Advanced Systems Lab (Fall'16) – Third Milestone

Name: *Rishu Agrawal*

Legi number: *15-945-355*

Grading

Section	Points
1	
2	
3	
4	
5	
Total	

MODEL ASSUMPTIONS

In order to provide a solid basis for the models, certain assumptions have been made for the analysis. In reality, these assumptions hold too, given how the system is designed. So before delving into the specifics of each model and juxtapose them to the observed measurements, I would like to list the major assumptions that have been made. Of which, these are:

- **Closed System assumption:** There are only a finite number of clients in the system, and all of them generate a request which is also present within the system. If there are *N* clients generating the workload, there are always *N* requests circulating within the system. This assumption is true by design of the workloads. The number of clients in the system are fixed and each client waits for a response before generating the next request. As a result, the throughput = arrival rate.
- **Invulnerable Requests:** Requests never fail. A request generated by the client is always serviced, and never gets dropped.
- **FCFS discipline:** All the queues that are being modelled are serviced using an FCFS discipline.
- **Infinite buffer for all queues:** None of the queues modelled have a finite buffer. This implies, each queue can hold an infinite number of jobs.
- **Infinite capacity:** The system is designed to cater any number of clients and all the data utilized for modelling was using this assumption.

1 System as One Unit

An M/M/1 queue is commonly used to model single processor systems with the underlying assumption that interarrival times and the service times are exponentially distributed and there is only one server. But as is the case in our middleware system, neither do we have a single server, nor is the system a single processor system. In the case of the stability trace, we use a machine with 8 cores and at most 27 worker threads are working concurrently. I do not expect the model to be able to explain the real-life behavior of my middleware implementation as it does not account for the multiple memcached servers running in parallel or the usage of concurrent threads. This is a crucial aspect in explaining the performance of my middleware system as the service rate is majorly impacted by CPU-resource sharing and parallelism which the M/M/1 queue model does not capture. Another key aspect that the model is incapable of explaining is the difference in performance by the different worker threads (SET and GET) as the SET worker threads are asynchronous and the GET worker threads are synchronous. The model described also can't differentiate the performance impact of each type of request separately.

I model my middleware system as a M/M/1 queue as shown in *Figure 1*:

- Since I don't have the necessary logs to precisely derive the time spent in the network, I model the system by representing it as a black box.
- The T_{server} calculated in the Middleware logs is inclusive of the network time taken by the worker thread to send and receive the response time from the memcached servers. It is an upper-bound on the service time of the model.
- The time required for hashing and parsing the requests are assumed to be negligible.
- The memaslap clients generate the requests that are pushed to the *request queue*.
- The *service provider* then pulls a request from *the request queue* sequentially, processes it and returns the response to the respective Memaslap client.

In order to make comparisons between the modelled parameters and the actual observations, I use data from the ***trace1*** experiment. The first thing I did was to verify whether Little's Law was valid for the real-life measurements. According to the output from memaslap clients, the real-life data looks as

follows:

Aggregated average throughput (Arrival rate) = 12025 ops/sec

Average response time = 16046.67 microseconds

By Little's law,

Mean Number of Jobs in System = Arrival rate X Mean response time

Thus we have, Mean Number of Jobs in System = 12025 * 0.016046.67 = 192.9 which looks reasonable after accounting for the approximations in the memaslap measurements. The number of jobs has to be 192 as we used 64 virtual clients per memaslap machine and we have 3 memaslap machines and the system is a closed system. Now, since the M/M/1 queue model also follows the Little's law, we can use this information to estimate the other parameters in the model.

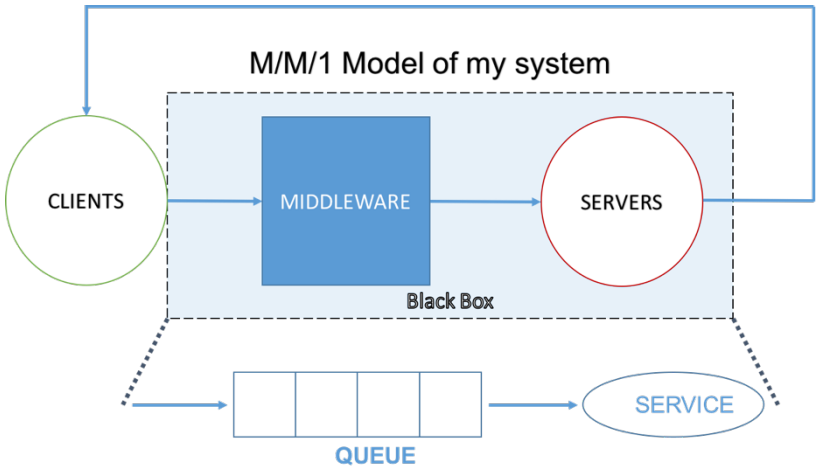


Figure 1. Middleware system modelled as a M/M/1 queue using a black box approach

Input to the Model: Mean Arrival Rate (λ) = 12025, Mean number of jobs in system ($E[n]$) = 192

Table 1: Analysis of modelled and real-life behavior of system for trace1 experiment

Parameters	**M/M/1 queue**	**Observed by experiment**
Traffic Intensity ρ	0.994819	-
Service rate μ	12087.63 ops/sec	>= 501.32 ops/sec
Mean Service Time $E[S]$	82.73 microseconds	<= 1994.72 microseconds
Mean # jobs in the System $E[n]$	192	192.9
Mean # jobs in the queue $E[n_q]$	191	>= 165
Mean Response time $E[r]$	16046.67 microseconds	16046.67 microseconds
Mean Waiting time $E[w]$	15963.94 microseconds	>= 14041.95 microseconds

As, it can be seen from the Table above, the traffic intensity $\rho = 0.99$ is less than 1, meaning that the system is stable. In the case of the M/M/1 queue model, once we have 2 input parameters, the other

metrics are computed directly using the formulae from the book. In the case of the experimental data, I need to make some analytical decisions. Since I don't have an exact estimate on the parallelism factor, I use bounds to express it. I know that at most, at a given time, 27 worker threads can work concurrently, thus the minimum number of jobs that have to wait in the queue are 192 – 27 = 165. The reason that I say at most 27 threads may work at the same time is because the load may not be perfectly balanced and all read threads from the thread pool may not be in use at the same time. Similarly, since T_{server} was an upper bound on the service time for a single request, the mean waiting time for each request has to be at least $E[r] - T_{server}$ = 14041.95 microseconds.

Comparing the model and the real-life behavior of the system, I can tell that the similarities end after the verification of Little's Law. The service time observed by the system in reality is greater than the service time given by the model by a factor of almost **24.** This is because of the fact that the M/M/1 model does not account for the concurrent threads and in the system. However, it is in line with the notion that this number should be less than 27. The semantic interpretation of this number is that in our system, on an average, at most 24 requests are processed concurrently. However, this number has to be taken with a pinch of salt as it has been inflated by including the network time. But it is helpful in highlighting the deficiencies of the M/M/1 queue model in capturing the scale and concurrency of my middleware system as it is too simplistic.

2 Analysis of System Based on Scalability Data

An M/M/m queue is commonly used to model multi-processor systems or devices that have several identical servers with the underlying assumption that interarrival times and the service times are exponentially distributed and there is only one queue for waiting. Since the middleware runs on a multiprocessor system (8 cores) with several threads running concurrently, the M/M/m queue model is certainly a better candidate than the basic M/M/1 queue model for my system. However, I do not expect the model to be able to completely explain the real-life behavior of my middleware implementation for all configurations as it treats all the "m" servers or devices to be identical. This is usually not the case as the performance of each device is dependent on the load each device receives and usually have a variation in performance. This is a crucial aspect in explaining the scalability of servers for my middleware system as the service rate is majorly impacted by CPU-resource sharing and context switching and not all "m" servers scale as expected. Also, if we consider each worker thread to be a separate server or device, the differences in the performance of SET worker threads and GET worker threads will be considered to be identical by this model which is obviously not true as seen in Milestone 2. Moreover, the M/M/m queue model assumes a single queue which is again not apropos to explain my system as both the SET and GET requests have different T_{server} and T_{queue} times because of their asynchronous and synchronous nature.

I model my middleware system as a M/M/m queue as shown in *Figure 2*:

- Since I don't have the necessary logs to precisely derive the time spent in the network, I model the system by representing it as a black box.
- The T_{server} calculated in the Middleware logs is inclusive of the network time taken by the worker thread to send and receive the response time from the memcached servers. It is an upper-bound on the service time of the model.
- The time required for hashing and parsing the requests are assumed to be negligible.
- The memaslap clients generate the requests that are pushed to the *request queue*.
- One of the free "m" *service providers* then pulls a request from *the request queue* sequentially, processes it and returns the response to the respective Memaslap client.
- I choose m = Number of servers * Number of worker threads since each thread has a connection to the memcached server and works independently to serve requests and can be abstracted as a single device in the model.

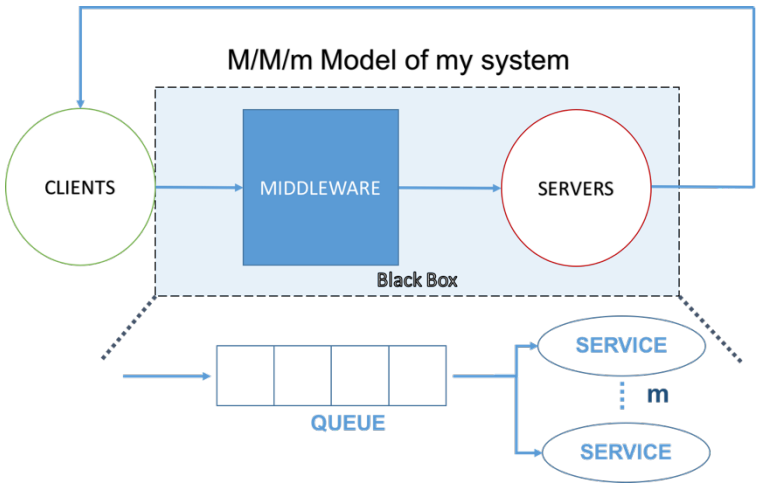


Figure 2. Middleware system modelled as a M/M/m queue using a black box approach

In order to make comparisons between the modelled parameters and the actual observations, I use configurations from the ***Maximum Throughput*** and ***Effect of Replication*** experiments from Milestone 2. The first thing I did was to verify whether Little's Law was valid for the real-life measurements. As was the case above, the Little's law is indeed valid for all the experiments. I shall divide my analysis in two parts for each experiment type.

2.1 M/M/m models from "*Maximum Throughput*" Experiments

In this section, I use configurations from the ***vclientexp*** experiment for 16 threads and run additional experiments for some configurations to get data for the following experiments:

Number of Memcached Servers	5
Number of Client Machines	3
Virtual clients / Machine	**2, 4, 8, 16, 24,** 32, 64, 80, 96, 112, 128, 144, 192
Workload	Key 16B, Value 128B - Small, Writes/Reads Ratio: 0.00/1.00
Replication (Rep)	No Replication (R=1)
Number of Threads in Thread Pool	16
Runtime x repetitions	300s x 10
Log files	**newvclientexp**, vclientexp

The reason that I choose these experiments is because from Milestone 2, we know the configuration at which the throughput saturates and we know the maximum throughput for this configuration. Using this info, we can now say that the service rate for the M/M/80 (m = 16*5, since there are no write requests) model = max. throughput/m = 14241/80 = 178.0125 ops/sec. Using this and the information and the mean number of jobs in the system, I try to estimate the arrival rate and mean response time given by the model and compare it to the measured average throughput and average response time by the real data for different number of clients. I get the graph shown in ***Figure 3.***

As we can see from the graph, the estimation from the model and the experimental values are almost similar after the saturation point, i.e. 288 clients. This is the point where the system is running at a traffic intensity of almost 1. When the system is under-saturated, the predicted arrival rate or throughput from the model varies by about 2000 ops/sec and the mean response time varies by about 2-3 milliseconds. The reason that the model represents the real life data as well as it does in this situation is largely because of two reasons:

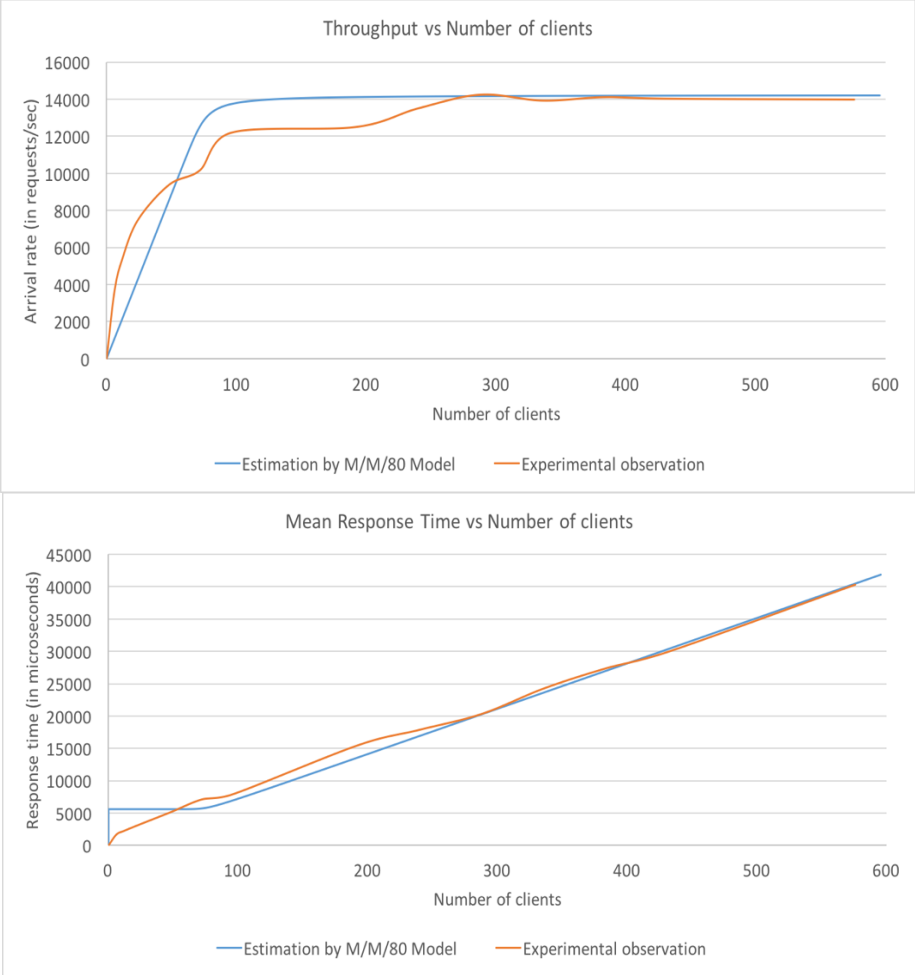


Figure 3. The estimated and actual throughput and mean response time

- This configuration of experiments only has synchronous GET requests and no SET requests. This means that if the load balancing is equal, the "m" servers in the M/M/m model are identical.
- The number of memcached servers are unchanged. Therefore, the sharing of resources affects all the configurations equally and despite the change in number of requests in the system, the usage of the same service rate of 178.0125 ops/sec is justified.

2.2 M/M/m models from "*Effect of Replication*" Experiments

In this section, I use configurations from the ***repexp*** experiment for 16 threads, 80 virtual clients, no replication (R=1) and number of memcached servers = 3, 5 and 7. I choose these experiments is for 2 reasons. Firstly, I want to show that increasing the number of servers (or m) while modelling my middleware system as the M/M/m queue also changes the service rate for each configuration and a single service rate can't be used to model systems with more servers. Secondly, I try to explain the shortcomings of the modelling the entire system with GET and SET requests as a single M/M/m queue.

In the models for these configurations, we cannot choose the service rate from the section above as our system now also has SET requests and has a write % of 5%. Also, in the M/M/m models constructed from these experiments, m = Memcached servers * (16 + 1):
Input to the Model: Mean Arrival Rate (λ), Mean number of jobs in system ($E[n]$)

Table 2: Analysis of modelled and real-life behavior of system for experiments, the rates are in ops/sec and time in microseconds

	λ = 12877.2, E[n] = 239.53		λ = 12648.8, E[n] = 238.91		λ = 12162.2, E[n] = 239.78	
Parameters	**M/M/51 queue**	**Observed by experiment**	**M/M/85 queue**	**Observed by experiment**	**M/M/119 queue**	**Observed by experiment**
Traffic Intensity ρ	0.994956	-	0.993988	-	0.992618	-
Service rate μ	253.77	>= 463.84	149.70	>= 470.26	102.96	>= 476.85
Mean Service Time $E[S]$	3940.51	<= 2155.88	6679.60	<= 2126.47	9712.19	<= 2097.08
Mean # jobs in the queue $E[n_q]$	189	>= 189	155	>= 155	121	>= 121
Mean Response time $E[r]$	18601.40	18601.40	18888.18	18888.18	19715.24	19715.24
Mean Waiting time $E[w]$	14660.88	>= 16455.42	12208.56	>= 16761.71	10003.05	>= 17618.16

As in the case of M/M/1 queue model, I use bounds to express the parameter values of mean service and waiting time and mean number of jobs in the queue and system for real-life data. If we analyse the values from ***Table 2***, we can observe that for the M/M/m models above the service rate(μ) changes for every configuration as "m" changes whereas in the case of the experimental observation, this value is almost constant at around 470 ops/sec. If we choose a constant service rate in an M/M/m model from, say m=51, and use E[n] = 240 to try to predict the arrival rate(λ), we will see that as "m" increases, λ also increases leading to perfect speed-up. This behavior can be seen in the table below where I predict the λ for models with m = 85 and 119 (i.e. 5 and 7 memcached servers) using μ = 253.77:

λ (in ops/sec)	m = 85	m = 119
Predicted	21441 ops/sec	29976 ops/sec
Measured	12648.8 ops/sec	12162.2 ops/sec

Because the M/M/m queue model assumes that all the "m" devices are identical, increasing m for a fixed service rate leads to an increase in λ. In the case of the real life system, things aren't so straightforward. The "m" threads aren't the same as each other. The worker threads consist of the synchronous read threads and the asynchronous write threads. Moreover, the load on each of these threads is different. The asynchronous writer threads in this set of experiments satisfy 5% of the total requests and and the synchronous read threads satisfy 95% of the total requests. We also know that the write requests don't spend time in the queue and are immediately forwarded for service to the memcached server which is a stark contradiction to the M/M/m queue design which assumes that a request stays in the queue until the one of the "m" servers is free to serve it. In other words, the M/M/m model assumes a synchronous design.

Another thing to note is that the M/M/m model attributes this service rate to all the "m" devices which means that all the devices are active at a given instant. However, in the case of the real life system, this is not the case entirely. As seen in Milestone 2, CPU sharing and context switching plays a crucial part in the performance of the system as increasing the number of memcached servers (and thus the total number of threads in the middleware) does not lead to an increase in performance. In fact, the throughput almost remains constant and the speed-up factor is almost 1 instead of being something much greater than 1. This is also the reason why the mean service time (E[S]) for the actual system is

almost same for all the 3 configurations at around 2100 microseconds. In order to estimate the the slowdown experienced by the threads due to the CPU sharing and context switching, I use the product of m and service rate estimated by the M/M/m queue model and divide it by the actual service time. This number gives us a bound on the expected effective number of threads that are active in the system. This can also be termed as the *parallelism factor*. These values are shown in the table below:

λ (in ops/sec)	m = 51	m = 85	m = 119
Expected number of effective threads	27.90	27.05	25.69
Model E[s]/Actual E[s]	1.82	3.14	4.63

One thing to note is that this number doesn't mean that out of 51 threads, 27 threads are active. It has to be interpreted as number of concurrent requests that are being served at a given instant, since there are several factors at play. The writer threads are asynchronous and we don't have an estimate of how many requests are being satisfied concurrently by each instance. Also, the % of writes is very low compared to the % of reads and it is very likely that a majority of the 28 requests being served are reads. Since this number is same across all the configurations, it is a direct effect of the increase in servers causing no benefit in performance of the system in real life due to the overhead caused by the CPU-sharing policies and context switching in the middleware because of the large number of threads.

Thus, the learnings from the M/M/m queue to model my system can be summarized as follows:

- They work well when the Middleware only has synchronous GET requests.
- For a fixed m and same request type, the M/M/m queue model is acceptably capable to capture the effect of scalability in the number of clients and predict the behavior of the system.
- The predictions become more confident in the regions of saturation.
- However, they aren't capable of predicting the true behavior of the system caused by the scalability of the memcached servers when both SET and GET requests are present without knowing the arrival rate, i.e. a fixed service rate cannot be used to predict the behavior of my middleware system.
- When the arrival rate is known, the M/M/m queue model can be used to get a bound on the parallelism factor in the middleware.

3 System as Network of Queues

In this section, I build a network of queues for the whole system and use configurations from the ***writeexp*** experiments from the ***"Effect of Writes"*** section from Milestone 2 to compare the results. I use data for the case of 5 memcached servers, no replication and 1%, 5%, 10% writes. *Figure 4* manifests the queueing network architecture that I use to model my system

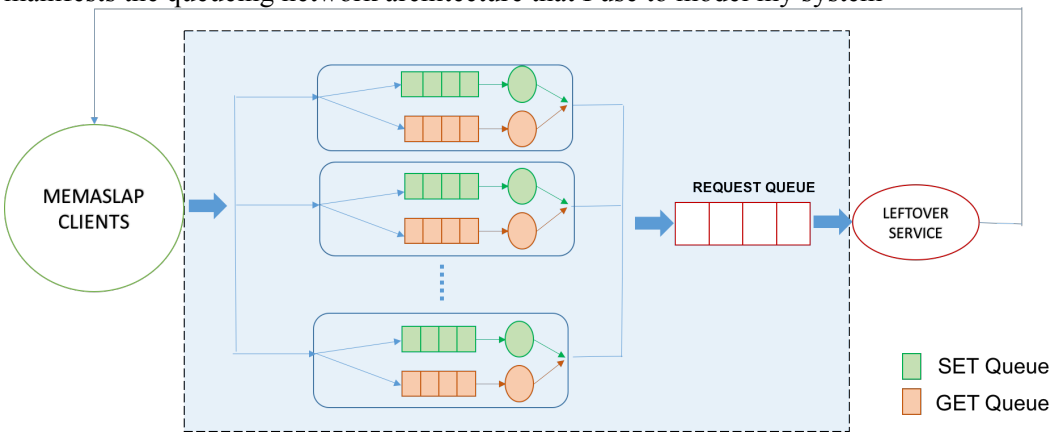


Figure 4. Modelling whole system as network of queues

Due to the limited measurements from the Middleware logs, I had to abstract several components from the queueing network. There are no provisions that allow me to compute the waiting time of a request before it is read by the main thread (Manager Class[1]), the time required for parsing and hashing, and the time a request spends waiting until it is sent back to a Memaslap client directly. However, I can jointly calculate the time a request spends while it is parsed, hashed and waits before it can be written back to a memaslap client: I call this time T_{lag} and it can be abstracted as $T_{lag} = T_{mw} - T_{queue} - T_{server}$.

Next, I try to estimate the time a request spends in the network when it is sent from the Memaslap clients to the Middleware and vice-versa by the main thread. I do this by calculating $T_{Network}$ which I define as the joint time a request spends in a network when it is sent from Memaslap clients to the Middleware, the time it waits in the network queue before it is read by the main thread and the time it spends in the network when it is sent back from the Middleware to the Memaslap client. This gives: $T_{Network} = Memaslap\ Response\ time - T_{mw}$.

I conceptualize $T_{Network}$ as a metric that is similar to the think time Z since each request spends this time waiting before it is actually received by the Middleware. Since parsing, hashing and writing to the Memaslap clients are done by the main receiving thread (in the Manager Class[1]), I combine them and term them as ***"Leftover Service"***. I model the Leftover Service as a M/M/1 queueing model, where $E[r] = T_{lag}$ and the arrival rate (λ) is the measured throughput, since all the requests undergo parsing and hashing and are sent back to the Memaslap clients. Based on these two parameters, I estimate the Mean Service Rate for Leftover service.

In addition to this, I also use the M/M/1 queueing model to model the GET operations. From previous sections, it would seem that the most obvious choice would be to model it as a M/M/m queue, where m is number of threads in the thread pool. However, as seen from before, not all the threads in the thread pool are utilized and the ***"effective"*** number of threads used is unknown. The T_{server} for GET requests represents the service time without parallelisation. Since, I do not know the "effective" number of threads used in the thread pool I can't use T_{server} for GET requests directly as the service time and model it as M/M/m, since m is still unknown. However, I know the T_{queue}, which takes into consideration the average number of threads utilized in the thread pool because if more threads are utilized, the T_{queue} time decreases since the requests shall then be pulled from the queue by more

threads more frequently. Thus, I can model the GET operations as an M/M/1 model and use: $E[r] = T_{queue} + T_{server}$. Moreover, the arrival rate can be chosen as the product of measured throughput and the visit ratio.

Moreover, to ensure that modelling the GET device as an M/M/1 queue is the right choice, I ran the **M**ean **V**alue **A**nalysis (MVA) algorithm modelling GET requests as M/M/1 and as M/M/m (m = number of threads in the thread pool) and compared the results. The results, using M/M/1 model for GET devices was as expected, whereas modelling it as an M/M/m queue resulted in very low utilization numbers.

I also model SET operations (device) as an M/M/1 queueing model (since it has one thread per server). The T_{server} time (for SETs only), doesn't include an overhead introduced by the CPU context switches (the starting and stopping of threads, due to inactivity), however, the T_{queue} time does include it. Therefore, instead of using T_{server} as the E[S] in M/M/1 model, the better approach is to calculate the Mean Service Time using $E[r] = T_{queue} + T_{server}$ and use the arrival rate as the product of measured throughput and the visit ratio. Thus, all devices (SET, GET, leftover) are modeled as an M/M/1 queueing model.

Since, in this section, I use a configuration with 5 memcached servers, I have 5 SET and 5 GET devices in the queueing network (M/M/1 models), each associated with a memcached server and one Leftover service. To run the MVA algorithm and find a bottleneck device, I need the visit ratios and Mean Service Time for each device (***Table 3***). The visit ratios for SET are calculated as $\frac{1}{5}$*$\boldsymbol{write\ Percentage}$ and for GET as $\frac{1}{5}$*$\boldsymbol{Read\ Percentage}$ since we assume equal load balancing. I calculate the Mean Service Time for each device based on the respective E[r], and throughput of the device as $\lambda * visit\ ratio$ - where arrival rate (λ) is the measured throughput. The results of the calculations for the GET, SET and Leftover device in the case of 5 memcached servers, 1% writes and No replication are presented in ***Table 3***.

Table 3: Measured Data with predictions using M/M/1 queueing model for each device

Parameters	Devices		
	i_{th} SET	i_{th} GET	Leftover
Visit Ratio	(1/100)*(1/5)	(99/100)*(1/5)	1
TPS per instance (TPS*Visit Ratio)	24.66	2441.57	12331.2
$E[r]$ $(in\ \mu s)$	1831.50	6203.464	8863.264
Estimated μ (Ops/Sec)	570.66	2602.77	12444.02
Utilization ρ	0.0432	0.9380	0.9909
Estimated $E[S]$ $(in\ \mu s)$	1752.35	384.20	80.35
Estimated $E[n]$	0.045	15.149	109.292

Based on the estimated Mean Service Time and visit ratios for each device (depicted in Table above), using think time as $Z = Memaslap\ Response\ time - T_{mw} = 0.008976756\ \mu s$, and N=240 (80 clients * 3), I use the MVA algorithm and present the results in the ***Table 4***, below.

Parameters	i_{th} SET	i_{th} GET	Leftover	Whole System
Utilization (U)	0.04336	0.94131	0.99429	-
Average # of Jobs (Q)	0.045331	15.123104	97.081311	128.123492
Response Time R (μs)	1831.775	6172.668	7846.116	10354.424
Throughput X (Req/Sec)	24.7475	2450.0107	12373.791	12373.7916

Table 4. MVA results for the case of 5 Memcached Servers, 1% write, No replication (Based on data from writeexp)

As it can be seen from the ***Table 4*** above, the Utilization and number of jobs in the device, for each Device (SET, GET, Leftover) is close to the ones computed in ***Table 3***. In addition, the system throughput of 12373.7916 is approximately the same as the measured one for this configuration, i.e. 12331.2. Moreover, the system response time of 10354.424 microseconds is approximately the same as the T_{mw} time reported from the Middleware logs, i.e. 10692.644 microseconds. This is in accordance with my logic, since I evaluate the think time Z as the difference between the response time reported by the Memaslap clients and the Middleware logs, suggesting that abstracting the network time between the middleware and memaslap as think time is a good decision for model. The response time for the SET device is exactly the same as the measured one ***(Table 3)***, while for GET device the difference is approximately 31 microseconds. Based on the utilization values of the devices from the ***Table 4***, the leftover device is the bottleneck device, since it has the highest utilization of 99.429%. This means that the main thread (Manager Class[1]) is the bottleneck.

This makes total sense since all the arriving requests have to go through the single main thread, first to be read and then, once it is processed by the GET/SET threads, to be sent back to the memaslap clients. The GET device also has a high utilization, which is expected since the workload configuration for these experiments is 80 clients per memaslap machine, which is close to the workload (96 clients) achieving the maximum throughput (Milestone 2 Section 1). This implies that the system is saturated and GET devices are working at nearly full capacity. In conclusion, the queueing network architecture described in this section seems to be a good fit with respect to the actual Middleware design.

To further illustrate that the bottleneck device is truly the Leftover device, I additionally run the MVA algorithm for 5 Memcached Servers with 5% and 10% write percentages and present the utilization of the i_{th} GET, SET and Leftover device in the Figure below. In order to achieve the MVA results for 5% and 10% write configurations, the same steps were followed as in the case of 1%, which is described above. As it can be seen from ***Figure 5***, as the write percentage is increased, the utilization for i_{th} SET device increases, almost by a factor of the write percentage. This is in accordance with my logic, since the SET device now has more requests to process and thus the utilization increases. The utilization for the i_{th} GET device decreases, since there are less read operations to process, but not very significantly. This is due to the fact that the proportion of the GET requests that arrive in the system is still very high and thus utilization does not decrease significantly. Finally, the utilization of the leftover device, which is the bottleneck device, is at 99% and stays around the same value. This is due to the fact that the total number of requests (workload) is kept the same, and since all requests have to go through the bottleneck device, the utilization doesn't change.

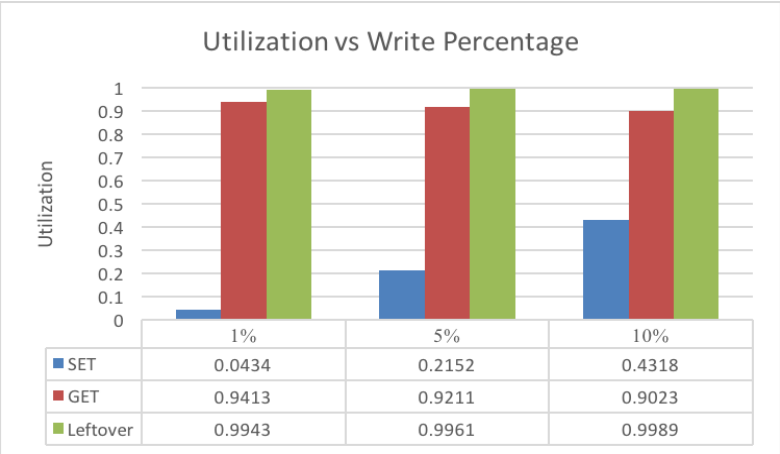


Figure 5. Utilization of i_{th} SET, GET device and Leftover device for different write percentages

In conclusion, the network of queues model enables a good analysis of the system, identifying the bottleneck of the system, i.e. the single main receiving thread (Manager Class[1]).

4 Factorial Experiment

To design a 2^k factorial experiment, I conducted a series of new experiments. The table below summarizes the parameters used in these experiments.

Number of Memcached Servers	5
Number of Client Machines	3
Virtual clients / Machine	**32, 96**
Workload	**Key 16B, Value 128B - Small, Key 16B, Value 512B - Large, Writes/Reads Ratio: 0.01/0.99, 0.10/0.90**
Replication (Rep)	Full Replication (Rep=5)
Number of Threads in Thread Pool	16
Runtime x repetitions(r)	180s x 5(r)
Log files	**factorialexp**

In these experiments, I vary 3 parameters, thus I introduce three factors (k=3) each with two levels. I chose the number of virtual clients and the write % as two of my factors because it exhibits a monotonic behavior for the experiments I conducted thus far. I choose to explore the effect of the request size mainly because it was recommended and it peaked my curiosity. However, I don't expect it to be a significant factor. The effect of each of the factor on the throughput is unidirectional. I define the three variables and their respective levels as shown below:

1. Number of Virtual Clients per Memaslap Machine (x_A)
    a. x_A = -1 if Number of Virtual Clients is 32
    b. x_A = 1 if Number of Virtual Clients is 96
2. Request Size (x_B)
    a. x_B = -1 if Requests Size is Small
    b. x_B = 1 if Request Size is Large
3. Percentage of Write operations (x_C)
    a. x_C = -1 if Percentage of Write Operations is 1%
    b. x_C = 1 if Percentage of Write Operations is 10%

The performance Y, representing the throughput of the System in Operations/Second, can be regressed on x_A , x_B and x_C using a nonlinear regression model of the form:

$$\widehat{Y_i} = q_0 * 1 + q_A * x_{Ai} + q_B * x_{Bi} + q_C * x_{Ci} + q_{AB} * x_{Ai} * x_{Bi} + q_{AC} * x_{Ai} * x_{Ci} + q_{BC} * x_{Bi} * x_{Ci} + q_{ABC} * x_{Ai} * x_{Bi} * x_{Ci}$$

where q represents the coefficients of respective factors. The table below is the sign table used to calculate the effect of the factors. The table presents the measured mean throughput for all combinations of the two levels of the three existing factors by averaging the throughput across 5 iterations (for each experiment).

Table 5: Sign Table Method of Calculating Effects

I	x_A	x_B	x_C	x_Ax_B	x_Ax_C	x_Bx_C	$x_Ax_Bx_C$	Mean Y
1	-1	-1	-1	1	1	1	-1	10173.2
1	-1	-1	1	1	-1	-1	1	9836.8
1	-1	1	-1	-1	1	-1	1	10029.2
1	-1	1	1	-1	-1	1	-1	9209.4
1	1	-1	-1	-1	-1	1	1	13628.2
1	1	-1	1	-1	1	-1	-1	12674.4
1	1	1	-1	1	-1	-1	-1	13094.2
1	1	1	1	1	1	1	1	11623.2
q_0	q_A	q_B	q_C	q_{AB}	q_{AC}	q_{BC}	q_{ABC}	
11283.5	**1471.4**	**-294.5**	**-447.6**	**-101.7**	**-158.5**	**-125.1**	**-4.225**	Total/8

The last row of the table above, shows the calculated coefficients for each of the factors and the interaction between the factors, q_0 being the mean throughput of all experiments. The sign of the calculated coefficients for each of the factors tells us what kind of effect the factors have on the system. As seen from the table above, q_A has a positive value meaning that the increase in the Number of Virtual Clients results in the increase of the throughput (Operations/Second), this observation is expected and has been explored in detail in the Milestone 2, Section 1(Finding Maximum throughput by varying # of Virtual Clients and threads in the thread pool). The rest of the factors and the interaction between these factors have a negative impact on the throughput of the system since all the respective coefficients have a negative value. The next step is to find out the importance of each factor and the importance of the interaction between different factors. The importance of a factor is measured by the proportion of the total variation in the Mean Y (throughput) accounted towards the factor. To quantify the importance of factors, I calculated the variation for each of the factor and summed them up to get the Sum of Squares Total (SST), based on the formula in the book:

$$SST = SSA + SSB + SSC + SSAB + SSAC + SSBC + SSABC + SSE = 2^3 * r * q_A^2 + 2^3 * r * q_B^2 + 2^3 * r * q_C^2 + 2^3 * r * q_{AB}^2 + 2^3 * r * q_{AC}^2 + 2^3 * r * q_{BC}^2 + 2^3 * r * q_{ABC}^2 + \sum_i^8 \sum_j^r (Y_{ij} - \widehat{Y_i})^2$$

where, Y_{ij} is the j[th] repetition of the i[th] experiment, $\widehat{Y_i}$ is same as the Mean Y for each experiment, $r = 5$ is number of repetitions for each experiment and finally SSE represented by the last term in the formula is the Sum of Squared Errors.

Based, on this Formula I calculate the variation due to each of the factor and get the percentage of variation due to each of them (presented in the Table below).

Table 6: Percentage of Variation due to each of the factor and factor interaction

x_A	x_B	x_C	x_Ax_B	x_Ax_C	x_Bx_C	$x_Ax_Bx_C$	Error
84.01%	3.37%	7.78%	0.40%	0.98%	0.61%	0%	2.86%

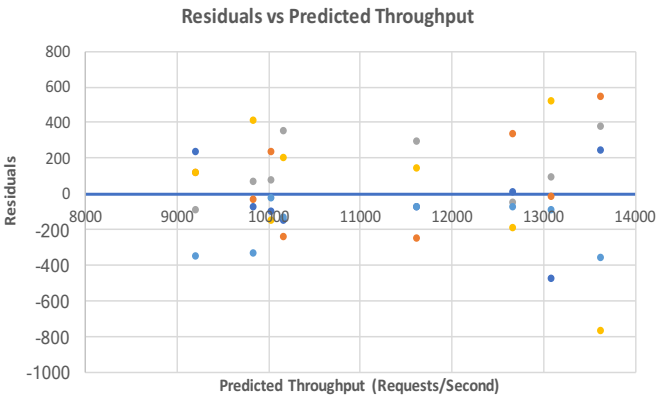
As it can be seen from the table, factor x_A, Number of Virtual Clients per Memaslap Machine, explains 84% of the existing variation, making it an important factor and worth exploring further on. The next factor with the highest percentage of variation– 7.78% - is x_C, the Percentage of Write Operations. This indicates that this factor also has an impact on the performance of the system (throughput), though not as significant and is much less important than Number of Virtual Clients. The rest of the factors and interaction between the factors have negligible contribution to the variation and thus are not significant parameters and can be disregarded (including the Request Size parameter). Moreover, since the computed coefficients (the effects) for factors are random variables, I calculate 90% confidence interval with $2^3 * (r-1) = 32$ degrees of freedom for them. I calculate the standard deviation (needed for confidence interval) of each of the coefficient (effect), based on the formula from the book:

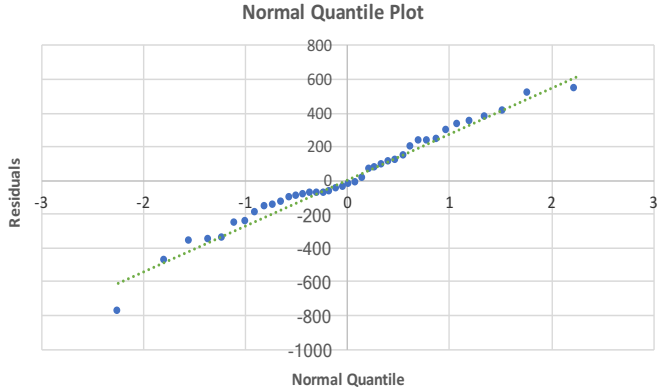
$$S_{q0} = S_{qA} = \cdots = \frac{S_e}{\sqrt{2^3 * r}},\ where\ S_e = \sqrt{\frac{SSE}{2^3 * (r-1)}}\ is\ standard\ deviation\ of\ errors$$

The table below presents the confidence intervals for each of the coefficients. Only q_0 and q_A confidence intervals are within the 10% interval around their respective values, while the confidence interval for the rest of the coefficients lies beyond the 10% interval. Moreover, the confidence interval for q_{ABC} includes 0. This, further suggests the significance of Factor A, Number of Virtual Clients per Memaslap Machine, and the insignificance of the other factors.

Table 7: Confidence Intervals for the coefficients of factors

q_0	q_A	q_B	q_C	q_{AB}	q_{AC}	q_{BC}	q_{ABC}
(11202.2, 11364.8)	(1390.2, 1552.7)	(-375.8, -213.3)	(-528.9, -366.4)	(-183, -20.5)	(-239.8, -77.3)	(-206.3, -43.8)	(-85.5, 77)





5 Interactive Law Verification

Although, I have somewhat verified the Interactive Law for almost all the experiments I conducted, in this section, I check the validity of the ***"Effect of Writes"*** experiments from Milestone 2 using the Interactive Law. In these experiments, I vary the replication Factor (Rep=1, Rep=Full), number of memcached servers (3, 5, 7), and percentage of write Operations (1%, 5%, 10%). The detailed description of experiment setup is presented in the table below.

Number of Memcached Servers	**3, 5, 7**
Number of Client Machines	3
Virtual clients / Machine	80
Workload	Key 16B, Value 128B, **Writes/Reads Ratio:** **0.01/0.99,** **0.05/0.95,** **0.10/0.90**
Replication (Rep)	**No Replication (Rep=1),** **Full Replication (Rep=S)**
Number of Threads in Thread Pool	16
Runtime x repetitions	180s x 5
Log files	**writeexp**

The Interactive Response Time Law (IRTL) states that:

$$R = \frac{N}{X} - Z,$$

where R is the Response time (in seconds),
N is total number of clients,
X is the throughput of the system (in Operations/Second) and
Z is think time of clients (in seconds).

In the experiments described in the table above, the number of virtual clients per memaslap machine is fixed to 80, resulting in total number of clients to be:

$$N = 80 * 3 = 240.$$

The think time Z can be estimated as

$$Z = \frac{N}{X} - R$$

I reckon that the system should have a think time of zero, if anything, because the memaslap client is designed to send a new request as soon as it receives a response from the system. One can say that even if there is a think time in the memaslap client machines because it has to parse the response to check if it is valid, this delay is negligible as the requests are sent instantaneously. This can be verified by the computation from the table below where:

$$IRT = \frac{N}{X}$$

	Percentage Write Operations	3 Memcached Servers		5 Memcached Servers		7 Memcached Servers	
		Rep=1	Rep=Full	Rep=1	Rep=Full	Rep=1	Rep=Full
IRTL Response Time (IRT)	1%	18192.84	18793.16	19462.82	19322.42	19569.79	19584.80
	5%	18637.59	18712.86	18974.13	19951.45	19733.27	19850.13
	10%	18739.75	19351.09	19118.32	20901.91	20035.06	22580.58
Measured Response Time (R)	1%	18286.2	18909.67	19668.53	19531.46	19676.53	19694.07
	5%	18725.73	18807.6	19110.66	20059.93	19918.60	19951.13
	10%	18843.33	19451	19223.6	21006	20141.53	22762.67
Z = IRT - R	1%	-93.35	-116.50	-205.70	-209.03	-106.74	-109.26
	5%	-88.14	-94.73	-136.53	-108.48	-185.33	-101.00
	10%	-103.58	-99.90	-105.27	-104.08	-106.47	-182.08
Difference in %	1%	**0.51**	**0.62**	**1.05**	**1.07**	**0.54**	**0.55**
	5%	**0.47**	**0.50**	**0.71**	**0.54**	**0.93**	**0.51**
	10%	**0.55**	**0.51**	**0.55**	**0.50**	**0.53**	**0.80**

The negative values of Z from the table above should not be interpreted as a negative think time. I investigated as to why I get these negative values and I found out that the negative values above are due to numerical approximations in the memaslap output and the approximations in the computation of N/X where X is the averaged throughput. Besides, the difference in % is less than 1% in most of the cases.

This further solidifies my reasoning that the think time is negligible and can be considered to be 0 because it is so small that even the approximation errors are greater than it. Therefore, the interactive law is valid for the set of experiments.

This tells us that the response time and the aggregated throughput are indeed inversely proportional to each other and their product is the total number of clients in the system.

Logfile listing

Each logfile has a README.txt file inside which maps the corresponding experiment number to the parameters used to run that experiment.

Short name	**Location**
trace1	https://gitlab.inf.ethz.ch/ragrawal/asl-fall16-project/blob/master/Experiment Logs/MWTrace3clients3replicas/trace1.zip
newvclientexp	https://gitlab.inf.ethz.ch/ragrawal/asl-fall16-project/blob/master/Experiment Logs/newvclientexp.zip
vclientexp	https://gitlab.inf.ethz.ch/ragrawal/asl-fall16-project/blob/master/Experiment Logs/vclientexp.zip
repexp	https://gitlab.inf.ethz.ch/ragrawal/asl-fall16-project/blob/master/Experiment Logs/repexp.zip
factorialexp	https://gitlab.inf.ethz.ch/ragrawal/asl-fall16-project/blob/master/Experiment Logs/factorialexp.zip
writeexp	https://gitlab.inf.ethz.ch/ragrawal/asl-fall16-project/blob/master/Experiment Logs/writeexp.zip