Team 4
Experimental Evaluation

18-749: Fault-Tolerant Distributed Systems
Priya Narasimhan
Spring 2006

Jonathan Gray

Megan Hyland

Mike Mishkin

Bryan Murawski

Prameet Shah

Joe Trapasso

Experimental Evaluation Data:

Data for the project was obtained by creating Perl scripts to gather probe data for each of the 48 experimental situations and writing the collected data to labeled output files. The experiments tested system response for varying numbers of clients (1, 4, 7, 10), varying request rates (0, 20, 40 milliseconds), and varying reply sizes (original size, 256, 512, 1024 bytes). A general description of our application is a distributed auction system that allows users to buy and sell items in an auction plaza. Clients have the ability to create accounts, view account data, post new auctions, post bids, view bid history, and check on the status of an auction. The two different invocations that were used in these scenarios were AuctionPost and AuctionView, described briefly below.

No.	Method Name	Input Parameters	Exception	Return Value	Description	Info.
1	Auction Post	Auction aucObj	▪ Invalid auctionInfo ▪ User Not Logged In	Void	This method allows a user to post auctions with specific information about the product.	Two way, Database Access, Request size = 114 bytes, Reply size = 102 bytes
2	AuctionView	String AuctionID	▪ Invalid Auction	Auction aucObj	This method allows a user to retrieve a specific auction with a given AuctionID string	Two way, Database Access, Request size = 4 bytes, Reply size = 94 bytes

The first of these invocations was invoked 5,000 consecutive times, and then the scripts switched to the second function for the final 5,000 invocations in each experiment. The next step involved creating scripts to sort the gathered data in Matlab, as specified in lecture, and then creating graphs to analyze the system performance, based on several factors:

$$\text{Latency}(i) = P_2(i) - P_1(i)$$

$$\text{Server}(i) = P_5(i) - P_4(i)$$

$$\text{Middleware} = \text{Latency}(i) - \text{Server}(i)$$

$$\text{Req\_Rate}(i) = \frac{10^6}{\left(P_4(i) - P_4(i-1)\right)}$$

$$\text{Throughput} = \frac{10^6}{\left(P_4(i) - P_4(i-1)\right)} \cdot \text{Reply Size}$$

With these calculations, we used Matlab to develop plots of the data, which can be found and analyzed in this document.

Experimental Evaluation Analysis:

Below is a detailed explanation of the analysis performed on the data collected from the experiments.

1. Line Plots of Latency for increasing numbers of clients and different reply sizes

The first thing discovered while plotting latency for the clients was that there was a noticeable difference in latency on each individual server's data. This is due to the fact that when we ran the experiments, we utilized 2 different functions: AuctionPost and AuctionView. AuctionPost takes much longer as a transaction due to its increased interaction with the database, adding new information to the database, whereas AuctionView simply returns the requested data. Therefore, the first 5000 invocations in each of our datasets tend to have a noticeably larger latency, as seen below in Figure 1.

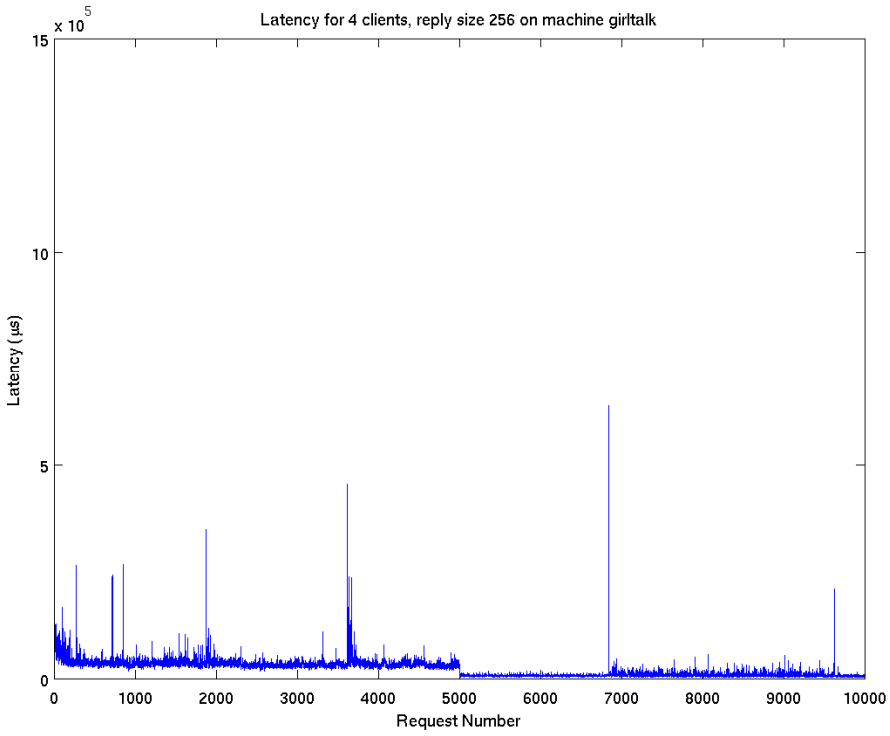


Figure 1: Latency for 4 cli, 256 bytes, girltalk

Also, we found that when the experiments were run the first time with 10 clients and a reply size different than our original size, we saw a dramatic increase in latency, as shown in the Figure 2 below. We have chosen to show the results for latency for 10 clients on the Chess server, but similar results occurred for all of the 10 client cases with larger than normal reply size. This graph represents requests with a reply size of 256, showing the problem described. The most likely reason for this behavior is that the allocation of additional memory for the larger than normal reply sizes slowed down the server with 10 simultaneous clients. Since there were so many clients allocating additional memory and then releasing it, the garbage collector most likely began to run more frequently, contributing to the higher latency. This effect may have been amplified towards the end as server logs were also being stored on the heap. Java likely ran out of heap space and was forced to run the garbage collector even more frequently.

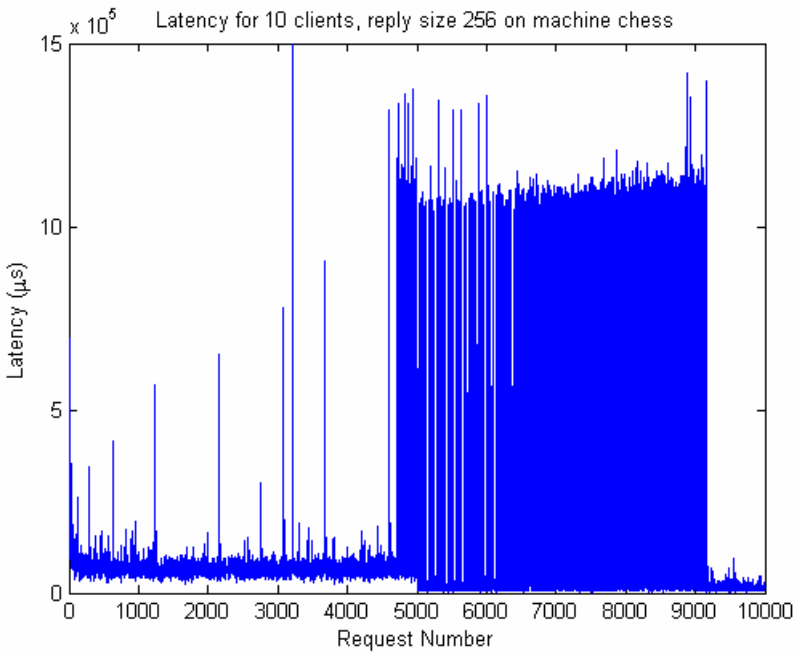


Figure 2: Latency for 10 cli, 256 bytes, chess

However, after repeating the experiments with an increased heap size of 300 MB, the total number of spikes in latency was greatly reduced, and the overall data acted closer to expected results. Due to these variations in latency and insights into our application and evaluation behaviors, we decided to include each of our 88 latency graphs to better show trends in our data. These graphs are differentiated by number of clients (1, 4, 7, 10) and request size (original, 256, 512, 1024) and then individual graphs were made for each client. They show more expected and predictable behavior for the course of the experiments, and the remaining spikes can be explained by the high load on the servers. These graphs can be found in Appendix A at the end of this document.

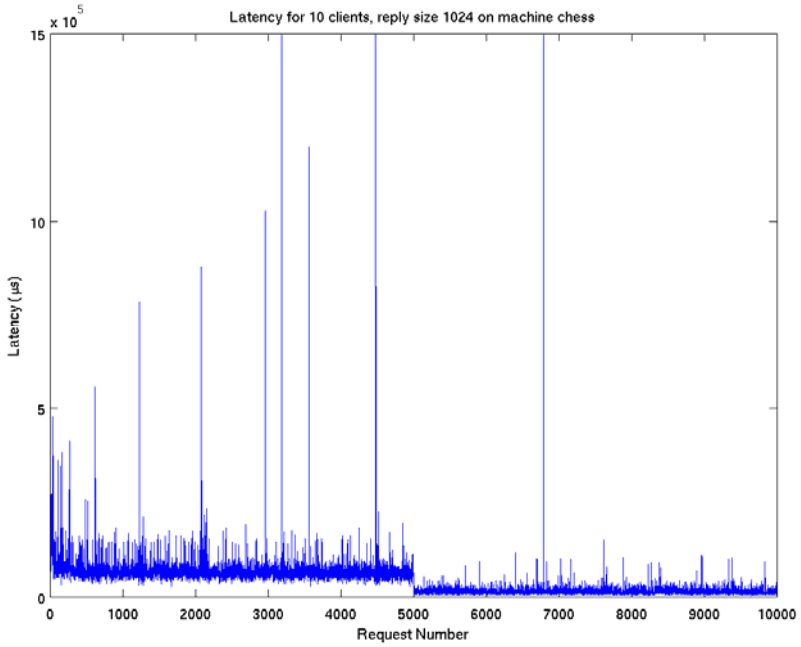


Figure 3: Latency for 10 cli, 1024 bytes, chess

We also combined data collected across all servers to graph max, mean, and 99% latency for each reply size, given a certain number of clients. Below are the lines graphs for max (Figure 4), 99% (Figure 5), and mean (Figure 6) latency for all aggregated data:

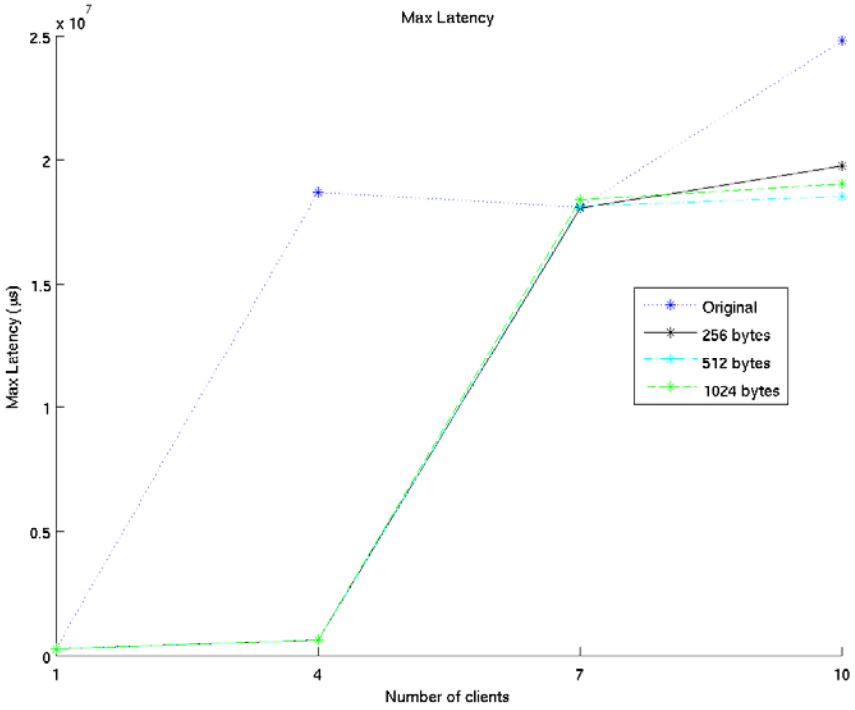


Figure 4: Max Latency

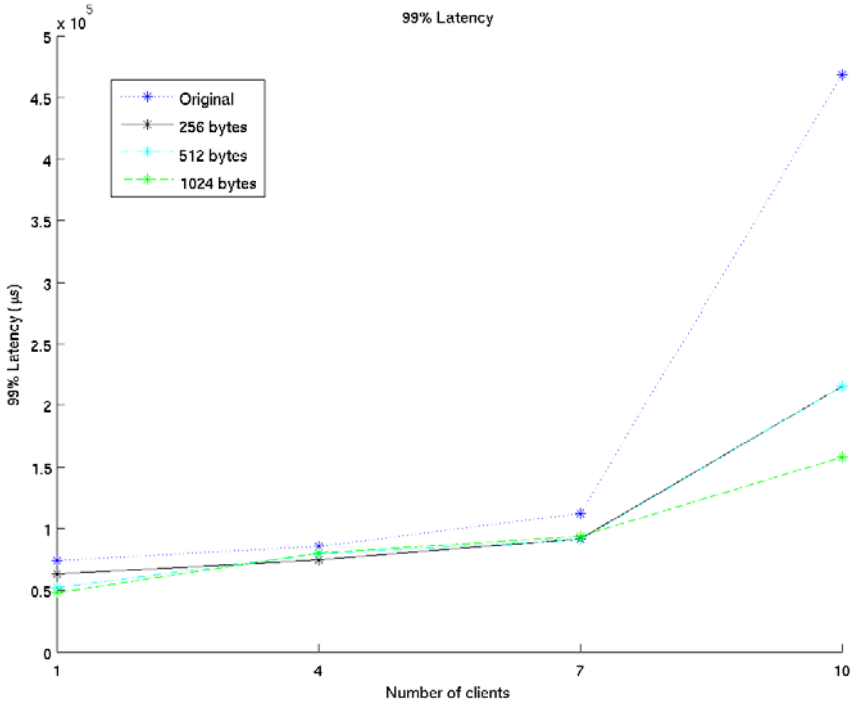


Figure 5: 99% Latency

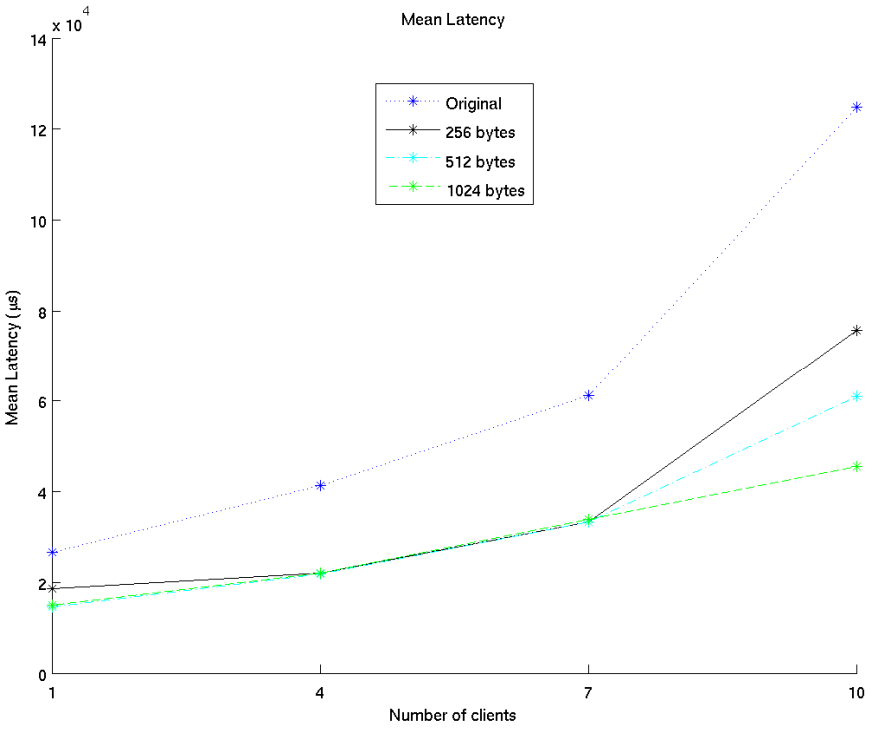


Figure 6: Mean Latency

It is clear from the graphs that after removing the top 1% of values, the 99% latency follows the mean latency much more closely. Both the 99% and the mean show observable upward trends in latency as number of clients increases for every reply size. One interesting trend we observed in the mean latency was that the original request size showed higher latencies in all cases than the other request sizes, shown in Figure 6. We believe that this is due to a Java optimization to our remote procedure call code causing less memory to be allocated and decreasing overall latency for the invocation when there is a non-standard reply size. We were unable to test this theory, however, due to the fact that the MySQL server was at 99% load for several days, and then failed to respond.

2. Area Plots (mean, max) latency and (mean, 99%) latency

The next analysis of data was done by taking the latency, as calculated in Part 1, and plotting both the mean and max values (sorted by increasing mean values) and then performing the same plot using mean and 99% values. The data was sorted capturing mean, maximum and 99th percentile data for each of the 48 different experiments. The graphs below show the maximum and mean latencies plotted together and also zoomed in due to the over 2 orders of magnitude difference in scale. Figure 8 shows the normal view for the mean and max latencies. Figure 7 is zoomed in by 2 orders of magnitude to show more closely the upward trend of latency across the experiments.

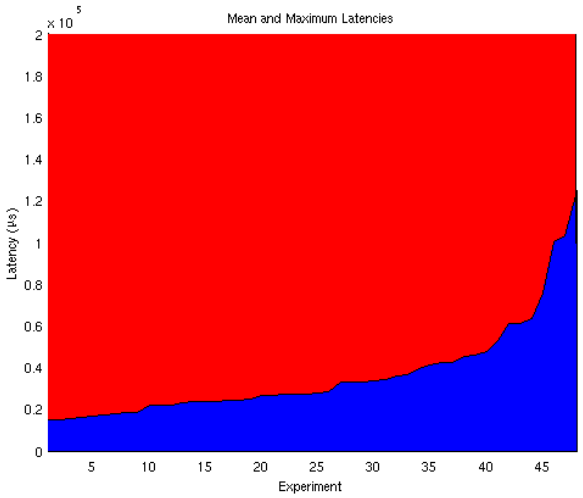


Figure 7: Mean/Max Latency Area Plots Zoomed

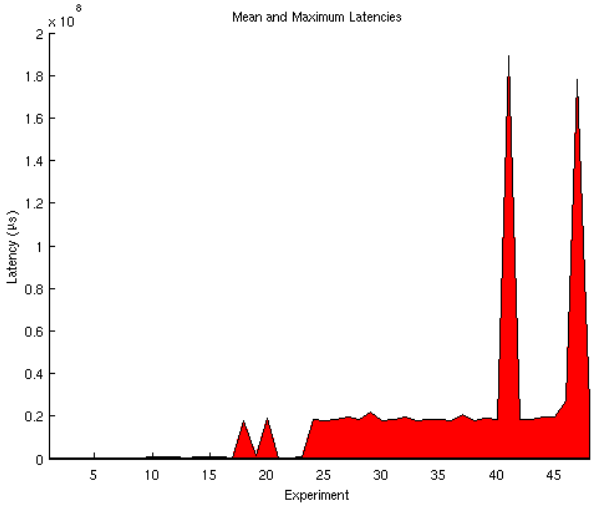


Figure 8: Mean/Max Latency Area Plots, Standard View

We believe the large spikes shown in Figure 8 come from the indeterminism in the server and the fact that there are random periods of high latency. They are much larger and more common towards the end because when we reran the tests with a larger heap many of our classmates were putting a huge load on the server, which significantly decreased our performance under 10 clients.

Figure 9 is a graph of the data comparing the mean values and the 99%. The plot shows that the magic 1% (in terms of the first half of the experiments) does indeed have a great effect on data, as the 99% nearly matches the mean values for the experiments. The difference between the mean and 99% latencies, however, show that there was greater than 1% spikes in the experimentation, most likely due to high server load.

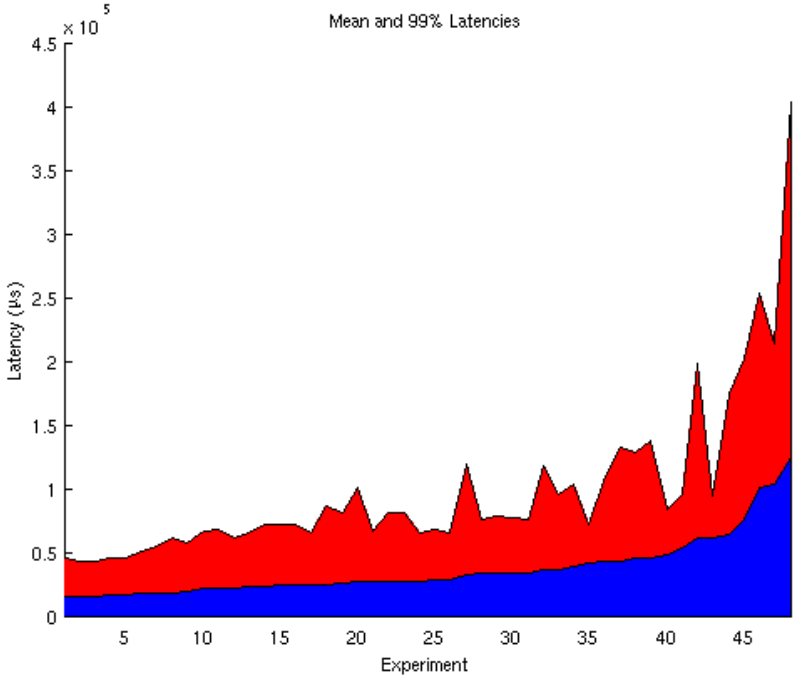


Figure 9: Mean/99% Latency

3. Bar Graph of latency component break down and normal requests

The third step in data analysis involved quantifying the outliers for the server and the middleware. Some of the most interesting data is contained in the figures below. Figure 10 shows the typical outliers for varying 1 client experiments. It is clear that middleware time is very consistent across different reply sizes, but server time is much greater in the original return size. This further supports our argument that in the non-standard return cases, java is doing some type of optimization to our code because we are not returning the data calculated in the method.

Figure 11 depicts the wildly different outliers we found once we increased past the single client experiments. With 4, 7, and 10 clients, we found that some outliers were caused by high server latency while others were from high middleware latency. As clients increase, both network traffic and server load increase. With this comes a greater likelihood of Ethernet collisions for both client to server (part of the middleware latency) as well as server to database (part of the server latency).

Lastly, Figure 12 gives our average outliers across different numbers of client, with all other experimental variables fixed. The average outlier greatly increases in total latency time as clients increase, as expected, and the single client case is far below all other cases. Average latency time is split somewhat evenly between server and middleware, so it is difficult to draw any conclusions about the real culprit of these extremely high latencies. More testing is necessary to determine the effects of the server to database latency and it's affect on total server latency. This could be done with two additional probes recording times into and out of the database; these bar graphs could then be split three ways into server, middleware, and database time. However to get useful data, the database server would need to be far less utilized than it was during our experiments due to other teams' experiments being run concurrently.

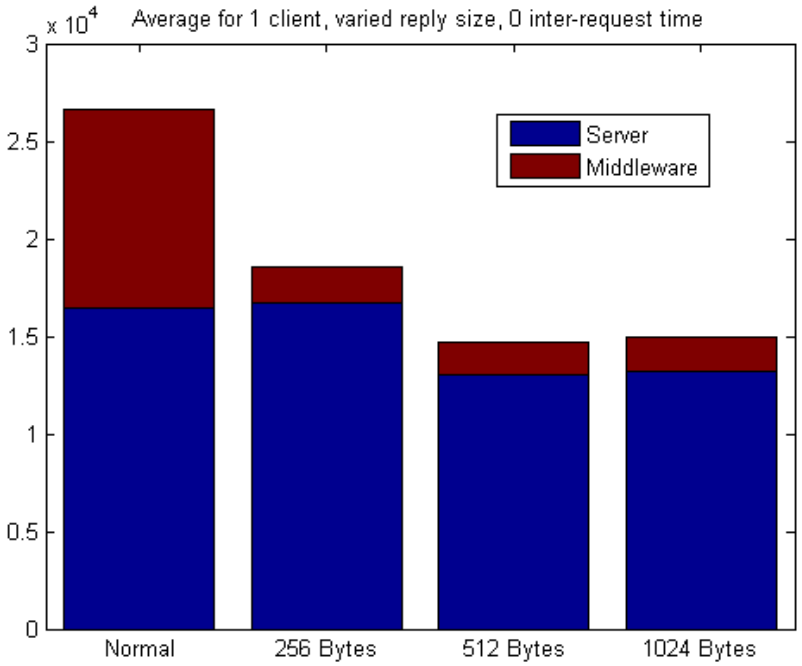


Figure 10: Server/Middleware Bar graph for 1 client/0 request time

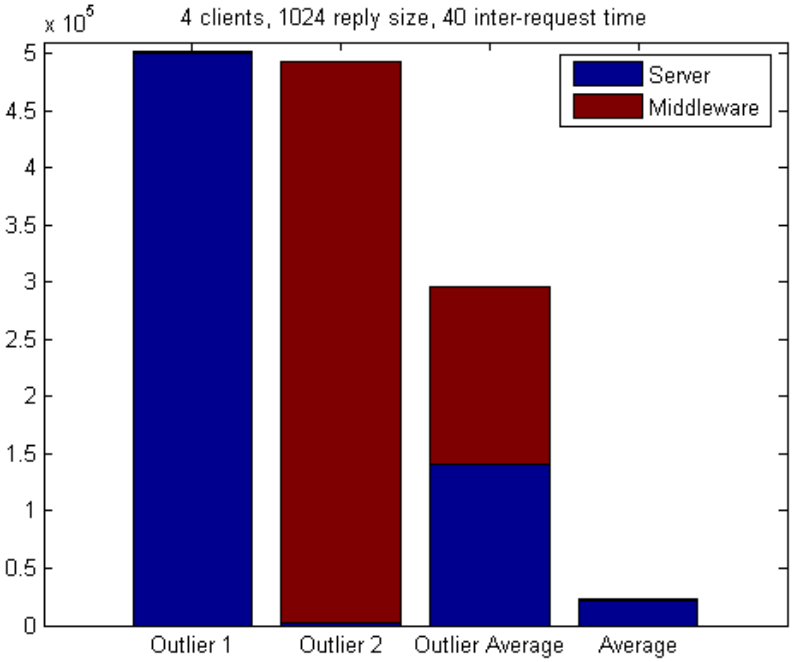


Figure 11: Server/Middleware Bar graph for 4 clients/40 request time/1024 reply size

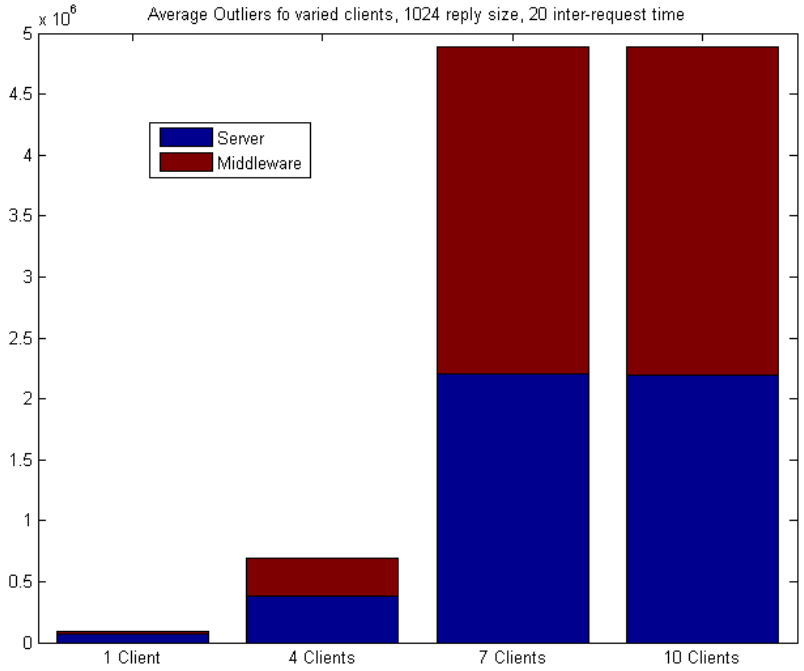


Figure 12: Server/Middleware Bar graph for 20 request time/1024 reply size

4. 3D Scatter plots of reply size by request rate by max and 99% latency

Another analysis of the data was performed by graphing the maximum and 99th percentile of latency based on reply size and request rate of the invocation, as seen below.

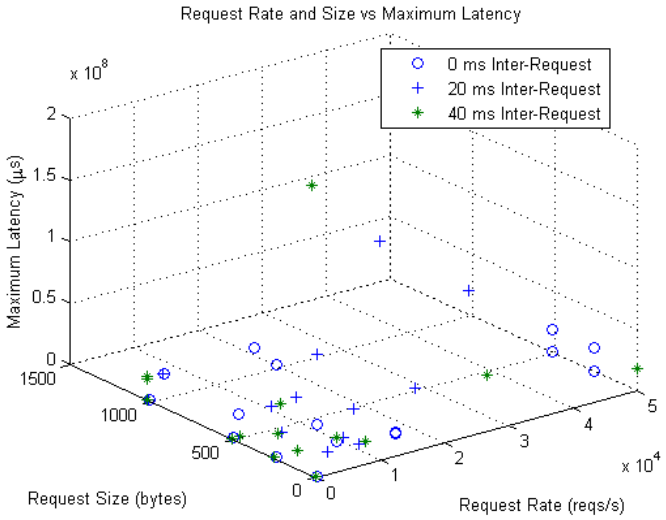


Figure 13: Request Rate vs. Size vs. Max Latency

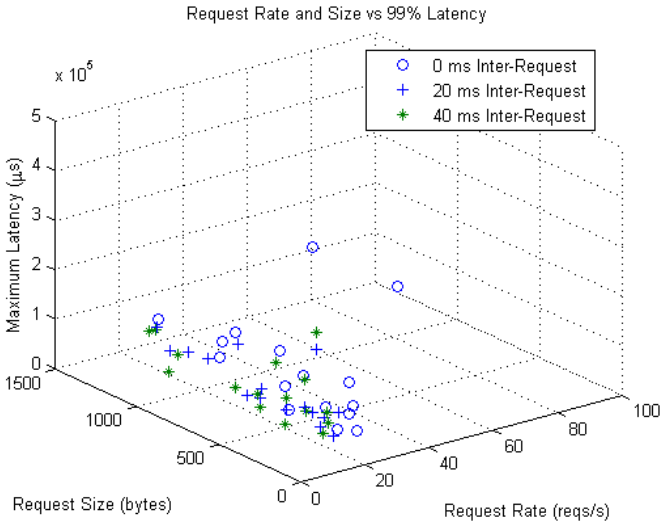


Figure 14: Request Rate vs. Size vs. 99% Latency

By plotting these figures, we were hoping to notice some trends between the Request Arrival Rate and the Maximum Latency for various request sizes. In general, we noticed scattered results for the Maximum case, and a cluster of recorded values once we removed the highest 1%, showing interdependence between request arrival and latency. Ideally here, we would notice increasing latency with increasing request rate. However, we believe that because each client can only handle one request at a time, the data did not follow such a trend. Infinite (or at least a much larger number of clients) may have changed the results. Increasing latency seems to correlate with decreasing request rates. With only 1-10 clients, there is a strong link between and limit on the values of request rate and latency. If latency increases, then clients who must send requests sequentially are forced to wait for longer responses. This will limit their ability to invoke more requests and thus the overall request rate is limited.

Below is a 2-dimensional graph showing the recorded data of request rate as compared with 99% latency. This plot shows the how latency and request rate are dependent on each other and are therefore limited to be in that cluster shown as described above.

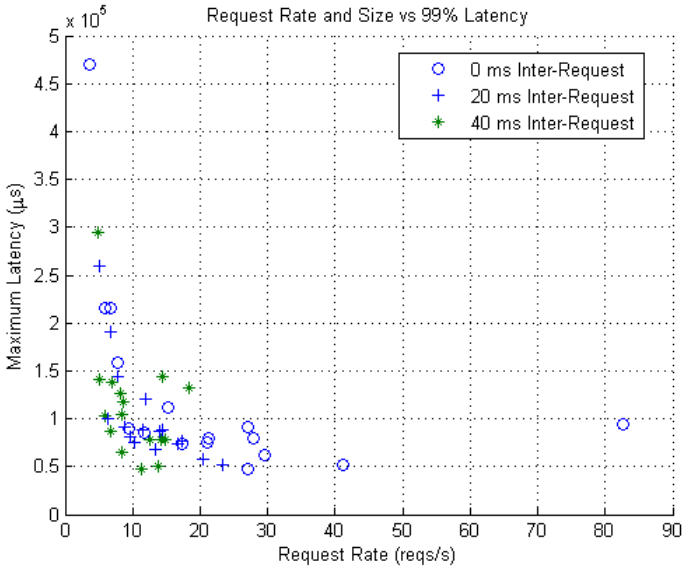


Figure 15: Request Rate vs. Latency

5. Latency vs. Throughput

The final comparison was made to consider the relationship between latency on the client side, timed from the request until the return of the invocation, to the throughput of the overall replies returned from the server, as shown in Figure 16.

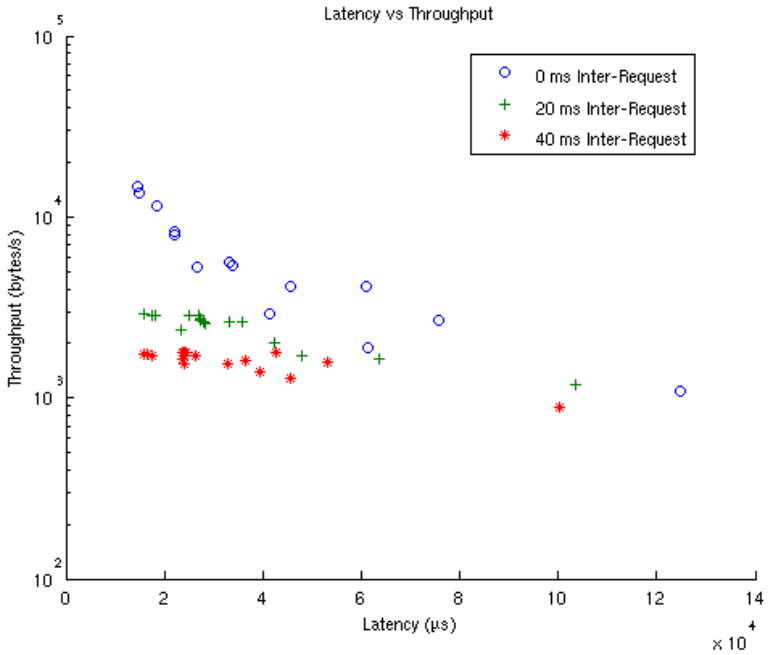
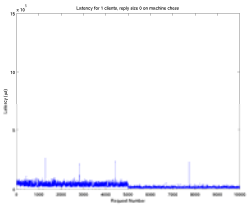
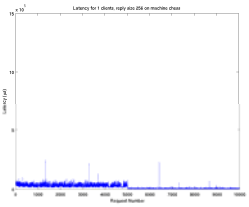
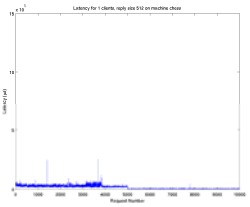
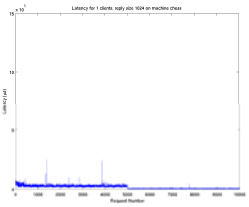


Figure 16: Latency vs. Throughput Scatter plot

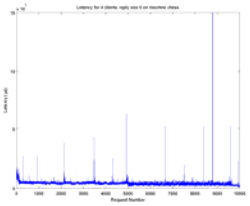
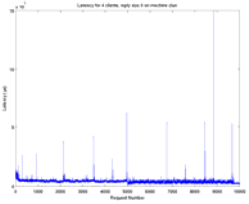
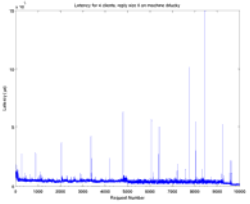
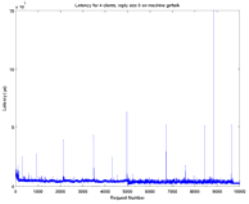
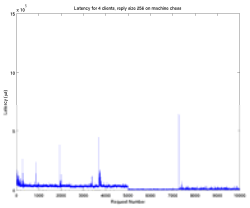
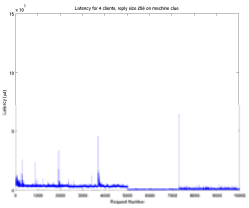
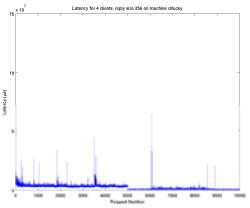
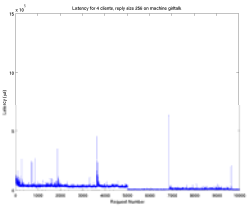
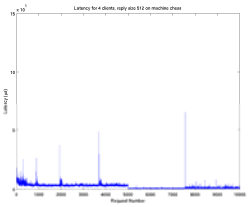
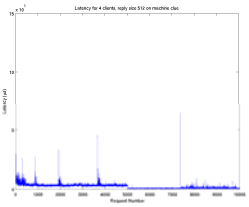
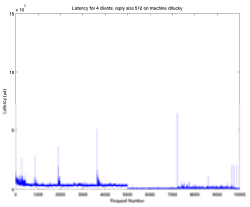
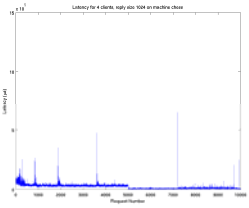
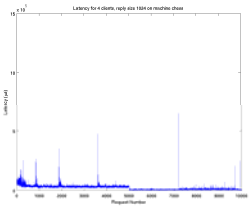
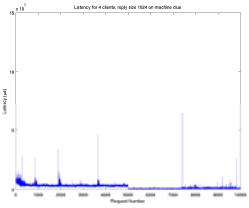
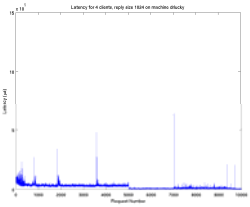
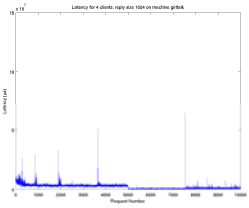
The plot is shown with a logarithmic scale, indicating a downward trend in throughput as latency increases. This makes sense with our predictions in that less data, in terms of bytes, is being processed when the observed latency is increasing. This trend is best observed above in the case of 0 inter-request time.

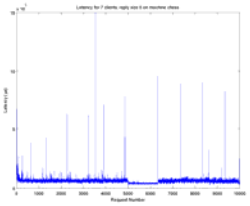
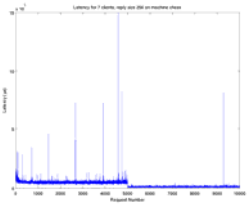
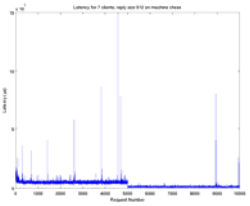
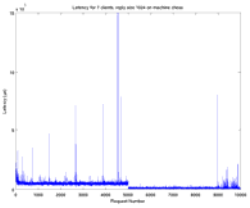
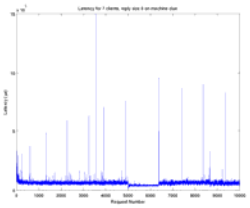
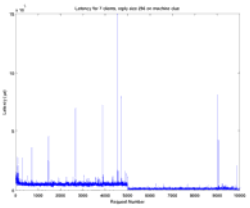
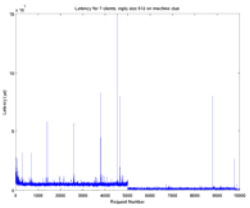
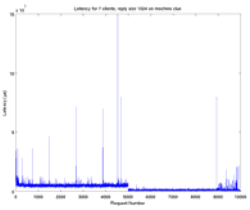
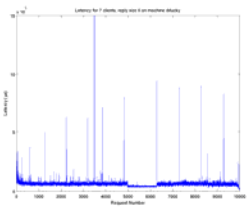
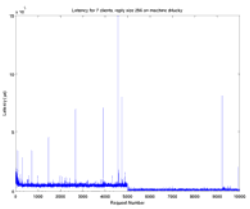
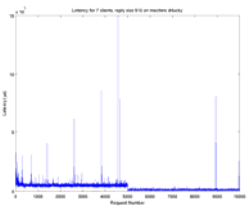
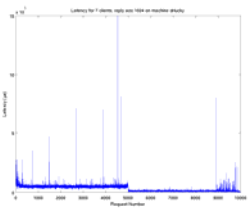
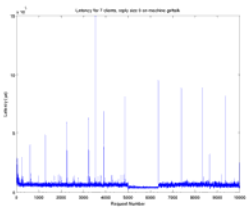
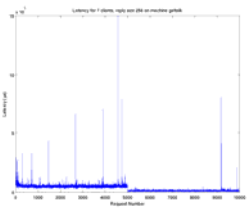
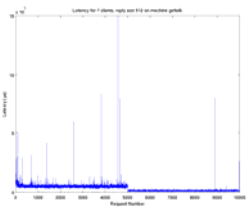
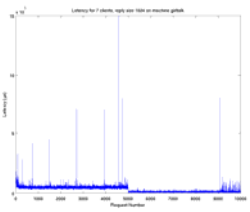
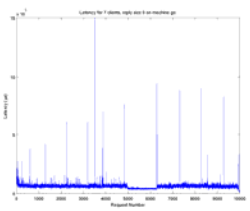
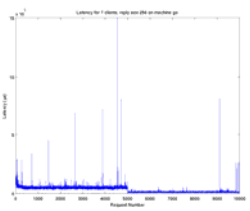
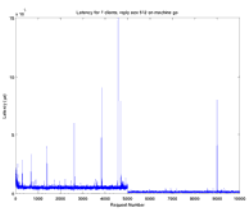
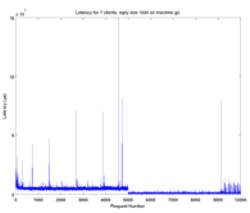
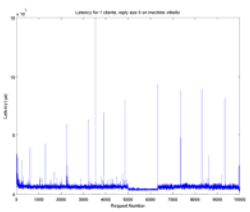
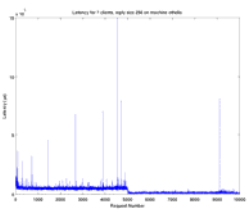
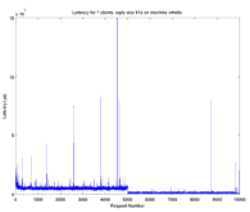
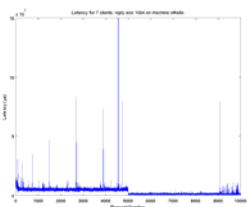
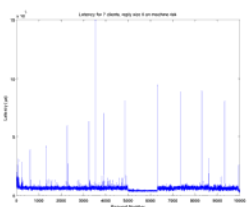
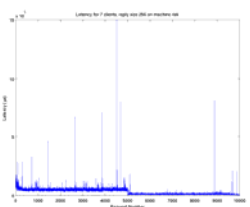
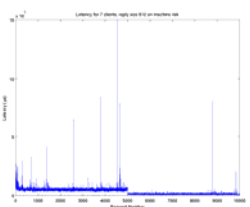
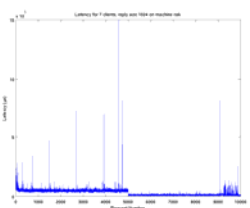
Appendix A

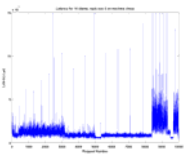
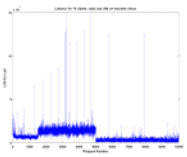
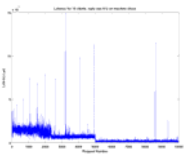
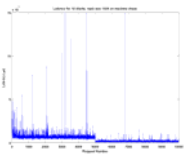
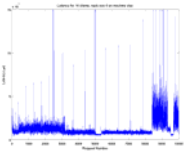
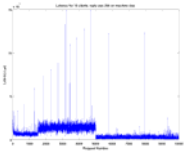
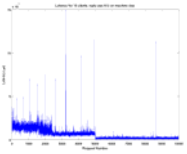
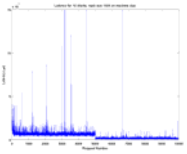
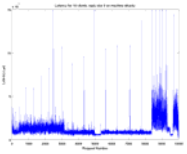
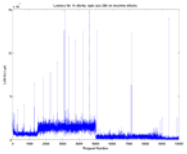
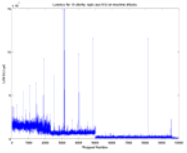
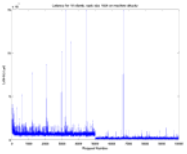
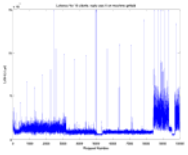
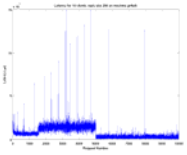
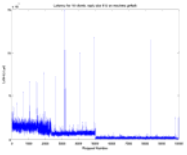
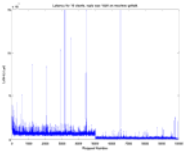
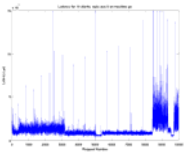
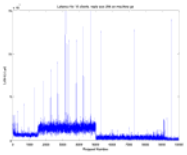
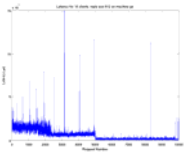
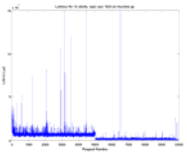
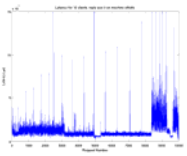
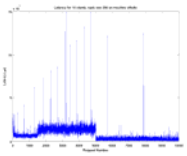
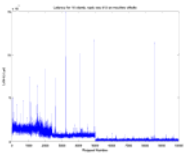
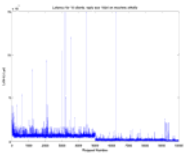
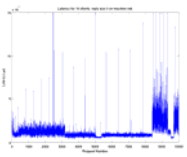
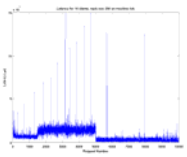
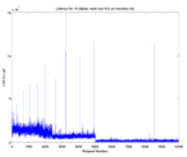
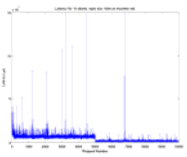
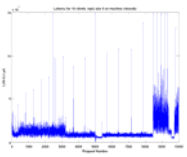
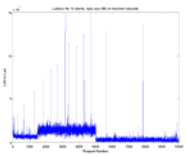
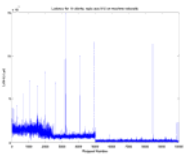
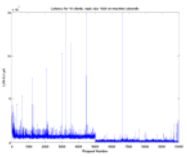
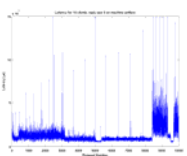
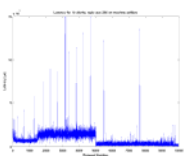
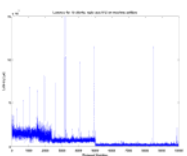
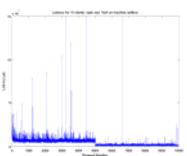
1 Client

	Original Reply	256 Byte Reply	512 Byte Reply	1024 Byte Reply
Client 1/1				

4 Clients

	Original Reply	256 Byte Reply	512 Byte Reply	1024 Byte Reply
Client 1/4				
Client 2/4				
Client 3/4				
Client 4/4				

7 Clients	Original Reply	256 Byte Reply	512 Byte Reply	1024 Byte Reply
Client 1/7				
Client 2/7				
Client 3/7				
Client 4/7				
Client 5/7				
Client 6/7				
Client 7/7				

10 Clients	Original Reply	256 Byte Reply	512 Byte Reply	1024 Byte Reply
Client 1/10				
Client 2/10				
Client 3/10				
Client 4/10				
Client 5/10				
Client 6/10				
Client 7/10				
Client 8/10				
Client 9/10				
Client 10/10