Advanced Systems Lab (Fall'16) – Third Milestone

Name: *Iveri Prangishvili*
Legi number: *15-926-140*

Grading

Section	**Points**
1	
2	
3	
4	
5	
Total	

1 System as One Unit

To build an M/M/1 model of the entire Middleware system based on the stability trace, I consider the Middleware and the Memcached Servers as the *Service* and the entire Middleware System as the *Black Box* queueing *System*. Black box includes everything (Middleware, Memcached Servers, Networking between Memaslap and Middleware) apart from the Memaslap Clients (see Figure 1). Each virtual client from a Memaslap machine sends a request to the *system*, waits until it receives the response and only then sends another request, meaning that the model can be treated as a *Closed*

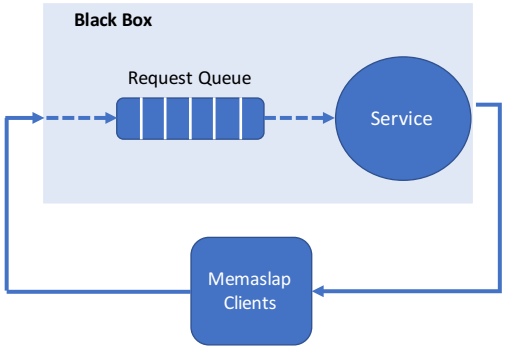


Figure 1: M/M/1 Model of the Whole System

System. Hence, the number of requests in the *black box* should equal to number of Virtual Clients, and consequently the number of requests arriving to the *system* is equal to the number of requests leaving (Completed) the *System*. Before, I construct a M/M/1 Model of the *System*, I check if Little's Law holds for the *System*. $Mean\ Number\ of\ Jobs\ in\ the\ System\ (N) = Arrival\ Rate\ (X) * Mean\ Response\ Time(R) = 10400 * 0.018471 = 192.09$, where X is the measured Memaslap throughput (Requests/Second) aggregated across 3 Client Machines and R is measured Memaslap Response Time averaged across 3 Client Machines. The resulting Number of jobs in the *System* from Little's Law is as expected, since there is a total of 192 Virtual Clients, and hence should be 192 requests in the system (Black Box). To construct a M/M/1 model based on the stability trace (*M1_1h_trace_instrumentation*), I need to calculate two parameters: Mean Arrival Rate (λ) and Mean Service Rate (μ). Since, Model is treated as a *Closed System,* Mean Arrival Rate (λ) is same as the measured throughput of the system $\lambda = 10400\ Requests/Second$. To find Mean Service Rate (μ), I look at the formula for Mean number of jobs in the System from M/M/1 model and use the fact that it should be 192, $E[n] = \frac{\rho}{1-\rho} = 192, where\ \rho = \frac{\lambda}{\mu}$, resulting in $\mu = \frac{193*\lambda}{192} = 10454.17\ Requests/Second$. Based on these two parameters Mean Arrival Rate (λ) and Mean Service Rate (μ), I calculate the rest of the measures presented in the Table below.

Table 1: M/M/1 Model values vs Measured values; Based on M1_1h_trace_instrumentation data

Parameters	Model	Memaslap and Middleware
Traffic Intensity ρ	0.9948	-
Mean Service Time $E[S]$	95.656 μs	$T_{Server} = 1725.305\ \mu s$
Mean # jobs in the System $E[n]$	192	192.09
Standard Deviation of $std[n]$	192.49	-

Mean # jobs in the queue $E[n_q]$	191.005	-
Standard Deviation of $std[n_q]$	192.49	-
Mean Response time $E[r]$	18461.54 μs	Mem Res. = 18471.67 μs
Standard Deviation of $std[r]$	18461.54 μs	Mem Std, Res = 14456.76 μs
Mean Waiting time $E[w]$	18365.88 μs	$T_{mw} - T_{Server} = 10320.117\ \mu s$
Standard Deviation of $std[w]$	18461.29 μs	-

From the Table 1, the Utilization – ρ – is 0.9948, which is less than 1, meaning that the *System* is in a stable state. In the Table 1, I compare the M/M/1 Model parameter values to the measured values from Memaslap clients and Middleware logs. The $\frac{T_{Server}}{E[S]} = 18.036$, meaning that the Mean Service Time $E[S]$ estimated through E[n] and λ, is 18.036 times smaller than T_{Server}. The reason behind this is that, M/M/1 model assumes that there is only one queue and a single service, and hence the requests are processed sequentially. However, in the Middleware there are several read and write queues, where each Read queue has several threads pulling requests in parallel from the queue and forwarding them to Memcached Servers. Because of the fact, that I use a black box approach, use the measured throughput (Requests/Second) as the arrival rate (λ) and the number of jobs in the system as the number of Virtual Clients, the M/M/1 model estimate of the Mean Service Time takes into consideration the change from the multithreaded architecture to single queue - single server one, thus explaining the reason why E[S] (Accounts for parallelism) is smaller than T_{Server}. Moreover, based on the configuration of the experiment there are in total 3*(8+1) = 27 threads in the Middleware, the ratio of $T_{Server}/E[S]$ can be perceived as the number of effective threads used in the Middleware, making it 18 out of 27. It is important to note that T_{Server} only includes service time associated with Memcached servers, excluding Reading/Writing from/to Memaslap by the main receiving thread (Server Class[1]) and hashing, which E[S] accounts for. In the table, above $E[r]$ is close to the Measured Memaslap Response time, which is expected, since $E[n]$ was used to estimate Mean Service Rate and according to Little's Law there is a relation between throughput (Requests/Second), number of jobs in the system and the Response time. I compare the mean waiting time of the model $E[w]$ to $T_{mw} - T_{Server}$. The difference between total time a request spends in the middleware and the service time is an estimate of the time a Request spends waiting in the Middleware. The M/M/1 model waiting time $E[w]$ is still higher than the measured quantity, this is due to the fact that, $E[r] = E[w] + E[S]$, as E[S] is lower E[W] should be higher to result in the same E[r] value (due to Little's Law).

Based on the description above, it is evident that M/M/1 model is not suited to model the Middleware, since the two have different architectures. Furthermore, M/M/1 is over simplistic model and the resulting measures do not reflect the behavior of the system.

[1] https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/src/Server.java

2 Analysis of System Based on Scalability Data

In this Section, I will construct several M/M/m models to analyze the system based on the scalability for two cases: 1. As the number of Virtual Clients increase and 2. As the number of Memcached Servers is increased. The Figure 2 below, presents the *black box* view of the system modeled as M/M/m queue. The experimental data in both sub-sections use **8** threads in the thread pool, thus each Get queue associated with a Memcached Server has **8** parallel threads working.

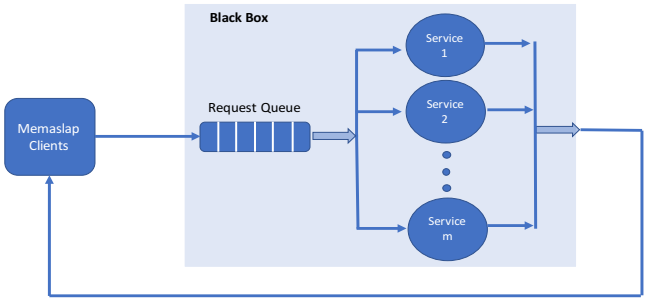


Figure 2: M/M/m Model of the System

2.1 Scalability with the increase of the Work Load

In this sub-section, I map the M/M/m model behavior to the scalability of the system as the number of virtual clients is increased. From the Milestone 2, Section 1 (M2_Sec1_Series1 and M2_Sec1_Series2 Data) experiments I obtain data for 32, 64, 112, 128, and 144 virtual clients with **8** threads in the thread pool. I also conducted additional experiment for this sub-section to obtain data for 2, 4, 8, 16, 24 Virtual Clients per Memaslap machine.

# Memcached Servers: 5, # Client Machines: 3, Run Time: 180s x 5 repetitions	
Virtual clients / Machine	**2, 4, 8, 16, 24**
Workload	**Key 16B, Value 128B, Writes/Reads Ratio: 0/1,**
Replication (Rep)	No Replication (Rep=1)
Number of Threads in Thread Pool	8
Log files	M3_Sec2

In Milestone 2, Section 1 (M2_Sec1_Series1 and M2_Sec1_Series2 Data) experiments, I achieve maximum throughput of 14320.2 Requests/Second with 8 threads in the thread pool at 112 virtual clients per Memaslap Machine, meaning that my System performs at full potential at this configuration. To construct the M/M/m model, I use as Mean Arrival Rate $\lambda = 14320.2\ Requests/Second$, set Mean number of jobs in the system $E[n] = 112 * 3 = 336$ and fit Mean Service Rate, resulting in $m * \mu = 14367.44\ Requests/Second$. Based on these two parameters I calculate the utilization $\rho = 0.9967$, which is less than 1, meaning that the System is stable. In this case, I set m to $Number\ of\ Memcached\ Servers\ (5) * Threads\ in\ thread\ pool\ (8) = 40$, since in the experimental data used, each Read queue associated with a Memcached Server has **8** threads in the thread pool, that work in parallel, meaning that there are in total 40 identical services (Since in these Experiment, ratios of Sets are set to 0 - Set threads are not use). The two plots below depict the predicted arrival rate (λ) and Expected Response Time ($E[r]$) by fitting Mean number of jobs in the system to $E[n] = 3 * Number\ of\ Virtual\ Clients$ and using the Mean Service Rate $\mu = \frac{14367.44}{40} = 359.86\ Requests/Second$ of the Saturated M/M/40 Model (Clients = 112 per Machine) and compare them to the measured Throughput and Response Time from Memaslap Machines.

As it can be seen from the Figure 3, the M/M/m model using μ calculated using max TPS from the 112 virtual clients per Machine (Total 336) doesn't predict accurately the systems behavior with

lower number of virtual clients for the throughput. This is logical since, the model assumes that the Mean Service Rate (μ) stays the same in between different number of virtual clients, which in the real

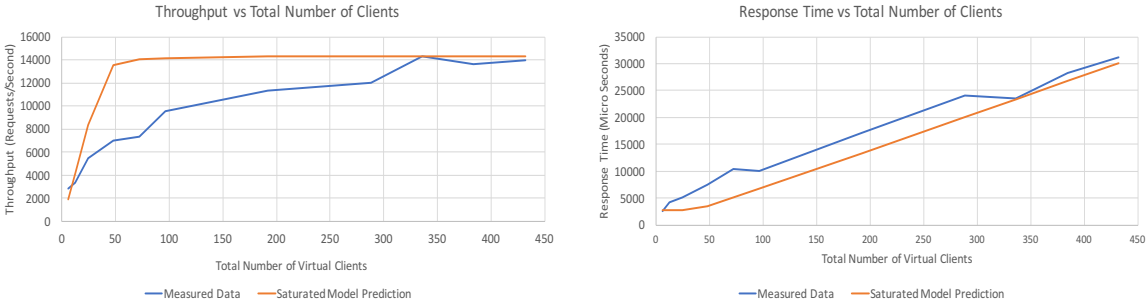


Figure 3: Saturated M/M/40 Model Prediction (112 Clients) vs Measured Data

system is not true. As the workload increases, there are many more requests to be read by the main receiving thread (Server Class[1]) and consequently more requests to be processed by the Read Threads, resulting in less CPU context switches and less CPU resources wasted. Hence, as workload increase the utilization of both the receiving main thread (Server class[1]) and reader threads increase gradually, thus also increasing Mean Service Rate gradually (Reaching max at 112 clients per VM). On the other hand, the model assumes that the Mean Service Rate (μ) stays always constant predicting the upper bound for the throughput (Requests/Second) since it has been initialized at the configuration, reaching maximum throughput (Requests/Second). The models predicted throughput increases from 0 to 50 requests, since M/M/m model is load dependent and since m = 40, if there are less than 40 requests in the Service the Model doesn't perform at full potential and will only use <40 identical services. As for the response time, the model prediction is close to the measured one (Figure 3 - right). E[r] = E[w] + E[S], where E[w] increases with higher number of virtual clients and the same behavior is observed with measured Response time, where waiting time also increases with number of clients. The predicted response time and the measured one, become similar when getting closer to 336 total Clients (the point where the E[S] is estimated from). In Table 2, I present the measurements for under saturated (8 clients per Machine = total of 24 clients) and saturated model (96 clients per machine, total of 288) based on the Mean Service rate, that is fitted using as the Arrival rate (λ) the measured throughput (at that configuration) and the assumption that E[n] = 8*3, 96*3 respectively and compare them to the M/M/m model predictions (with μ estimated from 112 Clients per Machine) and the Measured experimental data values.

Table 2: Compare fitted M/M/40 models to Model prediction (based on 112 clients per Machine) to Measured data from M3_Sec2, M2_Sec1_Series1.

	Under Saturated (Total 24 Clients)			Saturated (Total 288 Clients)		
Parameters	Model Prediction	Fitted Model	Measured Data	Model Prediction	Fitted Model	Measured Data
λ | TPS (Req/Sec)	8402.4	5447.4	5447.4	14311.3	12008.8	12008.8
ρ	0.58482	0.58448	-	0.996092	0.9960948	-
$E[S] \mid T_{Server}$ (μs)	**2784.07**	**4291.84**	1881.91	**2784.07**	**3317.88**	1535.35
$E[n]$	23.39	23.38	27.514	287.255	287.406	289.7
$std[n]$	4.8401	4.8387	-	255.45	255.60	-
$E[n_q]$	0.00167	0.00165	-	247.412	247.562	-

[1] https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/src/Server.java

$std[n_q]$	0.0798	0.0793	-	255.296	255.447	-
$E[r]$ |Res. Time (μs)	2784.27	4292.15	5050.93	20071.94	23932.96	24124.4
$E[w] \mid T_L\ (\mu s)$	0.1988	0.3031	647.889	17287.86	20615.08	13988.05

As it can be seen from the Table above (where $T_L = T_{mw} - T_{server}$), the prediction of the model based on Max TPS for the under saturated case (total of 24 clients), differs from the fitted model at that configuration. The Mean Service Time differs between the Fitted and prediction model by the factor of $\frac{4291.84}{2784.07} = 1.5415$, meaning that Fitted Mean Service Time is that larger than the one assumed from the maximum configuration. The Mean waiting time is also larger in the Fitted model, due to the fact arrival rate is lower (Measured throughput), and the same goes for the Expected Response Time, since it is the aggregation of the two. While, the difference between the Service Times for the Saturated model (total of 288 clients) is of the factor of $\frac{3317.88}{2784.07} = 1.1917$, which is lower than for under saturated one. This further supports the point, that as the workload changes the Mean Service Time changes as well, decreasing till a workload configuration of 112 clients per machine, which results in highest throughput of the system. This suggests, that there exists a tradeoff between devices in the system, specifically the main receiving thread (Server Class[1]) and the Read threads, that can't be described with M/M/m queuing model. Finally, Comparing the fitted models to the measured data, one can see that the expected Response times are approximately same, which is due to the Little's law since TPS, number of clients and the response time are all connected. As for the comparison between the Measured T_{Server} and fitted $E[S]$, fitted Mean Service Time ($E[S]$) is 2.28 and 2.16 times larger than T_{Server} for under saturated (24 clients) and saturated (288 clients) models respectively. The reason behind this is that, T_{Server} includes only Memcached Server processing time, while $E[S]$ additionally includes service times associated with reading/writing from main thread (Server Class) and hashing.

2.2 Scalability with the increase in Number of Memcached Servers

In this sub-section I build 3 M/M/m models for 3, 5 and 7 number of Memcached Servers with No replication, 5% writes based on M2_Sec2 Data and investigate the relation to the scalability of the system. To build M/M/m models I need two parameters: Mean Arrival Rate (λ) and Mean Service Rate (μ). Mean Arrival Rate λ = 11292.60, 12253.40, 11858.40 Requests/Second, are measured throughput for 3, 5 and 7 Memcached servers respectively. I estimate Mean Service Rate μ for 3, 5, 7 Memcached Servers by using arrival rates and setting Mean number of jobs in the system $E[n] = 96 * 3 = 288$, for all three cases since number of virtual clients Per Memaslap Machine is fixed to 96. Finally, I set m to 3*9, 5*9, and 7*9 respectively, since there are 8 read threads in the thread pool and one write thread per Memcached server, generalizing it to 9 identical services per Server (in total 9*Server number).

Table 3: Fitted M/M/m models for 3, 5, 7 Servers based on M2_Sec2 data.

	3 Memcached Servers M = 27; M/M/27		5 Memcached Servers M = 45; M/M/45		7 Memcached Servers M = 63; M/M/63	
Parameters	Fitted Model	Measured Data	Fitted Model	Measured Data	Fitted Model	Measured Data
ρ	0.9963	–	0.99602	–	0.9957	–
$E[S] \mid T_{Server}\ (\mu s)$	2382.01	1303.53	3657.85	1165.05	5290.09	1143.078
$E[n]$	287.334	289.65	287.348	289.62	287.386	289.37
$std[n]$	267.084	–	251.036	–	234.66	–

[1] https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/src/Server.java

$E[n_q]$	260.435	–	242.528	–	224.654	–
$std[n_q]$	266.985	–	250.86	–	234.404	–
$E[r]$ |Res. Time (μs)	25444.47	25650.13	23450.54	23636.13	24234.83	24402.6
$E[w] \mid T_L\ (\mu s)$	23062.45	15836.22	19792.69	13117.69	18944.74	12810.64

Table 3 (where $T_L = T_{mw} - T_{server}$), depicts the measurements of M/M/m models for each case. As it can be seen, as the number of Memcached Servers is increased, fitted $E[S]$ increase as well, while measured T_{Server} stays around the same. This behavior is expected, since as the number of servers increase so does the total number of threads in the system, resulting in the increase in the demand for the CPU resources. As the CPU resources are distributed, some parts of the system get slower (main receiving thread, Server Class[1]). This behavior is reflected in E[S], since it includes services other than Reading/Writing to Memcached, while measured T_{Server} stays approximately the same because it doesn't take this trade off into consideration. The ratio of E[S]/ T_{Server} is 1.827, 3.139, 4.627 for the case of 3, 5, 7 servers respectively, these factors represent a value by which E[S] is increased due to the other parts of the system (tradeoff). If the total number of Threads in the system 27, 45, 63 are divided by these factors the resulting values represent the number of effective threads used: 14.778, 14.336, 13.616. These numbers are approximately the same meaning that the effective number of threads that system can utilize without slowing down other parts of the system is staying constant. The Expected Response time is close to the measured one from Memaslap due to Little's law (Described in previous sub-section) and is increasing as expected with number of servers. The expected waiting time is consistently larger than T_L across different configurations, since E[w] in addition to including T_L (approximately a time request spends inside the middleware waiting) includes the existing network lag between Memaslap and Middleware, due to the fact that Mean arrival rate is computed through E[n] (which equals to the number of virtual clients) and Arrival Rate, which equals the measured throughput. Moreover, the ratio of E[w]/ T_L for 3, 5, 7 servers is 1.456, 1.50, 1.47 and stays approximately the same. Finally, I present a table with the model predictions of the throughput for the case of 5 and 7 Memcached servers based on the Mean Service Rate of M/M/27 model (3 Server one) and assuming E[n] = 288.

Table 4:Prediction for 5 and 7 Memcached Servers based on Mean Service rate estimated for 3 Memcached Servers

Prediction	ρ	λ (Req/Sec)	$E[S](\mu s)$	$E[n]$	$E[n_q]$	$E[r](\mu s)$	$E[w](\mu s)$
M/M/45	0.99602	18816.49	2382.01	287.3809	242.559	15272.81	12890.80
M/M/63	0.99574	26335.79	2382.01	287.477	224.745	10915.85	8533.84

As it can be seen the estimated arrival rate of M/M/45 and M/M/63: 18816.49, 26335.79 Requests/Second is much higher than the measured one 12253.40, 11858.40 and moreover doesn't follow the trend of decreasing throughput from 5 to 7 servers. The Expected Response time from the predicted models also decreases, since the expected waiting time decreased, which is due to the fact that number of jobs in the queue decreased from 242 to 224 as there are more identical services (from 45 to 63) to process the requests.

In conclusion, the M/M/m models can not relate to the system behavior for the scalability data. In the case of increasing workload, the M/M/m model follows the trend of increase in the Response Time however fails to map to the system behavior and take into consideration the tradeoff inside the Middleware. With the case of increasing number of Servers, the M/M/m queuing model fails to relate to the trend as the increase in the number of Memcached servers from 5 to 7 results in the decrease in the measured throughput due to some parts of the system slowing down (Due to Limited CPU resources), while the predicted throughput keeps increasing as there are more identical servers (larger m) and consequently increasing the value of $m * \mu$.

[1] https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/src/Server.java

3 System as Network of Queues

In this section, I build a network of queues for the whole system and compare the results to the experimental data. The data used in this section is from Milestone 2, Section 3 (M2_Sec3 Data), specifically for the case of 5 Memcached Servers, No Replication and 1%, 5%, 10% writes. The Figure Below presents the queueing Network architecture.

Due the limited measurements from the Middleware logs, I had to abstract several components from

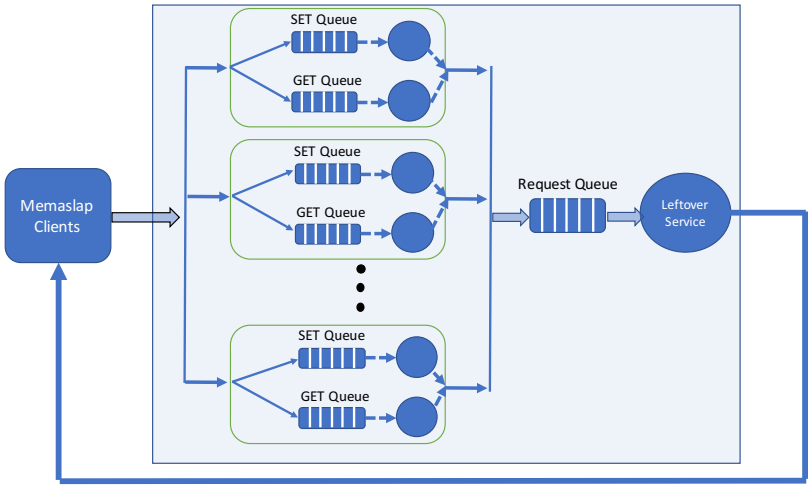


Figure 4: Modeling Whole System as Network of Queues

the queueing network. I have no way of getting a time request spends waiting until it is read by a main thread (Server Class[1]), until it is hashed and until it is sent back to Memaslap by main thread (Server Class[1]). Moreover, the service times for these services can't be derived through instrumentation logs. However, I can calculate a time for a request (which was already processed by GET/SET and got response), which includes: the waiting time required to get a write service (to Memaslap) by the main thread (Server Class[1]), the service time for writing to the Memaslap the main thread (Server Class[1]) and hashing Service, this time is $T_{left} = T_{mw} - T_{queue} - T_{server}$. Next, I don't know the exact time the request spends in the network (From Memaslap to Middleware or from Middleware to Memaslap). However, I can calculate the time for a request, which includes: the network time (including both ways from and to Middleware) and the time spent waiting to be read by the main thread (Server Class[1]) in the middleware, this time is $T_{NL} = Memaslap\ Response\ time - T_{mw}$. I can think of T_{NL} as the think time Z, since each request spends that much time before it is received (Read by the main thread[1] and first timestamp made) in the Middleware. Since, hashing and writing to Memaslap are done by main receiving Thread (Server Class[1]) I combine the two and call it a Leftover Service. I model the Leftover Service as M/M/1 queueing model, where $E[r] = T_{left}$ and since all requests have to be hashed and sent back to Memaslap, I use measured throughput as λ. Based on these two parameters I estimate Mean Service Rate for Leftover service. To model GET operation I also use M/M/1 queueing model, the natural choice would be to model it as M/M/m, where m is number of threads in the thread pool, however not all threads in the thread pool are utilized and the "effective" number of threads used is unknown. The T_{server} (for GET only) measurement from Middleware logs represents the service time of one thread. Since, I do not know the "effective" number of thread used in the thread pool I can't use T_{server} directly and model it as M/M/m, since m will be unknown. However, I know the T_{queue} (for GET only) time, which takes into consideration the average number of threads utilized in the thread pool, if more threads are utilized the T_{queue} time will decrease, since requests will be pulled from queue by more threads more frequently. Thus, I can model GET operation as M/M/1 model and use as $E[r] = T_{queue} + T_{server}$, and as arrival rate the

[1] https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/src/Server.java

measured throughput times the visit ratio. Moreover, to make sure that modeling GET device as M/M/1 queue is the right choice, I ran mean value analysis (MVA) algorithm modeling get at M/M/1 and as M/M/m (m = number of threads in thread pool) and compared the results. The results, using M/M/1 model for GET devices was per logic, while modeling it is M/M/m resulted in low utilization which doesn't reflect system behavior. I model SET operation (device) as 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, however T_{queue} time (for SET operation only) does include it. Therefore, instead of using T_{server} as the E[S] in M/M/1 model, better way is to calculate the Mean Service Time is using $E[r] = T_{queue} + T_{server}$ (Measures for SET operations only) and set as arrival rate, the product of measured throughput and the visit ratio. Thus, all devices (SET, GET, leftover) are modeled as M/M/1 queueing model. Since, in this section I use 5 Memcached Server configuration, in the queueing network I have 5 SET and 5 GET devices (M/M/1 models), each associated with a Memcached server, and one Leftover service (M/M/1).

To run a MVA algorithm and find a bottleneck device, I need the visit ratios and Mean Service Times for each device (Table 5). The visit ratios are calculated for SETs as $\frac{1}{5} * write\ Percentage$ and for GETs as $\frac{1}{5} * Read\ Percentage$. I calculate the Mean Service Times for each device based on the respective E[r], and throughput of the device - $\lambda * visit\ ratio$ - where arrival rate (λ) is the measured throughput. The results of the calculations for a GET, SET device and Leftover device for the case of 5 Memcached Servers, 1% writes and No replication is presented in Table below.

Table 5: Measured Data (Based on M2_Sec3 Data) with predictions using M/M/1 queueing model for Each device

Parameters	Devices		
	ith SET	ith GET	Leftover
Visit Ratio	(1/100)*(1/5)	(99/100)*(1/5)	1
TPS per instance (TPS*visit Ratio)	24.172	2393.028	12086
$E[r]$ (μs)	1914.949	6203.464	8863.264
Estimated μ (Req/Sec)	546.379	2554.228	12198.825
Utilization ρ	0.0442	0.9369	0.9907
Estimated $E[S]$ (μs)	1830.231	391.508	81.9751
Estimated $E[n]$	0.046288	14.84507	107.121405

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=288 (96 clients * 3), I use MVA algorithm and present the results in the Table 6, below.

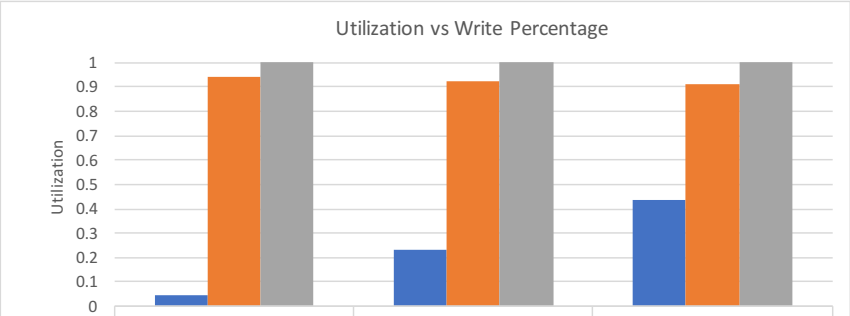
Table 6: MVA results for the case of 5 Memcached Servers, 1% write, No replication (Based on M2_Sec3 data)

Parameters	ith SET	ith GET	Leftover	Whole System
Utilization (U)	0.044605	0.94461	0.99891	-
Average # of Jobs (Q)	0.046687	16.71907	94.7836	178.6125
Response Time R (μs)	1915.676	6929.453	7778.31	14657.625
Throughput X (Req/Sec)	24.37127	2412.75623	12185.637	12185.6375

As it can be seen from the Table 6 (MVA results) above, the Utilization and number of jobs in the device, for each Device (SET, GET, Leftover) is close to the ones computed in Table 5. In addition, the System throughput of 12185.6375 is approximately the same as the measured one for this configuration 12086. Moreover, the System response time of 14657.625 Micro Seconds is approximately the same as the T_{mw} time reported from the Middleware logs 15023.644 Micro

Seconds. This is in accordance with logic, since I take as Think Time Z, the difference between the response time reported by Memaslap and Middleware time, suggesting that abstracting 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 5), while for GET device the difference is approximately 700 Micro Second. Based on the utilization of devices from the Table 6, Leftover device is the bottleneck device, since it has the highest utilization of 99.891%. This means that the main Receiving thread (Server Class[1]) is the bottleneck. This makes sense, since all arriving requests have to go through single main receiving thread (Server Class[1]), first to be read, hashed and then once processed by GET and SET threads, to be sent to the Memaslap through this main thread (Server Class[1]) again. GET device also has a high utilization, which is expected since the workload configuration for these experiments is 96 clients per Memaslap Machine, which is close to the workload (112 clients) achieving the maximum throughput (Milestone 2 Section 1). Meaning that the System is saturated, and GET devices are working at nearly full capacity. To summarize, the queuing 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 Leftover device, I run MVA algorithm for 5 Memcached Servers with 5% and 10% write percentages and present the utilization of ith GET, SET device and Leftover device in the Figure below (to get MVA results for 5% and 10% write operation configurations, same steps were taken as with the case of 1%, which is described above). As it can be seen from the Figure 5, as the Write percentage is increased the Utilization for ith SET device increases. This is in accordance with logic, since SET device has more requests to process and thus the utilization increases. The utilization for ith GET device decreases, since there are less read operations to process, but not significantly. This is due to the fact that proportion of the GET request that arrive to the system is still high and thus utilization is not decreasing significantly. Finally, the utilization of Leftover device, which is the bottleneck device, is 99% and stay around the same value. This is due to the fact that total number of requests (workload) is kept the same, and since all requests have to go through bottleneck device the utilization doesn't change.



	1%	5%	10%
SET	0.044605063	0.229903401	0.438567266
GET	0.944613369	0.92390066	0.914354996
Leftover	0.998918857	0.999844338	0.999931431

Figure 5: Utilization of ith SET, GET device and Leftover device as write percentages change. Based on (M2_Sec3 data)

In Conclusion, the network of queues model enables a good analysis of the system, identifying the bottleneck of the system in the single main receiving thread (Server Class[1]) as expected.

[1] https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/src/Server.java

4 Factorial Experiment

To design a 2^k factorial experiment, I conduct a series of new experiments for this section. 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	8
Runtime x repetitions(r)	180s x 5(r)
Total Number of Experiments conducted:	8 * repetitions (5)
Log files	M3_Sec4

In these experiments, I vary 3 parameters, thus I introduce three Factors (k=3) each with two levels. Below I define three variables and their respective levels.

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 (Value 128B)
    b. x_B = 1 if Request Size is Large (Value 512B)
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 Effect of each of the factor on the throughput is unidirectional, given the ranges for the values for each of the factor described above. For the number of virtual clients (x_A) the throughput increases in the given range, as for Request Size (x_B) and the Percentage of Writes (x_C) the throughput decreases for their respective ranges. For the experiment (M3_Sec4) configuration, I chose Full Replication to make the effect of Writes on the throughput more evident (based on the knowledge from Milestone 2 Section 2). The performance Y, representing the throughput of the System in Requests/Second, can be regressed on x_A , x_B and x_C using a nonlinear regression model of the form: $\widehat{Y_\iota} = q_0 * 1 + q_A * x_{Ai} + q_B * x_{Bi} + q_C * x_{Ci} + q_{AB} * x_{Ai} * x_{Bi} + q_{AC} * x_{Ai} * x_{Ci} + q_{BC} * x_{Bi} * x_{Ci} + q_{ABC} * x_{Ai} * x_{Bi} * x_{Ci}$, where q represents the coefficients of respective factors. The Table below is a sign table used to calculate the effects 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 7: 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 what kind of effect does the factors have on the system. As seen from the Table above q_A has a positive value meaning that increase in the Number of Virtual Clients results in the increase of the throughput (Requests/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). q_B and q_C coefficients have negative values, that indicates that the Request Size (x_B) the percentage of Writes(x_C) have a negative impact on the throughput of the system. The interaction terms between the factors have all negative impact on the throughput of the System, since respective coefficients have all negative value. The next, step is to find out the importance of each of the factors 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 sum them up to get the Sum of Squares Total (SST), based on the formula in the book: $SST = SSA + SSB + SSC + SSAB + SSAC + SSBC + SSABC + SSE$, where SSE represents the Sum of Squared Errors, which is calculated using the data from 5-repetition for each experiment. The table below presents the percentage of variation due to each of the factor.

Table 8: 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 Factor x_A, Number of Virtual Clients per Memaslap Machine, explains 84% of the existing variation, making it most important factor, which is expected since its levels change from under saturated case (32 clients) to near Saturated one (96 clients) and based on Milestone 2 Section 1 experiments the highest increase in the throughput is observed inside this region. The next factor with the highest percentage of variation – 7.78% - is x_C, Percentage of Write Operations. This indicates that this factor also has an important impact on the performance of the system (throughput). This is

logical, since the experiment is using Full Replication making the write operations more expensive than the read operations and since increasing the percentage of writes also results in the decrease of the percentage of reads, the system throughput decreases. x_B factors explain 3.37% of existing variation, which can be due to the fact that transferring a Request with larger value size (4 time bigger 512/128) will affect the network performance. The error term contributes 2.86% to the existing variation (calculated following 2^kr design), which is lower than the contribution of each of the factor. Finally, I compute 90% confidence interval for coefficients of factors with $2^3 * (r-1) = 32$ degrees of freedom, presented in Table below. As it can be seen only q_{ABC} coefficient includes 0 in its interval, meaning that the interaction between all three factors is not significant. The rest of the interaction factors are not significant due to small variation percentage; however, it is still in accordance with logic. The interaction between x_A *and* x_B(coefficient q_{AB}) make sense, since the network has to transfer more requests with larger value size, resulting in the decrease in the throughput. The interaction between x_A and x_C (coefficient q_{AC}) is also logical, since increasing percentage of writes with larger workload, results in more write operations, that are more expensive than read operation. Finally, the interaction term between x_B and x_C (coefficient q_{BC}), is due to the fact that, since there is a Full Replication, as write operations and the Request value size is increased, the network between the Set thread and the Memcached Servers suffer, because there is bigger traffic with 4 times bigger request values to transfer.

Table 9: 90% 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)

Finally, In the figure below I present a Residual vs Fitted Throughput (Requests/Second) and Quantile plot, to show that experimental errors are normally distributed (Right plot) and are not correlated (left plot)

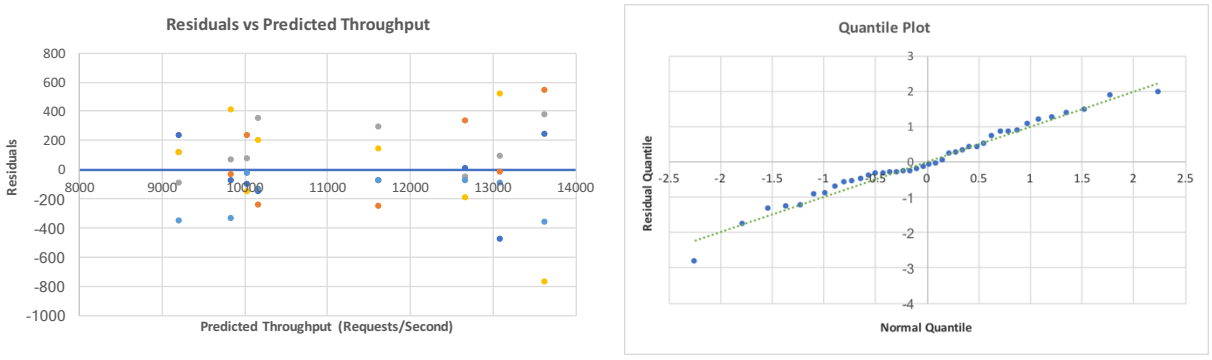


Figure 6: Residual vs fitted plot (left), Quantile plot (Right) based on M3_Sec4 Data

5 Interactive Law Verification

In this section I check the validity of all experiments (18 in total) from Milestone 2, Section 3 (M2_Sec3 Data) using 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 (from Mileston2 section 3) is presented in the Table below.

Number of Memcached Servers	**3, 5, 7**
Number of Client Machines	3
Virtual clients / Machine	96
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	8
Runtime x repetitions	180s x 5
Total Number of Experiments conducted:	18 * repetitions (5)
Log files	M2_Sec3

The Interactive Response Time Law (IRTL) formula is: $R = \frac{N}{X} - Z$, where R is the Response time (Seconds), N is total number of clients, X is the throughput of the system (Requests/Seconds) and Z is think time of clients (Seconds). In the Experiments described in the Table above, the number of virtual clients per Memaslap machine is fixed to 96, resulting in total number of clients: $N = 96 * 3 = 288$. I assume that think time $Z = 0$, because as soon as Memaslap Client gets response it sends a new request, without any waiting period. Hence, the Response time according to the interactive law is $R = \frac{288}{X}$, where X is the measured throughput of the system. The table below, presents calculated response time according to IRTL for each configuration in Micro Seconds, and the Measured Response time in Micro Seconds (based on M2_Sec3 Data). To compare the two, I calculate the difference between the **Measured response time and the Interactive Response time** and present the outcome in the last three rows in the Table below.

	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 (R)	1%	26635.09	26515.91	23829.22	23225.43	23470.35	23984.81
	5%	25503.42	26763.31	23503.68	24014.81	24286.58	25155.91
	10%	24984.38	26629.18	24020.41	25026.50	24849.43	26482.27
Measured Response	1%	26820.66	26646.66	24000.4	23349.66	23586.26	**24165.13**
	5%	25650.13	26895.06	23636.13	24137.46	24402.6	25276.46

Time (MR)	10%	25113	26805.46	24146.6	25144.46	24974.6	**26609.06**
Response Time Difference MR - R	1%	185.57 (0.69%)	130.75 (0.49%)	171.18 (0.71%)	124.23 (0.53%)	115.91 (0.49%)	180.32 (0.75%)
	5%	146.71 (0.57%)	131.75 (0.48%)	132.45 (0.56%)	122.66 (0.51%)	116.02 (0.47%)	120.55 (0.47%)
	10%	128.62 (0.51%)	176.28 (0.65%)	126.18 (0.52%)	117.96 (0.47%)	125.16 (0.50%)	126.8 (0.48%)

As it can be seen from the Table above, the response time based on the interactive law is almost the same as the measured response time from the Memaslap machines. If, I treated the think time Z as an unknown and Calculated it as the difference between the measured Memaslap Response Time and the IRTL Response Time, I would get the values presented in the last three rows in the table but with a **negative sign**. Having a think time Z, of negative value is not possible, therefore I set Z = 0 and continue with analysis. The difference between the calculated response time and the measured one is always less than 1% of the respective measured response time, making it negligible. Moreover, the existing difference can be accounted towards approximations in the Memaslap output values, as well as approximations in the average throughput X, which is used to calculate the IRTL response time N/X. Therefore, the Experiments from Milestone 2, Section 3 (M2_Sec3) are valid according to the Interactive Law, since the existing difference is negligible as noted above.

Log File Listing

Short Name	Location
M1_1h_trace_instrumentation	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Middleware/clients3_servers3_replication3/run_3.zip
M2_Sec1_Series1	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section1_Experiment_Series4.zip
M2_Sec1_Series2	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section1_Experiment_Series2.zip
M2_Sec2	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section2_Experiment_Series1.zip
M2_Sec3	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section3_Experiment_Series1.zip
M3_Sec2	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone3_section2_data.zip
M3_Sec4	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone3_Section4_data.zip