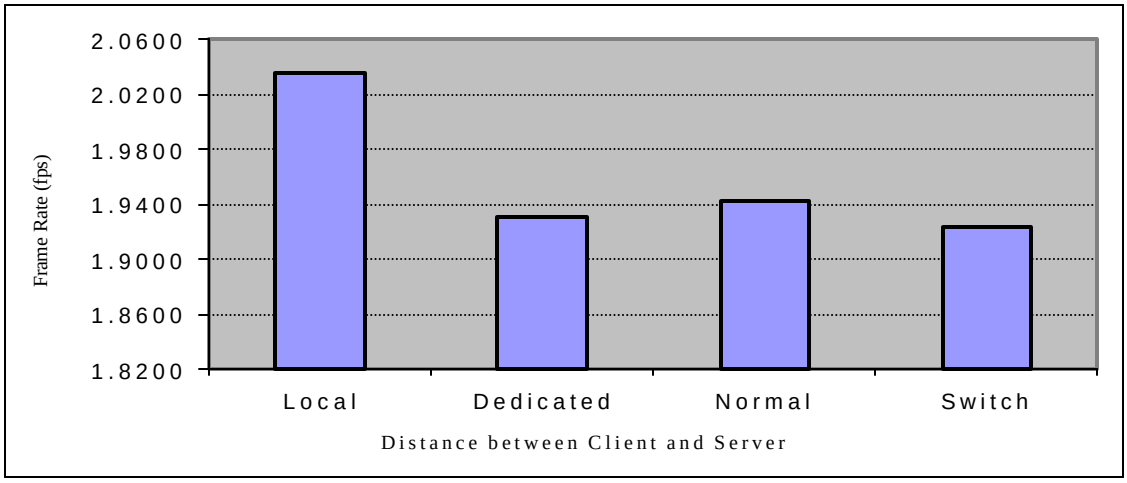
3.4 Just In Time (JIT) compilation

Our final variable was to toggle the JIT on and off. We noticed a substantial performance increase with the JIT turned on. We shall discuss this in detail in the next section.

4. Results

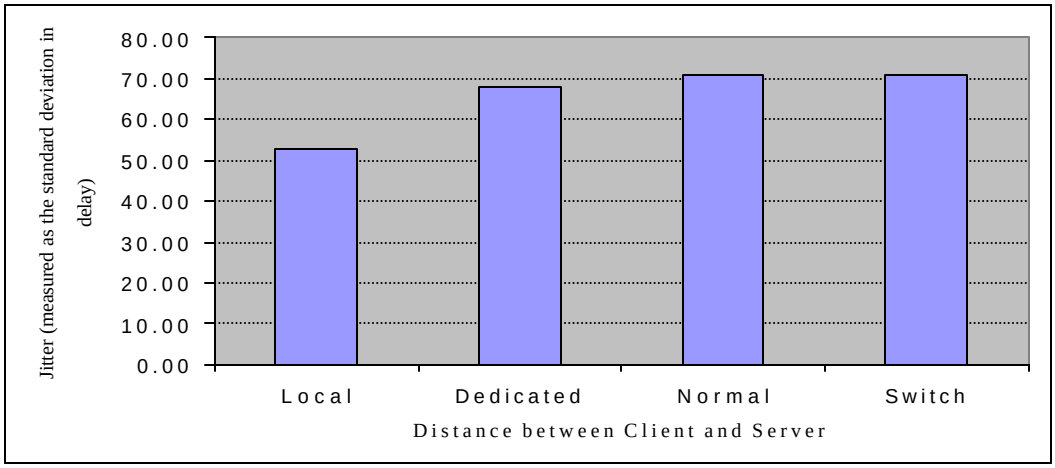
4.1 Underlying Network Setups

Our first set of tests shows that the access to local media is faster than access to media over the network. But, increasing the distance between the client and server over the network does not seem to matter. It did not show any significant difference or pattern. The following histogram shows these results.



Test I : Varying the distance between the client and server.
The Frame Rate for local access is faster than access over network. But over the network, results are almost the same, irrespective of the distance between the client and the server.

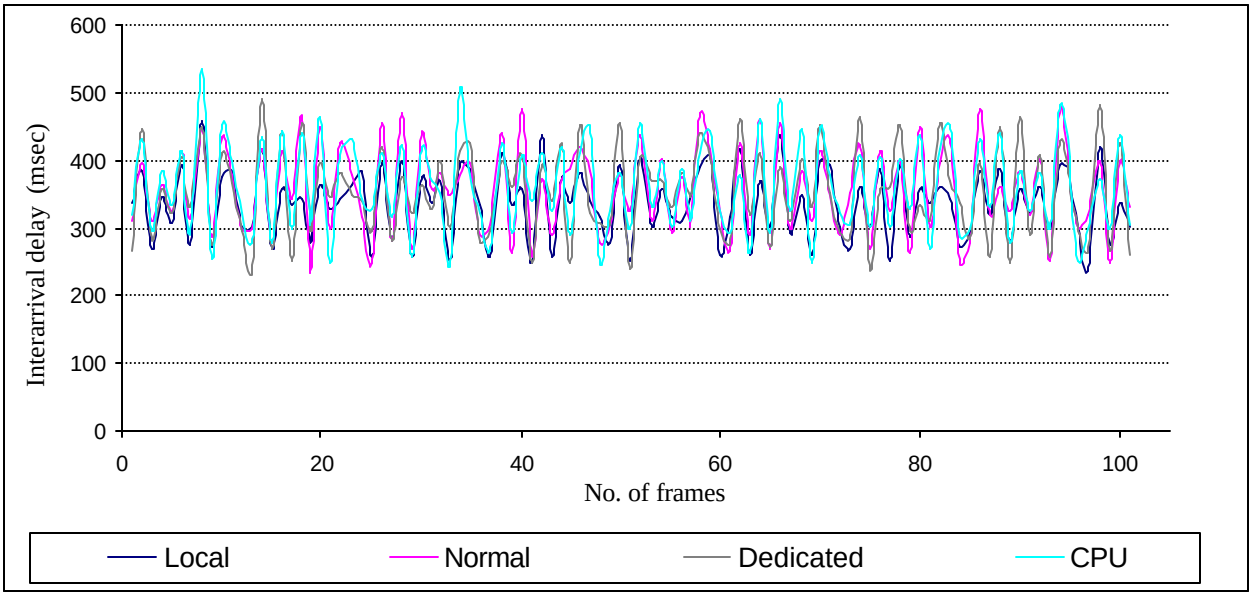
Like the frame rate, the jitter follows the same pattern. It is less for local access and more for network access and does not vary much when tested for different network setups, as seen in the following histogram



Test I : Varying the distance between the client and server.
The Jitter for local access is less than access over network. But over the network, results are almost the same, irrespective of the distance between the client and the server.

These results were consistent for two different MPEG files (launch.mpeg and maar_peet.mpeg, details of these MPEG files described later.)

The following graph shows the jitter when the MPEG file is accessed under different types of network settings, which we have specified in the previous sections.



Test I : Performance of theMPEG file under different network setups

Jitter here translates into the variation in the interarrival delay. The graph depicts that there is not too much of a difference in the jitter experienced when the MPEG file is served over different network settings. However, if we see the histogram above, we see that there is some amount of difference in the interarrival times of successive frames, when the MPEG file is accessed locally. Over the various network setups, i.e. when the distance between the client and the server increases, we don't see too much of a variation in jitter.

There was one setup, which we tested but cannot furnish the results that we obtained from that as we encountered inconsistent results due to a change in the hardware to the machine on which we ran the tests. We still need to probe the proposed setup in which the client and the server fall under different gateways under similar conditions, which existed when we ran the rest of the tests. Initially we hadn't configured the default gateway in Linux, due to which we couldn't access the network outside of the default gateway. We also encountered a minor problem whereby we had to change the monitor for the Linux box. However, once the default gateway was setup and the monitor replaced we have observed that the increase in performance has been twice what we had achieved before we made the above changes in the system settings. We have not been able to pinpoint the cause of this increase in performance and shall leave it for our future work.

4.2 Types of MPEG files

Our next goal was to find whether the type of MPEG file being served has any effects on the jitter and frame rate. We did find that the kind of content being served does have implications on the on jitter and frame rate. Our sample set consisted of four mpg files, each with its own characteristics, such as the difference in between successive frames. For e.g. a news clip has very little motion associated with it, as the object of focus (e.g. newsreader in a newsroom) doesn't change much with each successive frame. There might be a certain file in which the object of focus or the image changes quite frequently across consecutive frames, as in an action movie. A file in which the object of focus changes at certain intervals such as in the launching of a space shuttle, when initially we see gas fumes from the rear side of the shuttle and then as it launches, each of its different stages fall off after having taken the shuttle to the desired velocity and having consumed all the fuel that there might have been for that particular stage.

We include below a classification of the types of frames contained in the movies we used for performing our tests.

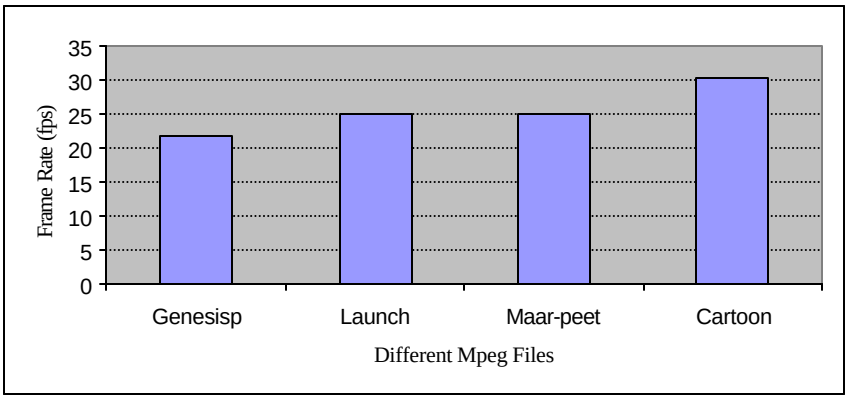
Movie name	Classification	I frames	P frames	B frames
Maar_peet.mpg *	Action - A lot of difference across consecutive frames.	25	50	220
Genesisp.mpg	Slow - Successive frames change after some period of time.	641	-	-
Launch.mpg	Static - Successive frames change after long periods of time.	409	-	-
Cartoon.mpg	Cartoon - Successive frames change quite frequently	109	109	432

* This was our default movie, which was used for comparing performance for all different setups

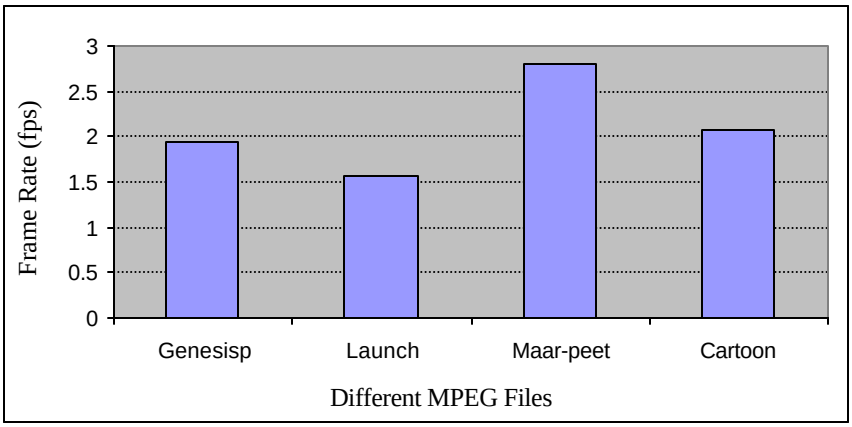
Following frame rates are obtained with the movie player written in C (mpeg_play):

Genesisp.mpg	21.69 frames/sec.	Launch.mpg	25.02 frames/sec.
Cartoon.mpg	30.34 frames/sec.	Maar_peet.mpg	25.06 frames/sec.

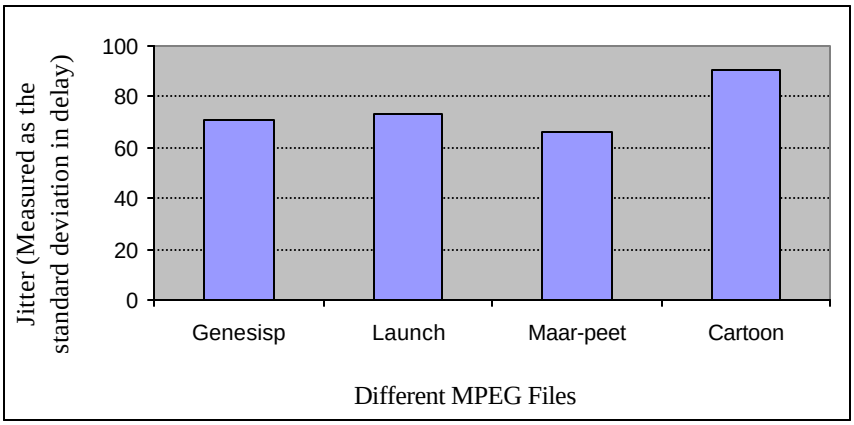
The following histograms make it evident that Java still lags in performance as compared to C.



Test II : Frame Rate using a C player (mpeg_play)

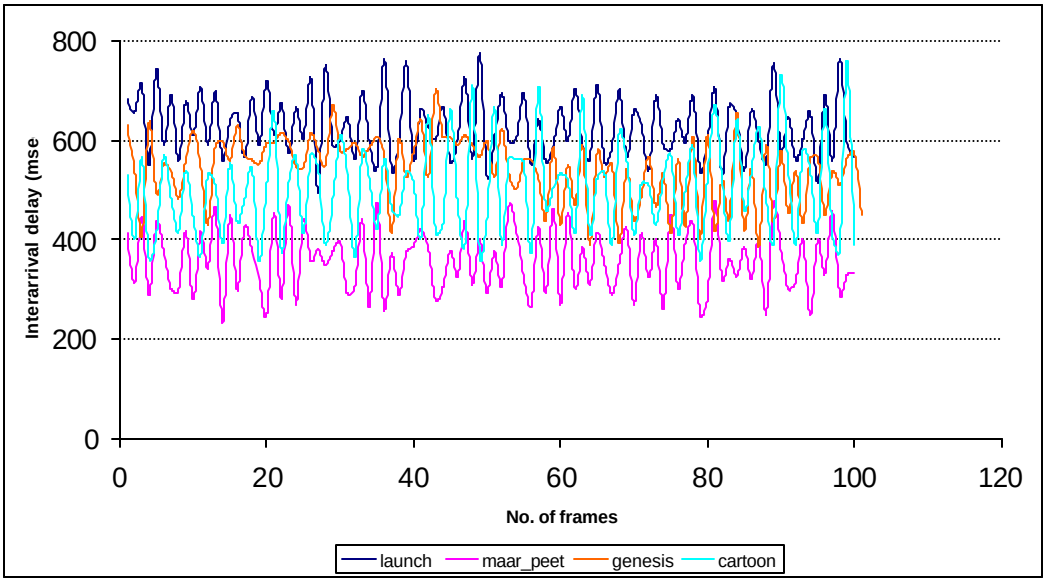


Test II : Frame Rate using our Java client : The frame rate pattern for different MPEG files using our Java client does not tally with that of the C player.



Test II : Jitter using our Java client :
It varies from file to file but there is no matching pattern with the frame rate

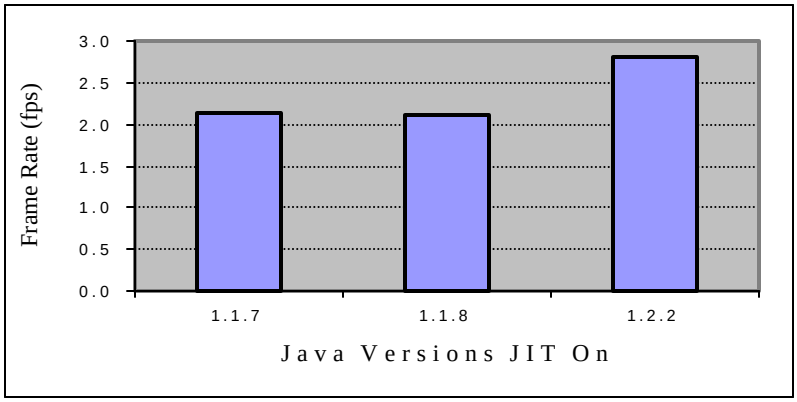
The Cartoon file, which performs the best for the C player, does not perform very well with our Java player. On the other hand, maar_peet (the action sequence) performs decently as compared to other files. It has I frames as half the number of P frames, where as cartoon has the same number of I and P frames. The cartoon file shows the maximum jitter, where as the action file shows the minimum. So it is unclear that whether the type of frame is responsible for all these results. Looking at launch and genesisp, we cannot say much either. Both have only I frames, with launch having a lower compression rate as compared to genesisp. Though both show comparable amount of jitter, but the frame rates are quite different, in fact, the results are opposite to what we get using the C player. A possible explanation for higher Jitter in cartoon over maar_peet and same jitter in genesisp and launch is the number of I frames. Since both genesisp and launch have only I frames, they show equal amount of jitter, but cartoon has more I frames than maar_peet, hence it shows more jitter. This is just a possible hypothesis and we do not have any concrete result to prove it. A lot of work needs to be done on this issue to crack down the real variables involved in these different MPEG files. The following graph shows the difference in performance for the different MPEG files :



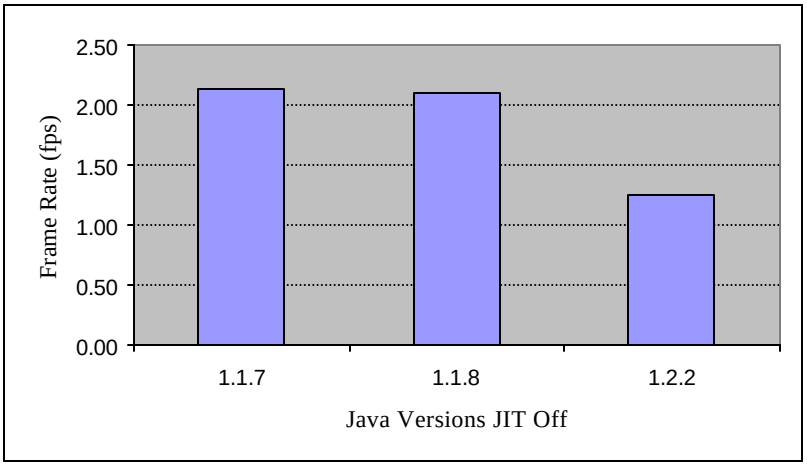
Test II : Performance difference between different MPEG files

4.3 Versions of Java

Our third set of tests was to try different versions of Java. We used JDK 1.1.7, 1.1.8 and 1.2.2. As expected the newer version of JDK i.e. 1.2.2 shows improvement in the performance. The older versions are supposed to have JIT on by default, but JDK 1.2.2 does not contain any built in JIT. So we tried JIT by Borland (JBuilder 3.5) with JDK1.2.2. We could not observe any difference with the Borland's version of JIT. Hence we concluded that Borland's JIT was not working for the Linux version. Then we tried SHU JIT by Shudo Kazuyuki [http://www.shudo.net/jit]. This showed a significant improvement in the performance. The performance difference in different versions of Java can be seen in the following histogram:

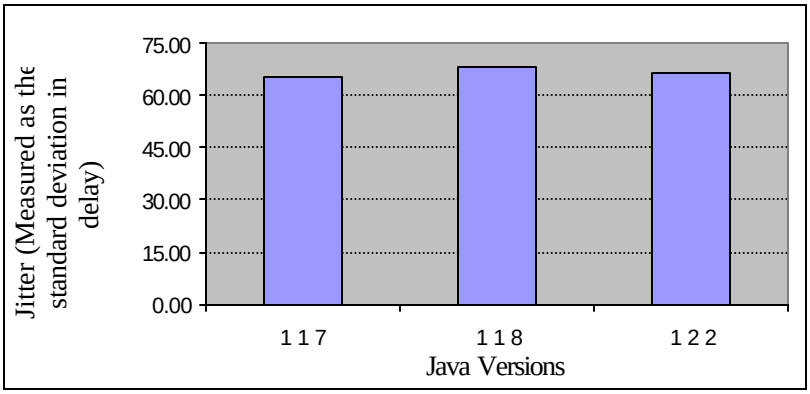


Test III : Different versions of Java with JIT on. The newer versions seems to be more promising



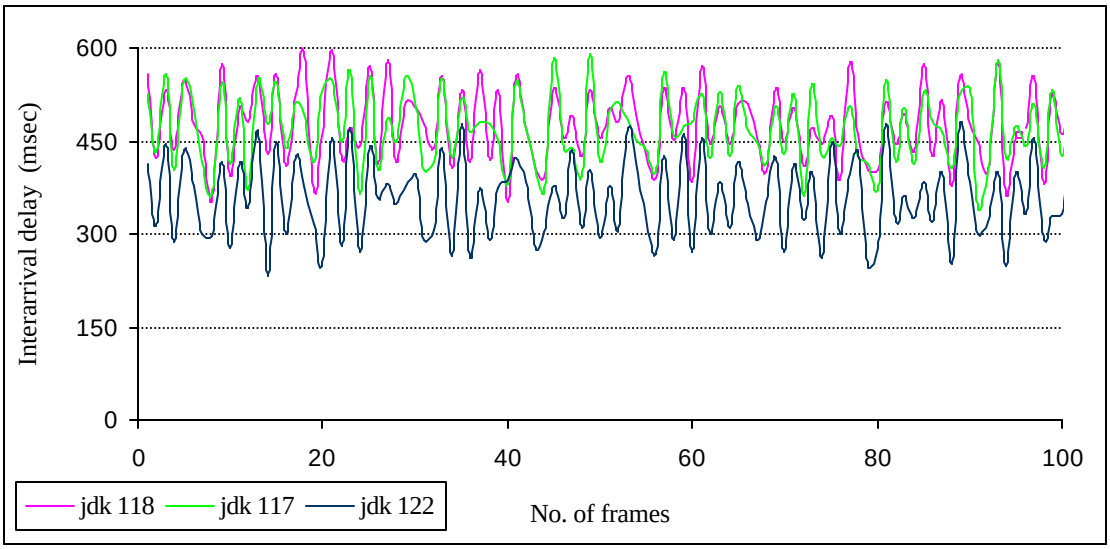
Test III : Different versions of Java with JIT off. It seems that the JIT is still on in the older versions.

The jitter seems to be the same for all of them. Though when we switched off JIT, jitter was slightly more for JDK 1.2.2. But, with JIT on, jitter is on comparable scale for all three of them.



Test III : Different Versions of Java with JIT on : The Jitter seems to be almost the same for all of them

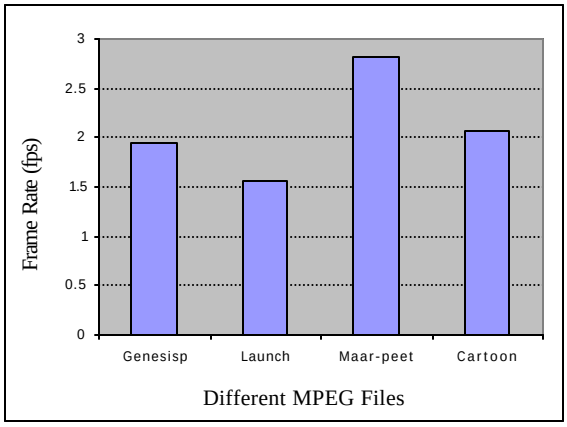
The newer version of Java, JDK 1.2.2, wins over the other two with a better frame rate and same amount of jitter. All our future work will be done using the newer version of Java JDK1.2.2, though we may try different JITs with it. We also may try to find out the reason why did the older versions perform similarly, though we explicitly switched the JIT on and off. The following graphs show the output we obtained from this set of tests:

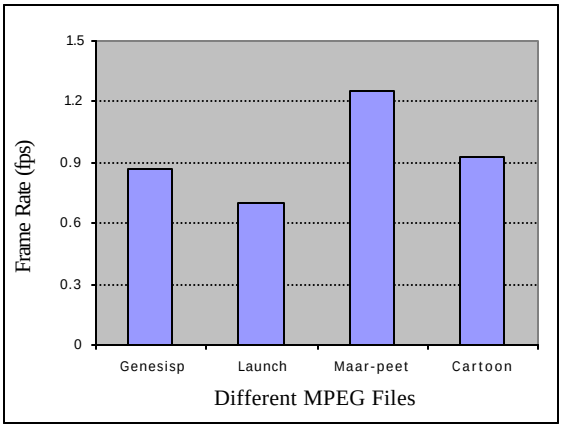


Test III : Performance for different versions of Java using the maar_peet.mpg file with JIT enabled.

4.4 JIT Compilation

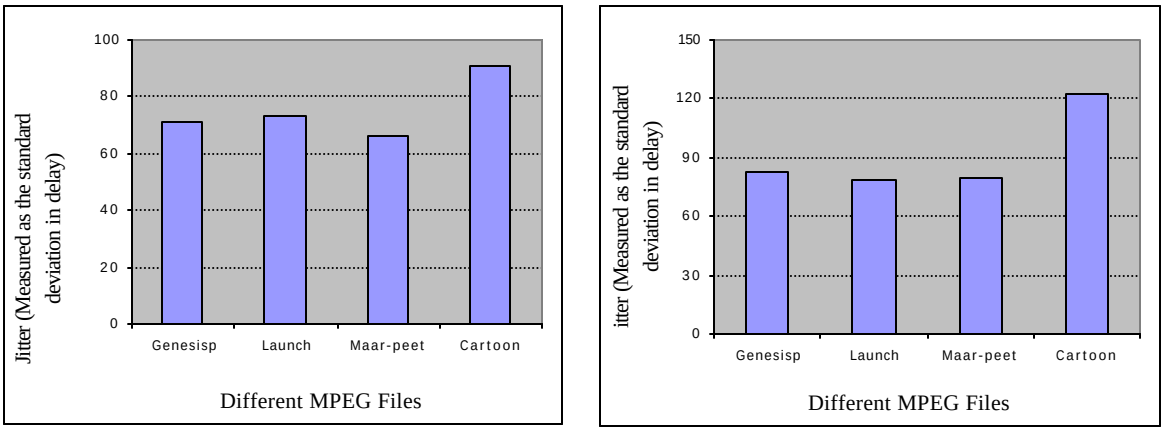
Our final parameter, JIT, was a part of all the three previous set of tests. All the tests discussed above were performed with JIT on as well as off, using JDK 1.2.2. Each showed a significant improvement in performance by the same scale. We shall site the different MPEG file test. For each MPEG file, the frame rate almost doubled when JIT was turned on, as seen from the following histograms. The left side histograms have JIT on and the right side has JIT off.





Test IV : Toggling the JIT on and off : The performance almost double with JIT on (left side)

JIT also affected the amount of jitter for the different MPEG files as seen in the following histograms. The left side graph is with JIT on and the one on the right side is with JIT off. Clearly, the jitter is less with JIT turned on.



Test IV : Toggling the JIT on and off : The jitter reduces with JIT on (left side)

The pattern remains almost the same over the different MPEG files in each case. Hence, the effect of JIT on jitter is quite consistent. In our future tests, we shall keep the JIT on, since we want minimum jitter and maximum frame rate.

5. Conclusions

The local access of the media verses access over the network shows 5% improvement for the frame rate and 25% less jitter. But there is no noticeable difference between the various network setups and neither could we observe any pattern from the number of tests we performed. It could be possible that we may need to perform these tests over a large number of times to observe some pattern, if at all such a pattern exists.

Our second set of tests shows that the type of MPEG file effects the frame rate as well as the jitter. The pattern is unclear as of now. Lot of work needs to be done on this issue. One possibility is that the type of frames in the file matter. For instance, the cartoon had equal number of I and P frames, whereas the action file (maar_peet.mpg) had double the number of P frames as compared to the number of I frames. The cartoon had the maximum jitter where as the action file has the lowest jitter. On the other hand, the files launch.mpg and genesisp.mpg have only I type frames, but they perform very differently as far as the frame rate is concerned. Using a C compiled player, launch.mpg shows a better frame rate, whereas with our Java player, genesisp.mpg shows a better frame rate. Both the files had similar amount of jitter. So, though we cannot arrive at any definite conclusion at this point, we can conclude that the type of file does matter.

Our next set of tests for different versions of Java shows that the newer version of JDK 1.2.2 proves to be promising. The frame rate is better (with JIT on) using the newer version but the jitter is slightly higher with JIT turned off. The previous versions do not show any change in performance with JIT on or off. This suggests that either JIT was always on or it was always off. Looking at the numbers, we suspect that it was always on. Our final set of tests of toggling JIT on

and off shows a significant difference. There is 100% improvement in the frame rate and 25% decrease in jitter with JIT on. The results were consistent for all different type of network tests and different MPEG files too. We can safely conclude that JIT significantly improves the performance of our Java MPEG player with a better frame rate and lower jitter.

Finally, the drastic improvement in the performance in the 5th phase of test I, after the insignificant change in the hardware, have led to the thought that there are still some unexplored variables involved. At this point of time it is difficult to conclude what can we derive from it, but we can surely say that a lot more work still needs to be done in this area.

6. Future Work

We have tested some of the possible variables to evaluate Java's performance. There are many more unexplored areas. Firstly there should be a code evaluation for the client and the server, which is necessary for certainty in our results. Then, we would like to try different network protocols such as using UDP, RMI or CORBA instead of TCP, as to determine if TCP is responsible for the jitter observed and if other protocols could help to reduce the jitter and improve performance. We would also like to try our experiments for different network buffer size for the client. Other variables can be finding any relationship between frame type and jitter. It could be possible that jitter is more for a certain frame type. The compression rate of the MPEG file should also be taken into consideration, because the genesisp.mpg file and launch.mpg file, in which the only difference is the compression rate and the number of frames, perform quite differently. Trying different JITs (such as JBuilder, TYA etc.) is another possible variable, though it should not vary the results much. We believe that our current work, along with the previous tests and these future proposed tests will help to understand and evaluate the performance of Java for streaming multimedia and hence will unfold various ways to improve its performance.

References

[1] Mark Claypool, Tom Coates, Shawn Hooley, Eric Shea and Chris Spellacy. Video Performace in Java. IRMA 2000 Multimedia Computing (MMC) Track. May 21-24, 2000

[2] Mark Claypool, Jonathan Tanner. The effects of jitter on the perpetual quality of video. *ACM Multimedia Conference*, Volume 2, Orlando, FL, October 30 - November 5, 1999.

[3] Tom R. Halfhill. How to soup up Java Part I. Byte Magazine. May 1998.

[4] Various resources on Java and JIT from the Internet, such as [http://java.sun.com] & [http://www.shudo.net/jit]