Advanced Systems Lab Report

Autumn Semester 2017

Name: IVAN TISHCHENKO
Legi: 17-945-536

Grading

Section	Points
1	
2	
3	
4	
5	
6	
7	
Total	

1 System Overview

This section contains the description of the implementation of the distributed middle-ware system which is designed to run on Microsoft Azure platform. All the work has been performed on the GNU/Linux operating system, to which the designed system is primarily targeted. The section highlights design decisions which were relevant while performing the experiments, explains how messages are parsed and how instrumentation is performed in a multi-threaded environment. The system is illustrated with a figure which contains all the threads and queues of the system (network and the memcached servers included), followed by explanations that show how requests of different types are handled in the system. The details regarding the artifacts related to the system are included at the README file at he git repository.

1.1 System's architecture

The following subsection describes overall system's architecture. The system designed for the course [1] contains the following components:

- **RunMW** [2] - Serves as the main entry point for the middle-ware system. Extended boilerplate originally according to the course specification. The client launches a command line application and specifies the command line parameters such as: IP address of the middle-ware, port on which middle-ware operates, IP addresses and corresponding port of the memcached servers, flag which enables sharded GETS.
- **MiddlewareMain** [3] - Main class of the system. Represents system's architecture and assembles its components. This class is responsible for starting middle-ware's components such as the NetThread and all the Workers. Furthermore it contains systems crucial architectural components such as workers pool, incoming request queue and cyclic counter for the round robin pattern. The entry point is the run method. The class is also contains the so called "shut down hook", which is triggered on systems exit.
- **NetThread** [4] - One of the most crucial components of the system's is it's network thread. NetThread extends Java's Thread class and is responsible for dealing with the incoming network requests from clients on the specified port. After accepting the requests NetThread queue the Request into the internal queue of requests. NetThread relies on the Java's NIO package, which is a natural choice in the setting where a single thread has to deal with multiple channels for data in the non-blocking IO fashion. The main method of the class is serve(), which contains a Selector object, which monitors channels for events. In NIO implementations the data is always read from a Channel into a Buffer, which doesn't cause blocking. This feature allows us accepting other client connections, while the channel is reading request data into a buffer.
- **Worker** [5] - This class represents another important architectural component, namely the Worker Thread. Before starting it's operation each instance of Worker opens the connections to the memcached servers, which remain open permanently until shutdown. The Worker thread dequeues the Request object from the internal jobs(request) queue

[1] `https://gitlab.ethz.ch/tivan/asl-fall17-project/tree/master/src/ch/ethz/asl/main`

[2] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/RunMW.java`

[3] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/MiddlewareMain.java`

[4] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/NetThread.java`

[5] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/Worker.java`

which is located in NetThread. After the job is taken from the queue, each Worker object is capable of handling SET, GET, MULTI-GET(both sharded and non-sharded) and UNSUPORTED requests. At the reception of all responses Worker threads forwards processed/assembles response back to clients. See next section for detailed description of request handling.

- **Request** [6] - The class is the representation of incoming requests. The same object is used to represent multiple types of requests. Request class serves as a data wrapper for requests representation containing fields such as Request type, Requests message and SocketChannel, which provides the gate of communication back to the client after the responses from servers have been acquired.

- **Parser** [7] - This class contains helper methods commonly used by other components in the system. It represents parsing, processing, information extraction and other related operations performed on system's incoming data. The main purpose of class is to decouple repetitive data manipulation tasks from architectural components.

- **CycleCounter** [8] - The class servers as an implementation of the modular counter (e.g. in case of 6 requests and 3 servers it's distributed to server with IDs as: 0, 1, 2, 0, 1, 2). Updating this counter is blocking, thus the object is thread-safe and can be shared across multiple instances. The main purpose of the CycleCounter is to implement the Round Robin (RR) scheduling pattern in order to distribute the load to multiple servers.

- **Statistics** [9] - The object is the structure implementation of the required instrumentation. Contains the fields with required statistics to be measured. Each worker thread would have it's own copy of Statistics object, which enables easy aggregation when the middle-ware shuts down.

- **ShutDownHook** [10] - The code contained in this objected is executed, when user shuts down the middle-ware application. ShutDownHook is a suitable place for aggregating all of the middle-ware's statistics and printing the final results, which were used later in the experiments, plotting and modeling.

- **Histogram** [11] - The class represents a histogram of request times with the step of 0.1 msec, which is printed on programs shut down. Takes the list of all response times as an input and assigns each response time value to the corresponding been.

1.2 Request handling

As depicted on the components figure the entry point of the Middle-ware for all Requests is the NetThread. NetThread listens for all incoming requests from clients Memtier $1...N$. Although NetThread is a single thread it is capable of handling multiple events in non blocking manner

[6] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/Request.java`

[7] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/Parser.java`

[8] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/CycleCounter.java`

[9] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/Statistics.java`

[10] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/ShutDownHook.java`

[11] `https://gitlab.ethz.ch/tivan/asl-fall17-project/blob/master/src/ch/ethz/asl/main/Histogram.java`

as described earlier. On reception NetThread reads text of the request and classifies it with the help of a method located in the Parser class. After construction of Request object NetThread puts the object into the request queue. The request queue is a blocking object, which could be safely shared among Workers 1...N from the thread pool. Afterwards the job is taken by the Worker. Each Worker has a socket stream to all of the memcached servers.

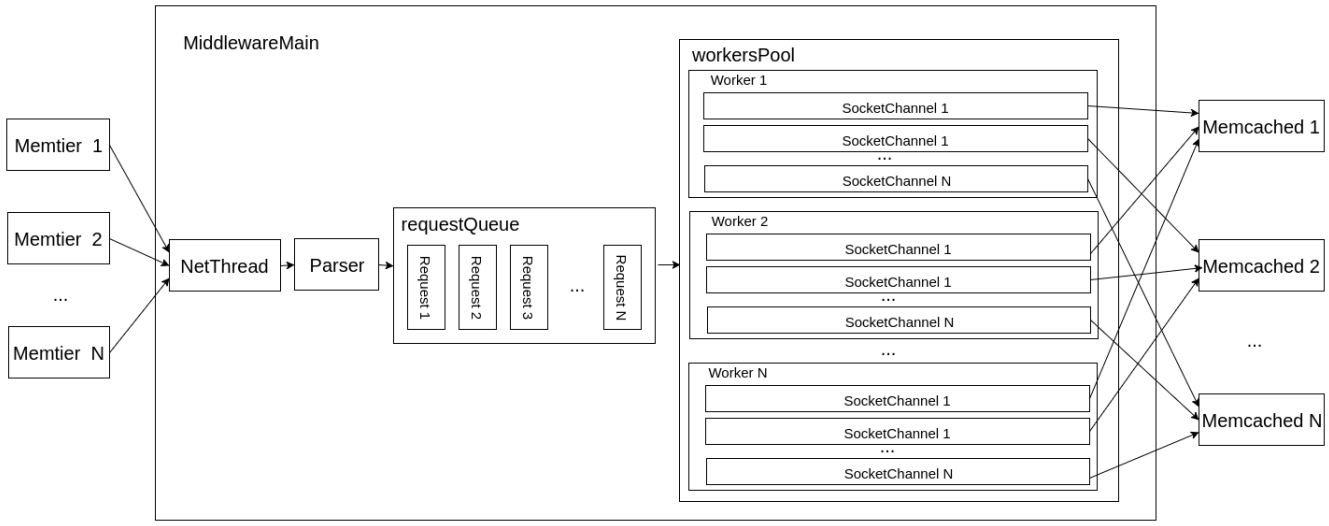


Figure 1: System's components

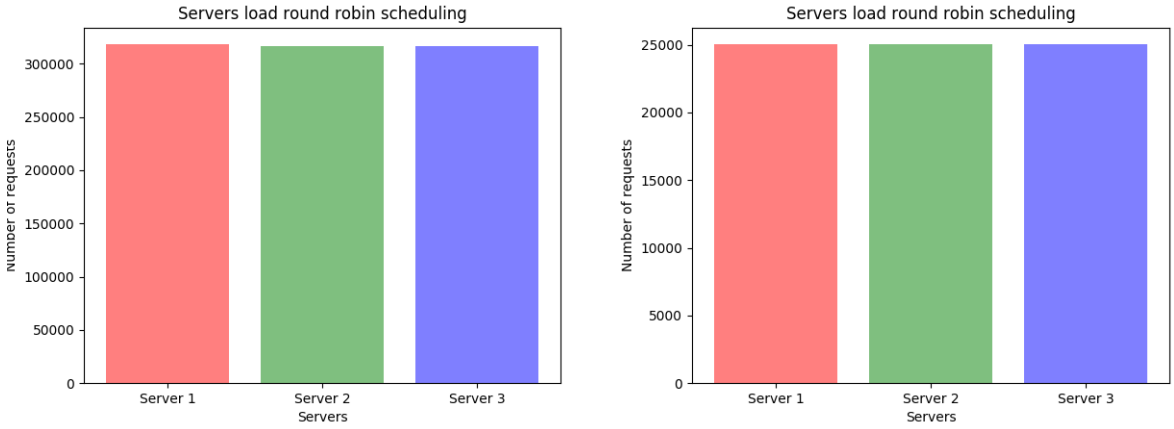
Based on the type of the Request, Workers send requests to servers:

- **SET**. In case of SET the data is replicated to all the memcached servers. In this case worker would make a blocking write thus waiting for all responses from all servers. Afterwards a worker parses all the response and checks for errors and if there is any it sends it back to client, in case there are no errors a single success message is selected and forwarded back to client.

- **GET**. In case of GET the request is forwarded to only a single memcached server, which is regulated by the Round Robin scheduling algorithm to distribute the load among the available servers. There is a single response message in this case which is forwarded to client. Figure 2a demonstrates equal round robin load scheduling while performing read-only workload.

- **Non-shared MULTI-GET** is similar in implementation to the GET, the only difference is that in non-sharded MULTI-GET multiple keys are requested/retrieved unlike a single key in GET, this method also relies on Round Robin scheduling as the GET one. The response from server is forwarded to client. Figure 2b demonstrates equal round robin load scheduling while performing read-only workload. The number of keys at the figure was

- **Sharded MULTI-GET** are split in multiple MULTI-GETS end sent to all servers. The objective of splitting is to split a large requests as equally as possible and provide the similar load to servers. After sending all the requests to server/servers, Workers wait until replies from all servers have been received. After reception of all replies, further processing and assembling is performed. As a result a single message is assembled where the ordering is the same as the client requested. Afterwards the data is finally being forwarded back to the client as a single message.



(a) Single GET load distribution

(b) Non sharded multi GET load distribution

Figure 2: Read only workload distribution

1.3 Instrumentation

Because of the multi-threaded nature of the application statistics are collected per Worker object. Concretely, each of Workers contains its own copy of the Statistics object as a field. The statistics are being gathered with the use of the Timer Task, which executes over a specified period of time. The statistics from this moment are being logged at the statistics.log. On middle-ware's shutdown there is a shutdown hook being executed, which aggregates all of the statistics from the log. Finally the aggregated statistics are printed to standard output stream. Statistics collected during instrumentation:

- Average throughput (ops/sec)
- Average response time (msec)
- Average queue length
- Average wait time in queue (msec)
- Average service time (msec)
- Number of SET
- Number of GET
- Number of MULTI GET
- Arrival rates
- Server loads
- Histogram response times (0.1 msec step)

For the modeling section the values of the arrival rates were heavily used. The arrival rates unlike other statistics are collected in the NetworkThreads. Each section there is a measurement of arriving jobs conducted. Afterwards the values are added to the lists. At the end of operation the middlewares outputs this list, from which the arrival rates for modeling are computed. The server load values for used to generate the histograms of equal server load distribution.

2 Baseline without Middleware (75 pts)

This section describes the baseline experiment with servers and client machines only and no middleware. The main purpose of the experiments is to demonstrates performance characteristics of the memtier clients and memcached servers before adding the middleware component.

The range of the victual clients is 1 to 33 with step 4. All experiments were repeated 3 times, each repletion was 1 minute long. The error bars represent the errors between the repetitions.

2.1 One Server

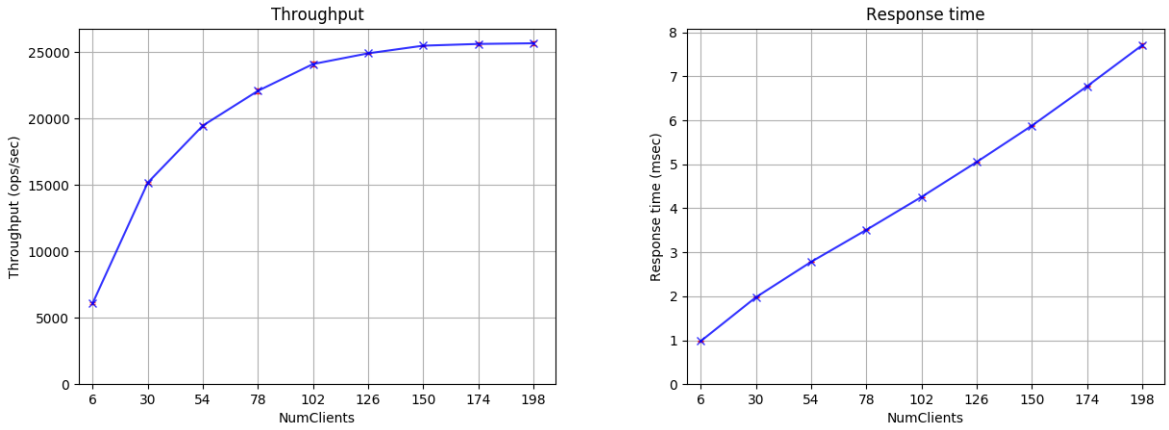
For both read-only and write-only workloads the throughput and the response time were measured as a function of NumClients and plotted later.

For this experiment 3 load generating machines of type A2 were used, containing one memtier (CT=2) each, the number of virtual clients (VC) per memtier was varied from 1 to 33 using step of 4 VC.

Ping analysis was performed before running the experiments to estimate the allocation of client machines in relation to server by the Azure environment. The client machines are located close to each other in terms of ping. However based on the ping we could deduce that first machine is located slightly closer to the server. That fact that machines are located closely to one another positively affects the experiment resulting in very low error in aggregation between the machines.

Client machine	Average ping time (msec)
tivanforaslvms1	1.187
tivanforaslvms2	1.389
tivanforaslvms3	1.356

After Performing write only workload the following figures for throughput and response time were acquired which are depicted at Figure 3.



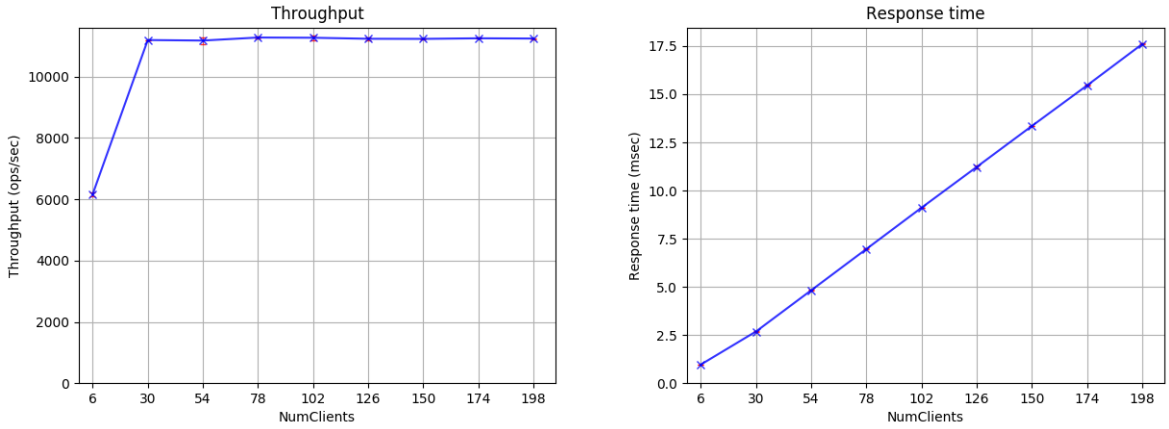
(a) SET only aggregated throughput

(b) SET only aggregated response time

Figure 3: Throughput and response time write only workload

From the figure we can see that the overall throughput grows with the number of clients and reaches the mark of over 25680 ops/sec. Before 150 NumClients clients mark server is under saturated. After that mark server is saturated, the throughput curve stays flat. The response time grows linearly with the increase of NumClients and peaks at 150 NumClients with the value of 7.7 msecs. Response time and throughput are positively correlated. Overall the figures dervived from the interactive law match the measured ones, for instance let us take the value of response time at 174 clients, which is 6.77. Using the interactive law $X = \frac{N}{R+T}$ we derive the value of 25701 ops/sec, which similar to the measured value of 25633 ops/sec.

Afterwards the GET only experiment was performed which depicted at Figure 4.



(a) GET only aggregated throughput

(b) GET only aggregated response time

Figure 4: Throughput and response time read only workload

The throughput rose rapidly after 30 clients. The server is saturated after that point. Overall throughput which reaches its maximum of 11277 ops/sec at 54. is less than in the SET only case. The response time which is 17 msec at 198 NumClients is significantly larger than in the SET only workload. The derived values for interactive law match the measurements, for instance 11255 ops/sec obtained from derivation versus 11250 ops/sec from measurements at the mark of 174 clients.

2.1.1 Explanation

After obtaining the results we could observe that the overall throughput of read only workload was significantly less than the throughput of the write only workload. The same tendency is observer with the response time where the performance of read only workload was significantly worse.

The difference between read only and write only results may be explained by the fact that the responses of read only workload's request are generally much larger in terms of size of the payload than during the write only workload. Therefore the throughput is less in the situation where the size of the data being manipulated and transfered is larger, which causes overhead for the network. Response time also increases if the size of the data transfered increases.

Relatively low number of the throughput of read only workload in the first configuration could be explained by looking at the network load at the second peak at Figure 7. The network load grows very rapidly compared to other experiments and stays flat until the end of the experiment. This issue results in a plot which also reaches maximum throughput rapidly and stays flat for the most part.

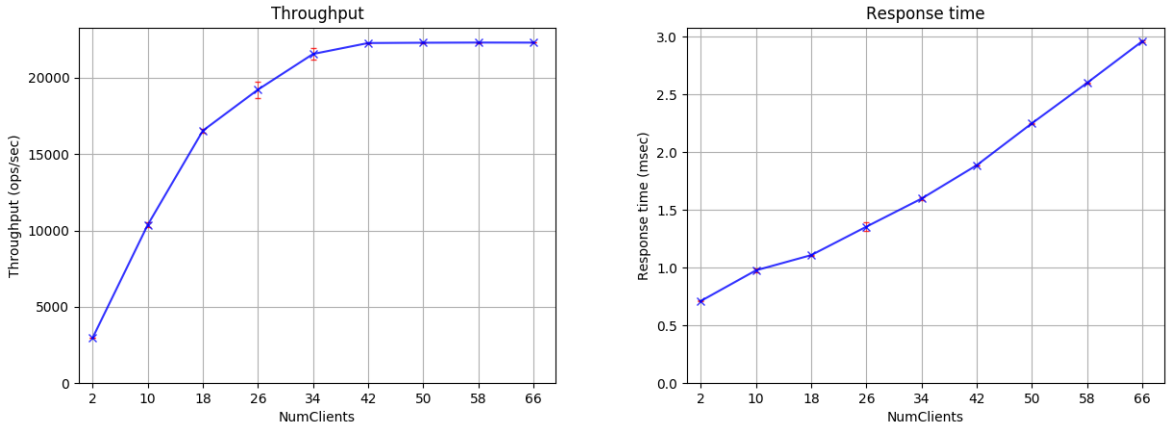
The value from 30 clients does not seem to grow more because it is basically limited by the network factor, this is supported by the fact that network plateaued at the value and did not grow at all (Figure 7, second peek corresponds to the current experiment). This behavior could be explained by Azures limitations of output network speed.

2.2 Two Servers

For the read-only and write-only workloads the throughput and response time were measured as a function of NumClients and plotted later.

For this experiment 1 machine of type A2 was used, with one memtier (CT=1) connected to each memcached instance (two memcache instances in total). Number of virtual clients (VC) per memtier thread was varied from 1 to 33 with step 4.

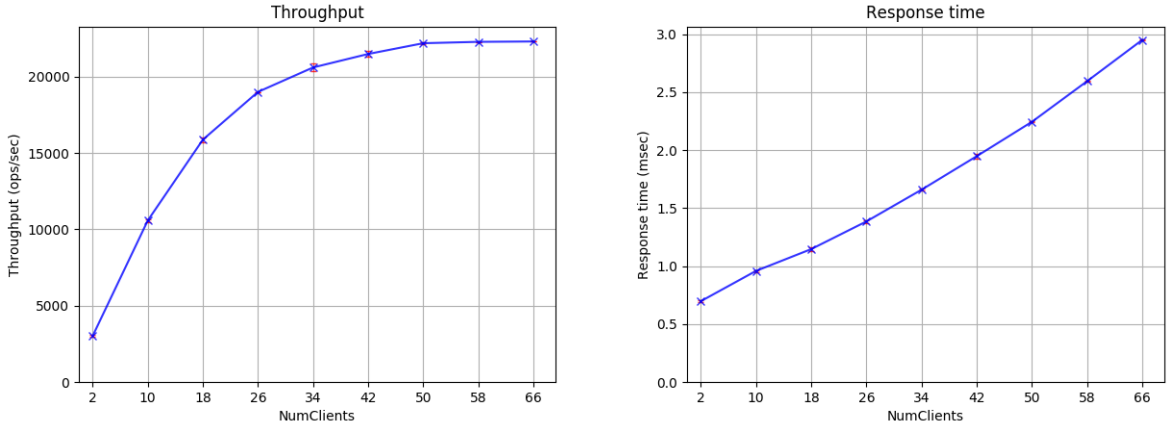
The results of write only experiment are depicted at Figure 5



(a) SET only aggregated throughput (b) SET only aggregated response time

Figure 5: Throughput and response time write only workload

With two client machines the maximum throughput gets to the mark of 22300 ops/sec. At 42 clients the servers begin to saturate, after that points throughput does not increase and the curve stays flat. The repose time peaks at 3 msecs at 66 NumClients. One could notice more rapid form of growth of response time compared to the previous configuration. The error bars have slightly increased compared to the previous experiment, however overall the error still remains rather minor.



(a) GET only aggregated throughput (b) GET only aggregated response time

Figure 6: Throughput and response time read only workload

Afterwards the read only workload has been performed which is illustrated on Figure 6. Overall the results are similar to the write only workload. The maximum value of throughput is reached at 66 NumClients at the 22305 opes/sec mark, the response time peaks at 66 NumClients with the value of 3 msecs. From the value of 50 NumClients the curve stays flat meaning that

the severs are saturated. The error bars are less than in write only workload.

2.2.1 Explanation

Comparing the results to the previous section's experiment the maximum throughput at write only workload is slightly lower than in the previous configuration which had more client machines but with only one memtier instance. In contrast the throughput for read only workload is larger in current experiment compared to the previous section.

When it comes to write only workloads of both configurations the saturation point for write only workload at the configurations with 2 memtier instances and one machine happened much earlier than at the one with 3 machines but 1 memtier instance. The reason for this is the number of servers and the number of client machines. At configuration number one the requests are sent to only one single server, however they are sent from completely different machines with different ping times to the servers, which means there would be network delays because of all the traffic, this allows even one memcached server to process jobs more effectively and not saturated so fast, concretely to process some jobs sent from some machine while the jobs from the other machine are still being sent and likely delayed at the network.

In contrast to second configuration the jobs are sent from two memtier instances but from the same machine, which means ping times are the same. The jobs are not slowed down by the network as much as during the previous case, although there are even 2 servers now.

When it comes to read only workload, as opposed to the read only experiment for the previous section, current configuration grows smoother. The saturation points occurs later. This behavior could be explained by analyzing the network load (figure 7), concretely one could see that the network load does grow smoother as opposed to rapid increase as in the previous configuration.

2.3 Summary

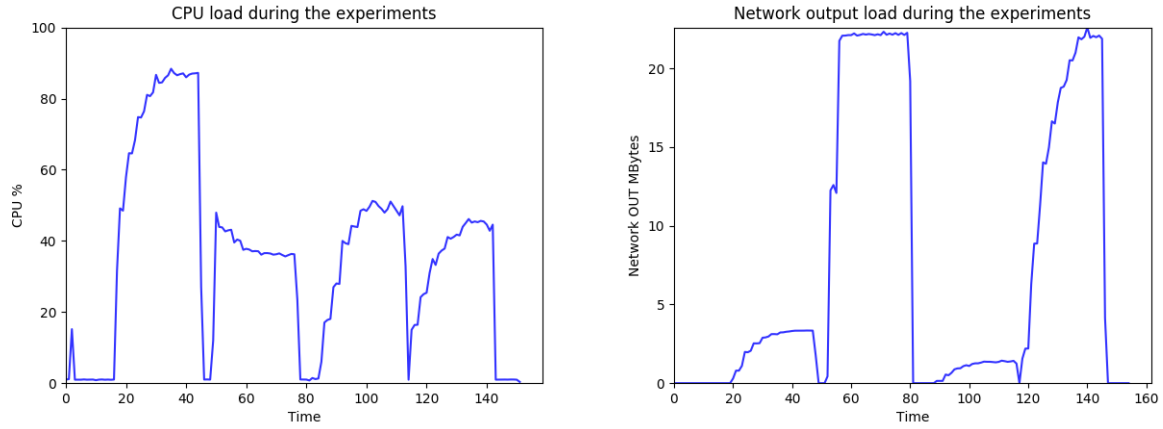
Based on the experiments conducted above the following table was acquired

Maximum throughput of different VMs.

	Read-only workload	Write-only workload	Configuration gives max. throughput
One memcached server	11276	25683	Reads 78 NumClients, writes 198 NumClients
One load generating VM	22302	22306	Reads 66 NumClients, writes 58 NumClients

In terms of write only workload the configuration which gave the maximum throughput is the configuration from the first experiment, specifically the one with 1 server, three machines, 1 memtier instance with 2 threads, however this figure was recorded at the larger number of NumClients point. When it comes to read only workload the second configuration with even smaller value of NumClients demonstrated significantly better results of throughput compared to the first configuration.

We could observe than the difference of throughput figures of write only experiment is significantly less than the difference of figures for read-only workload. This behavior is explained by the fact that read only workload puts a lot of pressure on the network out at the server side (Figure 7), because of this there was some limitation from network side which did not allow throughput to reach higher mark. In contrast, write only workload does not put a lot of pressure on the server network output.



(a) CPU load

(b) Network out load

Figure 7: Load of the components for baseline without middleware

To better interpret the final results one should consider the component's utilization. The CPU and Network load are plotted on Figure 7 which is based on the metrics provided by Microsoft Azure Metrics capabilities. At both graphs one could see four peaks which represent our four experiments: SETs with the first configuration, GETs with the first configuration, SETs with the second configuration, GETs with the second configuration. In general GET only workloads for both configurations were heavier on the network than the experiments for SET only workload.

Overall there is very little of an error and noise because the results across the repetitions and machines vary insignificantly, because the experiment was performed during nighttime when Azure environment produces the least noise.

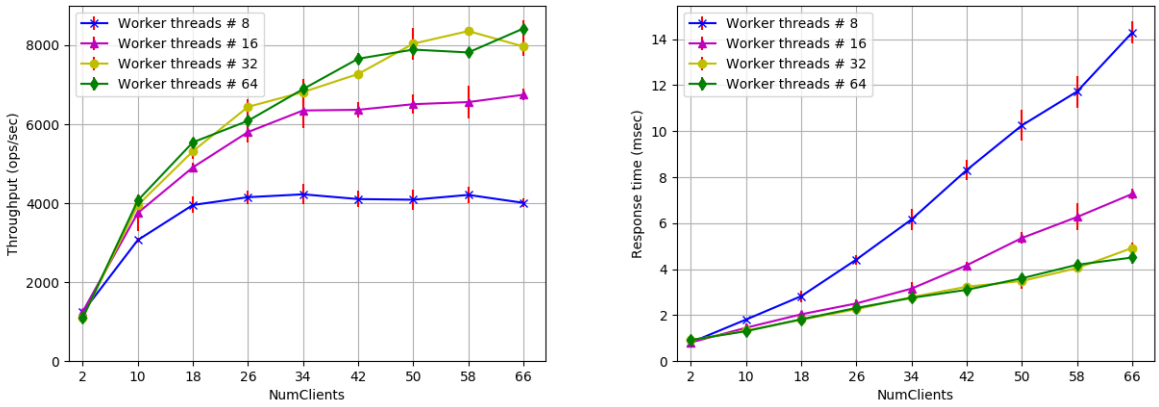
3 Baseline with Middleware (90 pts)

This section describes the set of experiments where 1 load generator A2 and 1 server A1 were used. The throughput and response times where measured as a function of the number of clients. Scaling of the virtual clients is performed as in the last sections. Provided plots of both throughput and response time are based on the data measured on the middleware through instrumentation. All experiments were repeated 3 times, each repletion was 1 minute long. The error bars represent the errors between the repetitions.

3.1 One Middleware

This configuration used one load generator A2 (one instance of memtier with CT=2), single middleware A4 and 1 server A1. The read-only and a write-only workload were performed with varying VC setting from 1 to 33 with step 4.

The experiment was repeated for all worker threads configurations.

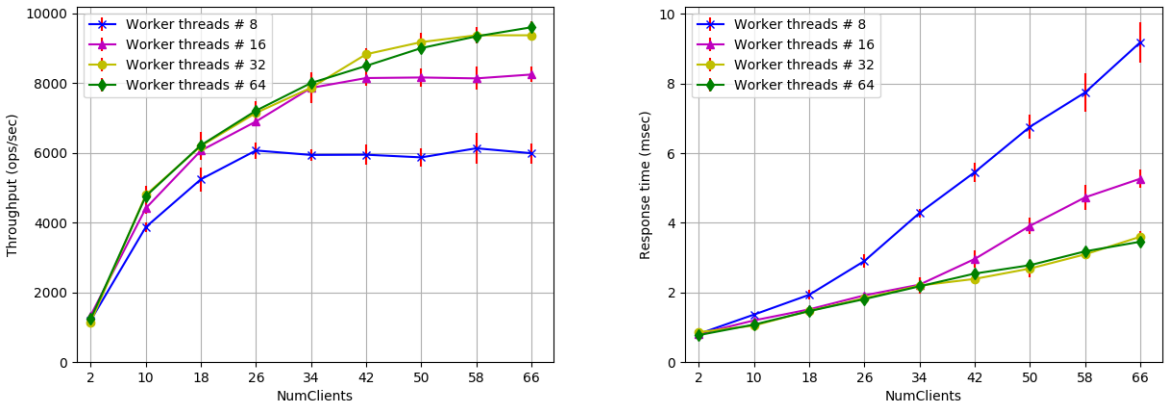


(a) SET only aggregated throughput (b) SET only aggregated response time

Figure 8: Throughput and response time, one middleware configuration

Figure 8 demonstrates the results of write only workload with single middleware. For all 4 worker threads configurations the throughput as well as the response time increase. The configuration with 8 worker threads saturates at 18 clients, sooner than all other configurations. The configurations with 32 and 64 worker threads deliver the best performance in terms of throughput, however the configuration with 64 worker threads demonstrates better performance in terms of response time.

When it comes to interactive law verification we could compute the value of response time using the equation $X = \frac{N}{R+T}$. Let us take point of 66 clients, 16 worker threads. We have 7.27 msec for response time and 6746.67 ops/sec for throughout. Using the equation we derive the value of 6877 ops/sec. In general the derived values matched the measurements, higher values of derived figures are explained by zero think time assumption.



(a) GET only aggregated throughput (b) GET only aggregated response time

Figure 9: Throughput and response time, one middleware configuration

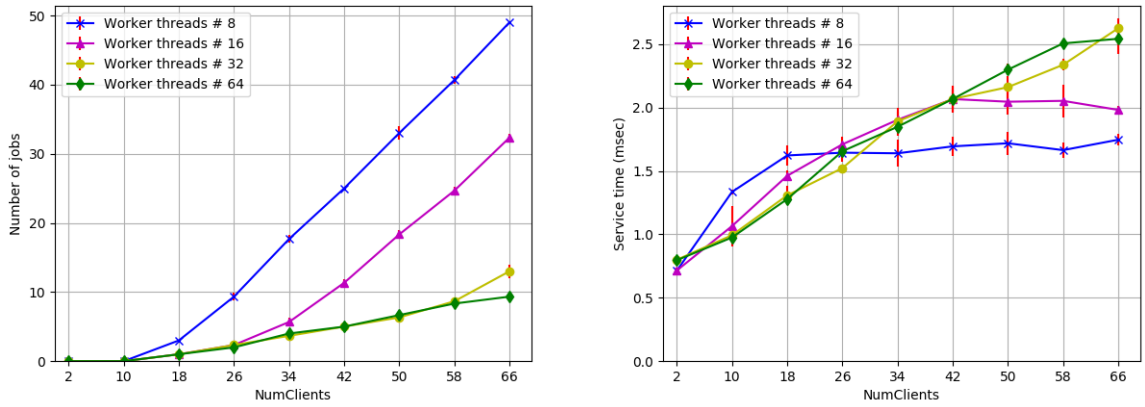
Followed by the write only experiment, the read only load experiment was performed. From Figure 9 experiment one could notice that 64 worker threads configuration demonstrated the best performance both in terms of throughput and response time. The configuration peaks at 66 clients. On the other hand the configuration with 8 threads showed the worst performance

both in regards of throughput and response time alike. The configuration is saturated after 18 clients the throughput curve stays flat and the response time graph has a knee at the same point.

We can observe that the interactive holds. Let us take 50 clients for 8 threads. The derived value is 5442 ops/sec, the measured one is 5987.33. In general the values are close, the discrepancy is explained by the zero think time assumption. The discrepancy for more worker threads configurations is larger because the think time increases with more worker threads, however we still assume think time as constant, specifically zero think time assumption.

3.1.1 Explanation

After analyzing the results of the experiment, one could state that the system demonstrates better performance with the rise of worker threads number. Comparing the write only and read only workload, the latter one demonstrated higher overall throughput for all worker threads configurations and lower response times as well.

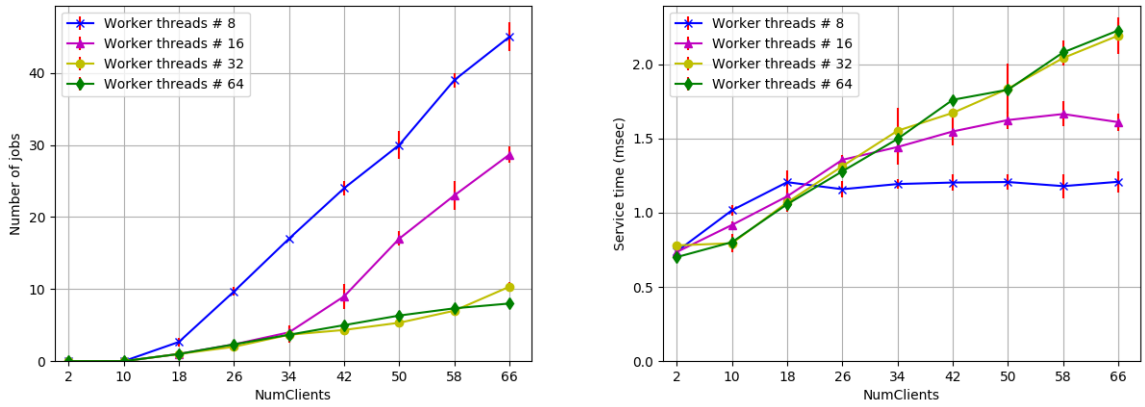


(a) Queue length

(b) Service time

Figure 10: Measurements of internal system components, single middleware, write only workload

During the write only experiment the throughput at 64 worker thread configuration kept rising even after the mark of 66 clients, therefore the saturation point is not visible in this range, and it likely appears at the higher mark. The absence of saturation point could be confirmed by the queues length values displayed at Figure 10. Even at the mark of 66 clients no knee was observed. The saturation point of 32 worker threads appears to be at 58 clients, we can observe the knee at the response time graph as well as at the queue length graphs. The saturation point for 16 worker threads is located at the mark of 34 clients, we can see that plot for throughput plateaus at that point, the response time has a knee at the point, so does the queuing time. The configurations with 8 worker threads saturated as the earliest, specifically the mark of 18 clients is the saturation point, we can see that throughput stayed flat at that point, response time plot contains a knee and the queue lengths changed the order of growth at that point.



(a) Queue length

(b) Service time

Figure 11: Measurements of internal system components, single middleware, read only workload

When it comes to read only workload the two best configurations were the ones with 32 and 64 clients. The saturations point for 32 workers threads appears at the mark of 58 clients. The saturation point for 64 workers threads is again not observed in the range. From the throughput plot 9 we see that both plateau after that point of 58 clients, the response times rise up as well as the queue sizes. The configuration with 16 threads saturate significantly earlier at the mark of 34 clients. Finally, the configuration with 8 threads saturated at the same point as during the write only workload, specifically at 18 clients.

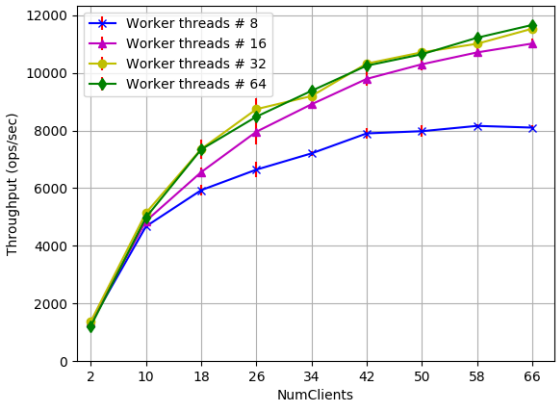
Speaking of internal components, the overall trend is that with the increase of worker threads queue length become less. The order of queue growth is more rapid with lower number of workers. This has an effect on saturation points, concretely the less worker threads are in middleware the earlier the saturation happens. Another important observation is that more worker threads cause service times to go up, which is explained by the fact that more worker threads means more pressure for the servers. The order of growth increases on curves with more worker threads.

3.2 Two Middlewares

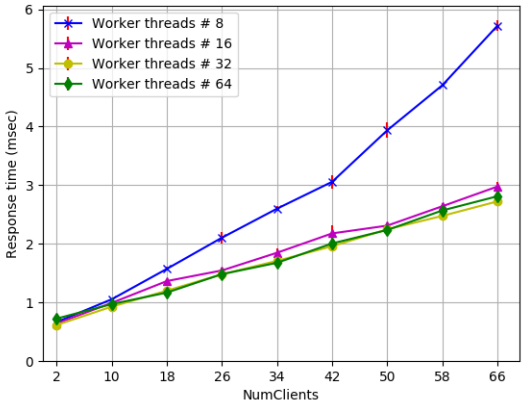
This configuration used one load generator A2 (two instances of memtier with CT=1), two middlewares A4 and 1 server A1. The read-only and a write-only workload were performed with varying VC setting from 1 to 33 with step 4.

The experiment was repeated for all worker threads configurations.

Figure 12 illustrates the result of the write only experiment with two middlewares. The configuration with 64 and 32 worker threads deliver the best performance both in terms of throughput and response time, in contrast the configuration with 8 delivers the worst performance. We can observe the same trend that some curves do not plateau in the proposed range as well as that some response time curves do not contain knees.

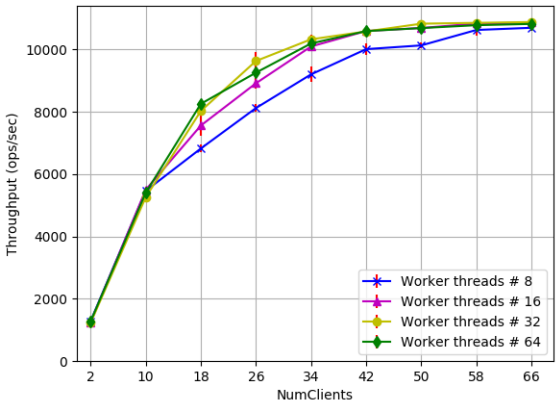


teu (a) SET only aggregated throughput

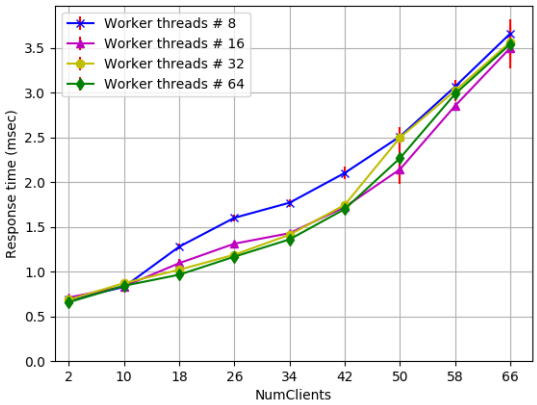


(b) SET only aggregated response time

Figure 12: Throughput and response time, two middlewares configuration



(a) GET only aggregated throughput



(b) GET only aggregated response time

Figure 13: Throughput and response time, two middlewares configuration

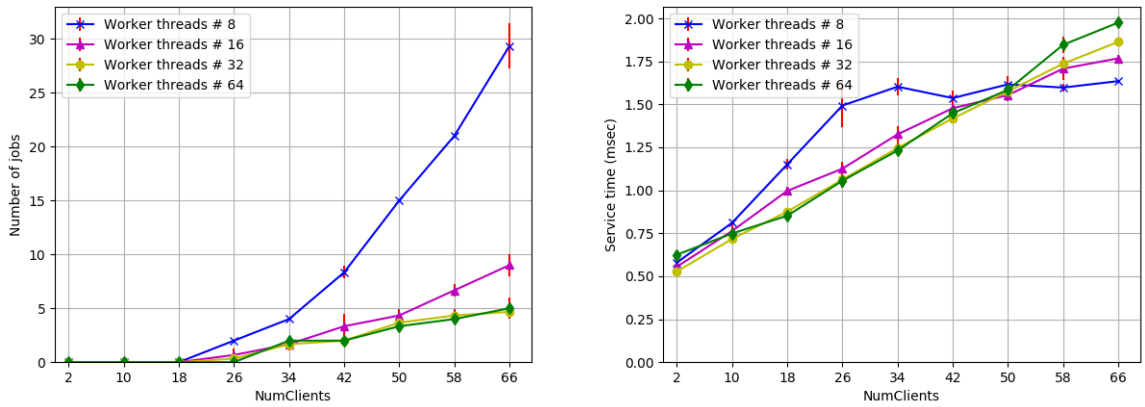
Figure 13 shows the result of the read only experiment with two middlewares. The throughput and response times curves of different configurations are located closer to each other than compared to the previous experiment. Another thing to notice is that the throughput curves look more saturated compared to the write only workload. This behavior is going to be explained in the later section.

3.2.1 Explanation

From the write only experiment we can see that the configuration that achieves the earliest saturation is the one with 8 worker threads. This happens at the mark of 42 clients, much later than during the previous section. We can see the plateau of throughput after that point as well we can see the knee at the response time at the mark of 42 clients (Figure 12), queues start to grow rapidly after 42.

However if one would consider other worker threads configuration with more threads, one could see the trend that there are no signs of reaching of saturation, especially at configurations

of 32 and 64 threads. Furthermore, there is an interesting tendency of decrease of service times with the increase of worker threads. The queue lengths are fairly small even at large numbers of clients for the configurations with 32 and 64 worker threads. This kind of behavior differs from the behavior of previous configuration. This is a strong indication that middle-ware is not the bottleneck in this setting for 16, 32 and 64 threads. Therefore we need to add the second client to explore the performance in more detail.



(a) Queue length

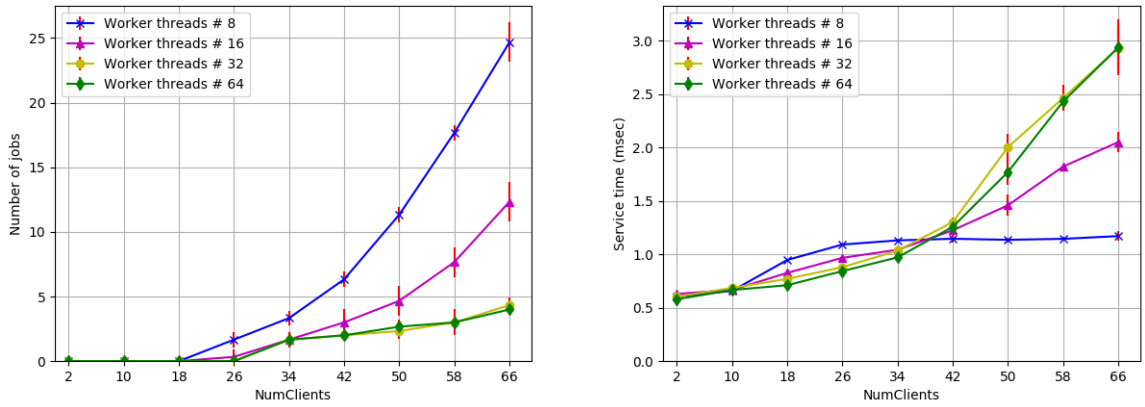
(b) Service time

Figure 14: Internal system components, double middleware, write only workload

After performing the experiment for the read only workload there was a tendency observed that the saturation is more distinct compared to the write only workload. We can see that the throughput curves for all worker thread configurations plateau, which is an indication that an increase in worker threads does not introduce a significant speedup as it was during the write only workload.

We can see that for the configuration with 8 threads, which was still the fastest to saturate, the saturation point lies at 34 clients, because throughput plateaus after that point and there are knees at response time plots and at the queues lengths plots as well. Figure 15 demonstrates the state of components. Notably, for this configuration we can see that service time did not increase much with the rise of clients which means middlewares was actually the bottleneck. In contrasts, the configuration with more threads had almost similar performance as we have seen it before, we can see that the service times went up with the increase of clients, however the queues sizes were relatively low. This indicates that memcached servers were the bottlenecks, at configurations with 32 and 64 threads. With 16 threads we observer similar behavior two both highest and lowest threads configuration, which indicates that there is a mixture of bottleneck from middleware and servers for that configuration.

Different kind of behavior of the experiment in this section is explained by the fact that we use two middlewares compared to the single one in the previous section. Double middleware means more capacity for the arriving jobs because there two queues in this setting. This means the queuing size would decrease compared to the case with one middleware. Another point to consider is the fact that two middlewares put significantly more load on the servers, compared to the single middleware setting.

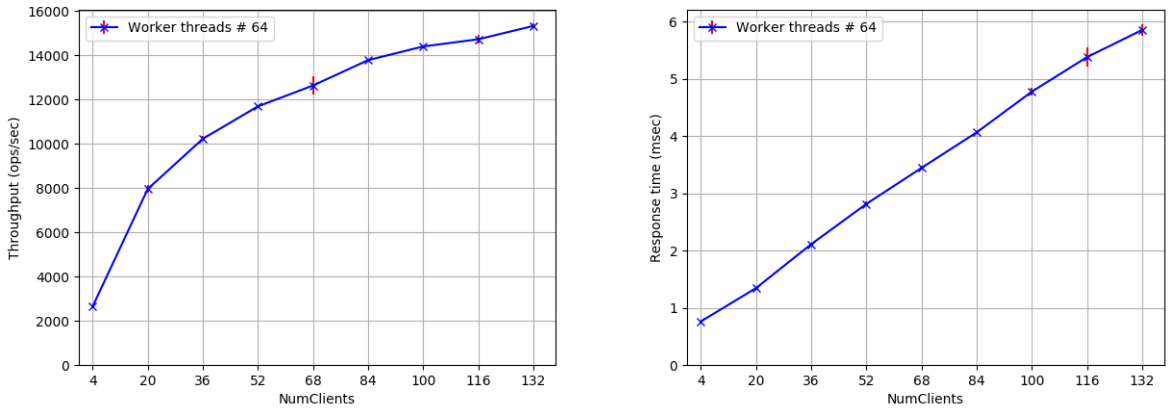


(a) Queue length (b) Service time

Figure 15: Internal system components, double middleware, read only workload

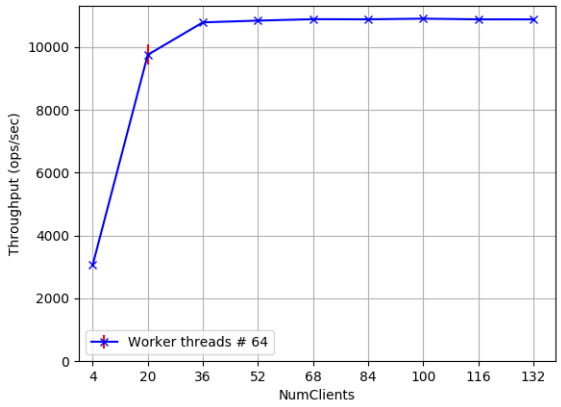
Overall, the experiments demonstrated that the system is not saturated, as it was previously mentioned. Therefore it makes sense to increase the load by performing and additional experiments with client machines increased to two instances of A2.

Figure 16 demonstrates the behavior of the system during the write only workload. Expectedly, with more clients the overall throughput rises and reaches the mark over 16000 req/sec. However the throughput still keeps rising and one could not observe the saturation point in the range.

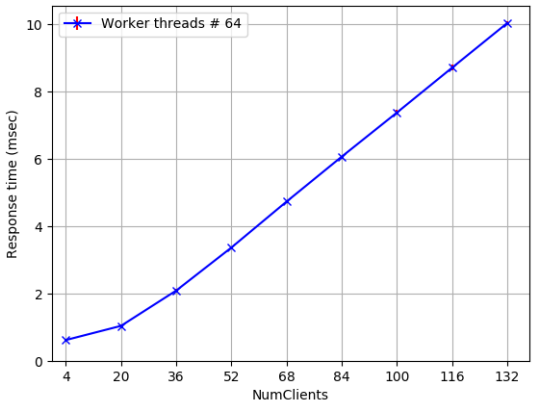


(a) SET only aggregated throughput (b) SET only aggregated response time

Figure 16: Expanded experiment, double middleware configuration, read only workload



(a) GET only aggregated throughput



(b) GET only aggregated response time

Figure 17: Expanded experiment, double middleware configuration, read only workload, additional client machine

Figure 17 shows the results of the system during the read only workload. The throughput only reaches 11000 at this time because the proper population has been performed. The system is saturated after 36 clients, the throughput curve is flat after that mark and there is a knee at the response time plot. The behavior of the curve reminds us about the trend in read only baseline experiment, where the network limited the growth.

3.3 Summary

Based on the experiments data the following table was produced:

Maximum throughput for one middleware.

	Throughput (ops/sec)	Response time (msec)	Average time in queue (msec)	Miss rate
Reads: Measured on middleware	9596.67	3.46	1.01	0.00 %
Reads: Measured on clients	9788.97	6.73	n/a	0.00 %
Writes: Measured on middleware	8424.67	4.51	1.25	n/a
Writes: Measured on clients	8596.35	7.72	n/a	n/a

Maximum throughput for two middlewares.

	Throughput (ops/sec)	Response time (msec)	Average time in queue (msec)	Miss rate
Reads: Measured on middleware	10882.66	3.56	0.55	0.00 %
Reads: Measured on clients	11093.17	5.96	n/a	0.00 %
Writes: Measured on middleware	11661.66	2.81	0.60	n/a
Writes: Measured on clients	11871.52	5.55	n/a	n/a

Overall, while comparing the values of throughput for the one and two middleware configurations one could notice a slight discrepancy between the values measured on the client and

the middleware. This can be explained by the fact that the middlewares are started slightly earlier, which results in zero requests measured, which in return results in slightly lower value of throughput. Response time measured on the middleware is lower that the one measured on the client due o the fact that the response time is measured as the difference of time arriving to the system and time when the job left the system, without taking in regard the network transfer time which is impossible to measure on the middleware. This results in larger response time values when measuring on clients.

The single middleware configuration gives larger figures in terms of both throughput and response time for read only workload than for write only workload. In contrast to the experiment without middle-ware where there was a significant difference of throughput figures of two type of workloads for the one server configuration and very similar figures with one load generator configuration. This is explained by the introduction of middleware, which balances the performance of different types of workloads. The overall throughput figures are lower than in the case with no middleware, because the bottleneck in form of middleware was introduced. The read workload gave better throughput and response time results, with less queuing in the system.

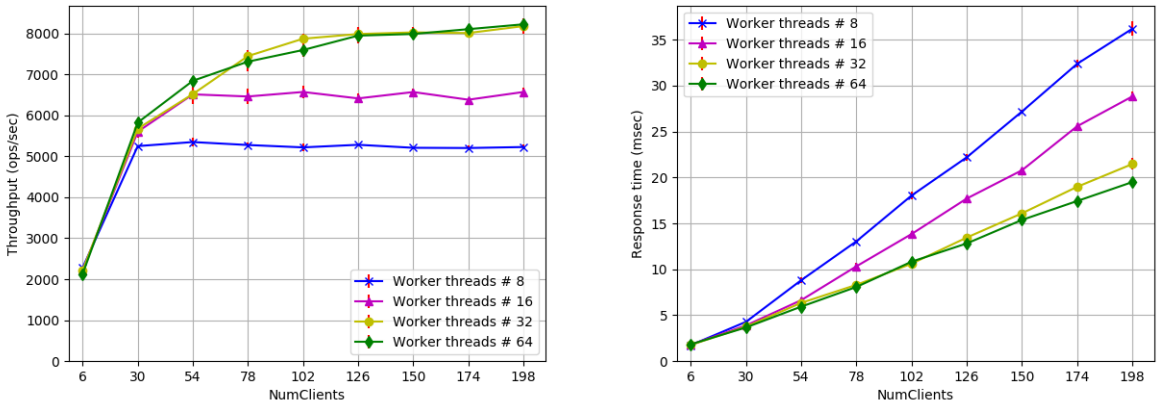
The double middleware configuration produced higher throughput figures and lower response time figures in overall results. This could be explained by the fact of introduction of the second middleware component, which doubled the amount of parallelism, which in return caused more processed jobs going through the system, which caused throughput figures to rise and response times to lower. The queuing time has also decreased compared to previous configuration because there are now more worker threads which can take the jobs from the queue, which obviously reduces both the times in queue.

The overall trend is that the difference of performance between read only and write only workload has decreased as compared to the experiment without the middleware, which is as mentioned earlier can be explain by introduction of middleware. In general, the throughput is lower than in the experiments prior to middleware introduction, which explained by the fact that all requests have to go through middleware, which slows down the response time (the job has to wait in queues, limited amount of parallelism etc.).

4 Throughput for Writes (90 pts)

4.1 Full System

The purpose of this section is to analyze the performance of the full system using write only workload. Three client machines of type A2 were connected to two middlewares of type A4, which were connected with three servers of type A1. All experiments were repeated 3 times, each repletion was 1 minute long. The error bars represent the errors between the repetitions.



(a) Aggregated throughput

(b) Aggregated response time

Figure 18: Throughput and response time, full system, throughput for writes experiment

Figure 18 demonstrates the throughput and response time measurements after performing the throughput for writes experiment.

Overall, the tendency is that with the increase of number of workers threads the throughput increases and the response time decreases. The characteristics of both throughput and response time do not vary significantly no matter of configuration before the reach mark of 30 clients. After that point the configurations which offer more parallelism start to deliver higher throughput. After the mark of 54 clients configurations with 32 and 64 threads start to deliver best performance and at the mark of 126 clients the configuration with the most threads outperforms the other configurations.

The configuration with 8 worker threads demonstrated the worst performance both in terms of throughput and response time. The system begins to saturate at the mark of 30 clients , the curve stays flat and does not increase after this value, one could also notice a knee at the same mark ob the response time.

The configuration with 16 worker threads follows the same pattern as the previous configuration. The system starts to saturate at the mark of 54 clients, the throughput remains the same after this point, there is a jump on response plot at the same mark.

The setting with 32 worker has shown significantly better performance after 54 clients, outperforming the previous configurations. The system begins to saturate at 102 clients. Similarly, we observe the same kind of growth with the 64 workers threads configuration. The configurations outperforms the 32 threads setting, delivering very similar figures in terms of throughput, however with the noticeably better results in response time, visible from the mark of 102 clients.

4.1.1 Explanation

To better understand the results obtained in the previous subsection, let us consider other characteristics rather than response time and throughput. Figure 19 contains the sizes of the queue and service times, both plotted as a function of number of clients.

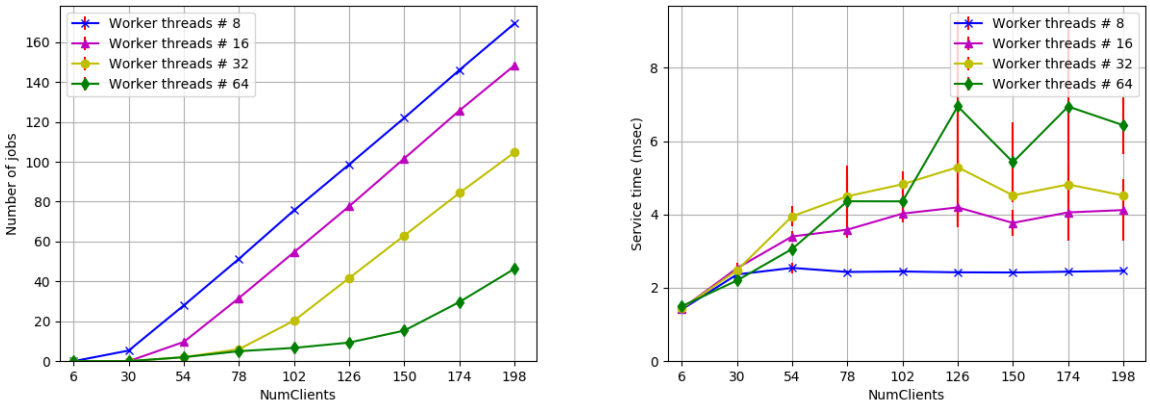
The throughput figures are lower and the response time is higher compared during the baseline experiment for write only workload, this is due to the fact that we use three servers now. Larger number of servers increases the service time as it could be seen from the service time plot, this is because the data has to be replicated to all three servers during the write only workload, which increases the service time, which in respect will increase the response time and decrease the throughput.

From the service time plot, one could notice that service times are correlated with the number of workers, the more workers threads are in the system, the higher the service times are. This tendency can be explained by the fact that the degree of parallelism rises with the increase of worker of threads. More threads means more generated load for the memcached servers. More jobs the servers obtain, the more time it takes for the server to respond to the middleware, which causes the service time to increase. Noticeably, the noise increases with the number of worker threads increased, this could be explained by the fact that the more worker threads one has the more measurement discrepancies are introduced.

When it comes to queue length, the number of jobs in the queue curves are negatively correlated with the number of worker threads. With the increase of number of worker threads in the middleware, the number of jobs in the queue drops down. Namely at 8 worker threads the number of jobs in the queue goes over 160, in contrast the configuration with the most amount of threads only goes over 40 jobs in the queue as well as it does maintain the lower order of growth.

The behavior of the queues is correlated with the saturation points. In other words, there is a knee visible at the curves of the queue length plot. For instance at the saturation point of 8 worker threads configuration, one could observe that the order of growth of the queue length changes rapidly at the mark of 30 clients. The same behavior is observed at 16 clients configuration, at the saturation point of 54, there is a knee observable on the plot. The same behavior is observed at other configurations such as 32 and 64 worker threads configurations, the jump for 32 threads as at 78, and for 64 threads it is located at 150 clients. Noticeably the order of function growth becomes the same after each curve goes over its saturation mark.

Overall, we can see that queues sizes keep increasing rapidly with more additional clients. The service times increase with more workers. This is an indication that middleware is limiting the throughput growth, this behavior was already observed during baseline with single middleware experiment.



(a) Queue length

(b) Service time

Figure 19: Measurements of internal system components

4.1.2 Determining the best configuration

Previously we have already discovered the performance of the system with 1 client and 1 server and had an opportunity to reason about the best configurations. Having performed the experiment with the full system one could select the configuration of worker threads which delivers

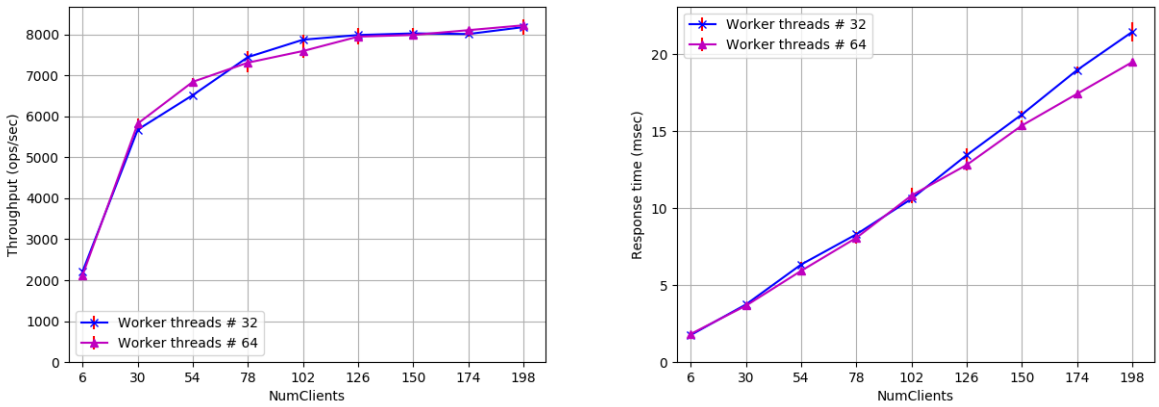
the highest throughput. The configuration selected in this section would be used in further experiments as well as in the section on modeling.

From the previous section we could observe that there are two configurations which have performed significantly better than the others. These two configurations are 32 and 64 worker threads (Figure 20). Before he mark of 30 clients these two configurations perform with no difference in regards with both throughput and response time.

At the period from 102 to 150 clients the configuration with 32 threads actually performs better than the configuration with more threads, this could be explained by the fact that at this number of clients more threads cause overhead, rather than benefit and make the computation faster. In other words the number of threads can be redundant in some cases, so the there is so called competition between the threads instead of expected performance speedup.

Passing the mark of 126 clients two configurations give almost identical results in terms of throughput. However, after considering the graphs for the response time, one could conclude that the configuration with 64 workers threads delivers similar throughput but at better numbers in terms of response time. Two graphs of response time start to diverge at the point of 102, the line for 32 worker threads starts to change the order of growth, however the version with 64 worker threads remains the same order of growth, because of more parallelism available. Overall, the configuration with 64 worker threads provides better results in terms of response on the whole range of clients.

All things considered, the configuration with the most amount of worker threads namely the configuration with 64 threads has provided the best overall performance. Thus the configuration with 64 worker threads would be interchangeably used as the maximum throughput configuration. Further experiments as well the section on modeling will fix the number of threads at 64.



(a) Throughput

(b) Response time

Figure 20: Best configuration selection

4.2 Summary

After conducting the experiments described in the previous sections the following table was constructed based on the data acquired. The entries correspond to the maximum throughput point for each corresponding configuration.

Maximum throughput for the full system

	WT=8	WT=16	WT=32	WT=64
Throughput (Middleware) (ops/sec)	5347.67	6569.67	8177.33	8221.00
Throughput (Derived from MW response time) (ops/sec)	6125.82	7367.19	9222.84	10152.23
Throughput (Client) (ops/sec)	5423.33	6598.37	8053.64	8204.04
Average time in queue (msec)	9.93	14.98	19.70	7.46
Average length of queue	14.00	27.33	52.33	23.17
Average time waiting for memcached (msec)	2.54	4.02	4.52	6.43

Overall, there is a tendency of throughput increases between maximum throughput points of all configuration, when measured on the middleware. The same tendency is observed when performing the same measurements on the client. There is a slight discrepancy between throughput results measured on the middleware and the throughput measured on the client. The figures on the middleware are slightly lower for all configurations than the ones from the client, this behavior could be explained by the fact that the middleware has been started slightly earlier before memtier clients. Because of the earlier start middleware started recording the arrivals of the jobs before any actual jobs were generated, that obviously caused zero jobs arrival measured at the statistics logs, which later were used for plotting as well as for the content of the tables.

The second entry of the table, which is throughput derived from the response time measured at the middleware used the interactive law equation $X = \frac{N}{R+Z}$. The overall trend continues to hold, the throughput increases with the increase of worker threads. In general, the figures are higher than throughput measured at the client and directly at middleware. This could be explained by the fact that the response time measured on the middleware is not the equal values to the response time measured on the client. The way the response time measured on the middleware is time when the job was received from the socket minus the time the job was sent back to the client. This value would always inevitably be less than the value measured on the client, because it does not take in regards the network transfer time, which is never a constant and varies a lot because of the network environment factors. Considering the equation, because of the response time values being lower, the throughput becomes larger compared to the ones measured on the client and directly on the middleware.

Another point to consider is the tendency of increasing discrepancies in the throughput values measurements between the throughput derived from the response time and the values measured directly. In other words, the more worker threads we introduce the more difference we obtain in the derived values and the value measured directly. This could be explained by the fact that with the increase of number of worker threads more network overhead arises, because obviously more threads generate greater load. Due to more threads it takes more time to transfer the response back to client. The so called service times increase as it was showed at Figure 19. All things considered, due to the reason that response time measured on the middleware does not include this transfer time, as it was described earlier, the difference between directly measured throughput and throughput derived from response time rises.

Noticeably, the time in queue in the table does increase until the 32 worker threads configuration and reaches the minimum at 64 worker threads. This behavior is explained by the reason that maximum throughput configurations do not happen at the same mark of clients. Namely the maximum throughput for 32 and 64 worker threads happens at 33 clients, in contrast to 16 worker threads where the maximum is at the mark of 17 clients as well in the case with 8 worker threads where the maximum is observed at 9 clients. The time spent in queue is correlated with the increase of of virtual clients.

Considered this information, it particularly makes sense to look at the configurations of 32 and 64 threads which have the maximum at the same point at 33 clients. The queuing time

for the 64 workers is 2.64 times less, which is explained by the fact that there are more threads performing the tasks, which means there are more jobs that could be taken from the queue. This causes the job to spend less time in the queue on average. The value of time in queue for 8 workers is less than with the configuration with 16 workers because there are only 9 clients compared to 17 clients at the corresponding maximums. Less clients means there would be less jobs generates, which in return means the queues would be shorter and the average queue time would be less. The same pattern is observed while comparing 16 and 32 threads configurations, because the maximum throughput client number is 17 and 33 respectively.

The same observation is noticeable in the queue length values as it was the case with queuing time. These two metrics are correlated because less jobs in the queue causes the requests to be dequeued earlier and spend less time in the queue. This is again explained by the fact that not all maximum throughput configurations were reached at the same mark of clients. In general, less clients will inevitably mean shorter queues and less of queuing time.

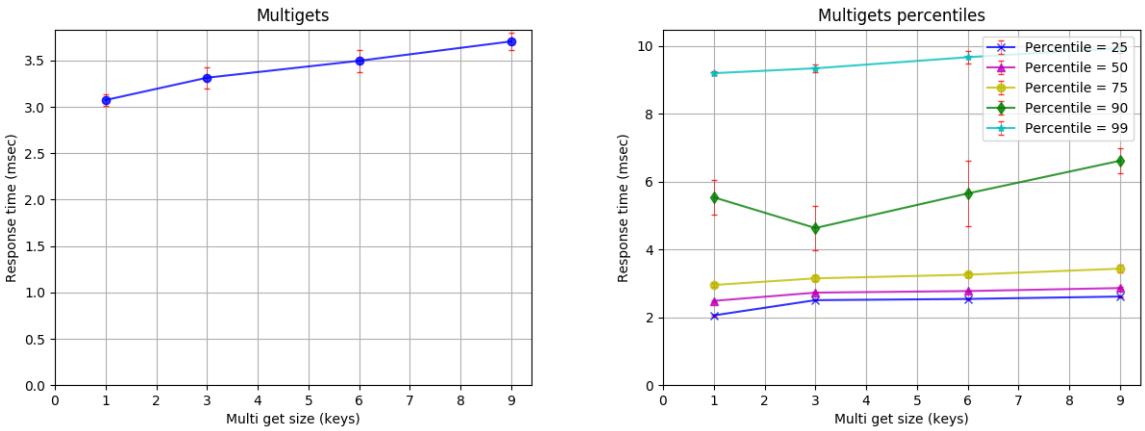
5 Gets and Multi-gets (90 pts)

The purpose of this section is to explore the behavior of the system during the multi-get workload. Further the behavior of two modes, namely sharded and non sharded were compared and analyzed together. For these experiments there were used three clients of type A2, two A4 for middlewares and three A1 for servers. All of the experiments were conducted using the maximum throughput configuration, which is 64 worker threads as determined earlier.

It is important to mention that the number of keys in a multi-get request was fixed for all of the experiments in the section. All of the memtier command have the form of –**ratio 1:i**, where i would vary in the key range which is 1, 3, 6 and 9. In other words one will not receive a request with fewer number of keys than specified.

5.1 Sharded Case

This subsection describes the multi-gets experiment with keys in range 1, 3, 6 and 9 while the sharding flag was enabled on the middleware application. All experiments were repeated 3 times, each repletion was 1 minute long. The error bars represent the errors between the repetitions.



(a) Response time

(b) Response time percentiles

Figure 21: Response times, sharded case

Figure 21 demonstrates the results of responses time after performing the experiment with sharding. The main tendency of the plot is the correlation of the response time and the key size of the multi-get. The response time increases the more keys size becomes. However there variation of response time is not large and lies in the range of 3 msec to 3.75 msec.

When it comes to percentile data, one could see similar kind of growth. 50% of request were completed under 3 msec, 75% were completed in less than 4 msec. The upper limit was at around 10 msec, under which the 99% percentile is located.

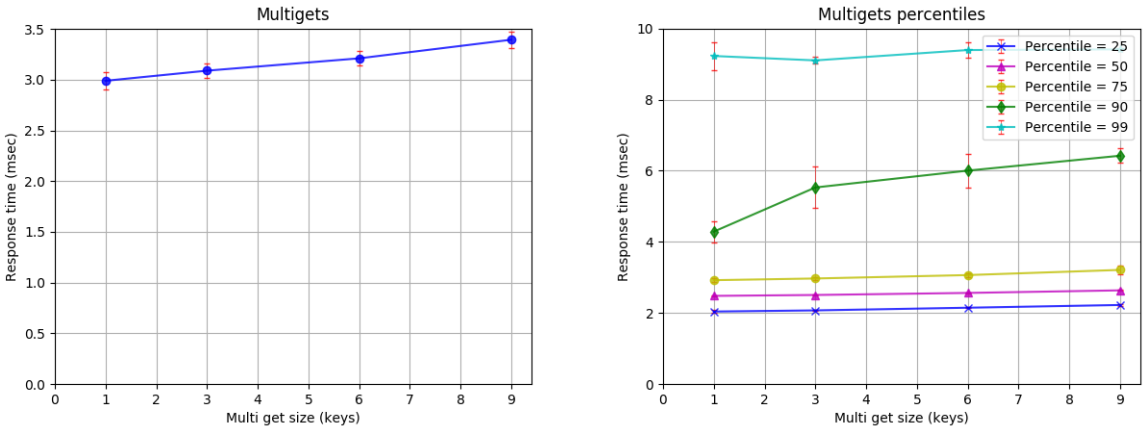
5.1.1 Explanation

The linear increase of the response time with the growth of the multi-get key size can be explained by the fact that the size of the response increases as the keysize grows. This means more data is being transfered back from the server to the middleware as well more data transfered between middleware and clients. This issue causes the service time go up as its depicted at Figure 27. Along with more service time the queening time also increases as illustrated at 28. These factors are positively correlated with response time, specifically they cause it to rise, as we see at the response time graph. The growth of all these three functions remains very similar.

One more thing worth mentioning is the fact that the response time of 90th percentile at the position of 1 multi get key is actually located higher that the one with 3 keys. This could be explained by the fact of sharding, with only one key, which cannot be sharded between the machines, this is why this example is located higher than the one with 3 keys, where sharding is obviously takes place.

5.2 Non-sharded Case

This subsection describes the multi-gets experiment with keys in range 1, 3, 6 and 9 while the sharding flag was disabled on the middleware application.



(a) Response time

(b) Response time percentiles

Figure 22: Response times, non-sharded case

Figure 22 contains the response times as functions of the keysize. One could observe the linear growth of the response time as in the case with sharding. The value range starts from 3 msec, which is slightly lower than during the sharded case, afterwards the response time function continues to grow to reach the value slightly under 3.5 msec, which is lower than in the sharded case.

Noticeably the percentiles for 25, 50, 75 percent remain the very little order of growth unlike with the sharding case. The 99th percentile was located lower compared to the sharded case.

5.2.1 Explanation

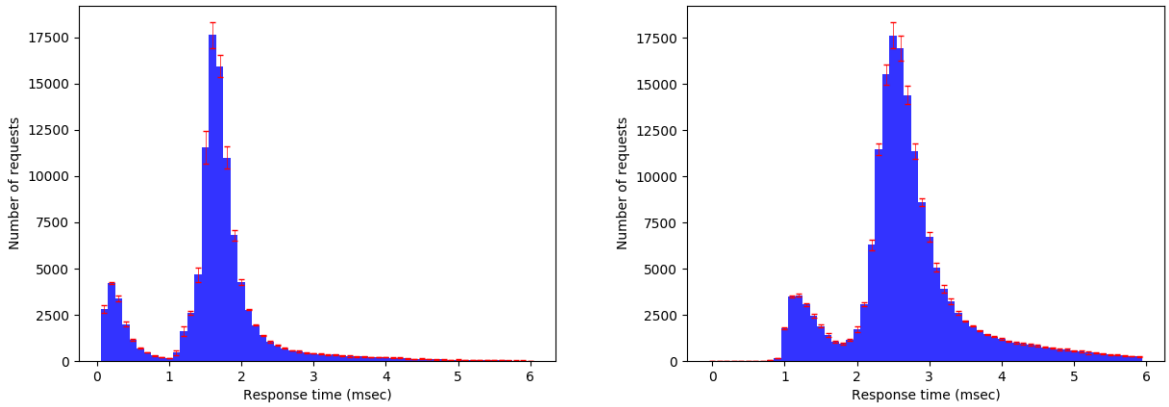
The linear increase in response time as explained as previously, the increase in key size causes the response data to enhance, which in return causes more overhead on the network. From the Figure 27 we can observe that the service time rises with more multi-get keys, however with the different rate of growth compared to the sharded case. So do the queuing times as illustrated at Figure 28. These factors as mentioned previous are correlated with the rising response time.

Noticeably for 75% of requests the response time does not demonstrate large growth as opposed wo sharding case where the growth was more visible. The 90th percentile at the non sharded case has a minimum at 1 client as opposed to the same case on the plot for sharded multigets. The same kind of reasoning as in the previous experiment is the explanation.

5.3 Histogram

For the case with 6 keys inside the multi-get, the following four histograms were constructed. All of them serve as the representation of the response time distribution. The bucket size was fixed at the value of 60 bins for all four histograms. The step size for all histograms is 0.1 msec.

The histograms from middlewares were produced from the instrumentation. On the other hands the histograms for clients were generated from the data produced by the memtier client, specifically memtier application produces the CDF of the response times, which can be easily converted to the from of PDF, which exactly suitable for the histogram.



(a) Measured on middleware

(b) Measured on client

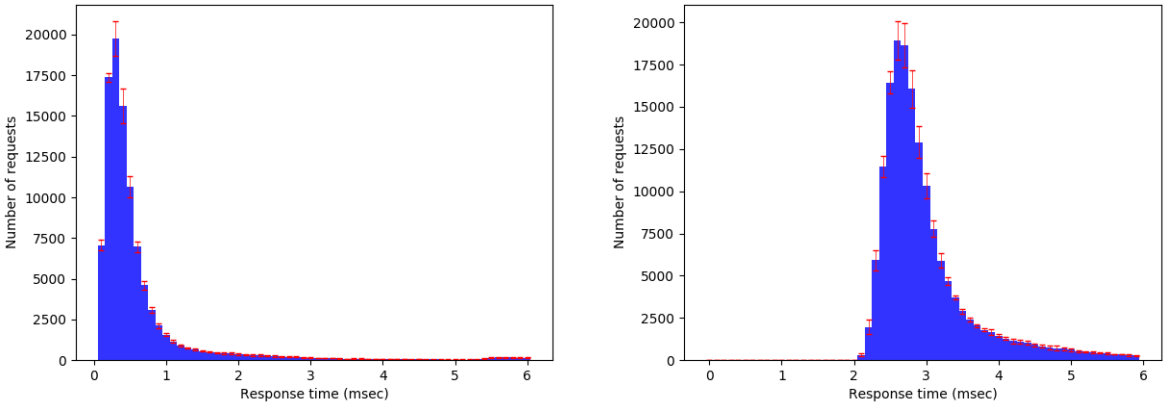
Figure 23: Histogram of multigets response times distribution, non-sharded case

Figure 23 contains the distribution of response time during the non-sharded experiment. Overall, both the histograms illustrate the same tendency and have the same shape. The main thing to observe is the shift of the histogram of the client.

This behavior is explained by the fact that the response time measured on the middleware does not include the network transfer time, which obviously cannot be measured from the middleware, as it was explained in the previous sections. Because of this reason the histogram from the clients looks like a shifted version of the histogram from the middleware. The shift is equal to 1 msec, which represents the network transfer part.

Small bump from 0 to 1 at the middlewre histogram and from 1 to 2 at the client can be explained by the fact of the mixed workload, which corresponds to the write only workload prior to read only one.

Another thing to notice is the fact that the histogram from the middleware has narrower shape compared to the one from the client. This could be explained by the averaging of the response time between multiple worker threads, which makes the data to lie closer to the mean. In contrast the client obtains more deviated results of the response time which contain more outliers.



(a) Measured on middleware

(b) Measured on client

Figure 24: Histogram of multigets response times distribution, sharded case

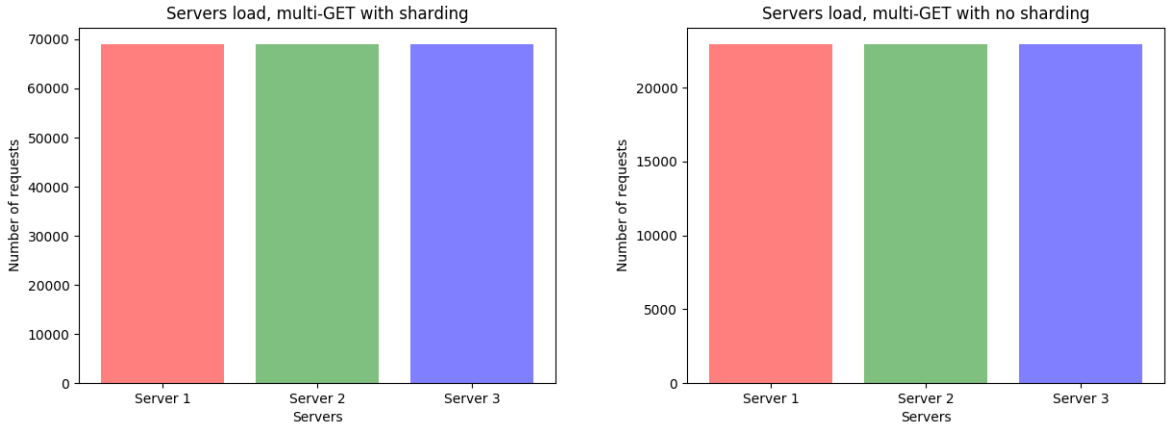
Figure 24 contains the histograms of the response time distribution during the sharded experiment. Fist thing to notice, the response time diagram of the client is shifted to the right compared to the one from the middleware histogram. The shift is due to the reason the response time from the middleware does not include network delays.

The shift of the diagrams is more visible than during the non sharded case. More of a shift could be explained by considering the figures of service time (figure 27). Overall the service time for sharded case is higher than during the case with no sharding. This means it takes servers longer to reply, which counts towards more delay to get the reply.

The client histogram again has wider shape than the one from the middleware due to the same reason as during the non sharded case. Both histograms have a lot of outliers values, which results in long tails of response times.

5.4 Summary

This section would analyze two modes in comparison to each other. Figure 25 demonstrates the server load distribution for both modes. During the operation at both modes the servers receive almost identical load of requests. One could observe that during the sharded mode the servers receive approximately three times more requests, this is explained by sharding between three servers.



(a) Sharded case

(b) Non sharded case

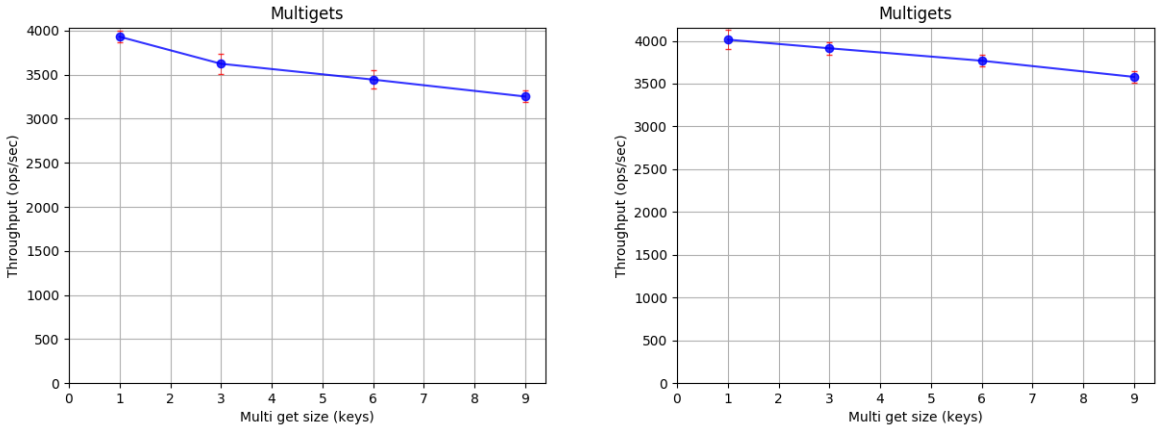
Figure 25: Load distribution multi get, number of keys is 9

While performing the comparison between two modes it makes sense not only look at the figures of response time, but also consider the figures of the throughput. Figure 26 demonstrates the throughput as the function of multi-get key size.

Overall, the throughput for both cases starts to plummet for both sharded and non sharded modes with the increase of multi-get key size. The maximum throughput for both modes was recorded at 1 multi get key, in contrast the minimum is observed at 9 keys. The reason for this behavior is the same as with the response time, more multi-get keys increase the service times, which in return decreases the throughput.

The throughput figures at 1 multi-get key look the same for both modes, this is explained by the fact that one key cannot be sharded, so two modes operate the same. However, when we go further to 3 multi-get keys the throughput decreases more during the sharded case. This is explained by the higher service time numbers (Figure 27), which means it takes longer time for servers to reply, that in return causes lower throughput numbers. The same trend continues over both sharded and non sharded curves until both reach the minimums of approximately 3250 and 3500 respectively. The figures of throughput demonstrate that both modes deliver similar performance in terms of throughput with sharded case being slighlty behind.

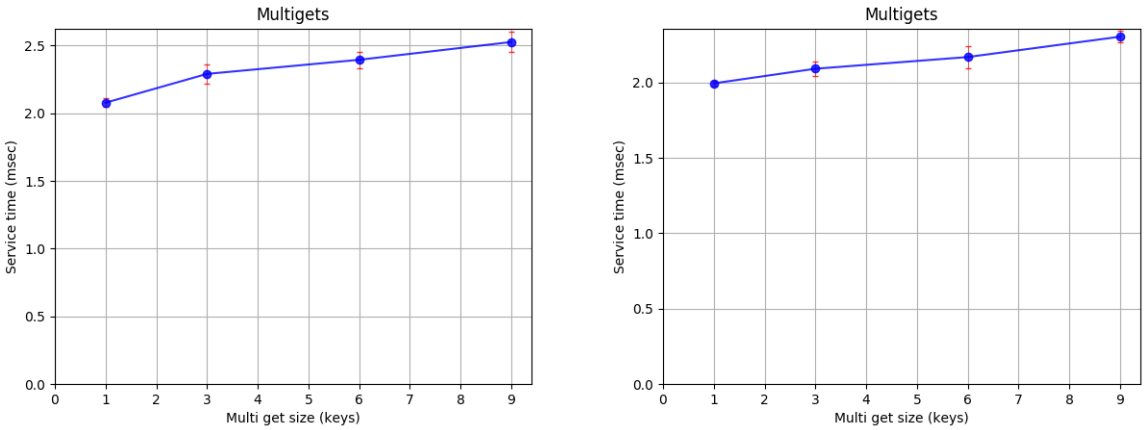
When it comes to interactive law, let us take the point of 9 keys for both sharded and non sharded mode, which have the values of throughput 3252.59 ops/sec and 3578.69 ops/sec respectively. The derivation with the interactive's law equation gives us 3243 ops/sec for sharded case and 3539 ops/sec for non sharded case. Overall the derived values from the interactive law match the measured ones.



(a) Sharded case (b) Non sharded case

Figure 26: Throughput as a function of key-size

Figure 27 contains the numbers of server services times for both modes. Overall the service times for both modes do not have very large difference. This could be explained by the fact that we only used 2 clients, which do not bring much divergence. However the service time during the sharded case has a faster rate of growth and reaches higher mark of time. During the 1 multi-get key point services times in both modes do not vary significantly, because there is nothing to shard for the sharded mode. Going over further key size introduces more difference between two modes. In general, sharded mode starts to take more time. The explanation for this is that there are more request being sent to servers because of request splitting, this in return puts more load on the servers, which causes the servers to take more time to reply.



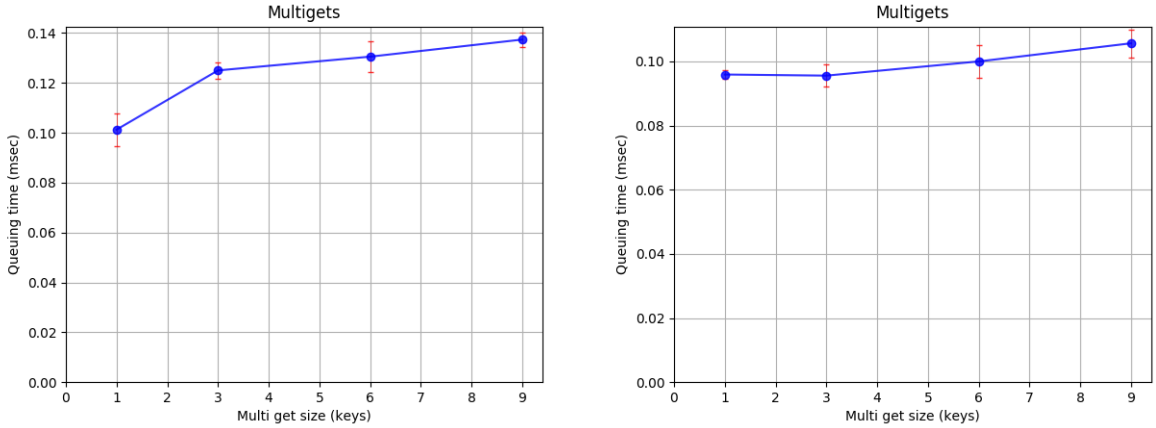
(a) Sharded case (b) Non sharded case

Figure 27: Service time as a function of key-size

The times of waiting in the queue are shown at Figure 28. The most noticeable tendency is that the queuing times are less on the full range of key-size for the non-sharded case. During the nonsharded experiment the queuing times were close to 0.10 msec, in contrast to sharded case which had more fluctuation from 0.10 msec to 0.14 msec.

Larger queuing times in sharded mode are determined by larger service times of the same mode. In other words it takes longer for servers to reply, because of this the jobs are dequeued

slower by the worker threads, which caused higher figures of queuing time in sharded mode. The same situation happens as with the service time happens with queuing times, at 1 keys there is very little difference between the metrics of two modes, because there is nothing to shard, however more and more divergence arises with the increase of multi-get key size.



(a) Sharded case

(b) Non sharded case

Figure 28: Queuing time as a function of key-size

In conclusion, two modes delivered very similar performance with the slight lead of non-sharded case. The reason for this is that the experiment was conducted at relatively low number of clients. Sharding is the technique which starts to pay back at the higher loads of clients. Obviously at this point of clients, which was fixed at 2 clients, it is hard to adequately compare two modes. Basically for 2 clients it is more preferable to use non sharded mode for all values of keys from range 1, 3, 6, 9.

6 2K Analysis (90 pts)

During the experiments in previous sections we demonstrated how various component of different configurations affect system's performance, however the individual affects of these system components have not been explored yet. The need for the experiment which would estimate the components individually is inevitable while performing system's analysis.

This section's experiments purpose is to highlight how the proposed parameters of the system namely: memcached servers, middlewares, worker threads inside middlewares affect our entire system's performance. The total number of proposed factors to test is 3, in addition to that fact the proposed number of repetitions is 3, therefore we have to explore the factorial model of 2^3r (the number of factors is k = 3, and the number of repetitions r = 3) which consists of 8 combinations of factors and takes in regard 3 repetitions. At the end, the analysis of these factors on overall system's performance based on the selected model will be performed.

To get the data 3 client machines of type A2 with 66 total virtual clients each were used to generate the load. The measurement of both throughput and response time of the system during the experiment were done as in the previous experiments, specifically the number of repetitions was fixed to three. When it comes to get, the experiment did not make used of multigets, but only concentrated on single GETS. Further, the following proposed parameters were investigated:

- Memcached servers: 2 and 3

- Middlewares: 1 and 2
- Worker threads per MW: 8 and 32

The experiments were repeated for write only, read only, and finally write-read types of workload. All the data which was used to determine the effect of the parameters on throughput and response time was taken from the side of client. All experiments were repeated 3 times, each repletion was 1 minute long.

Let A represent the first factor, namely the number of memcached servers, in analogy let's represent the second factor as B, which is the number of middlewares, finally let C represent the work threads per MW factor. Next, we define three variables to encode our combinations of factors as follows:

$$x_A = \begin{cases} -1 & \text{2 memcahced servers} \\ 1 & \text{3 memcahced servers} \end{cases}$$

$$x_B = \begin{cases} -1 & \text{1 middleware component} \\ 1 & \text{2 middleware components} \end{cases}$$

$$x_C = \begin{cases} -1 & \text{8 worker threads} \\ 1 & \text{32 worker threads} \end{cases}$$

Based on these variables we next define our nonlinear regression model as:

$$y = q_0 + q_A x_A + q_B x_B + q_C x_C + q_{AB} x_A x_B + q_{AC} x_A x_C + q_{BC} x_B x_C + q_{ABC} x_A x_B x_C + e$$

Where q_n correspond to affects of each individual factors, as well as the combined effects of factors. The final term e represents the error term, since we consider the $2^k r$ model, which takes in regards the repetitions.

Considering the information described earlier we now need to determine the effect coefficients. This will be performed by constructing the sign table presented during the exercise.

Further when we have the corresponding table for each subsection we need to determine the sum of the squared errors, notated as SSE. This value will be used later for estimation of error variance:

$$SSE = \sum_{i=1}^{2^3} \sum_{j=1}^{3} e_{ij}^2$$

In addition we need to determine the allocation of variation. Let the total variation be calculated as:

$$SST = SSA + SSB + SSC + SSAB + SSAC + SSBC + SSABC + SSE$$

Various components that make up our allocation of variation are computed with the following equations:

$$SSA = 2^3 \cdot 3 \cdot q_A^2 \qquad SSAB = 2^3 \cdot 3 \cdot q_{AB}^2$$
$$SSB = 2^3 \cdot 3 \cdot q_B^2 \qquad SSAC = 2^3 \cdot 3 \cdot q_{AC}^2 \qquad SSABC = 2^3 \cdot 3 \cdot q_{ABC}^2$$
$$SSC = 2^3 \cdot 3 \cdot q_C^2 \qquad SSBC = 2^3 \cdot 3 \cdot q_{BC}^2$$

The final equation for SST now takes the form:

$$SST = 2^3 \cdot 3 \cdot q_A^2 + 2^3 \cdot 3 \cdot q_B^2 + 2^3 \cdot 3 \cdot q_C^2 + 2^3 \cdot 3 \cdot q_{AB}^2 + 2^3 \cdot 3 \cdot q_{AC}^2 + 2^3 \cdot 3 \cdot q_{BC}^2 + 2^3 \cdot 3 \cdot q_{ABC}^2 + \sum_{i,j} e_{ij}^2$$

Once we have all of the above variables we can compute the variation of each parameter, which would be used for the analysis. The variation of a factor could be obtained by dividing the variation of specific factor by the total variation. In the following subsections the experiment would be repeated for three types of workloads.

6.1 Write-only workload analysis

After performing the write only workload there results for throughput of three repetitions from the memtier client were put into Table 1.

Exp.	Effects								Measured throughput			Mean of y_n	Errors		
i	I	A	B	C	AB	AC	BC	ABC	y_{i3}	y_{i1}	y_{i2}	y_i	e_{i1}	e_{i2}	e_{i3}
1	1	-1	-1	-1	1	1	1	-1	2475	2604	2702	2594	118.67	-10.33	-108.33
2	1	-1	-1	1	1	-1	-1	1	5985	6001	5069	5685	-300.00	-316.00	616.00
3	1	-1	1	-1	-1	1	-1	1	4836	4644	4696	4725	-110.67	81.33	29.33
4	1	-1	1	1	-1	-1	1	-1	10446	10256	9908	10203	-242.67	-52.67	295.33
5	1	1	-1	-1	-1	-1	1	1	1933	1990	2035	1986	53.00	-4.00	-49.00
6	1	1	-1	1	-1	1	-1	-1	4513	4457	4433	4468	-45.33	10.67	34.67
7	1	1	1	-1	1	-1	-1	-1	3700	3742	3711	3718	17.67	-24.33	6.67
8	1	1	1	1	1	1	1	1	8462	8504	8222	8396	-66.00	-108.00	174.00

Table 1: Factorial experiment for throughput, write-only workload

Table 2 contains the results of response time from the same experiment.

Exp.	Effects								Response time			Mean of y_n	Errors		
i	I	A	B	C	AB	AC	BC	ABC	y_{i3}	y_{i1}	y_{i2}	y_i	e_{i1}	e_{i2}	e_{i3}
1	1	-1	-1	-1	1	1	1	-1	77	73	70	73	-3.67	0.33	3.33
2	1	-1	-1	1	1	-1	-1	1	31	32	37	33	2.33	1.33	-3.67
3	1	-1	1	-1	-1	1	-1	1	40	42	41	41	1.00	-1.00	0.00
4	1	-1	1	1	-1	-1	1	-1	18	18	19	18	0.33	0.33	-0.67
5	1	1	-1	-1	-1	-1	1	1	99	96	93	96	-3.00	0.00	3.00
6	1	1	-1	1	-1	1	-1	-1	42	42	43	42	0.33	0.33	-0.67
7	1	1	1	-1	1	-1	-1	-1	53	51	52	52	-1.00	1.00	0.00
8	1	1	1	1	1	1	1	1	22	22	23	22	0.33	0.33	-0.67

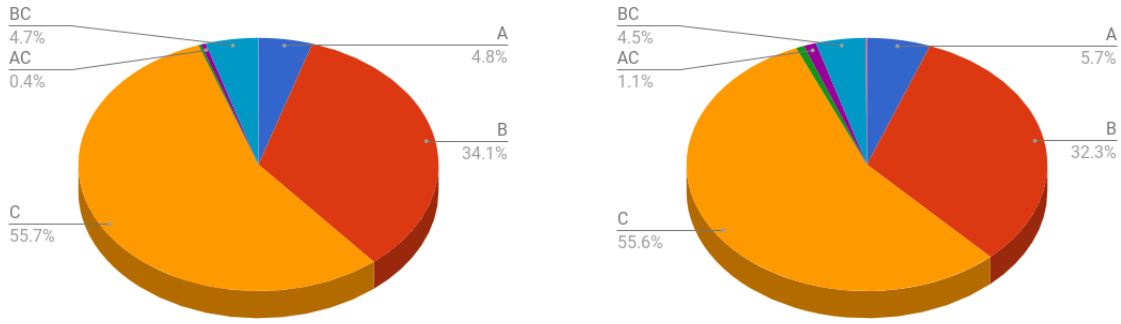
Table 2: Factorial experiment for response time, write-only workload

	Throughput analysis		Response time analysis	
Parameter	Mean estimate	Variation (%)	Mean estimate	Variation %
q_0	5221.83	-	47.33	-
q_A	-580	4.82%	5.83	5.65%
q_B	1538.75	33.93%	-13.92	32.17%
q_C	1966.17	55.39%	-18.25	55.32%
q_{AB}	-123.75	0.22%	-2.08	0.72%
q_{AC}	-176.17	0.44%	-2.58	1.11%
q_{BC}	572.92	4.70%	5.17	4.43%
q_{ABC}	-23.75	0.01%	0.83	0.12%

Table 3: Allocation of variation, write-only workload

The results of variation of each factor were put in the Table 3. From the table we see that the average throughput is 5221.83 ops/sec. The throughput is mostly affected by the value of number of worker threads inside the middleware, which is at 55.39% percent. This behavior is confirmed by the baseline experiment, where we saw significant performance jump with the addition of more worker threads. The second largest factor in terms of variation is the number of middlewares, which contributed to 34.1% of throughput variation. Number of servers and number of middlewares combined with worker threads number introduced similar minor variations of 4.7 % and 4.8% respectively.

Average response time was 47.33 msec. The figures for the response demonstrate the same tendency as with throughput, with the number of workers threads contributing the most to the variation at 55.32 %. The second largest factor is number of middlewares, which is 32.3 %. Number of servers and interactive effect of middlewares number and workers threads number made rather small contribution to variation art 5.7% and 4.5% respectively.



(a) Throughput (b) Response time

Figure 29: Variation chart, write only workload

6.2 Read-only workload analysis

After performing the read only workload there results for throughput of three repetitions from the memtier client were put into Table 4.

Exp.	Effects								Measured throughput			Mean of y_n	Errors		
i	I	A	B	C	AB	AC	BC	ABC	y_{i3}	y_{i1}	y_{i2}	y_i	e_{i1}	e_{i2}	e_{i3}
1	1	-1	-1	-1	1	1	1	-1	4275	4331	4331	4312	37.33	-18.67	-18.67
2	1	-1	-1	1	1	-1	-1	1	9635	9619	9504	9586	-49.00	-33.00	82.00
3	1	-1	1	-1	-1	1	-1	1	8055	7966	8045	8022	-33.00	56.00	-23.00
4	1	-1	1	1	-1	-1	1	-1	17831	16581	17534	17315	-515.67	734.33	-218.67
5	1	1	-1	-1	-1	-1	1	1	4973	5153	5213	5113	140.00	-40.00	-100.00
6	1	1	-1	1	-1	1	-1	-1	10943	10127	10037	10369	-574.00	242.00	332.00
7	1	1	1	-1	1	-1	-1	-1	10087	9142	9264	9498	-589.33	355.67	233.67
8	1	1	1	1	1	1	1	1	18440	18126	15765	17444	-996.33	-682.33	1678.67

Table 4: Factorial experiment for throughput, read-only workload

Alternatively the number of response time were put into Table 5.

Exp.	Effects								Response time			Mean of y_n	Errors		
i	I	A	B	C	AB	AC	BC	ABC	y_{i3}	y_{i1}	y_{i2}	y_i	e_{i1}	e_{i2}	e_{i3}
1	1	-1	-1	-1	1	1	1	-1	44	44	44	44.00	0.00	0.00	0.00
2	1	-1	-1	1	1	-1	-1	1	19	19	20	19.33	0.33	0.33	-0.67
3	1	-1	1	-1	-1	1	-1	1	23	24	24	23.67	0.67	-0.33	-0.33
4	1	-1	1	1	-1	-1	1	-1	10	11	10	10.33	0.33	-0.67	0.33
5	1	1	-1	-1	-1	-1	1	1	38	37	36	37.00	-1.00	0.00	1.00
6	1	1	-1	1	-1	1	-1	-1	17	18	19	18.00	1.00	0.00	-1.00
7	1	1	1	-1	1	-1	-1	-1	19	21	21	20.33	1.33	-0.67	-0.67
8	1	1	1	1	1	1	1	1	10	10	6	8.67	-1.33	-1.33	2.67

Table 5: Factorial experiment for response time, read-only workload

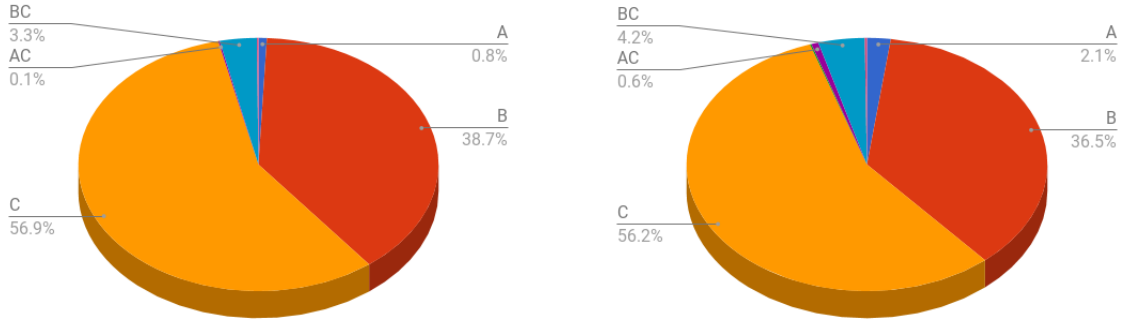
	Throughput analysis		Response time analysis	
Parameter	Mean estimate	Variation (%)	Mean estimate	Variation %
q_0	10207.38	-	22.67	-
q_A	398.46	0.74%	-1.67	2.11%
q_B	2862.29	38.25%	-6.92	36.30%
q_C	3471.13	56.25%	-8.58	55.90%
q_{AB}	2.54	0.00%	0.42	0.13%
q_{AC}	-170.63	0.14%	0.92	0.64%
q_{BC}	838.71	3.28%	2.33	4.13%
q_{ABC}	-166.21	0.13%	-0.50	0.19%

Table 6: Allocation of variation, read-only workload

Table 6 contains the variations of factors for both response time and throughput. Overall the results of variations are very similar to the ones from the write only workload. The average value of throughput is at 10207.38 ops/sec, and average response time was at 22.67 msec.

Noticeably, the largest factors towards the variation of both throughput and middleware were the number of worker threads, at 56.25% and 55.90% respectively. This is confirmed by the baseline's read only experiment, where the number of workers threads introduced the significant speedup of performance. The second largest variation contributor was the number of

middlewares, 38.25% for throughput, 36.30 % fro response time, the numbers are higher than in the write only workload similarly to the baseline experiment. Interactive effect of numer of middlewares and worker threads was at 3.28% and 4.13%. Number of servers and other interactive effects did not bring much variation again.



(a) Throughput (b) Response time

Figure 30: Variation chart, read only workload

6.3 Write-Read-50-50 workload analysis

The data of write-read experiment was recorded in tables 7 and 8.

Exp.	Effects								Measured throughput			Mean of y_n	Errors		
i	I	A	B	C	AB	AC	BC	ABC	y_{i3}	y_{i1}	y_{i2}	y_i	e_{i1}	e_{i2}	e_{i3}
1	1	-1	-1	-1	1	1	1	-1	3305	3427	3710	3481	175.67	53.67	-229.33
2	1	-1	-1	1	1	-1	-1	1	7246	6901	7026	7058	-188.33	156.67	31.67
3	1	-1	1	-1	-1	1	-1	1	6710	6206	6621	6512	-197.67	306.33	-108.67
4	1	-1	1	1	-1	-1	1	-1	13483	12910	13217	13203	-279.67	293.33	-13.67
5	1	1	-1	-1	-1	-1	1	1	3106	3319	3054	3160	53.67	-159.33	105.67
6	1	1	-1	1	-1	1	-1	-1	6247	6413	6488	6383	135.67	-30.33	-105.33
7	1	1	1	-1	1	-1	-1	-1	5908	5194	5582	5561	-346.67	367.33	-20.67
8	1	1	1	1	1	1	1	1	12733	11740	5895	10123	-2610.33	-1617.33	4227.67

Table 7: Factorial experiment for throughput, write-read workload

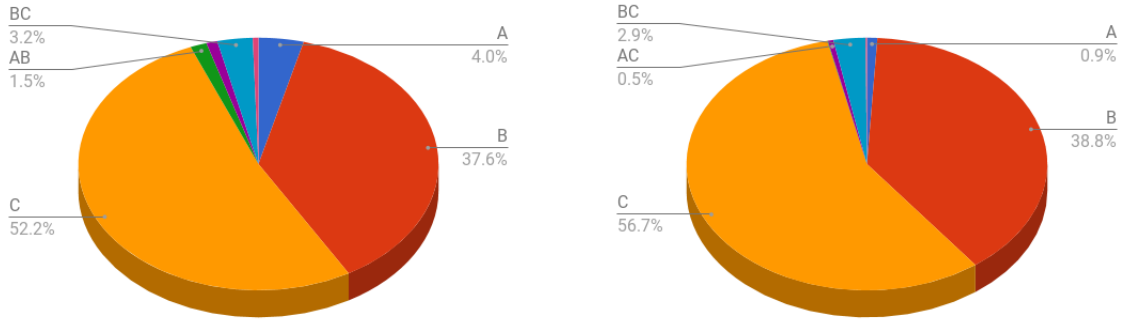
Exp.	Effects								Response time			Mean of y_n	Errors		
i	I	A	B	C	AB	AC	BC	ABC	y_{i3}	y_{i1}	y_{i2}	y_i	e_{i1}	e_{i2}	e_{i3}
1	1	-1	-1	-1	1	1	1	-1	58	56	51	55.00	-3.00	-1.00	4.00
2	1	-1	-1	1	1	-1	-1	1	26	27	27	26.67	0.67	-0.33	-0.33
3	1	-1	1	-1	-1	1	-1	1	28	31	29	29.33	1.33	-1.67	0.33
4	1	-1	1	1	-1	-1	1	-1	14	14	14	14.00	0.00	0.00	0.00
5	1	1	-1	-1	-1	-1	1	1	61	57	62	60.00	-1.00	3.00	-2.00
6	1	1	-1	1	-1	1	-1	-1	30	29	29	29.33	-0.67	0.33	0.33
7	1	1	1	-1	1	-1	-1	-1	33	37	35	35.00	2.00	-2.00	0.00
8	1	1	1	1	1	1	1	1	15	16	8	13.00	-2.00	-3.00	5.00

Table 8: Factorial experiment for response time, write-read workload

	Throughput analysis		Response time analysis	
Parameter	Mean estimate	Variation (%)	Mean estimate	Variation %
q_0	6935.04	-	22.67	-
q_A	-628.46	3.62%	-1.67	0.92%
q_B	1914.88	33.57%	-6.92	38.23%
q_C	2256.54	46.62%	-8.58	55.90%
q_{AB}	-379.46	1.32%	0.42	0.05%
q_{AC}	-310.46	0.88%	0.92	0.49%
q_{BC}	556.54	2.84%	2.33	2.83%
q_{ABC}	-221.96	0.45%	-0.50	0.11%

Table 9: Allocation of variation, write-read workload

Table 9 contains the variations of factors for the write-read workload. Average throughput and response times were at 6935.04 ops/sec and 22.67 msec marks respectively. The behavior remains the same as previously the number of worker threads is still the largest contributor to variation at 46.62% for throughput and 55.9% for response time. The second largest variation in introduced by the number of middlewares at similar marks for throughput and response time at 37.6% and 38.8%, respectively. All other factors produced relatively minor contribution to variation.



(a) Throughput (b) Response time

Figure 31: Variation chart, write-read workload

6.4 Summary

In conclusion, after performing the experiment on various types of workload we can state that the largest contributor to variation is the number of worker threads. This result is confirmed by the baseline and throughput for writes experiments, where we have seen large performance boost with the increase of worker threads from 8 to 32. The second largest variation factor was the number of middlewares, this behavior is confirmed by the baseline with middleware experiment where we observed performance boost with the additional middleware component.

7 Queuing Model (90 pts)

Previous sections gave us the opportunity to explore the entire system's performance on various types of configurations, types of workloads as well as the individual components contribution.

However for better interpretation of system's performance it makes sense to make use of numerous experimental results and internal instrumentation to design theoretical models of various types which could be later compared to the actual results. The need for such a model or even a group of model arises.

The purpose of this section is dedicated towards the design of three models, namely M/M/1 model, M/M/m model and the network of queues model. Each model would be compared to the actual data measured during the experiments as well as the performance of these models would be compared against each other.

7.1 M/M/1

The modeling of the system starts with the M/M/1 model, which would be described in this subsection. This model is the simplest of all three. The data which was used to design this queuing model is based on section for throughput for writes. The modeling was repeated for all worker threads configurations: 8, 16, 32, and 64 worker threads. It was decided to fix the number of virtual client to 33 clients, because it is regarded as the mark when the system is under the most pressure.

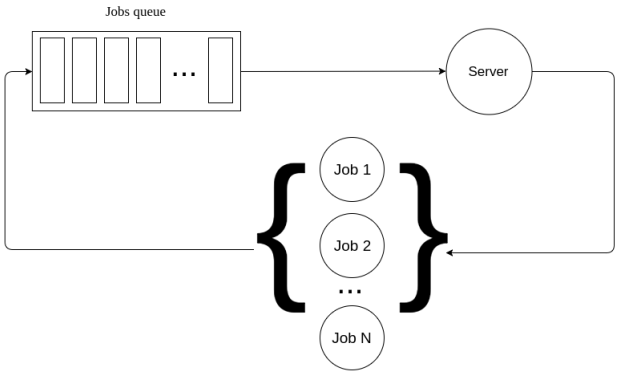


Figure 32: Modeling the system with M/M/1

Figure 32 represents the architecture of the model which was used. We can see that during this setting only one M/M/1 queue represented the entire system during modeling. This somewhat simplistic design will of course have the affect on the results, which are described later.

Before building the M/M/1 model one has to acquire two parameters, namely jobs arrival rate, notated as λ as well as the jobs service rate, notated as μ.

- To acquire λ the internal middleware instrumentation was used. Specifically the network thread of the middleware records the number of jobs arriving to the system at the one second interval. After obtaining all of the values of arrivals per seconds the average of this list is taken. The average of all repetitions of these computed values provides us with λ, which we later use for modeling.

  There is however another way to calculate the arrival rate, which the courses book proposes. Specifically arrival rate is total arrivals divided by total time. It was decided not to use this metric due to sufficient instrumentation.

- Next the value of μ was obtained. The value represents the maximum throughput configurations of the respective worker threads configurations. To get the value we need to

return back to throughput for writes experiment where one has to take the maximum point of each of the throughput curves. Theses values are exactly the jobs service rates for the corresponding worker threads configuration.

Little's law was used to estimate the number of jobs located in the system, specifically mean number in the system was defined as a product of arrival rate times mean time spent in the system.

The modeling begins with the value of worker threads at 64, the results of modeling are demonstrated at Table 10. The only input parameters for the model are jobs arrival rate which is 7538.59 ops/sec and jobs service rate which was 8221 ops/sec.

The first parameter which has to be checked is ρ, in this case it is equal to 0.9170, the values is rather high, however the value is lower than 1, which means the system is stable, but fairly saturated. The large value of ρ is explained by the large number of virtual clients used for the experiment, which put the pressure on the system. The throughput graphs from the section for throughput for writes experiments confirm the value.

The metric of probability of zero jobs in this system has rather low value of only 0.0830. Low value could be explained by large number of worker threads, specifically the tasks are parallelized by the 64 worker threads, which means more jobs could be taken form the queue, which causes the queue to be shorter on average. This behavior has been already confirmed by the figures of queuing lengths in section on throughput for writes which were much lower for 64 worker threads compared to configurations with fewer worker threads.

Mean number of jobs in the system has reached the value of 11.047 which is significantly lower than the estimation. The underestimation of the model could be explained by model's architecture. The entire system is modeled as just one single queue, apart from this the model does not assume any noise coming from the network delays. The value is confined by the idealistic view of this model, which assumes jobs enter and exit the single queue without delays and do not accumulate inside the queue much.

The same tendency is as with mean number of jobs in the system is observed with mean number of jobs in the queue. The model assumes that the majority of the jobs in the system are located in the queue. However the real measurements demonstrate that only less than half jobs are located in the queue. This could be again backed by the fact that this model does not take network delays into account.

The value of 1.465 msec was predicted by the model, when it comes to the response time. The value of the response time is lower than the predicted one. The estimation of mean waiting time, on the other hand was closer to the measured value. This explained by the fact that response times consists of multiple values, therefore the value converges more than other more discrete metrics. Overall higher values of measured results are explained by network delays.

Metric description	Variable	Estimated results	Measured results
Jobs arrival rate	λ	7538.59	-
Jobs service rate	μ	8221	-
Utilization	ρ	0.9170	-
Probability of zero jobs in the system	p_0	0.0830	-
Mean number of jobs in the system	$E[n]$	11.047	100.768 (estimation)
Mean number of jobs in the queue	$E[n_q]$	10.13	46
Mean response time	$E[r]$	1.465	19.567
Variance of the response time	$Var[r]$	0.002147	-
Mean waiting time	$E[w]$	1.3236	6.82
Variance of the waiting time	$Var[w]$	0.002132	-

Table 10: M/M/1 results comparison, 64 WT configuration

Further the same kind of modeling was performed with the 32 worker threads configurations, the results of this modeling are show at Table 11. The jobs arrival rate was 7230.32 ops/sec which is lower than in the previous case, the reason for this fewer worker threads, which are correlated with the throughput as it was demonstrated during previous sections. In contrast the value of jobs service rate has only slightly decreased compared to previous section, specifically to 8177 jobs/sec.

The figures for utilization are at 0.8842, which means the system is stable, however higher number indicates that the system is under heavy load. The value is very similar to the utilization value from the configuration with 64 worker threads. This detail is confirmed by nearly identical throughput and response time behavior, which was noticed during the corresponding experiment. The lower value of utilization than the one with 64 worker, can be explained by thread overhead which does not increase the performance significantly but rather introduces so called competition between the threads on the 8 core machine.

Similarly to the previous configuration the figures for mean number of jobs in the system from modeling are lower than the ones from the system. The same happens to mean number of jobs in the queue it is lower than the measurement. The reasoning behind it is the same as in the previous configurations. Despite lower modeled values the difference between number of jobs in the system and queue is significantly less in the case with more worker threads. This result is based on model's naive assumption of more worker threads meaning significantly less queuing.

Mean response time reached the value of 1.056 msec and mean waiting time was at 0.9340 msec. The most noticeable thing is that these values are lower than the same values from the real measurements of the system, however the difference of the values of response time and waiting time from modeling are similar to the ones from the real measurements. This is explained by model's assumption that waiting time is the largest contributor to response time. The real data shows us that time spent in queue is indeed regarded as the one which contributes the most to the response time.

Metric description	Variable	Estimated results	Measured results
Jobs arrival rate	λ	7230.32	-
Jobs service rate	μ	8177	-
Utilization	ρ	0.8842	-
Probability of zero jobs in the system	p_0	0.1158	-
Mean number of jobs in the system	$E[n]$	7.63755	126.212 (estimation)
Mean number of jobs in the queue	$E[n_q]$	6.7533	106
Mean response time	$E[r]$	1.056	21.991
Variance of the response time	$Var[r]$	0.00112	-
Mean waiting time	$E[w]$	0.9340	18.66
Variance of the waiting time	$Var[w]$	0.0011	-

Table 11: M/M/1 results comparison, 32 WT configuration

Table 12 shows the results for the modeling with 16 worker threads. The arrival rate is decreased to 5898.29 ops/sec compared to the previous configuration, so did jobs service rate to 6569 ops/sec. Lower arrival rate is explained by the fact that our is system is a closed system, specifically if a client does not receive the response he would send another request, thus reducing the amount of jobs arriving.

The utilization jumped up compared to the previous case and reached 0.8979, which is explained by the reduction of worker threads by the factor of two.

Mean number of jobs in the system and mean number of jobs in the queue is again explained by the model's assumption of only one server, which is not the case for the real system which contains a lot of jobs in the worker threads.

The same tendency holds for the very low discrepancies in response time and waiting time, which was exactly the case in the measured results. Thus the response time and the waiting time from the real measurements appear as the scaled version of the ones from modeling.

Metric description	Variable	Estimated results	Measured results
Jobs arrival rate	λ	5898.29	-
Jobs service rate	μ	6569	-
Utilization	ρ	0.8979	-
Probability of zero jobs in the system	p_0	0.1021	-
Mean number of jobs in the system	$E[n]$	8.7940	139.677 (estimation)
Mean number of jobs in the queue	$E[n_q]$	7.896	150
Mean response time	$E[r]$	1.491	28.77
Variance of the response time	$Var[r]$	0.0022	-
Mean waiting time	$E[w]$	1.3387	27.18
Variance of the waiting time	$Var[w]$	0.0022	-

Table 12: M/M/1 results comparison, 16 WT configuration

The last configuration of worker threads for this model was the one with 8 worker threads, similarly the results can be found at Table 13. Both input values of jobs arrival rate and jobs service rate increased, 4894.72 ops/sec and 5347.67 ops/sec respectively.

The most noticeable thing is the increase of the utilization value which jumped to the value of 0.9153. This behavior was already observed during the corresponding experiment, specifically the system at 8 worker threads was saturated a relatively low mark of clients and remained saturated until 33 virtual clients mark and onwards.

Mean number of jobs in the queue as well as number of jobs in the system are lower than the measured ones because of the one server assumption of the model, as it was the case in the previous configuration.

The waiting time as well the response time of the model look again as the scaled versions of the waiting time and the response times from the measurement. There is a very little difference between the response times and waiting times at both modeling part and the measurements one, which is again explained by model's assumption that the response time mostly consists of waiting time.

Metric description	**Variable**	**Estimated results**	**Measured results**
Jobs arrival rate	λ	4894.72	-
Jobs service rate	μ	5347.67	-
Utilization	ρ	0.9153	-
Probability of zero jobs in the system	p_0	0.0847	-
Mean number of jobs in the system	$E[n]$	10.806	160.806 (estimation)
Mean number of jobs in the queue	$E[n_q]$	9.891	170
Mean response time	$E[r]$	2.077	35.305
Variance of the response time	$Var[r]$	0.00487	-
Mean waiting time	$E[w]$	2.0208	33.6
Variance of the waiting time	$Var[w]$	0.0048	-

Table 13: M/M/1 results comparison, 8 WT configuration

7.1.1 Summary

In conclusion, after performing the modeling for different worker thread configurations with M/M/1 model one could agree that the model is too simplistic for the accurate modeling of system's measured parameters, although the proposed model was able to demonstrate the main trends of the system such as system's saturation and variability of the parameters under various configurations.

The reasons for mismatching of some modeled parameters to the real ones could be explained by the following properties of this model:

- The model assumes only one server, which clearly is not the case in our system which was designed with the idea of multi-threading in mind. One server is simply not able to accurately model larger number of threads, although still might show the overall utilization trend.
- The propose model only makes use of two parameters, namely the value of arrival rate and service rate. Only two values are simply not enough to model a complex multi-threaded system
- The model does not take in regard the network delay part as well as various other hidden factors which influence the performance of computer system.

7.2 M/M/m

After performing the modeling with the M/M/1 model we observed that it is not accurate to model all worker threads with just one server.

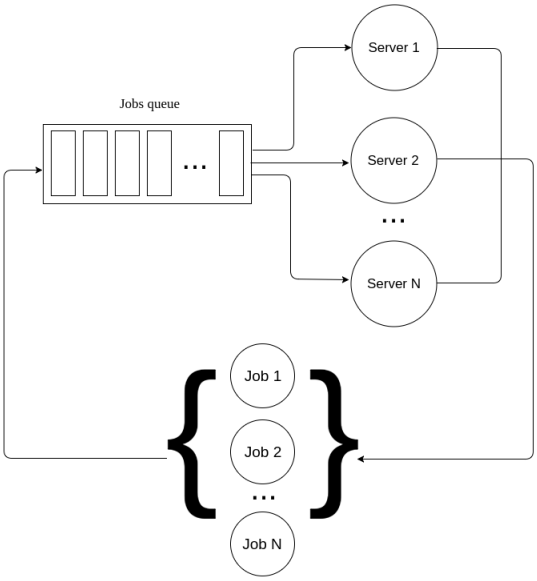


Figure 33: Modeling the system with M/M/m

Therefore the need for the more accurate model which takes in regard each individual worker threads of our multi-threaded software arises. This section will describe the results of modeling using the M/M/m model.

The general architecture of the model proposed in the section is illustrated at 33. In this model each worker thread of the midlewares was represented as one server. Therefore the model which would be used in the rest of the section will be M/M/128. To be more specific, 128 corresponds to 128 worker threads, because 64 worker threads middleware configuration was selected as best performing configuration. It was decided to vary the number of virtual clients in the range of 33, 17 and 9 clients to explore the behavior of the system on different loads.

In order to construct the M/M/64 model, one first has to acquire the following three parameters:

- Arrival rate λ. This metric is chosen as in the previous section and obtained from the internal instrumentation rather than from the analytical expression. The value of arrival rate is varied across the range of virtual clients.
- μ which stands for jobs service rate per server. This metric is similar to the jobs service rate in the M/M/1 model, with the difference of taking in regard each server. The value is fixed at 66 ops/sec per server, which the limit of throughput each worker thread on average can do. The value obviously stays fixed for various cases of virtual clients.
- m the third parameter for the model is the number of servers parameter, which is 128 in our case.

Firstly the modeling was performed on 33 virtual clients. The results are displayed at Table 14. The system achieve 0.89 at its utilization, which a similar values to the one in the M/M/1. Relatively high value of utilization is explained by the fact that system is under heavy load at that point, which makes sense with this number of clients.

Probability of zero jobs in the system is incredibly low, which is quite natural at this amount of clients as well as it is backed by the plot of queue length in the section for throughput for writes.

The probability of queuing is at 0.14. This value does not match the real-world scenario, because it is inevitable that there would definitely be some queuing at the system. The plot of queue length from the corresponding section as well as the corresponding instrumentation at this configuration that there are always jobs in the queue.

Mean number of jobs in the system was at 115.4 ops/sec, which is very close to the estimated value. Mean number of jobs in the queue was at the very low value of 1.174 jobs compared to 46 jobs measured from the experiments. Unlike the model in the previous section the M/M/m model does not assume that most of the jobs in the systems are jobs in the queue, which confirmed by real measurement, however not at such extreme as the modeling performed.

When it comes to response time model's results are quite close to the ones measured on the middleware, 15.31 msec and 19.567 msec respectively. In contrast the waiting time has somewhat larger discrepancy. The model has the opposite behavior to the one the model in the previous section produced. The M/M/m model assumes that the largest part of the response time comes from time spend in worker threads, from the real measurements of our system, we see that queuing time and memcached service time make equal contribution with queuing time contributing a bit more.

Metric description	**Variable**	**Estimated results**	**Measured results**
Jobs arrival rate	λ	7538.59	-
Jobs service rate	μ	66	-
Utilization	ρ	0.89	-
Probability of zero jobs in the system	p_0	2.39E-50	-
Probability of queuing	ϱ	0.14	-
Mean number of jobs in the system	$E[n]$	115.4	100.768 (estimation)
Mean number of jobs in the queue	$E[n_q]$	1.174	46
Mean response time	$E[r]$	15.31	19.567
Variance of the response time	$Var[r]$	0.23	-
Mean waiting time	$E[w]$	0.16	6.82
Variance of the waiting time	$Var[w]$	3.2E-4	-

Table 14: M/M/128 results comparison, VC = 33 configuration

Further, the number of virtual clients was reduced to 17. The results of modeling from such a setting are showed at Table 15. The arrival rate has decreased to the value of 7194.246 ops/sec due to the lower number of clients.

The value of utilization went down compared to the previous case. This explained by the fact that less clients put less load on the system, therefore the value of utilization decreases. The probability of zeros jobs in the system is slightly higher than in the previous configuration, however it still remains absolutely insignificant, because at 17 virtual clients there were never 0 jobs spotted at the plots or logfiles. The probability of queuing has decrease compared to the configuration with 33 virtual clients, which is a natural type of behavior to expect with the decrease of virtual clients.

When it comes to response time the difference between the modeled value and the measured value is not particularly large 15.19 msec and 10.389 msec respectively. The discrepancies of waiting times from modeling and the measurements are a bit larger than from the case with response times. However the difference between mean response time and mean waiting time have the same trend on modeling and measurements, this is explained by the assumption of the model that the queue is not full and that the majority of the jobs are in the workers threads, which matches the real world scenario.

Metric description	Variable	Estimated results	Measured results
Jobs arrival rate	λ	7194.246	-
Jobs service rate	μ	66	-
Utilization	ρ	0.85	-
Probability of zero jobs in the system	p_0	4.53E-48	-
Probability of queuing	ϱ	0.05	-
Mean number of jobs in the system	$E[n]$	109.29	44.576 (estimation)
Mean number of jobs in the queue	$E[n_q]$	0.282	7
Mean response time	$E[r]$	15.19	10.389
Variance of the response time	$Var[r]$	0.23	-
Mean waiting time	$E[w]$	0.039	1.13
Variance of the waiting time	$Var[w]$	6E-5	-

Table 15: M/M/128 results comparison, VC = 17 configuration

Finally, the system was modeled at the value of 9 virtual clients. The results of modeling are shown at Table 16. The input parameter of jobs arrival rate was 6333.46 ops/sec.

The most noticeable thing is the value of utilization, which plummeted to 0.75 compared to previous configurations. This demonstrates that the system is under less load. The value continues to drop even more rapidly with the decrease of virtual clients, for instance at 1 clients it was recorded at 0.26.

The probability of zero jobs in the system is larger compared to previous two configurations however it remains astonishingly low, so is the probability of queuing, which is very low, in contrast the measurements results where there are always jobs available in the queues and in the system.

The number of jobs in the system and in the queues from model's estimation were higher than the measured values of the same value. The number of jobs in the queue was modeled at the very low value. This matches the behavior of the real system which contained fewer jobs in the queue than previous configurations. This due to the fact that model still assumes the jobs spend a lot of time in the worker threads, which is less in the real system at his low mark of clients.

The values for response time was modeled lower than the actual values from measurements. This is because to model's assumptions that it takes a lot of time for the memcached service time and that the majority of the response time comes from it, which was actually the case during the real experiment, where queuing time was very low at 9 clients and did not contribute much to the response time.

Metric description	Variable	Estimated results	Measured results
Jobs arrival rate	λ	6333.46	-
Jobs service rate	μ	66	-
Utilization	ρ	0.75	-
Probability of zero jobs in the system	p_0	2.11E-42	-
Probability of queuing	ϱ	0.0011	-
Mean number of jobs in the system	$E[n]$	95.96	16.548 (estimation)
Mean number of jobs in the queue	$E[n_q]$	0.0033	2
Mean response time	$E[r]$	15.15	5.823
Variance of the response time	$Var[r]$	0.23	-
Mean waiting time	$E[w]$	0.00053	0.63
Variance of the waiting time	$Var[w]$	5E-6	-

Table 16: M/M/128 results comparison, VC = 9 configuration

7.2.1 Summary

In conclusion, the modeling with the M/M/128 model demonstrated better results matching than more simple M/M/1 model. The reason for this is the use of more servers to model each worker threads. This fact gave better matching figures in terms of number of jobs as well as the response and waiting times.

On our input parameters, unlike the M/M/1 model which assumed the majority of the response time comes only from waiting time, the M/M/128 model considered the fact memcached service time could be the largest contributor to the response time. In other words the M/M/1 model solely assumes the majority of the jobs should be located in the queue because it takes very little for the server to process the jobs, in contrast M/M/128 demonstrates that in some cases the majority of jobs could not only be located at the queue but at the worker threads as well.

7.3 Network of Queues

During previous section we tried to perform the modeling with different simple models. We attempted to view the entire system as just a single queue with just one single worker representing all the worker threads, as it was the case during the M/M/1 model. Next we brought our model to the new level representing each worker threads with a server, however still having just one single queue.

Despite of the fact that previous models demonstrated some common patterns with real measurement of the system and shown overall trends with the change of the load and configurations, these kind of simple models are not particularly useful for accurate performance estimations.

Therefore the need for a more complex and accurate model arises. The way to construct a more complex model is to network various types of queues which would take into consideration various components of the system. The input parameters for the queues inside the network were taken from the instrumentation of the middleware from section 3. **Please note that for this section's modeling experiment form section 3 was repeated and the data from the new run was used**. In this section we would first design such a network network of queues, later we will compare the results of such model to real components of the system. At the end an analysis of the utilization of each component as well as the analysis of bottlenecks will be performed with the MVA algorithm.

It was decided to fix the amount of worker threads in middlewares at 64 threads. The modeling for read only and write only workloads were performed separately.

7.4 Single middleware

The network of queues consists of two parts, namely the modeling with a single middleware and the modeling with double middlewares as it was the case during the corresponding experiments in section 3. This section will describe the model which was created from the experiment with single middleware.

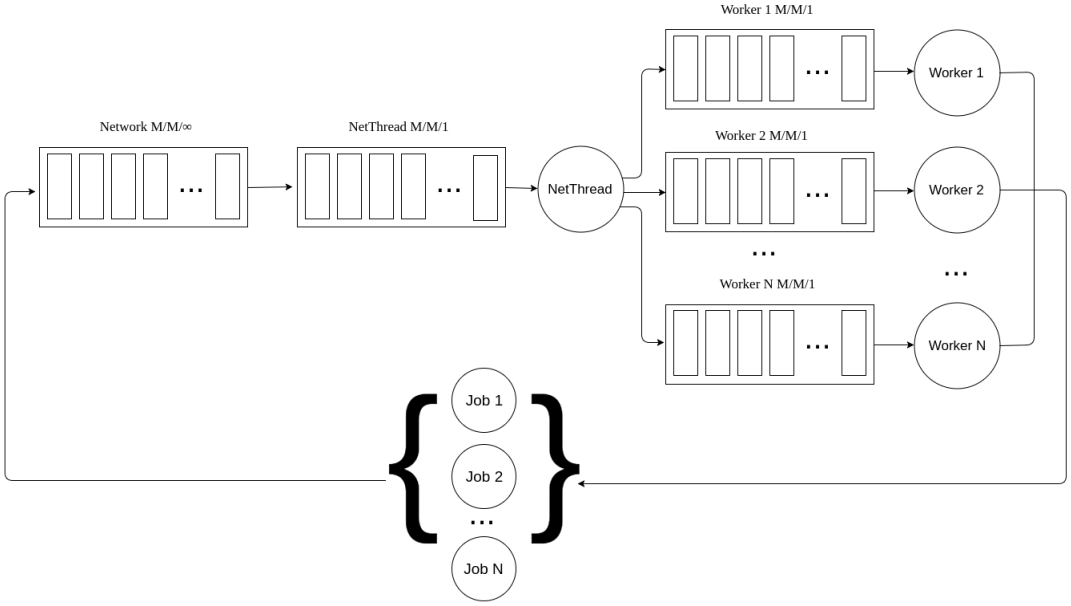


Figure 34: Network of queues modeling, single middleware

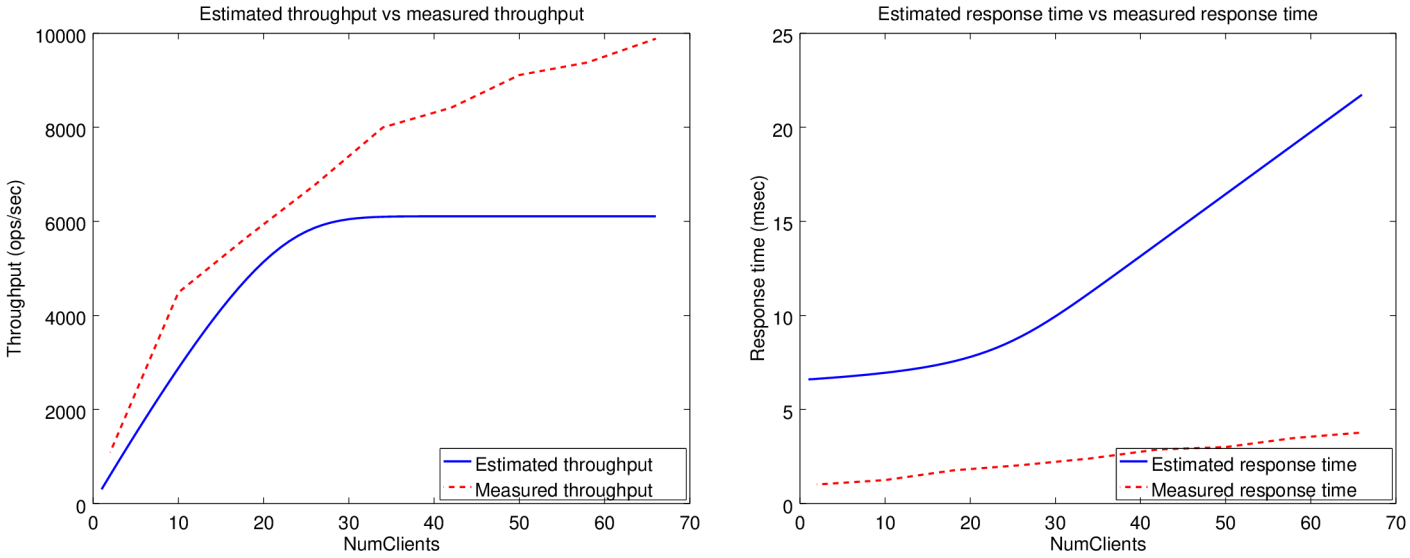
Figure 34 illustrates the general architecture of the proposed model. The model which will be used is the closed model, which means it has no external arrivals and departures. The network component of our system will be modeled as the so called network device, which will correspond to the single M/M/∞ queue. Service time for this device was acquired from the network ping times from that experiment.

From the network queue the jobs will be forwarded to the NetThread queue, which is represents the job queue of our middleware. The best way to model this component is to use the M/M/1 queue, because the NetworkThread object could be represented by one server quite accurately.

From the NetworkThread's queue the jobs will go to worker queues. As it was observed during previous modeling, experiments and from the experiment there is queuing behavior happening in each worker threads. Therefore it was decided to represent each worker as M/M/1 queue. The service time was taken from the instrumentation.

It was decided not to represent the servers as separate queue elements, the reason for this is the fact that the service times of the worker queues contain the part of the service time of the memcached servers. Therefore, the corresponding M/M/1 queues not only attempt to model the workers but also the memcached servers. The main reason for this decision is the instrumentation, the data which was obtained during the experiments matches this set of

components. There are however other more complex ways to model the system, this however requires more scrutinized instrumentation, which is beyond of the requirements scope.



(a) Throughput estimation

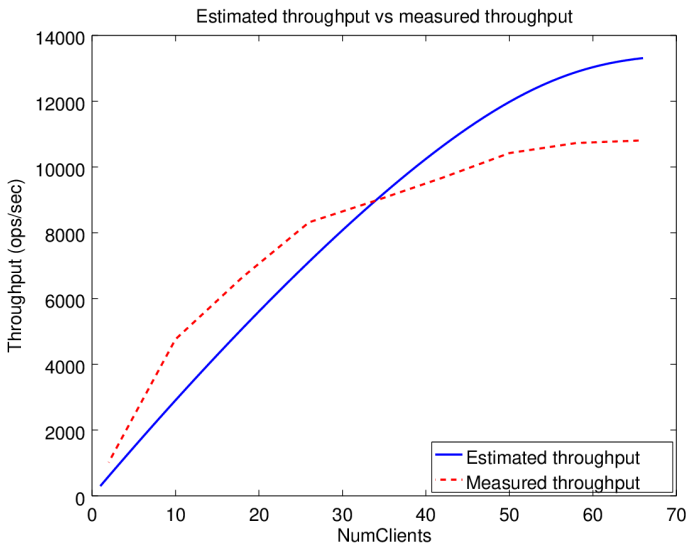
(b) Response time estimation

Figure 35: Estimated throughput and response time, write only workload, single middleware

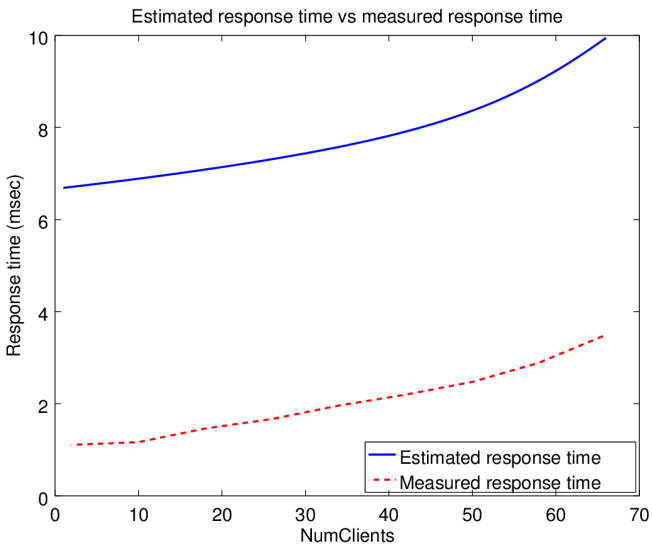
Figure 35 illustrates the results of modeling of the write only workload. One could observe that the figures of estimated throughput and the measured one share similarities before the mark of 30 clients. However the values of throughput begin to diverge after the 30 clients mark, measured curve continues to go up rapidly and still remains the growth even at 70 clients, in contrasts the modeled curve predicts the saturation after the mark of 30 clients.

The same behavior is observed with the response time curves. The theoretical curve estimates the saturation at 30 clients however the real curve continues to grow linearly and does not contain any knees, meaning the system is not close to saturation.

The behavior of lower figures of the model could be explained by the input parameters for the modeling. The parameters which we inputed were taken from the measurements when the system was not any close to saturation. Because of that the model predicted worse performance than in reality, producing overall lower figures of throughout and larger figures of response time.



(a) Throughput estimation



(b) Response time estimation

Figure 36: Estimated throughput and response time, read only workload, single middleware

Further the modeling for the read only workload was performed, the results are shown at Figure 36. Overall, there is a better match of figures of both throughput and response time. The throughput was predicted slightly higher than during the measurements unlike during the write only workload.

The response time figures do not match as accurately as the throughput ones, however two curves maintain identical order of growth. In general, the figures for read only experiment gave better estimation than the write only ones, this is because the input parameters for the model were taken from time when system was saturated, which did not cause underestimation as before.

		Write only workload		**Read only workload**	
Device name	**Queue type**	X_i	U_i	X_i	U_i
Network	$M/M/\infty$	3030	11.42000	6603	24.88388
Network thread	$M/M/1$	3030	1.00000	6603	0.99044
Worker thread N	$M/M/1$	47.348	0.11837	103.17	0.28578

Table 17: Network of queues modeling, single middleware

When it comes to finding the bottleneck of the system we should consider the values of utilization of components, which could be found at Table 17. From the table once could observe that the largest value of utilization came from the Network device at 11.42 and 24.88 for write only and read only workloads respectively. Because this device has the largest value of utilization it is regarded as the bottleneck. Next comes the network threads with the values close to 1 for both read and write workloads. Worker threads have significantly less values of utilization of 0.12 and 0.29 respectively.

7.5 Double middlewares

It is now time to model the second configuration of the system of the same experiment. The corresponding experiment used two middleware instances, this section will describe the proposed

model for the setting.

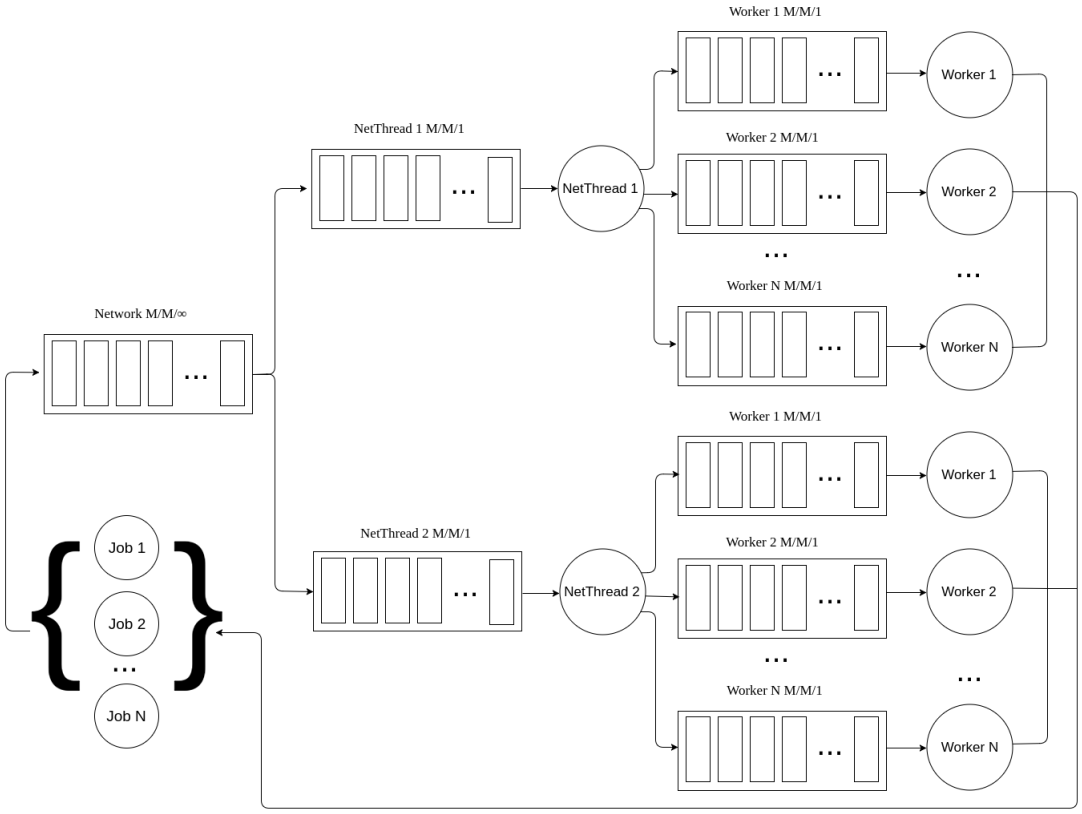
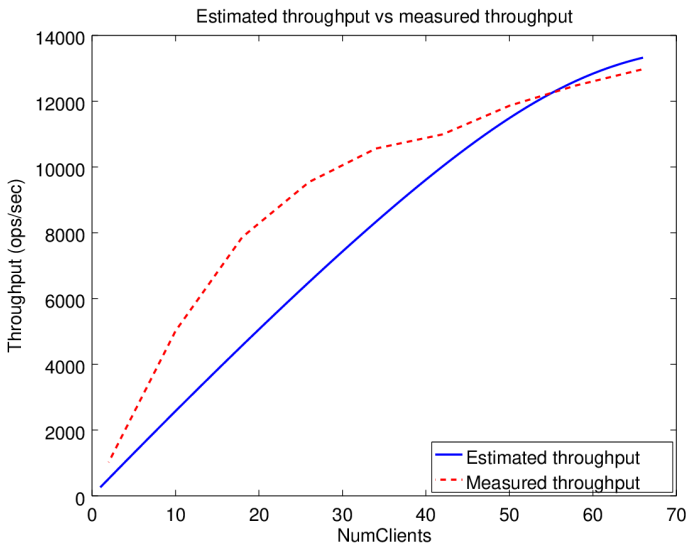


Figure 37: Network of queues modeling, double middleware

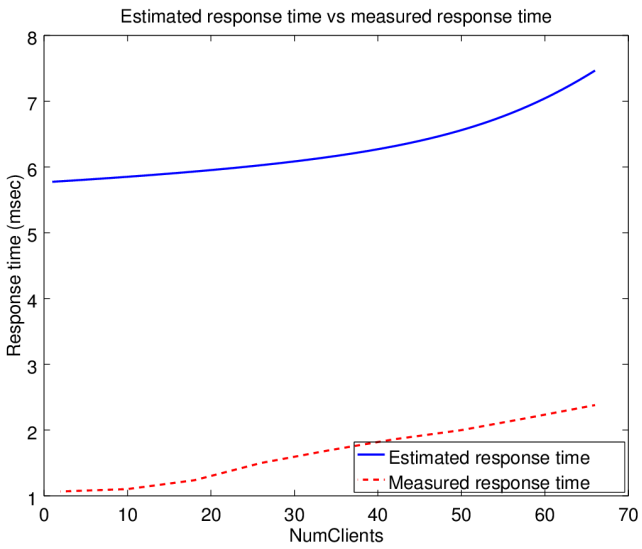
The general architecture of the constructed model is depicted at Figure 37. As previously it was decided to represent the network as the network device, specifically the M/M/∞ queue was chosen to simulate that device. From the infinity queue the jobs will follow into the middlewares. It was decided to model each middleware separately with a M/M/1 instance for each. The choice is motivated by the fact that two middlewares are independent components, which have their own internal queues and not identical figures of internal measurements.

Each middleware's network thread queue would guide the jobs into the worker threads queues, which as previously be modeled as M/M/1 queue for each worker thread. in total we would have 128 M/M/1 queues to represent all the workers threads in each middleware, each having 64 worker threads in the pool. It was decided again not to model the memcached servers as queues because the service times of the worker components already contains part of the service time of the memcached servers. The system is again closed as during the previous section.

Figure 39 demonstrates the results for modeling of the write only workload. The throughput curve predicted lower figures of throughput before 50 clients, from 50 clients the throughput values converged for the modeled curve and for the estimated one. The predicted values of response time are predicted higher that the actual ones, however having similar order of growth. Overall there is a match in the values, however from the model's figures we can see that the system begins to saturate in contrasts the real figures do not show signs of saturation.



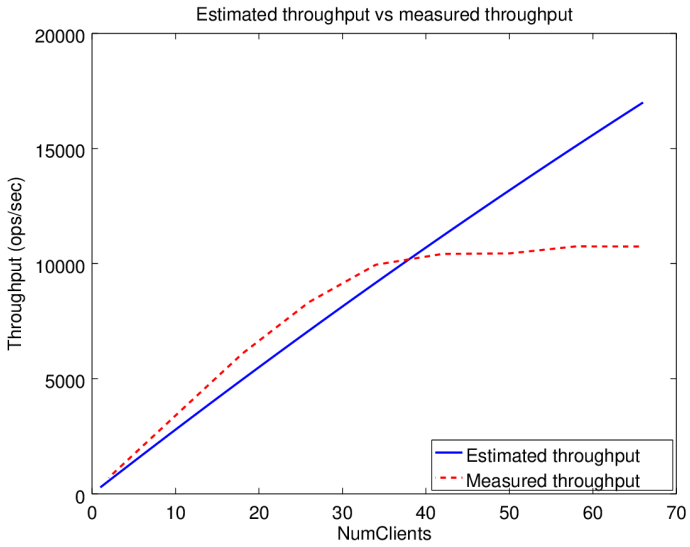
(a) Throughput estimation



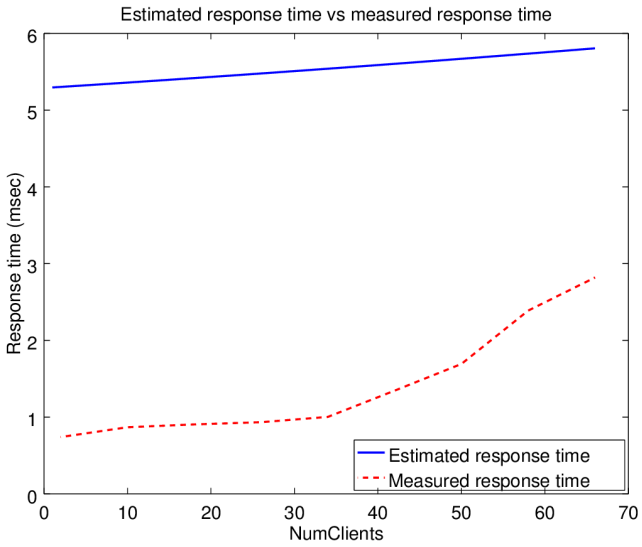
(b) Response time estimation

Figure 38: Estimated throughput and response time, write only workload, double middleware

It was decided to redo the read only experiment from the section baseline with middleware for better modeling. Originally the experiment was performed without proper population of the databases, which resulted in high number of caches misses, which in return enhanced the values of total throughput. The experiment was repeated so we could perform proper modeling.



(a) Throughput estimation



(b) Response time estimation

Figure 39: Estimated throughput and response time, read only workload, double middleware

Figure 39 demonstrates the results of modeling during the read only workload. The throughput number grew linearly exactly as the model predicted before 30 clients. However after the value of 30 the real system began to saturate, in contrast the model still predicted growth of throughput and no knee at response time. This model overestimates system's performance.

		Write only workload		Read only workload	
Device name	**Queue type**	X_i	U_i	X_i	U_i
Network	$M/M/\infty$	8836	33.29782	11272	34.96023
Network thread in MW 1	$M/M/1$	4418	0.96750	5636	0.30792
Network thread in MW 2	$M/M/1$	4418	0.80404	5636	0.13448
Worker thread N in MW 1	$M/M/1$	69.028	0.12977	88.063	0.17914
Worker thread N in MW 2	$M/M/1$	69.028	0.11942	88.063	0.20032

Table 18: Network of queues modeling, double middleware

Table 18 contains the values of utilization for the bottleneck analysis. From the table the most observable device in terms of utilization is the network device which has the largest values of utilization of 33.3 and 34.96 for write only and read only workloads. According to the largest values of utilization this is the bottleneck of the setting. The network threads had the second largest values of utilization , network thread of first middleware had the utilization of 0.97 during write only mode and 0.31 during read only mode. The network thread of the second middleware showed the utilization of 0.8 during write only and 0.13 during read only. The network threads in both middleware showed overall similar utilization. One thing to notice is that during the read only workload worker threads in the second middleware were more of a bottleneck than the network thread in the corresponding configuration. This is explained by the fact that service times of each middleware are not identical because of the cloud environment.

7.6 Summary

In conclusion, the modeling with the network of queues demonstrated better result match than the previous models such as M/M/1 and M/M/128. The reason for better result match is the fact that we model system's components in detail, in contrast to more simple model where the entire system was regarded as just one queue.

The network of queues model predicted the network as the largest bottleneck, however during the experiments we could see that often other components were the bottlenecks. The second largest bottleneck was the network thread, this behavior was frequently seen during the experiments when the queues were full and the system was saturated.