

FTEX Real-Time Experimental Evaluation Report

18-749 Fault-Tolerant Distributed Systems

Team 3

By Ernest Chan
Yong Khong Chong
Han Chun Lim
Ahmad Imran Sulaiman
Yik Shing (Arnold) Yip (Chief Experimenter)

Experiment parameters

Replication Style: Passive

Number of clients: 1, 4, 7, 10

Number of invocated methods: 4

Methods invocated: putOrder(), getAccount(), login(), getTransactions()

Number of invocations per method: 2500

Total invocations per client per experiment: 1000 (2500 invocations X 4 methods)

Inter-request times: 0ms, 20ms, 40ms

Reply sizes: 16 bytes (Original), 256 bytes, 512 bytes, 1024 bytes

Description of Client Invocations

- putOrder()
 - Places an order to trade a particular stock
 - Searches the database for other compatible orders
 - Execute orders if matching orders founds
 - Inserts new entry into database a new order
 - Database Access: Find min/max by price, Insert new Entry

- getAccount()
 - Finds and returns an account with the same username argument
 - Database Access: Find by primary key

- login()
 - Finds an account with username and determines if password in account is the same as password inserted
 - Database Access: Find by primary key

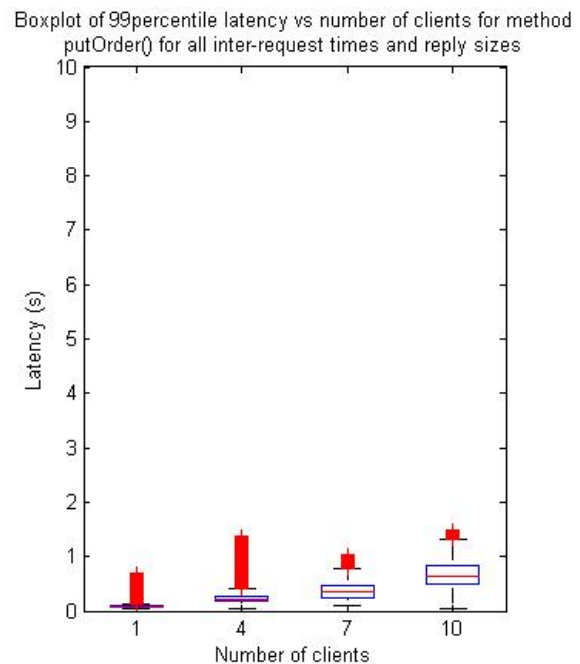
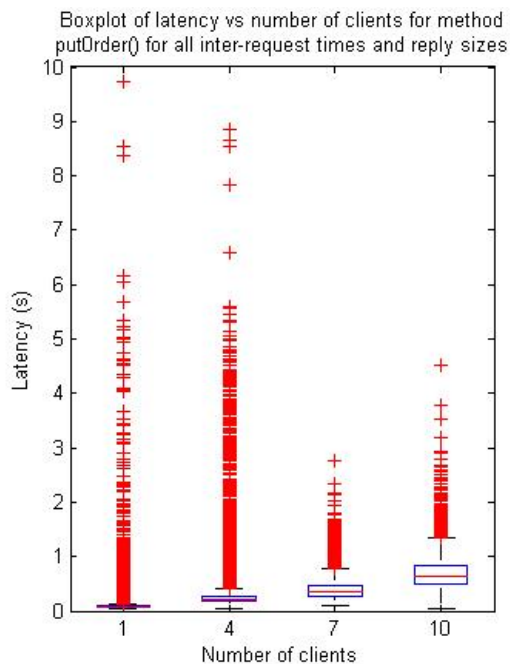
- getTransactions()
 - Given a username, return all transactions executed by the username
 - Database Access: Find from table all with matching username

List of Client Invocations

Method	One-Way	DB_ACCESS	SZ_REQUEST	SZ_REPLY
putOrder()	No	Yes	Variable	16
getAccount()	No	Yes	Variable (user_name)	Variable
login()	No	Yes	Variable (user_name + password)	16
getTransaction()	No	Yes	Variable (user_name)	Variable

Database tables

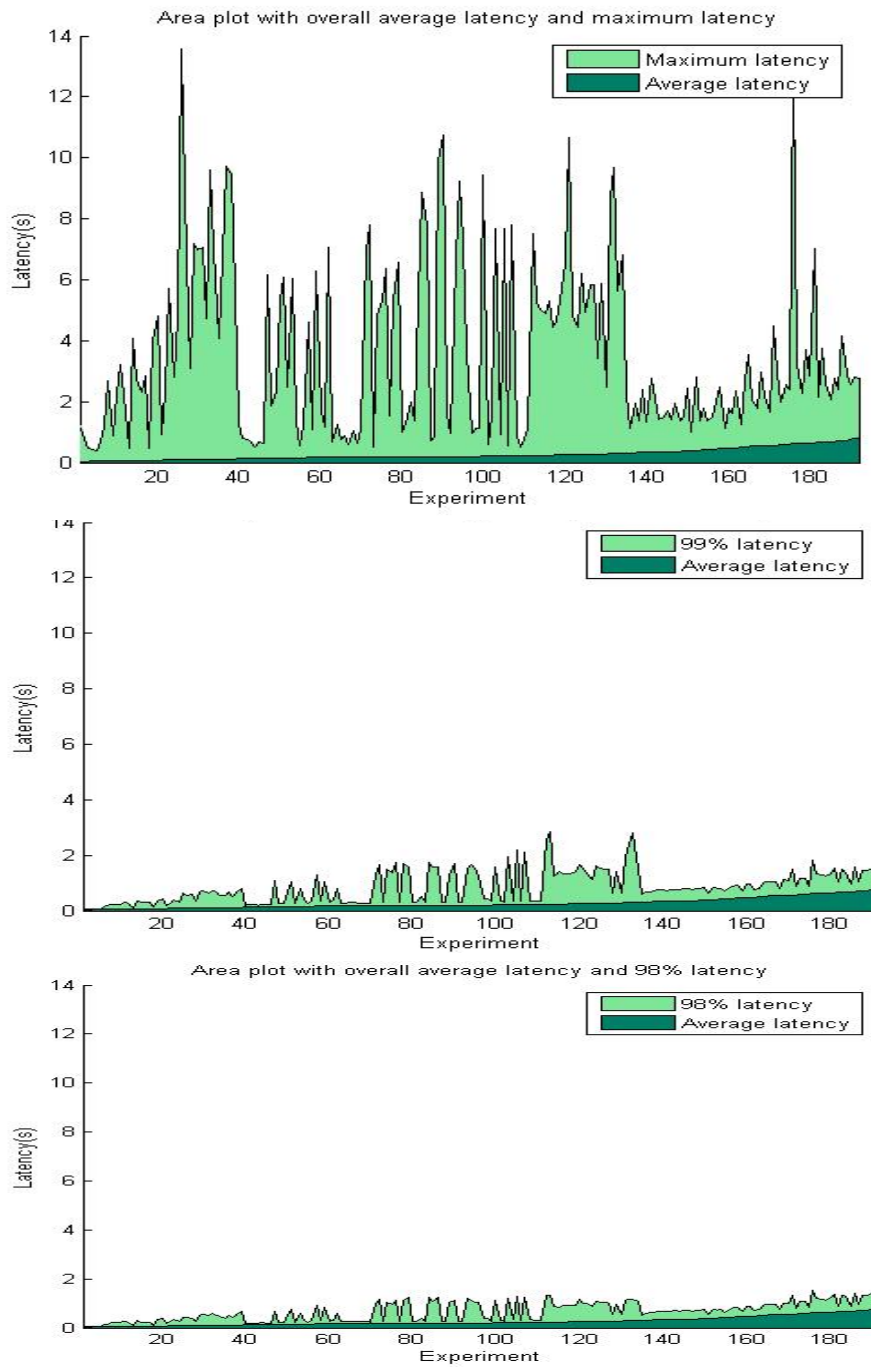
- OrderEJB
 - Contains details of an order, including stock name, quantity, buy/sell, and price
 - Used by putOrder() that finds items and adds items
 - Increases in size as trials that invoke putOrder are run()
- Account
 - Contains details of a user including username, password, last name and first name
 - Used by login() and getAccount() methods that only find and read items
 - No new entries are added during experiments
- TransactionEJB
 - Contains orders that have been completed, including buyer, seller, price, stock symbol and quantity
 - Used by getTransactions() that finds all items with a particular username
 - No new entries are added during experiments.

Number of Clients Effects

As the number of clients increases, it can be seen that the mean latency increases. Also more interestingly the standard deviations also increases as the number of clients increase. There quite a number of outliers (red-crosses) and a replot was done after removing 1% of the furthest outliers. The max latency decreases drastically as 1% of the outliers are removed. The box-plot above a plot for the invocation putOrder() with all inter-request times and reply sizes with the box-plot on the left for all data and 99 percentile of the data. Box-plots for other methods and line-plots of mean latency different inter-request times and replies are similar and concur with these hypotheses and are included in the appendix.

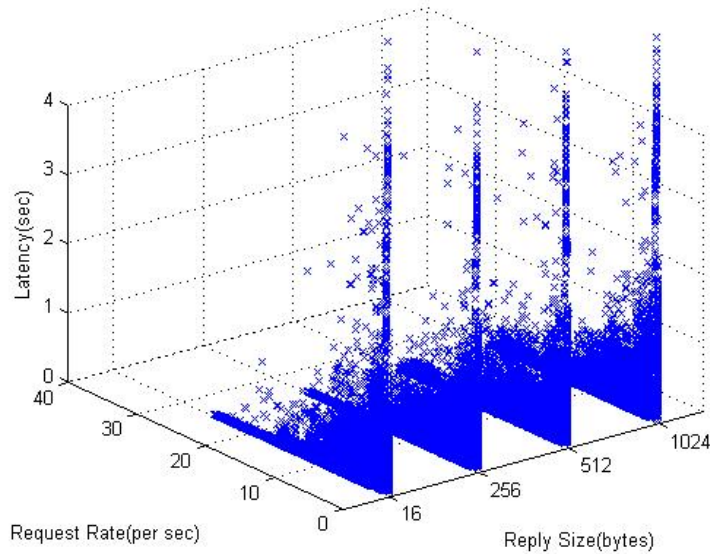
The Magical 1%

The following charts are used to study the “magical 1% theory”. The following graphs show mean, 99% and 98% latencies of all invocation methods for all 48 configurations. From these plots, we can see that a by eliminating just 1% of the largest measured latencies, most of the large latencies are eliminated. By removing a further, 1% there is not much improvement compared to the first 1%. Therefore, one can hypothesize that most of the measured latencies outliers are limited to 1%.

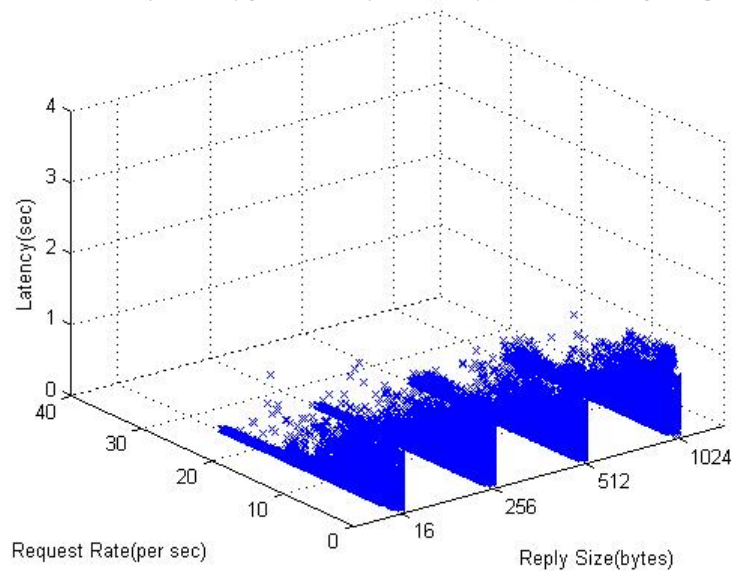


Request size and request rate impact on Max and 99% latency

3D scatter plots of reply size and request rate impact on maximum latency for login()



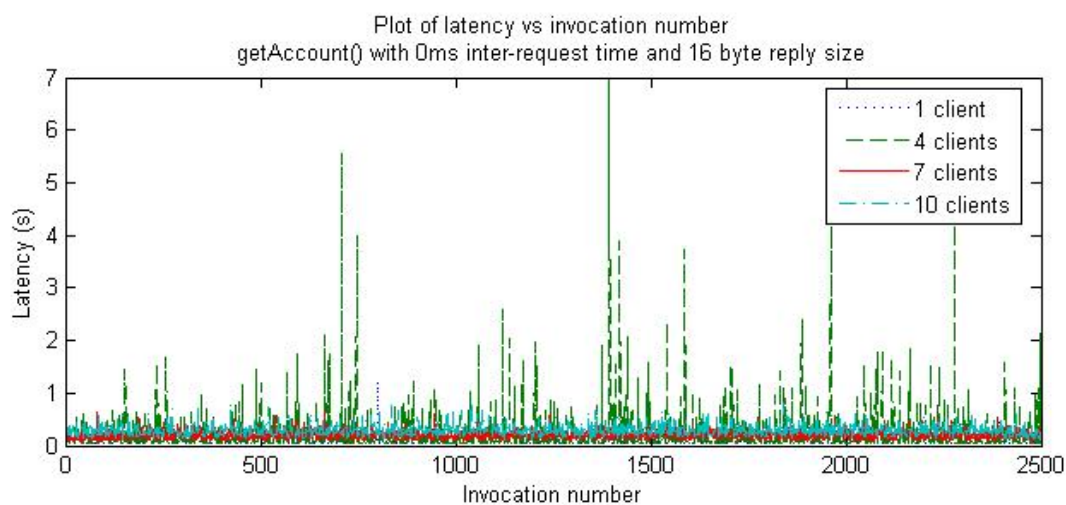
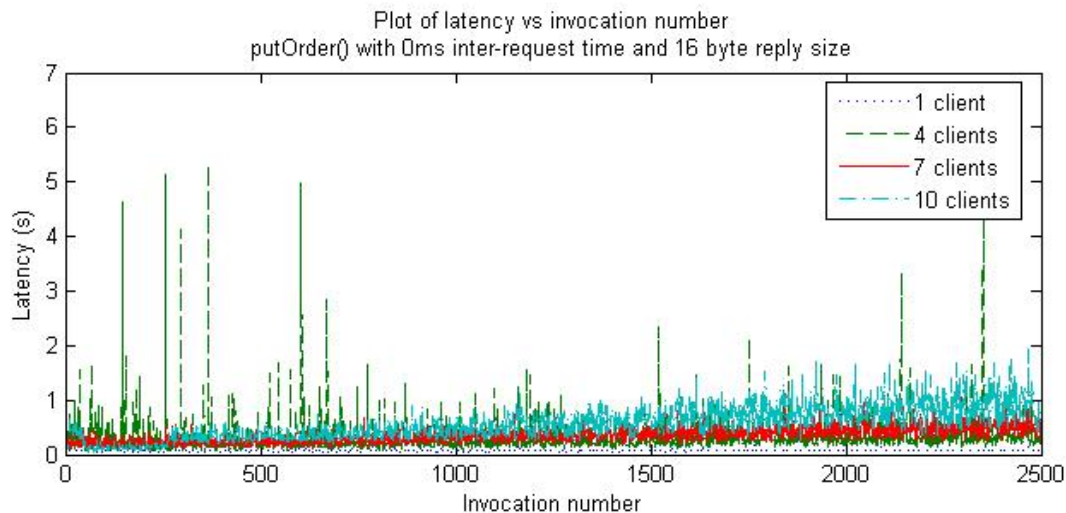
3D scatter plots of reply size and request rate impact on 99% latency for login()

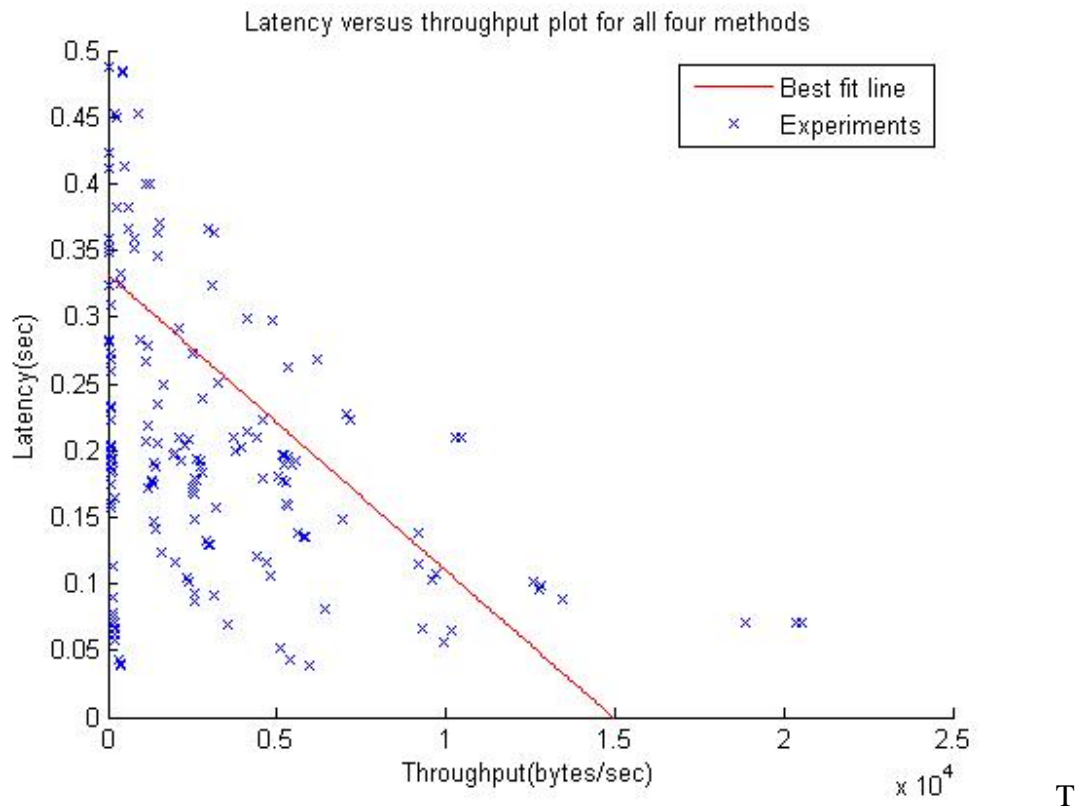


The graphs above show the reduction in latency as request rate increases. This observation contradicts common intuition that latency should increase as its gets more request. However, for maximum latencies, the relationship is inverse. One theory that when request are spread apart the server needs some time to “warm up” each time for reasons such as improper garbage collection or beans are passivated or destroyed, that would not happen if request are frequent enough. This causes significant “Start-up” delays. The change in request sizes does not really affect the latency. The usage of the 99th % however does help to make the data much clearer and easier to see trends.

Database Size Effects

For the method `putOrder()` which continually adds items for each invocation. Before each experiment set with a configuration, we would clear the database to minimize this effects. Therefore as we run our experiments, the latency for the client increases overtime as the database gets more filled. Each client run will put 2500 orders into the database. While we clean the database between experiments, we are unable to clean during the run. This makes a difference especially as we vary the number of clients. For one client there would only be 2500 entries at the end of the run. For 10 clients there would be 25000 entries. Therefore more time would be needed to find items within the database. Other methods invocated such as `getAccount()` do not add to the database and work with database that do not change therefore we do not see such affects. Therefore, when investigating the increase in latency as clients increase, it might be a good idea to look at the invocated methods (`getAccount()`, `login()`, `getTransactions()`) that do not have the added affects of a more bloated database. Notice in the graphs below how latency for a client increases as more invocations are made and the increase is more drastic when there are more clients running simultaneously only for method `putOrder()`.

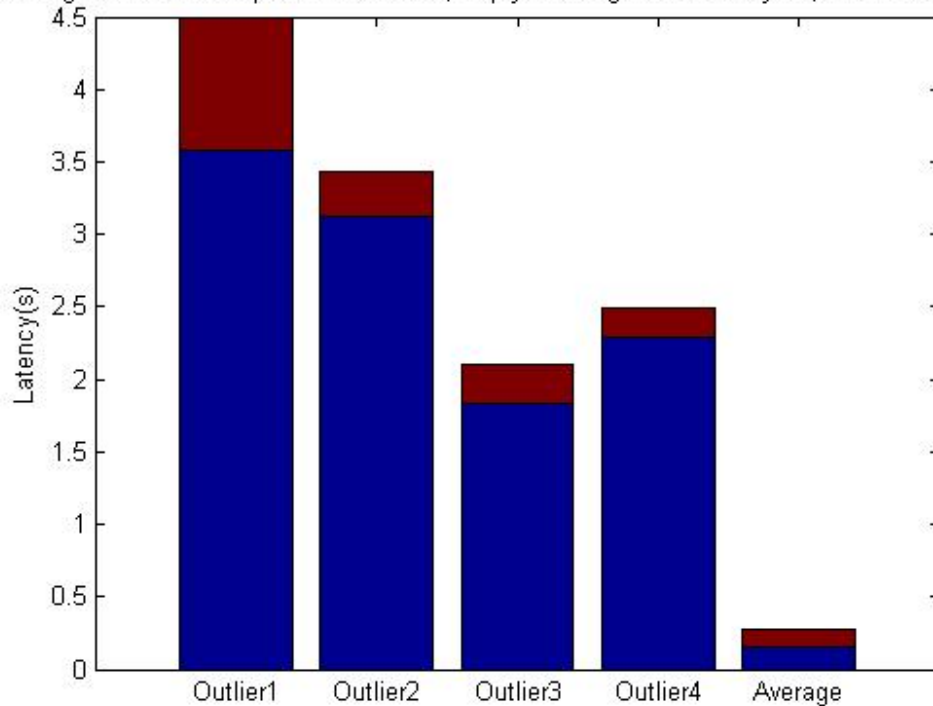


Latency vs Throughput

The scatter-plot above shows the inverse relation between latency and throughput for all the different invocation methods. The scatter dots are of the different experimental configurations. The best fit line further shows the inverse relation of latency vs throughput. Detailed scatter-plots of different methods show this result too and are included in the appendix.

Latency Component Breakdown for outliers

Bar graph of latency component break-down for different invocations outliers and average latency
 Configuration: Inter-request time - 0 ms, Reply message size - 16 bytes, No. of clients - 10



O

Outlier1 = Outliers for putOrder()

Outlier2 = Outliers for getAccount()

Outlier3 = Outliers for login()

Outlier4 = Outliers for getTransactions()

Average = Mean for all invocations

From the bar-graph above, first of all we can see that in average, latency in average is contributed by both the server and the middle-ware.

However, for the outliers, the majority of the latency is caused by delays in the middleware.

More bar-graph examples with similar conclusion are attached in the appendix.

Lessons Learned

- Communication within our team and with the TA's is very important.
- It is important to determine and confirm requirements and expectations
- Our system does not scale very well; the number of clients almost linearly increases the latency.
- Different graph styles can be used to make it easier to evaluate data. For example, a box plot makes it easy to see the distribution of data, while 3D scatter plots help determine trends with 3 variables.
- Analysis of data
 - maximum or average may not be the best measure, as in the case of the magical 1%
 - Using the 99 percentile helps eliminate outliers to make trends visible
 - Sometimes aggregate data is useful, other times a breakdown of different configurations can give a better picture
- Many factors can cause experimental data results to be skewed or invalid, as many factors as possible have to be taken into account when evaluating data. For example, system changes such as database size or processing load changes may significantly change data and lead to inconclusive or misleading results.

Appendix

Area plots with all invocations

Area plots with putOrder()

Area plots with getAccount()

Area plots with login()

Area plots with getTransactions()

Bar graphs for different configurations

Boxplot of latency vs #clients for all inter-request times and reply sizes

Line plot of 99% latency with 0 inter-request time for 4 invocations

Line plot of mean latency with 0 inter-request time for 4 invocations

Latency Vs throughput plots

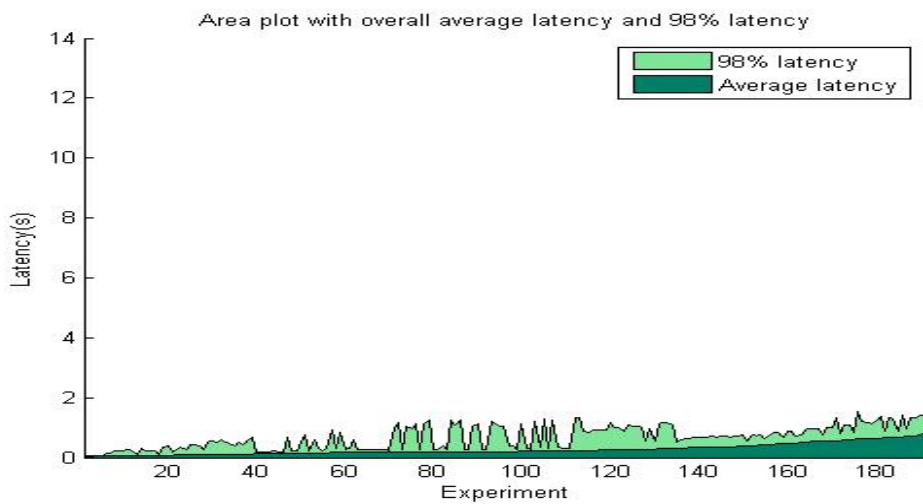
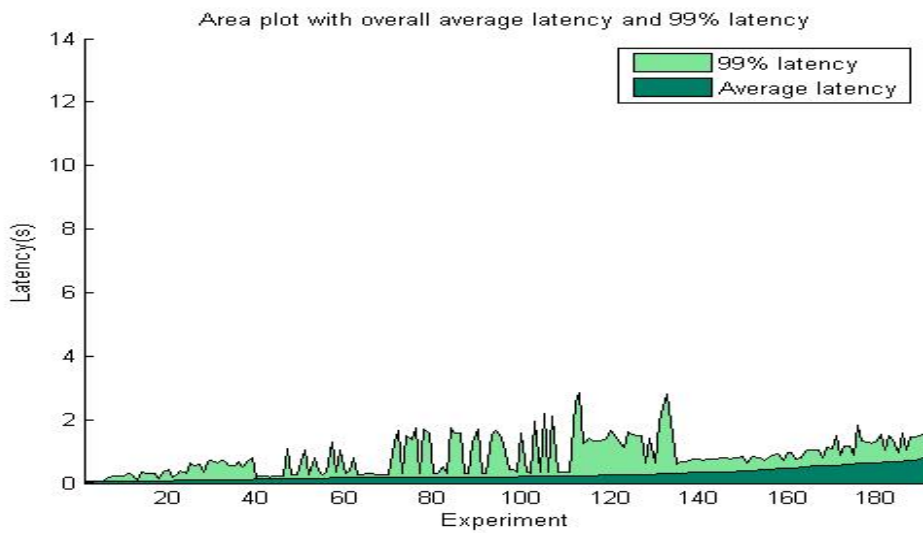
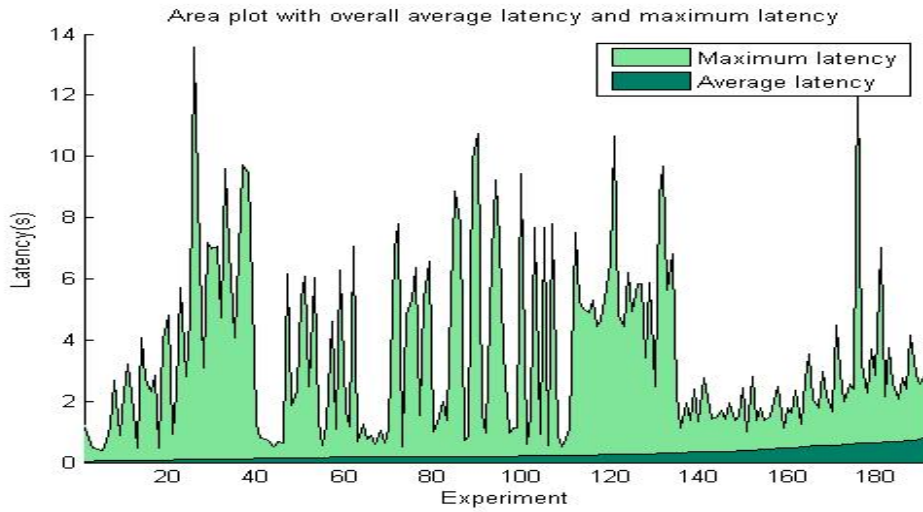
3D scatter plots for putOrder()

3D scatter plots for getAccount()

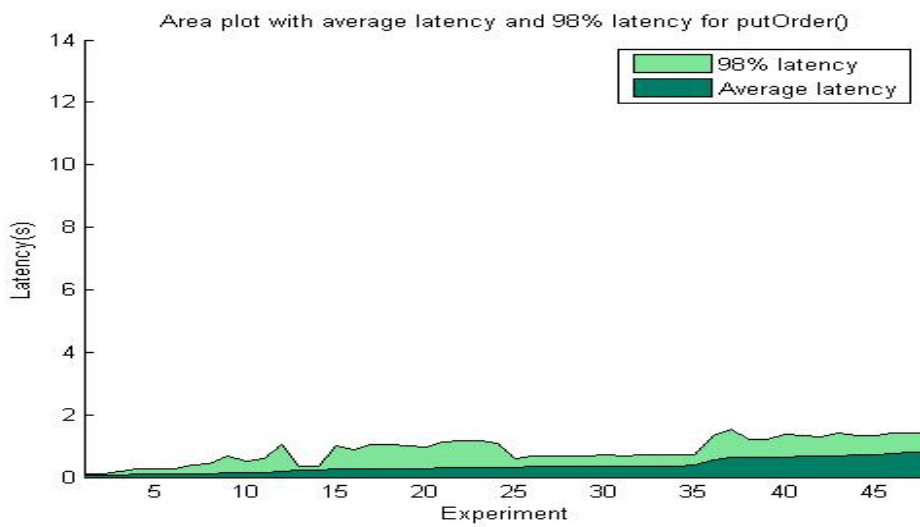
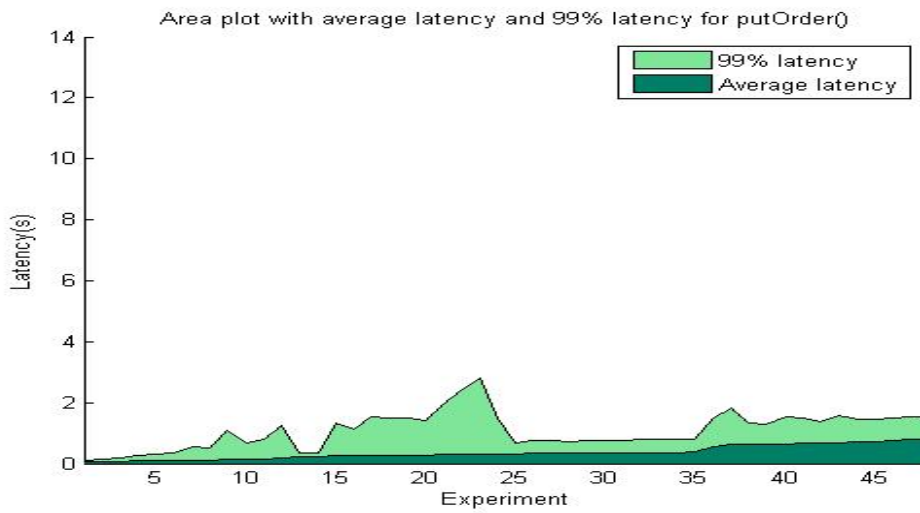
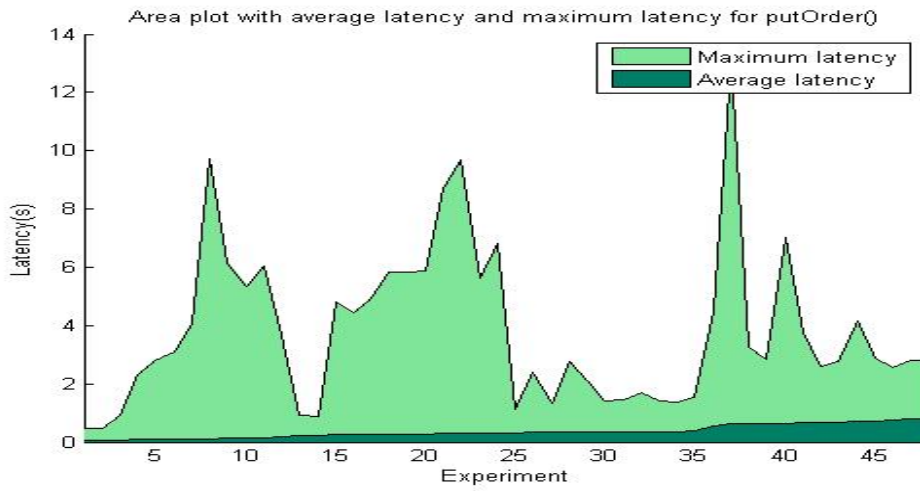
3D scatter plots for login()

3D scatter plots for getTransactions()

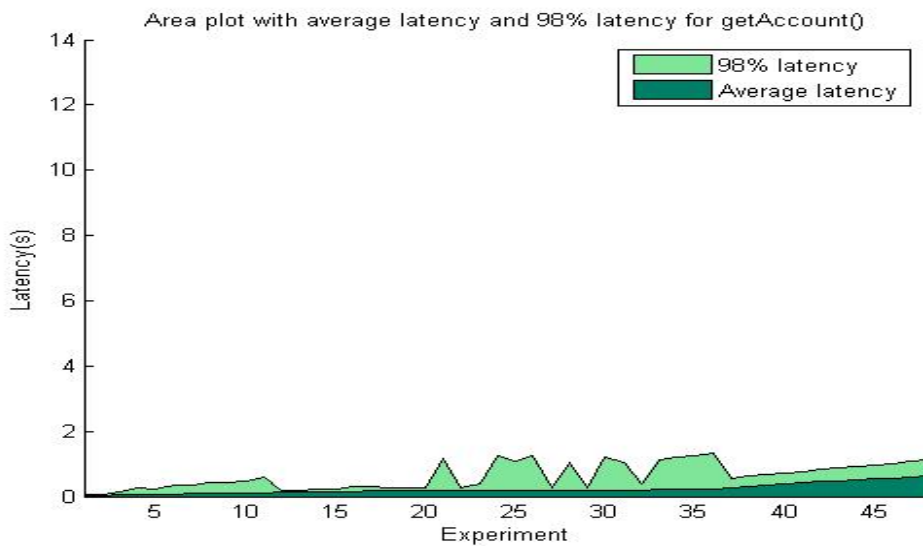
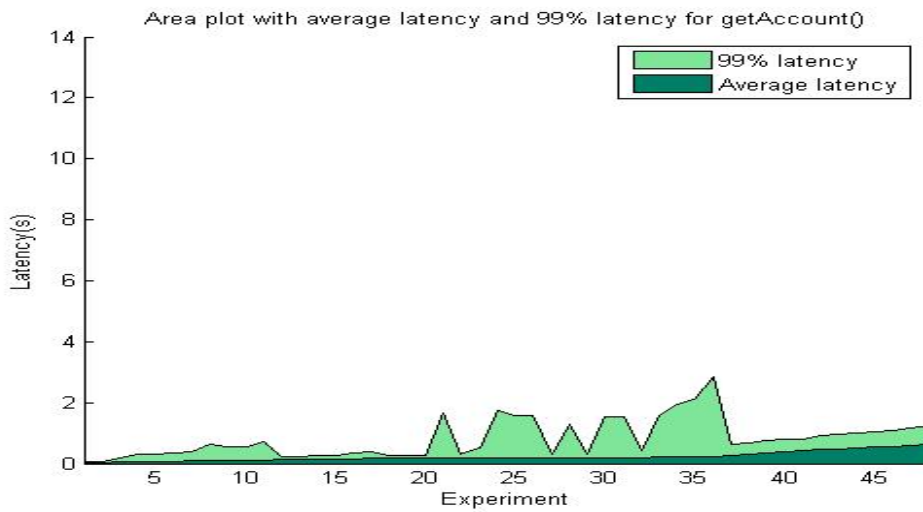
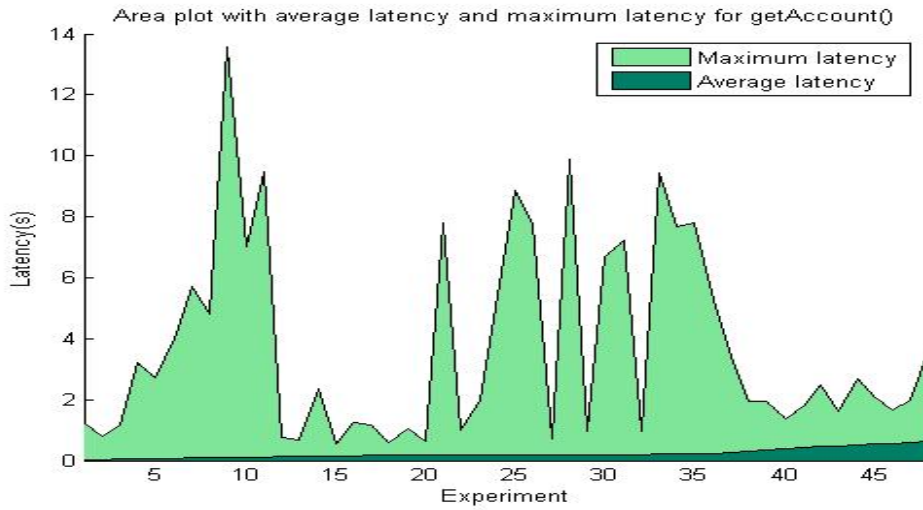
Area plots with all invocations



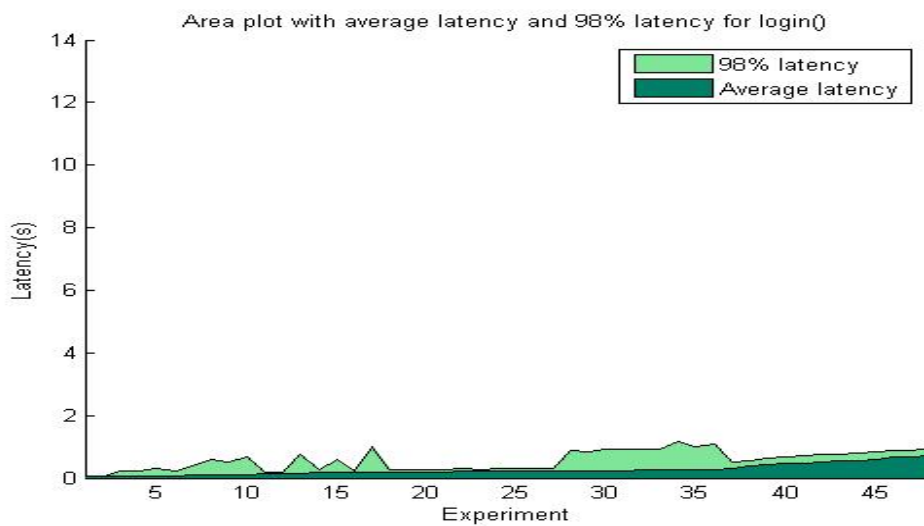
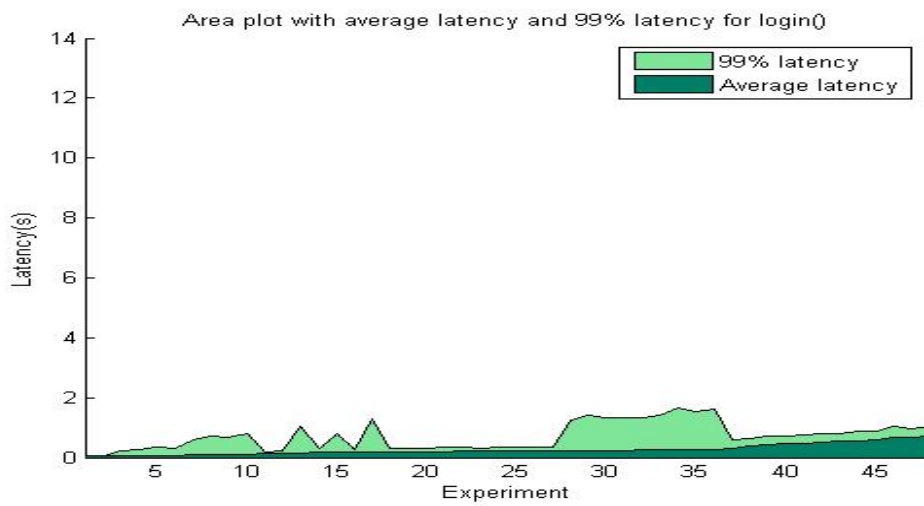
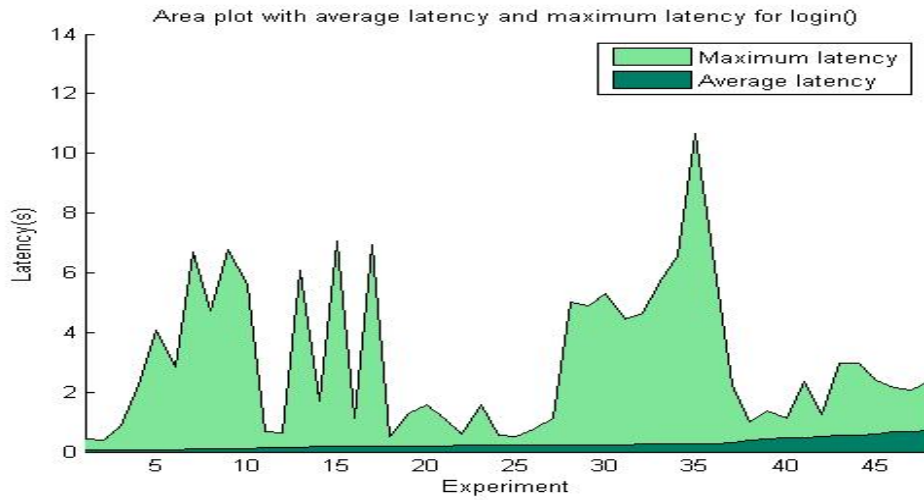
Area plots with putOrder()



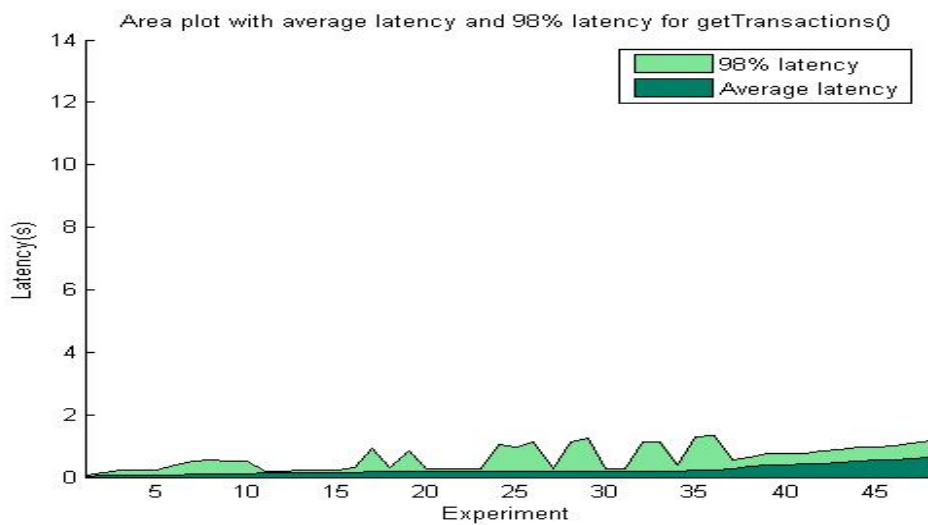
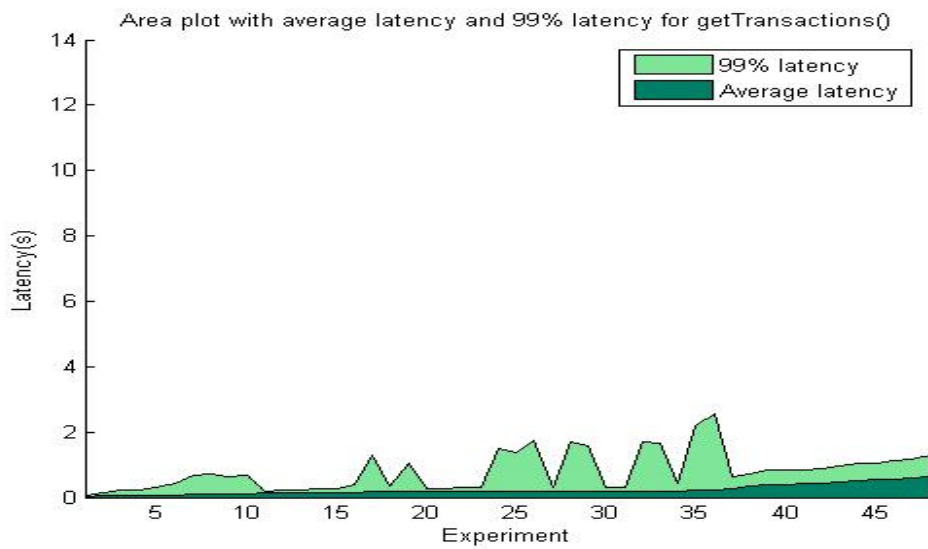
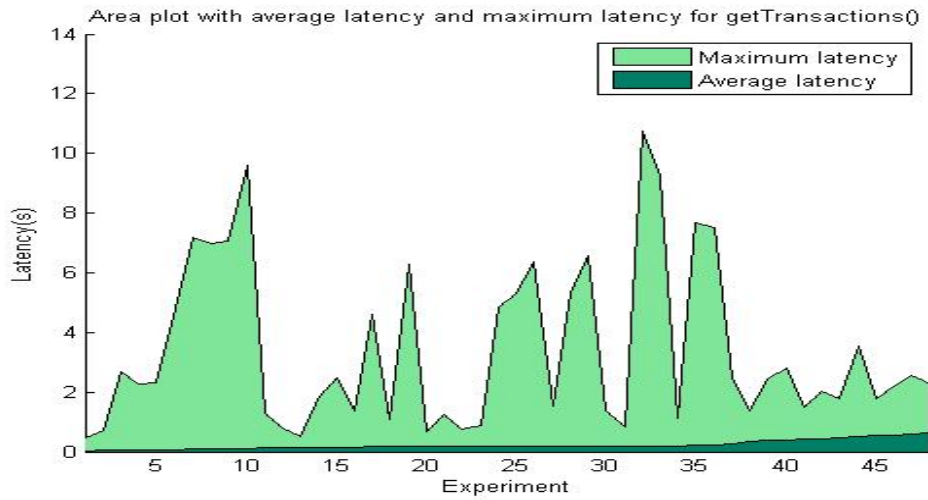
Area plots with getAccount()



Area plots with login()



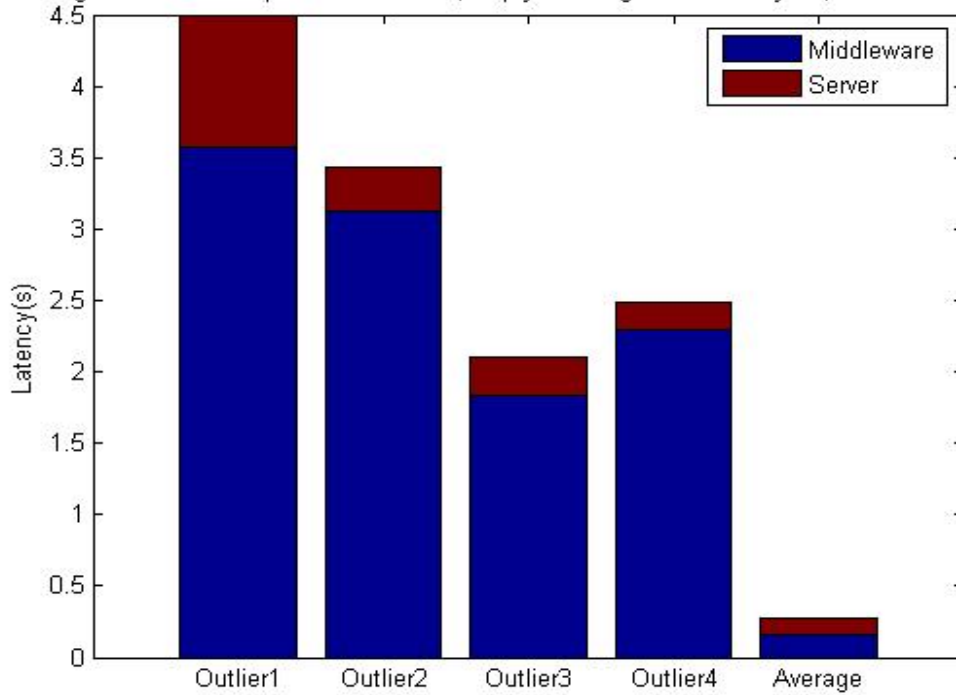
Area plots with getTransactions()



Bar graphs for different configurations

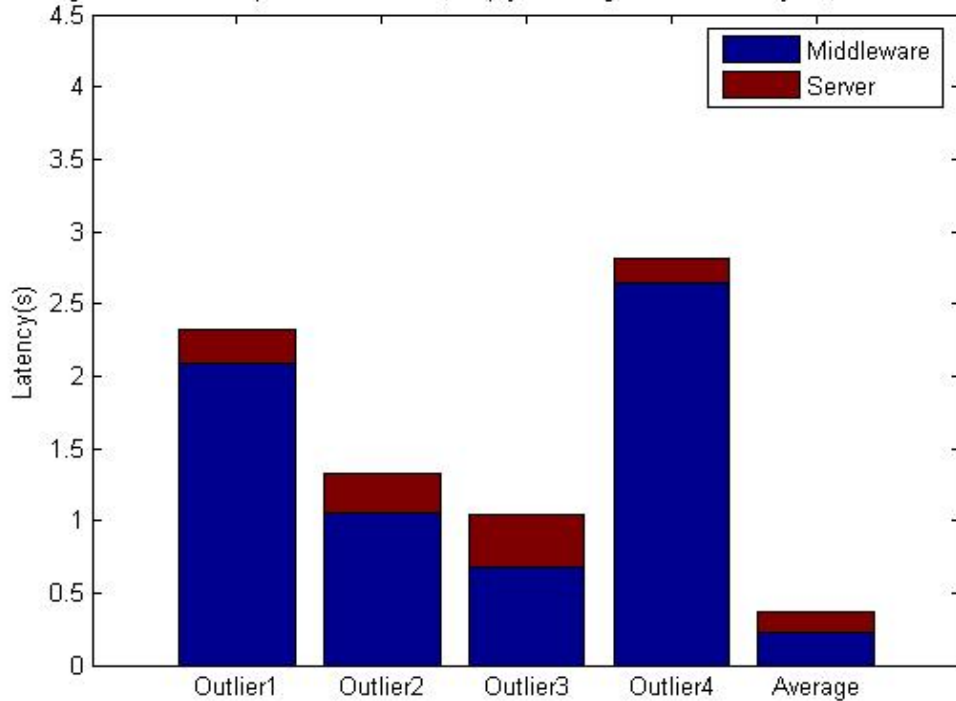
Bar graph of latency component break-down for different invocations outliers and average latency

Configuration: Inter-request time - 0 ms, Reply message size - 16 bytes, No. of clients - 10

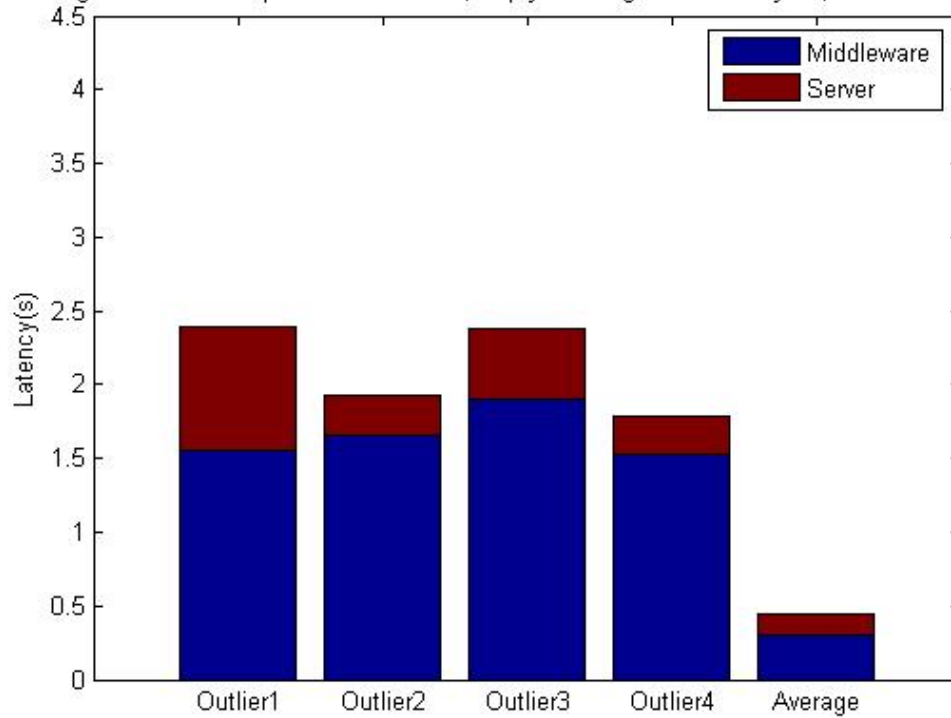


Bar graph of latency component break-down for different invocations outliers and average latency

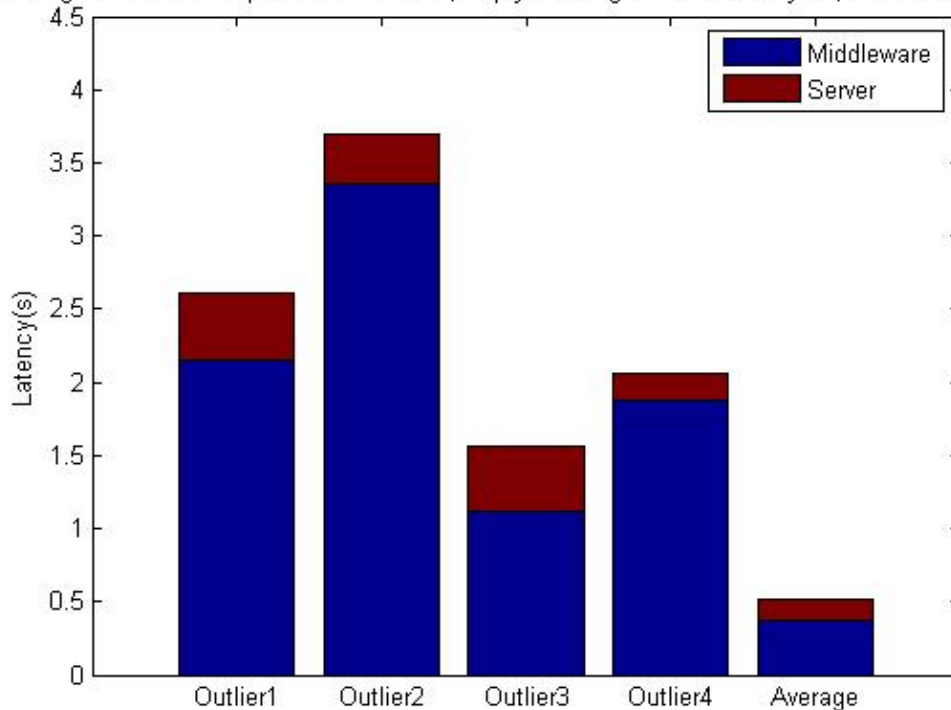
Configuration: Inter-request time - 0 ms, Reply message size - 1024 bytes, No. of clients - 10



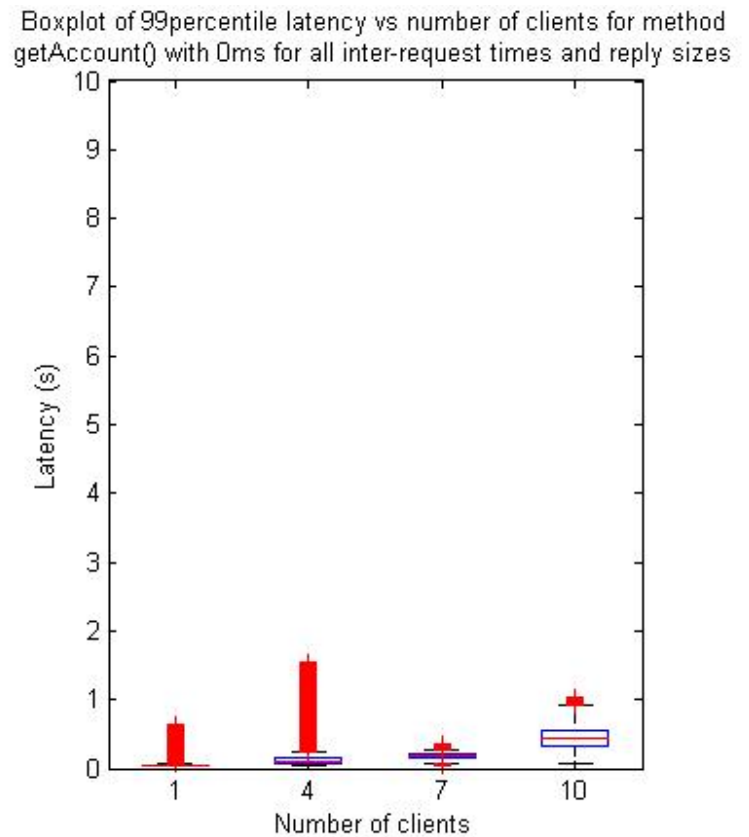
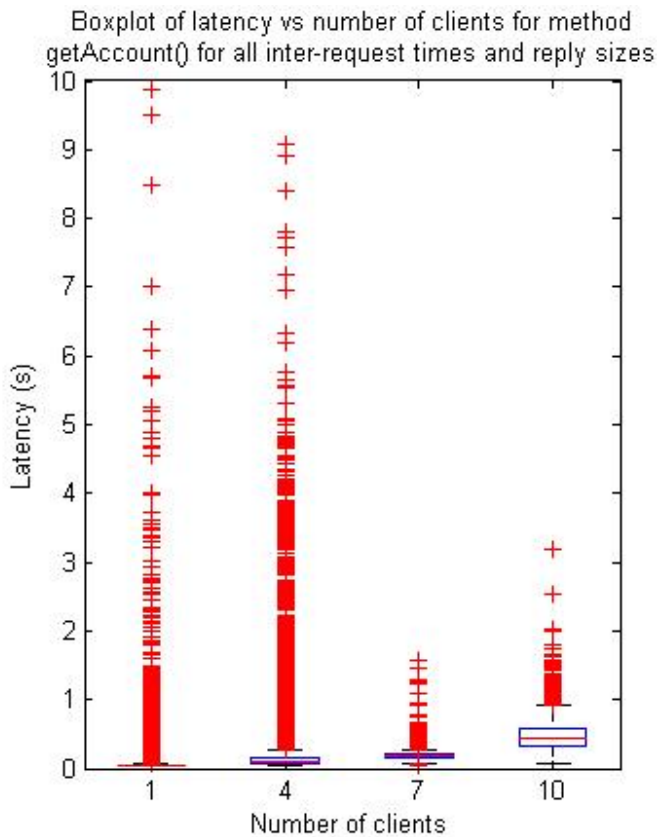
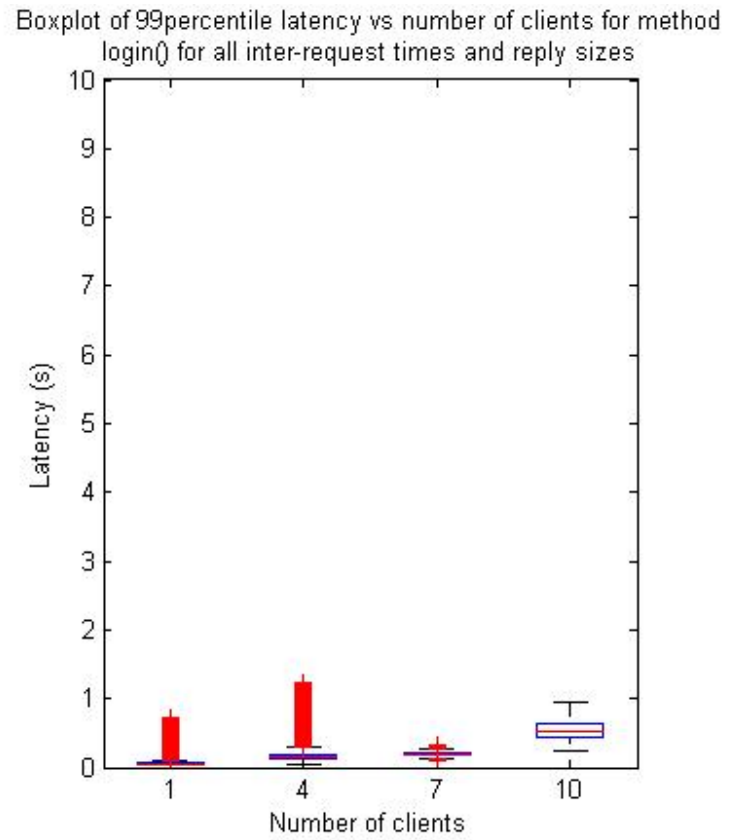
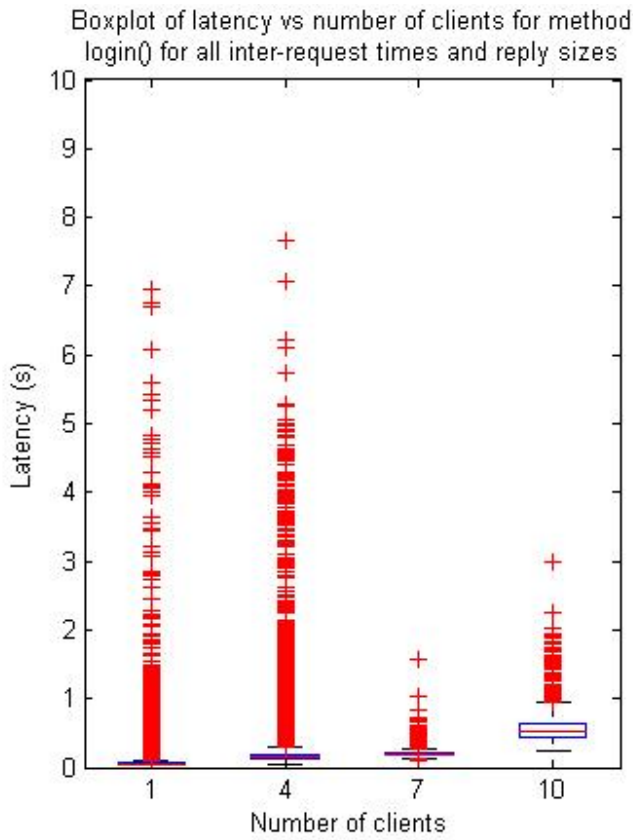
Bar graph of latency component break-down for different invocations outliers and average latency
Configuration: Inter-request time - 40 ms, Reply message size - 16 bytes, No. of clients - 10



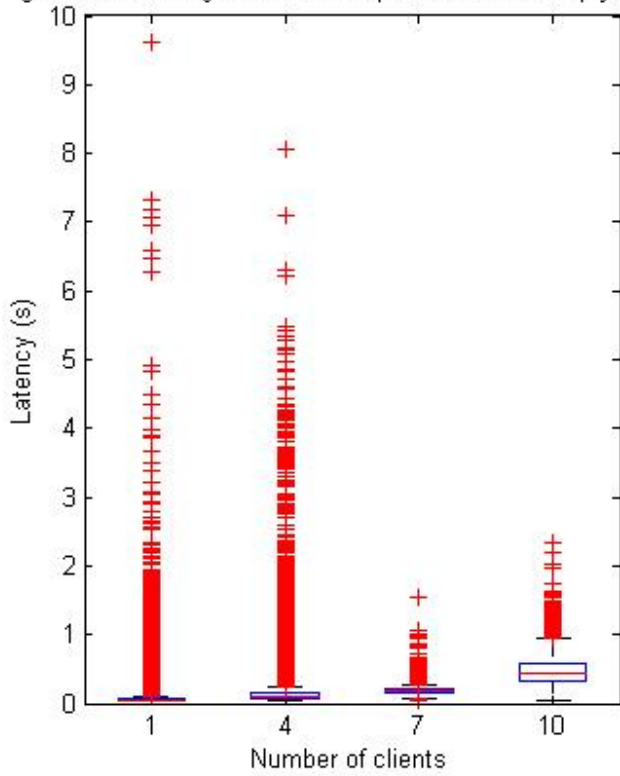
Bar graph of latency component break-down for different invocations outliers and average latency
Configuration: Inter-request time - 40 ms, Reply message size - 1024 bytes, No. of clients - 10



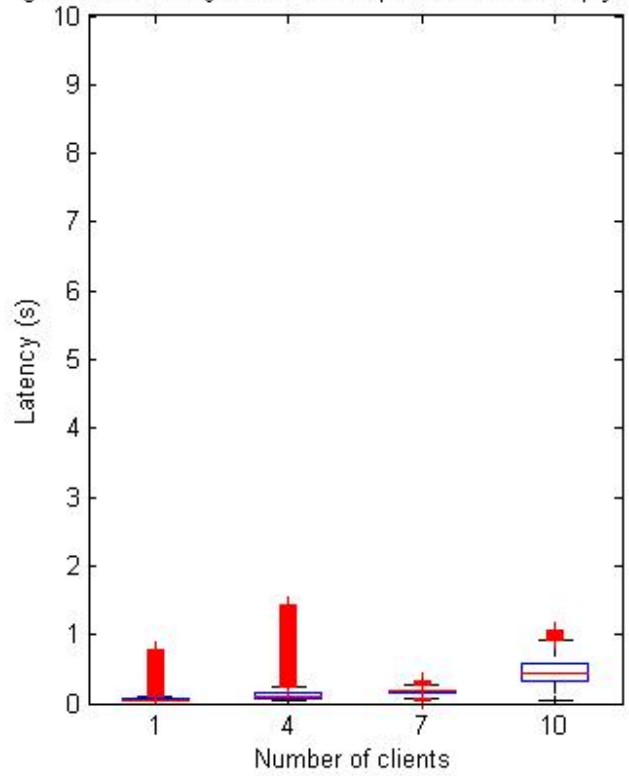
Boxplot of latency vs #clients for all inter-request times and reply sizes



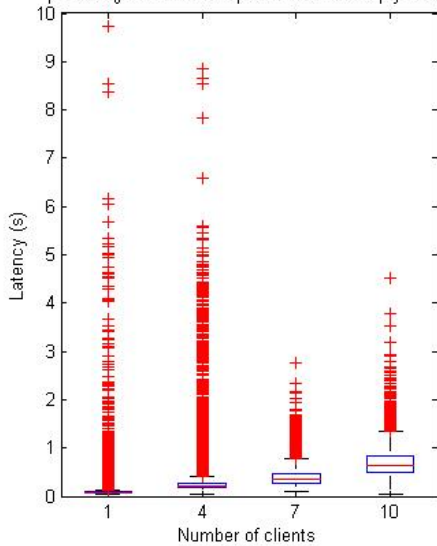
Boxplot of latency vs number of clients for method `getTransactions()` for all inter-request times and reply sizes



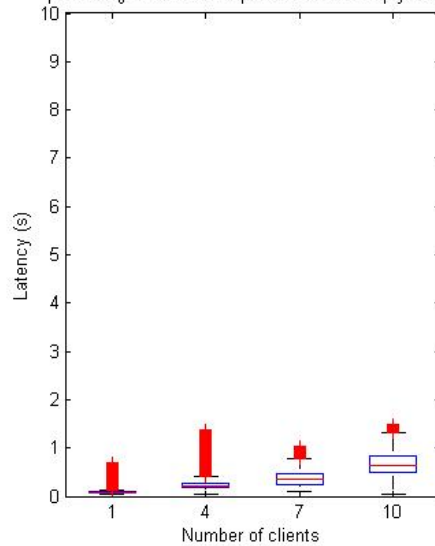
Boxplot of 99percentile latency vs number of clients for method `getTransactions()` for all inter-request times and reply sizes

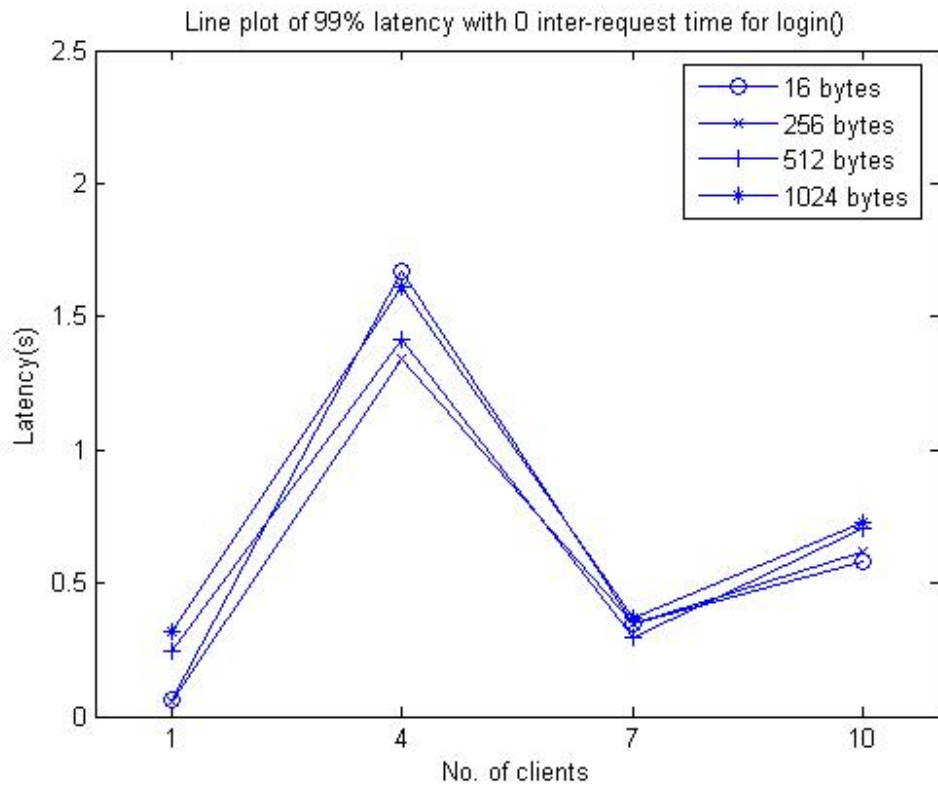
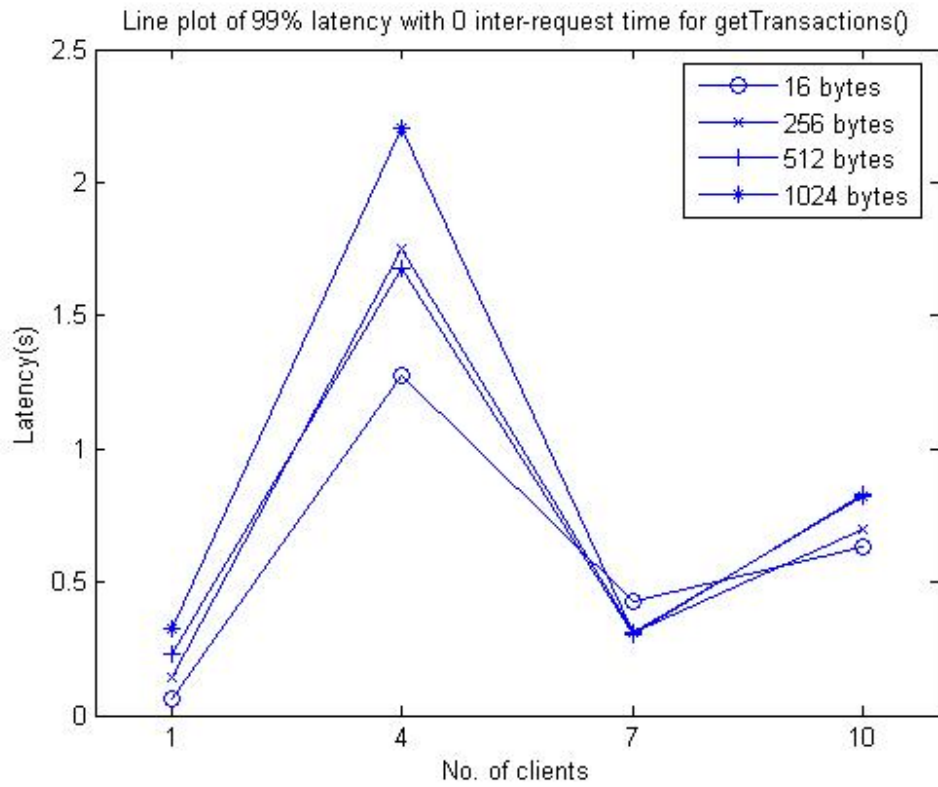


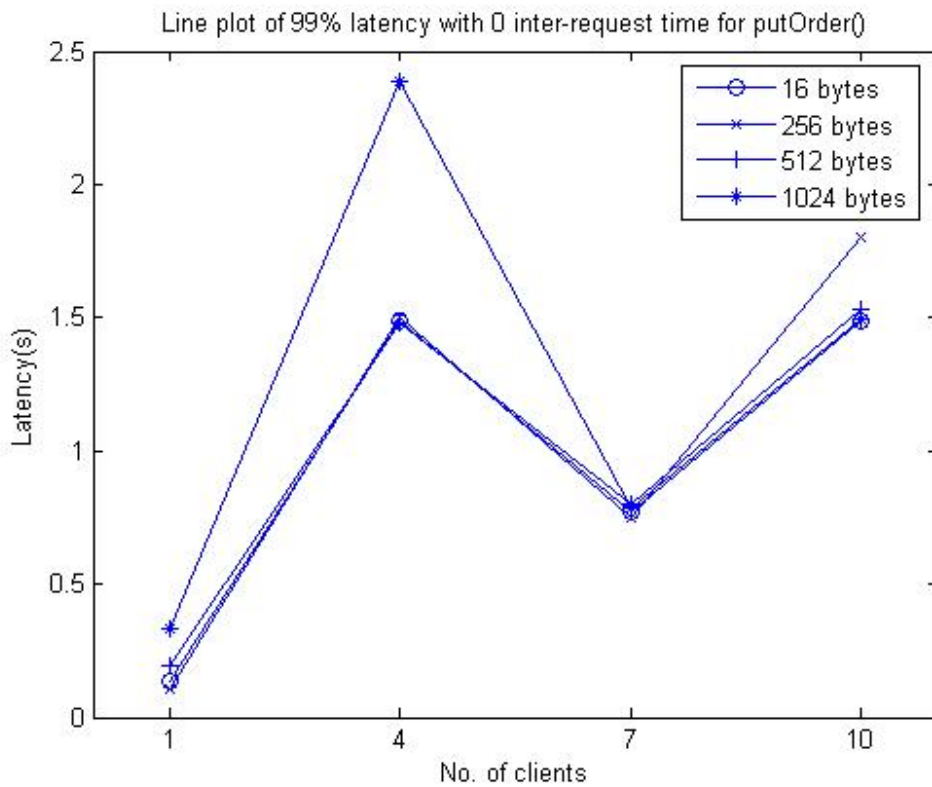
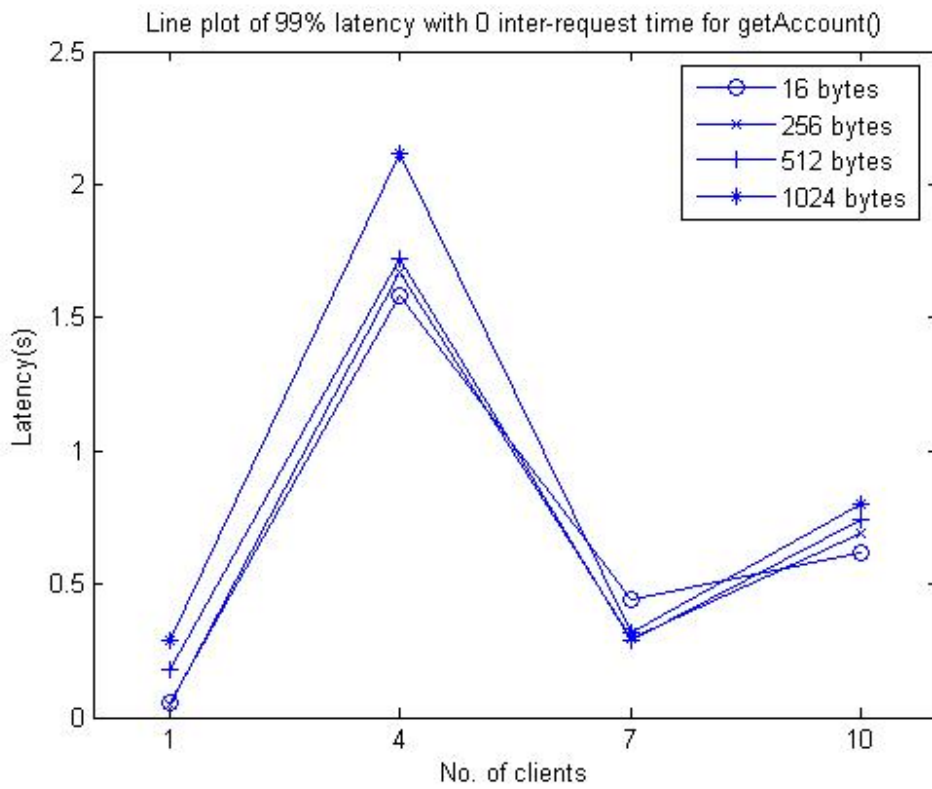
Boxplot of latency vs number of clients for method `putOrder()` for all inter-request times and reply sizes

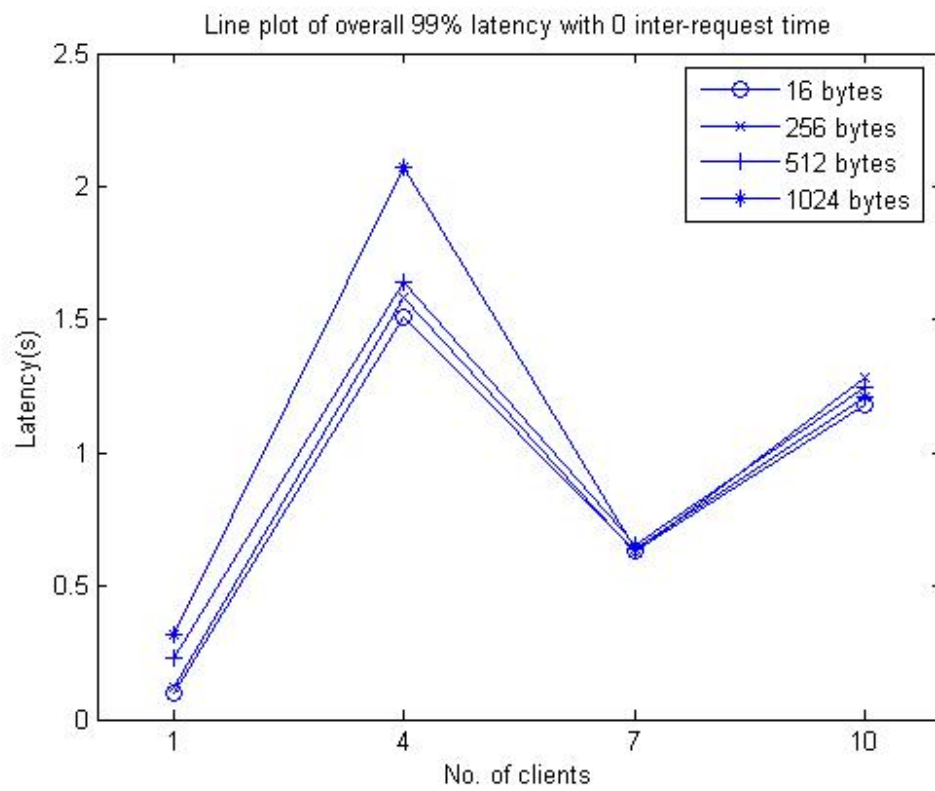


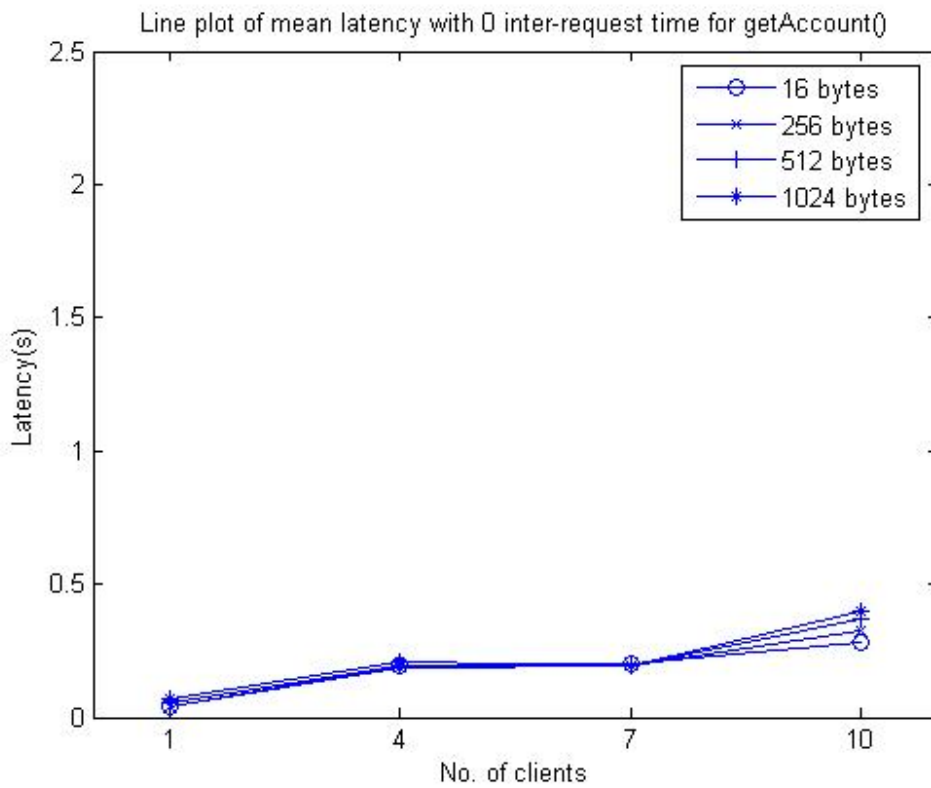
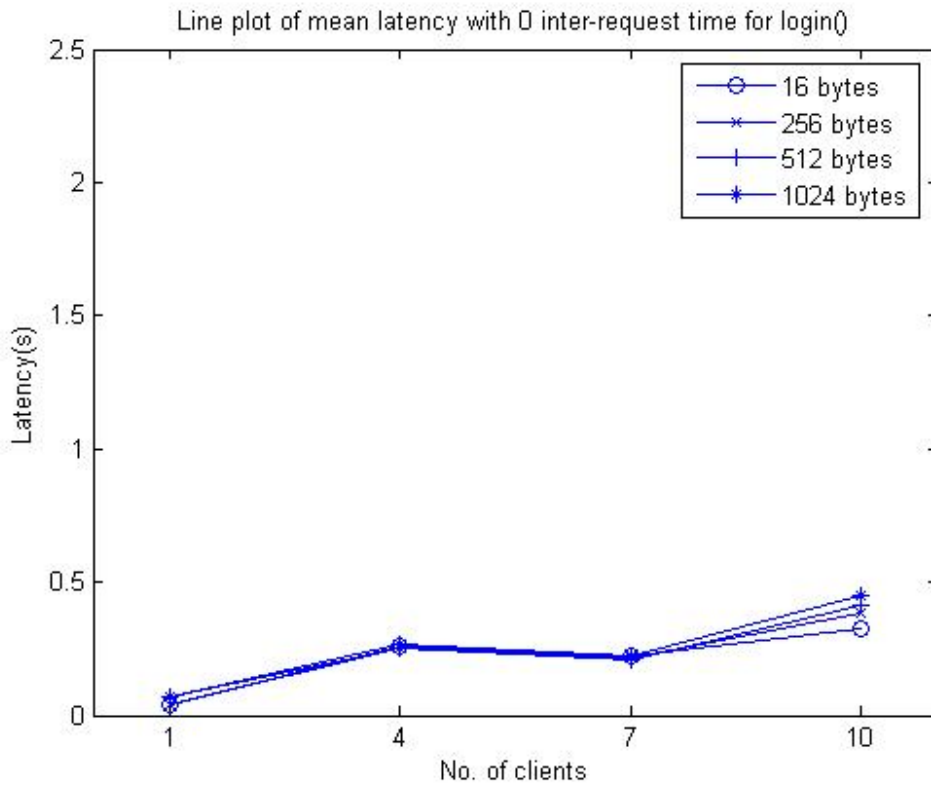
Boxplot of 99percentile latency vs number of clients for method `putOrder()` for all inter-request times and reply sizes

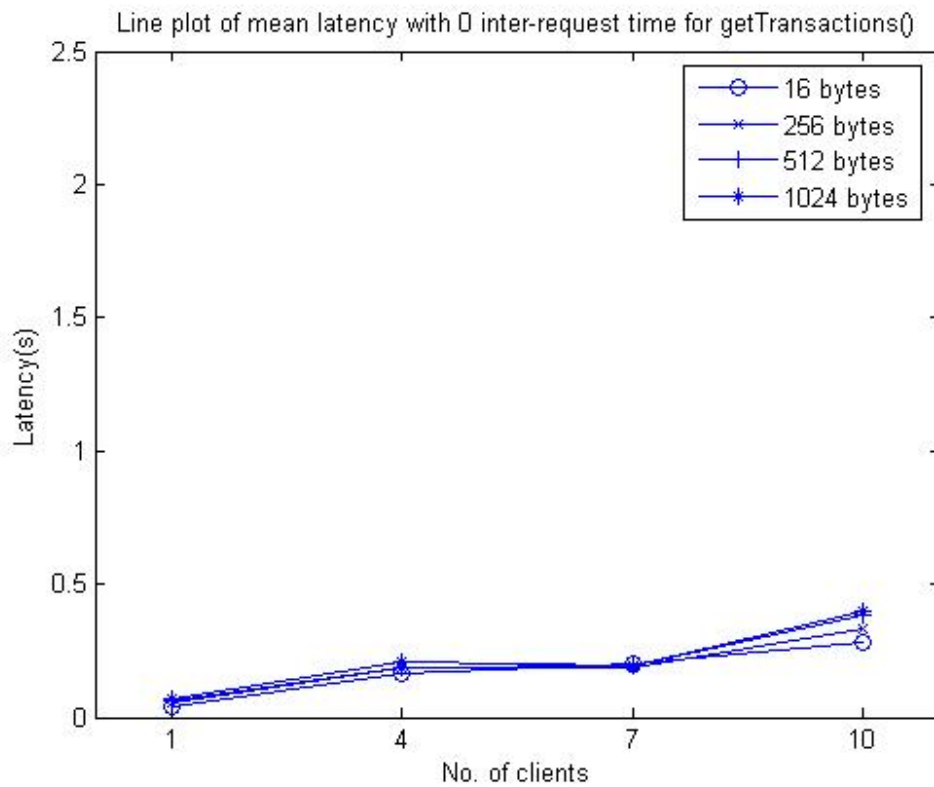
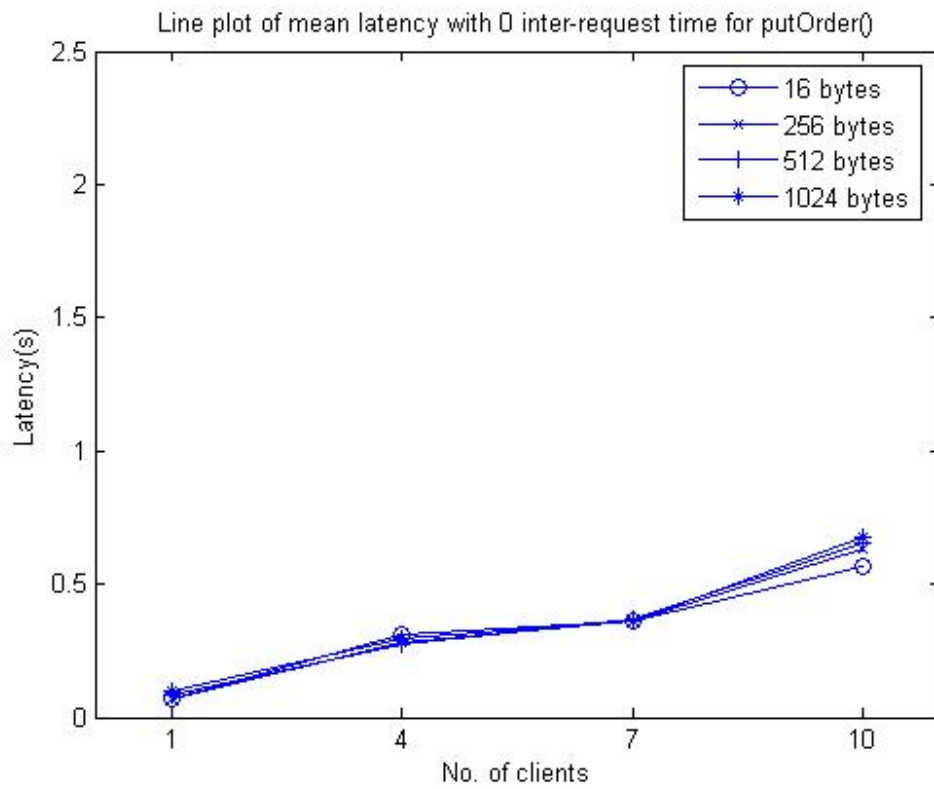


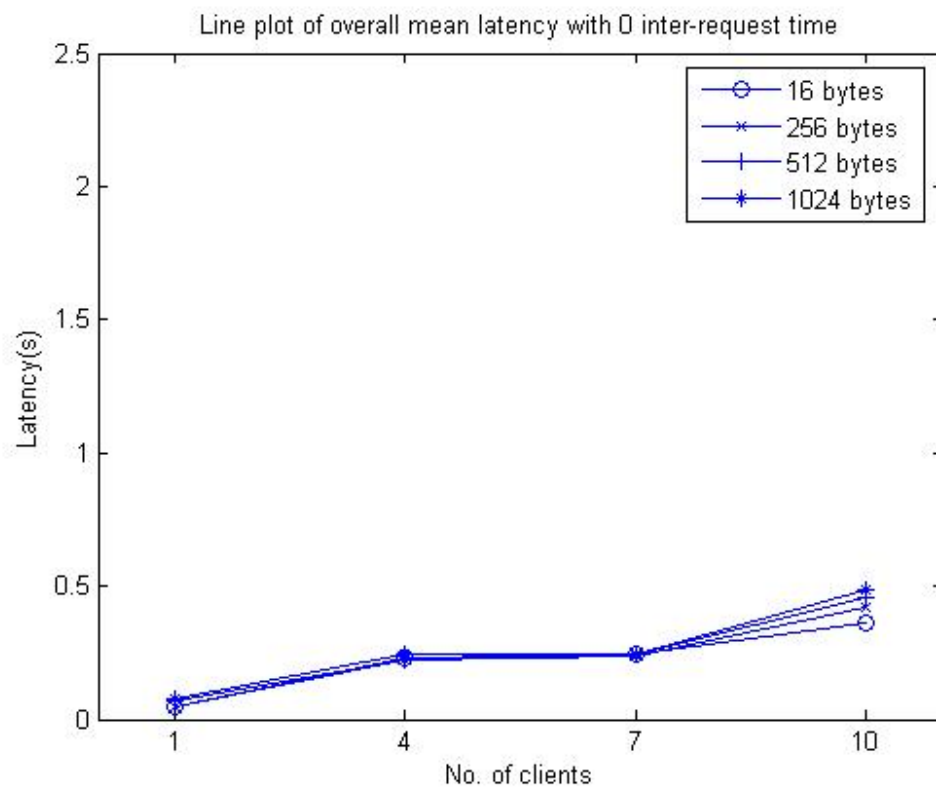
Line plot of 99% latency with 0 inter-request time for 4 invocations



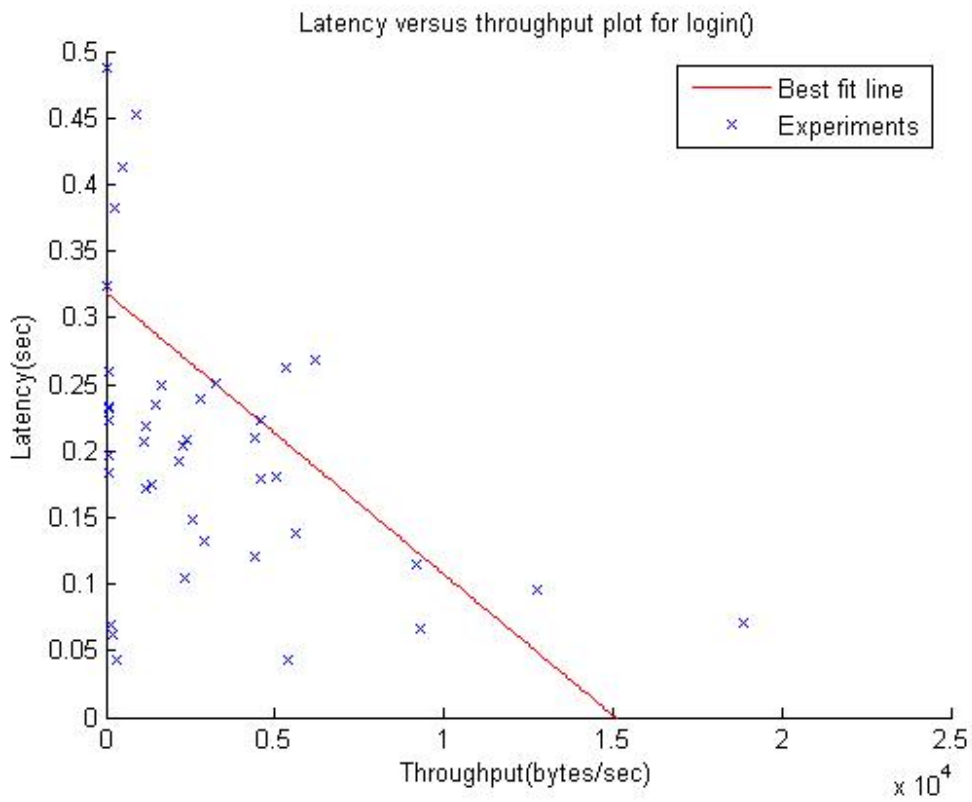
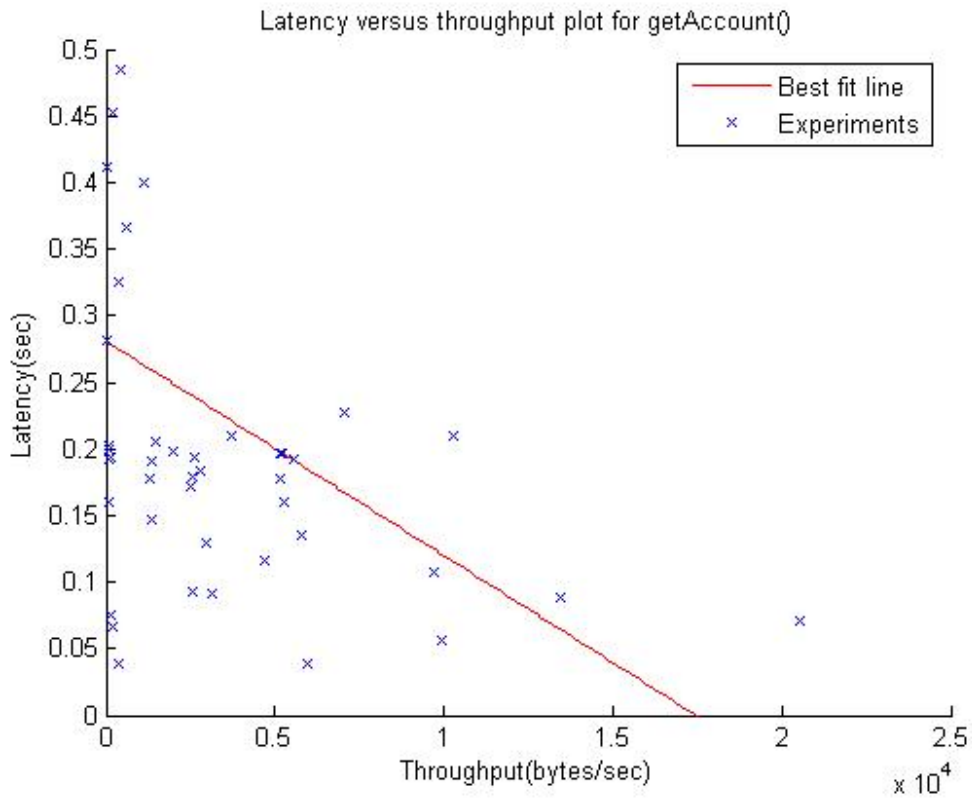


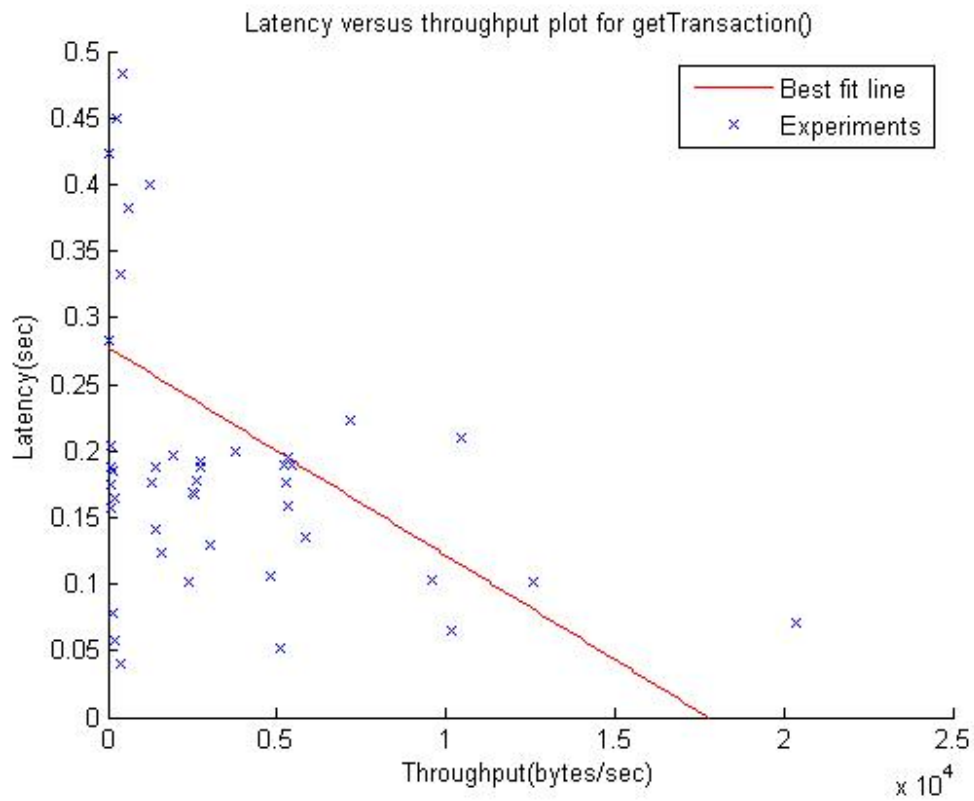
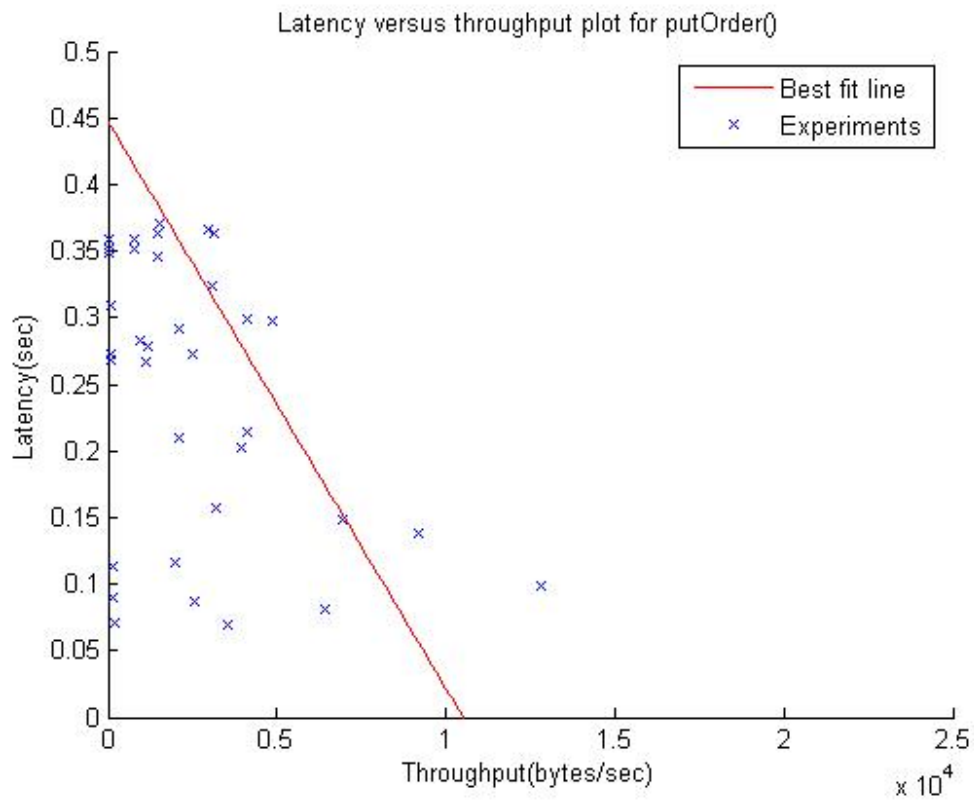
Line plot of mean latency with 0 inter-request time for 4 invocations





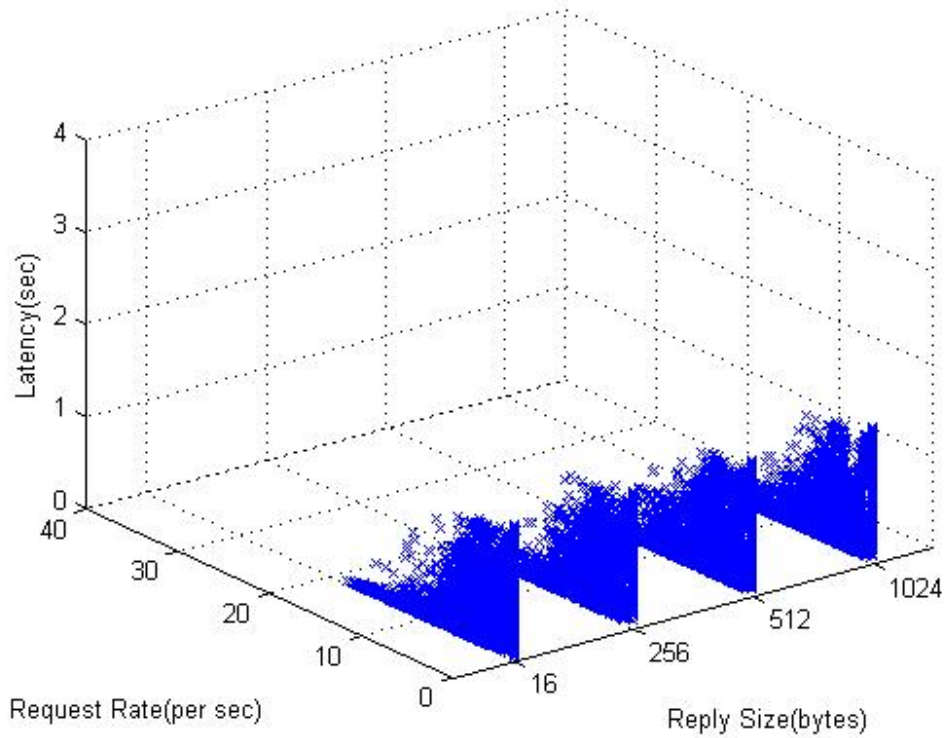
Latency Vs throughput plots



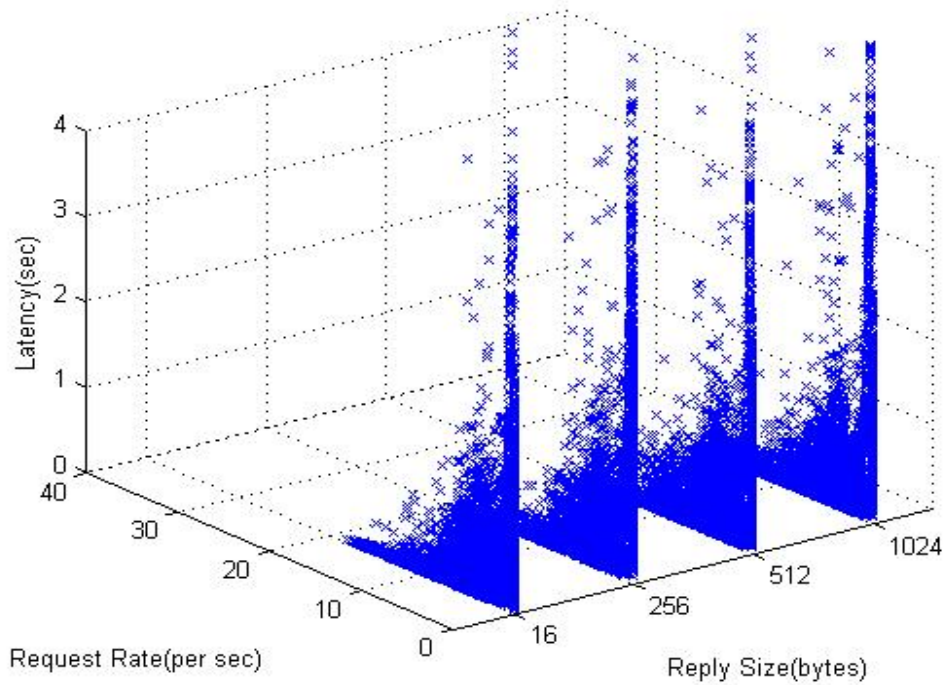


3D scatter plots for putOrder()

3D scatter plots of reply size and request rate impact on 99% latency for putOrder()

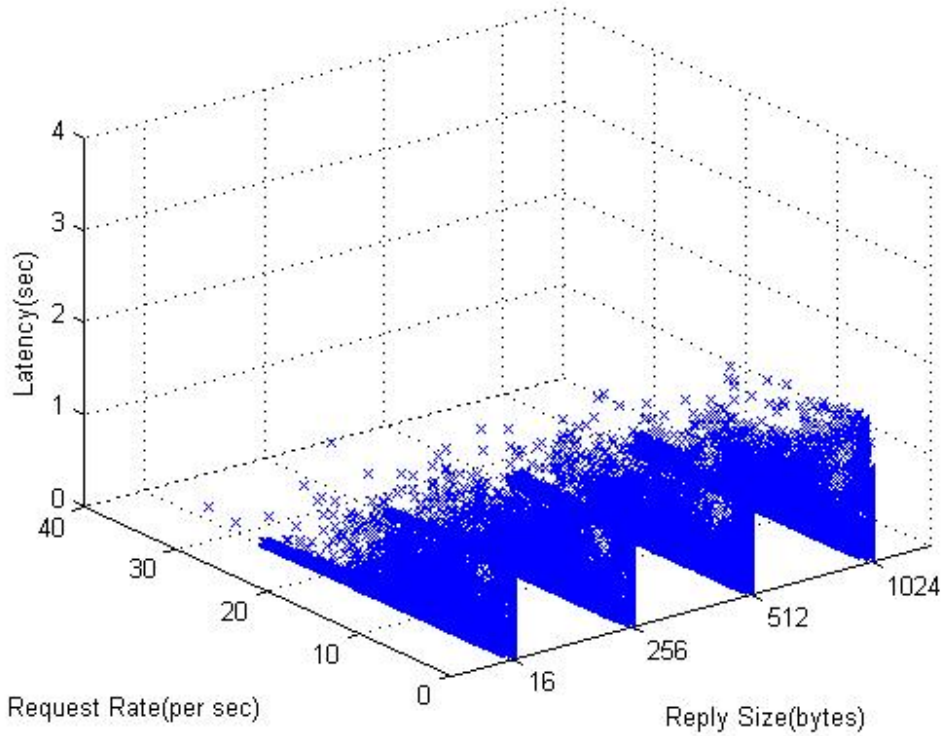


3D scatter plots of reply size and request rate impact on maximum latency putOrder()

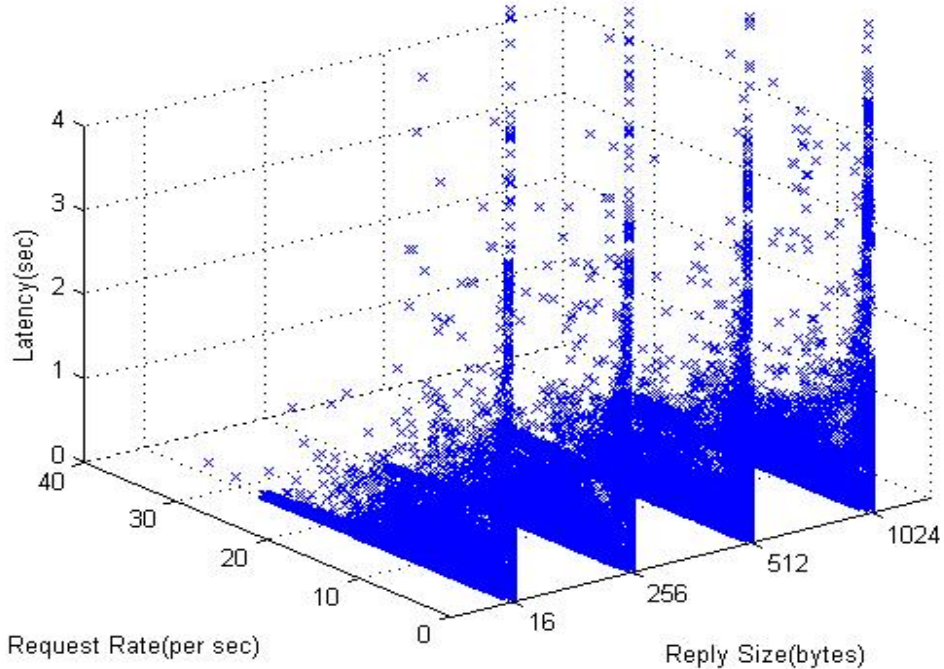


3D scatter plot for getAccount()

3D scatter plots of reply size and request rate impact on 99% latency for getAccount()

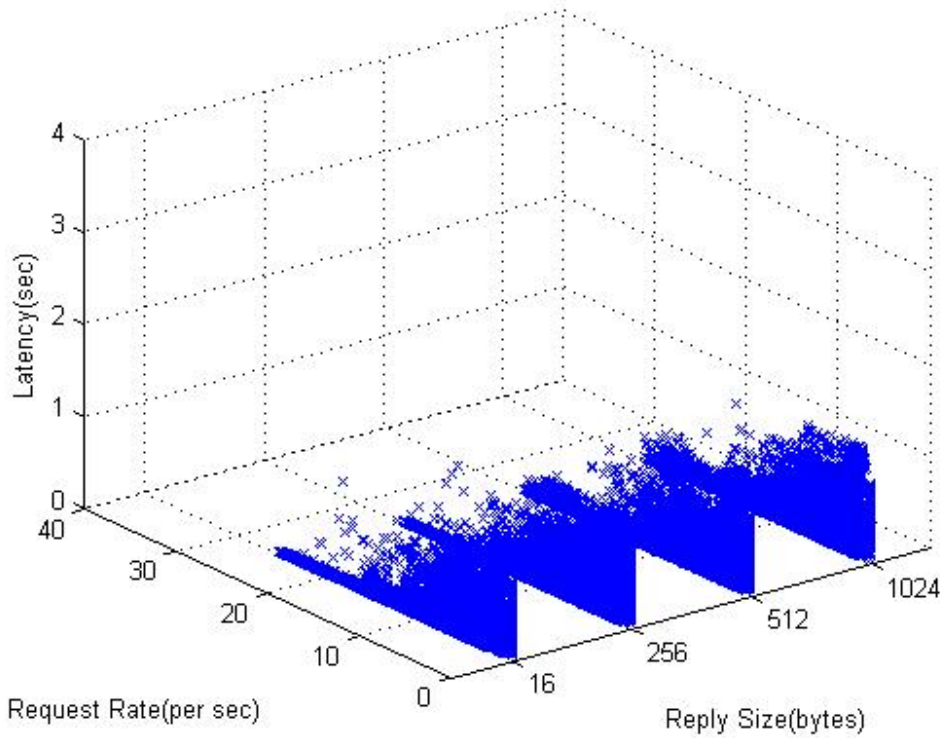


3D scatter plots of reply size and request rate impact on maximum latency for getAccount()

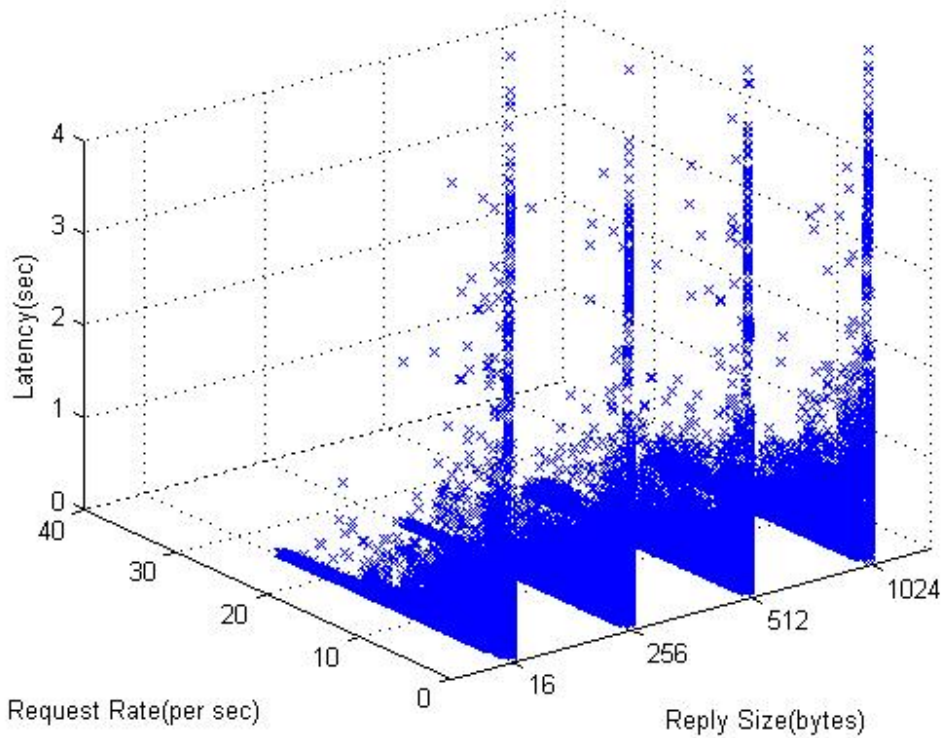


3D scatter plot for login()

3D scatter plots of reply size and request rate impact on 99% latency for login()

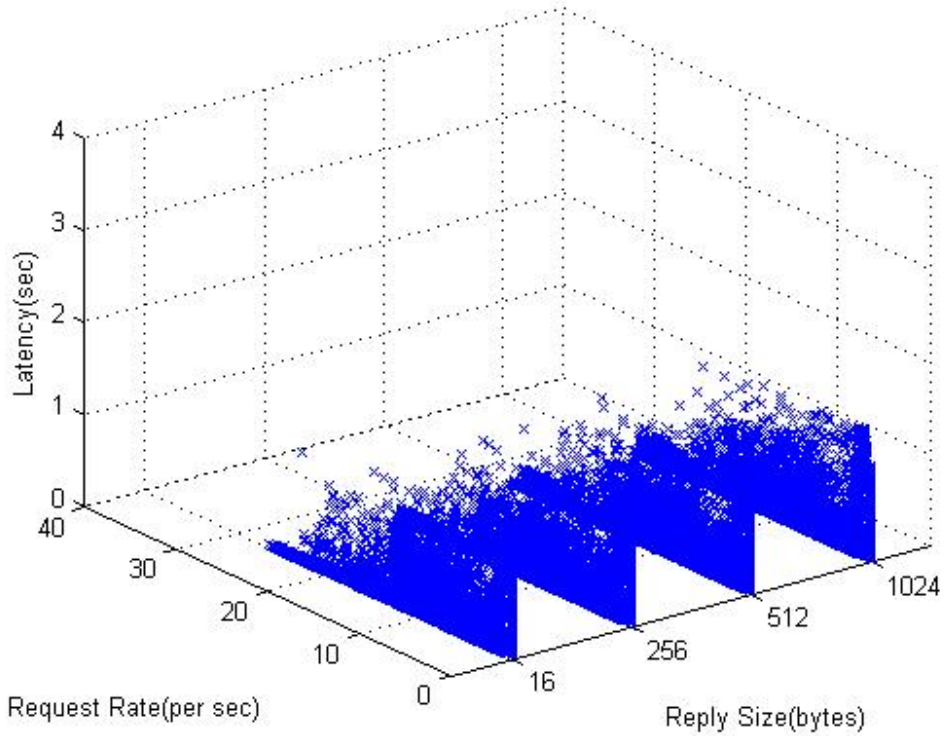


3D scatter plots of reply size and request rate impact on maximum latency for login()



3D scatter plot for getTransactions()

3D scatter plots of reply size and request rate impact on 99% latency for getTransactions()



3D scatter plots of reply size and request rate impact on maximum latency for getTransactions()

