Advanced Systems Lab (Fall'16) – Third Milestone

Name: *Teymur Babayev*
Legi number: *15-926-231*

Grading

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

1 System as One Unit

Several approaches have been considered in order to build M/M/1 model of the entire system. To define this model we basically need 2 parameters, one of which is λ – system throughput, and for my stability trace it is well defined and equals to 12262 ops/s (as stated in milestone 1 report). Choice of second parameter is based on whether we are using white box approach or black box approach. White box approach implies using measurements from MW logs and estimating service rate μ – with help of these measurements. One possible way to estimate μ from these measurements would be to use mean T_{server} measurement and do some modifications on it to obtain an estimation. However, there are several problems with this approach: T_{server} includes not only time for processing (serving) the request, but also time spent in network, and possibly some queue (waiting) time since requests might spend some time in internal memcached queue. Apart from that, T_{server} in our MW is associated with T_{queue}, however, T_{queue} is not the only place, where request is spending time waiting inside the MW (e.g., T_{left} is another such place), so by using T_{server} we would make some architectural assumptions which would only harm the simplistic M/M/1 model. Black box approach defines what exactly is a black box and operates with measurements obtained on the boundary of black box. In our case, black box would be everything apart from the clients (memaslap). To better illustrate how the whole system is split into clients and black box, I provide the following schematic:

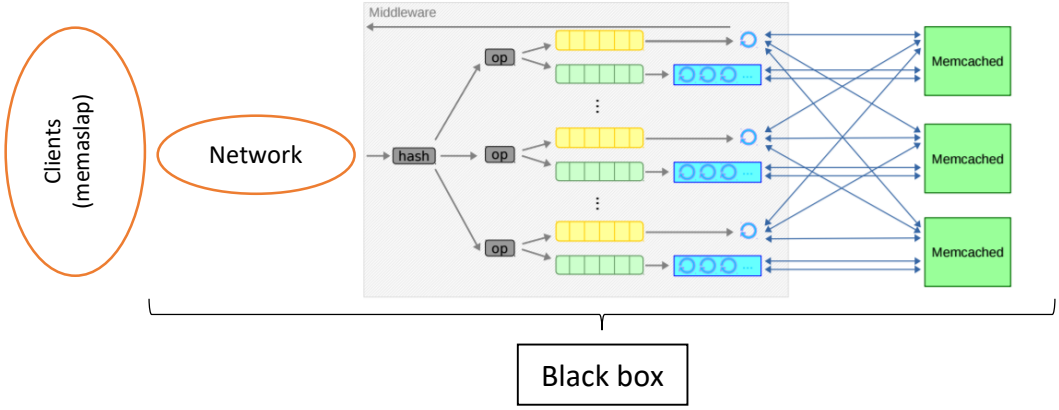


Figure 1: Black box definition

Measurements on the boundary of black box then would be: TPS, response time, number of virtual clients. Since we are in the closed system context, number of requests circulating in the black box then would be equal to number of virtual clients, that's why I choose this parameter to define my model. Looking at the formulas of M/M/1 model it is easy to derive that $\mu = \frac{\lambda(E[n]+1)}{E[n]}$. This way we will use λ and $E[n]$ to derive the first parameter of our model: μ = 12325.86. It is easy to see that Utilization (or Traffic Intensity) ρ is equal to 0.9948. The fact that utilization is less than 1 implies that the model is well defined and the system is in a stable state. Having ρ allows us to derive all other model outputs. which can be found in the below table together with corresponding MW and memaslap measurements:

Parameters	Model values	memaslap and MW values
λ or TPS	12262 ops/s	12262 ops/s
μ	12325.86 ops/s	-
ρ	0.9948	-
$E[s]$ or T_{server}	81.13 μs	1859.07 μs
$E[n]$	192	193.49436
Std. $E[n]$	192.49	-
$E[n_q]$	191.01	-
Std. $E[n_q]$	192.49	-
$E[r]$ or Res. time	15658.13 μs	15780 μs
Std. $E[r]$ or Std. res. time	15658.13 μs	10700.53 μs
$E[w]$ or $T_{queue} + T_{left}$	15577.00 μs	8869.77 μs
Std. $E[w]$ or Std. $T_{queue} + T_{left}$	15657.92 μs	9433.89 μs

Table 1: M/M/1 model values vs. actual data

In the table one may notice that $E[n]$ for memaslap data (calculated with Little's Law) is higher than 192. This is due to minor inconsistencies in memaslap measurements, which will be discussed in IRTL checks (section 5). One more thing to mention is that for actual measurements I take waiting time as $T_{queue} + T_{left}$: following my discussions in milestone 2, apart from 3-7 μs required for hashing, T_{left} consists of time spent waiting until request is sent back by my MiddlewareServer thread, that's why it is reasonable to add this time to "waiting time".

Above table shows that $E[r]$ is close to actual response time, but this is due to using $E[n]$ as an input – Little's Law implies strong connection between TPS, response time and # of jobs in system. Other values are hard to compare: $E[s]$ is 22.9 times lower than T_{server}. Complementary to that, $E[w]$ is higher than $T_{queue} + T_{left}$. In fact, in M/M/1 system ρ might be considered not only as utilization, but also as a portion of $E[w]$ in $E[r]$, which is exactly the case in our outputs. Considering such a differences, making some assumptions about std. deviations is not feasible.

Incompatibility of model output to actual measurements comes from the definition and architecture of M/M/1 model: this model assumes a single queue and single server – thus, sequential serving of jobs. These concepts are opposite to our MW + memcached combination: in reality there are several queues in the system, we use several servers and multithreading – all the things that model does not assume. The only way to perceive this model is "how the black box would behave if there would be no network, 1 big queue and 1 server". This explains why $E[s]$ is much lower than T_{server}: $E[s]$ should account for things being done in parallel. Otherwise service rate μ would be much lower and model would just not work at all. $E[w]$ thus is higher than our measurements, as a consequence of low $E[s]$. It's also worth to mention that even accounting for parallelism, $E[s]$ is incompatible to T_{server} since $E[s]$ is some generic service time, and T_{server} assumes only part of a total service time associated with memcached, however this is not the only place where job is "served" in our system (e.g., network, hashing, reading requests, sending back responses).

Discussions above demonstrate that M/M/1 is not a good choice to model our system due to difference in model and system architectures and oversimplification of behavior by model.

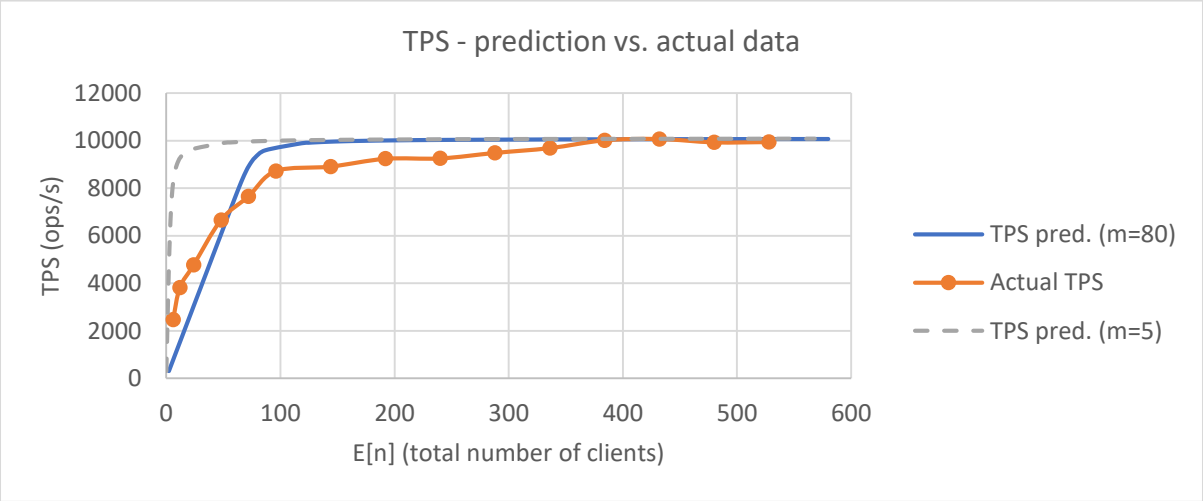
2 Analysis of System Based on Scalability Data

In this section I will be analyzing different M/M/m models using same black box approach with input parameters λ and $E[n]$ for initializing the models to analyze scalability in 2 dimensions: scalability with increase in load (#clients) and scalability with increase in resources (#servers).

2.1 Scalability with increase in load

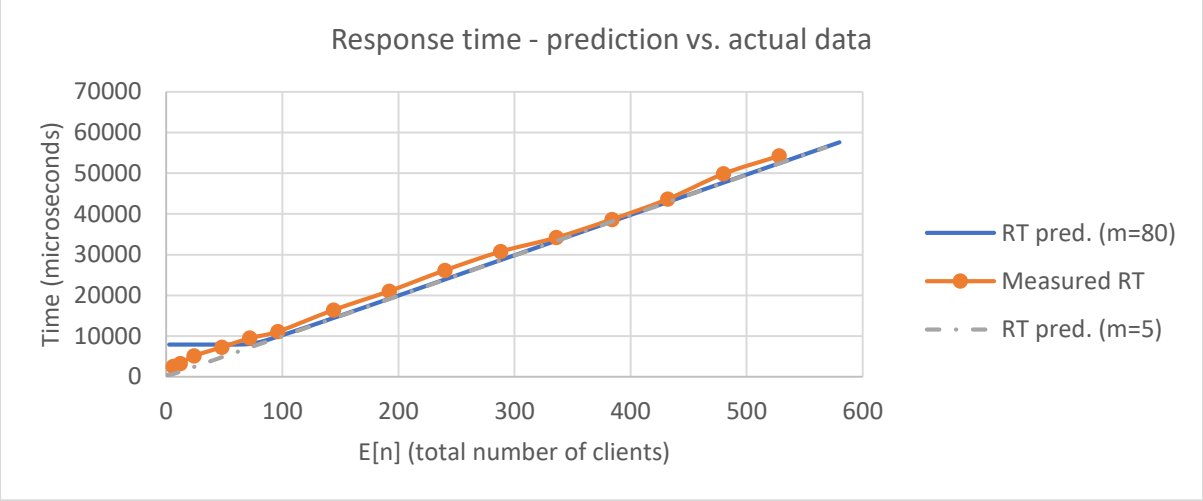
To analyze scalability with increase in number of clients, I will be using results of max. TPS experiment from MS2. However, in order to have full picture for comparison and analysis, I conducted new experiment. This experiment exactly follows setup of maximum TPS experiment of MS2 (setup table on page 3 of MS2 report), apart from the fact that number of clients per VM was: 2, 4, 8, 16, 24. Log files can be found in **mmm_tps** log file.

I will be analyzing model and measured data only for thread pool size 16 – this was my best TP size and I'll be looking into scalability with increase in resources in the next subsection. We know from MS2 report (section 1, page 5) that max. achievable TPS was 10069 with 144 clients / VM. Assuming that this is best my system was able to deliver, I initialize M/M/m model with λ = 10069 and $E[n]$ = 432 (3*144). With these inputs I can only infer that $m * \mu$ = 10096.72. This yields ρ = 0.9973, which makes sense since our system is operating on its limit. However, explicit choice over m still should be done. It would make sense if m would be equal to number of servers or number of worker threads in the system. Since the choice is not obvious, I initialized 2 models with $m = 5$ and $m = 80$ (thread pool size * number servers). I used these models to make predictions for λ for different $E[n]$'s and fixed μ. Following plot contains predictions for 2 models and actual measurement for TPS:



Plot 1: M/M/m models: TPS prediction vs. actual data

From above plot it is seen that actual results are pretty close to theoretical predictions for m = 80. In contrast, line for m = 5 reaches max. TPS much earlier and is not so close to system behavior. It is also seen that at total # of clients = 48, 384 and 432, actual data almost overlaps with theoretical line. These discussions imply that m = 80 is probably a better choice for a model. However, to have a full picture we need to observe res. time plot, which comes next:



Plot 2: M/M/m models: Res. time prediction vs. actual data

In contrast to TPS plot, RT plot shows opposite behavior for different m's: m = 5 fits actual res. time line better, and m = 80 line has a knee, defining a lower bound for $E[r]$ (which is equal to model's $E[s]$). This implies that M/M/m is not able to truly capture system behavior – either TPS or RT lines will not fit good with real data. It also worth saying that measured RT also contains a knee at point of ~80 clients, but change in slope is not big and can be hardly observed on the plot. Choosing m to be in between 5 and 80 would make bad fit for both TPS and RT. However, I am choosing m as 80 since higher m will yield higher model outputs (e.g. $E[s]$), closer to real measurements (m = 5 would make them badly compatible as in M/M/1 case). To analyze how real behavior is different from theoretical, I provide following table for $E[n]$ = 24 and 192 (actual TPS at these points is higher and lower than model TPS predictions). Also, I initialized 2 new models for these $E[n]$'s, using measured TPS at these points as λ for generating models (this will yield different $E[s]$ than the one in our prediction model). Table follows:

	24 clients (total)			192 clients (total)		
Parameter	Prediction model	Fitted model	Measurements	Prediction model	Fitted model	Measurements
λ or TPS (ops/s)	3029	4763.4	4763.4	10015.5	9236.75	9236.75
ρ	0.307	0.301	-	0.992	0.992	-
$E[s]$ or T_{server} (μs)	7923.37	5050.51	1713.29	7923.37	8591.36	2320.77
$E[n]$	24	24	24.47[i]	192	192	194.21[i]
Std. $E[n]$	4.96	4.90	-	123.94	123.90	-
$E[n_q]$	3.2E-19	9.7E-20	-	112.82	112.78	-
Std. $E[n_q]$	7.8E-10	4.2E-10	-	123.32	123.28	-
$E[r]$ or Res. time (μs)	7923.37	5050.51	5137.20	19187.79	20801.01	21025.29
$E[w]$ or $T_{queue} + T_{left}$ (μs)	1.03E-16	2.03E-17	2062.96	11264.42	12209.65	11434.33

Table 2: M/M/m models' outputs vs. actual data (models have m = 80)

First thing to mention is that $E[s]$ of all models are incompatible with T_{server} values. It follows the same logic as in M/M/1 model: since $E[s]$ is an output of a model, we have no control of what it consists of. In fact, we may assume, that this is some generic service time, including not only T_{server}, but service time for reading a request, writing back response, network cost, hashing. Second, T_{server} increases when going from 24 to 192 clients. Despite T_{server} is considered as 1 component of $E[s]$, in

our prediction model we use constant $E[s]$, which doesn't depict how things work in reality. Change in T_{server} implies than in reality we are dealing with load dependent service times. It's also proved by the fact that $E[s]$ for fitted models differ across 24 / 192 clients. In 24 clients case, fitted $E[s]$ is 3ms lower than constant $E[s]$ of a prediction model (7.9ms vs 5ms). In contrast, opposite happens for 192 clients case: fitted E[s] is 0.7ms higher than predicted E[s] (8.6ms vs 7.9ms).
Predicted $E[s]$ of 7.9ms defines lower bound for $E[r]$, that's why in 24 clients setting predicted TPS in lower than the actual. In contrast, actual TPS for 192 clients is lower than predicted: this can be explained by overhead created by the thread contention, since as we discovered in [MS2 section 1] that we are saturated at ~370 clients, and this is the point where threads are becoming effective, because queue never gets empty after that. The prediction model is not able to capture such a behavior: it is not accounting for effects of multithreading.
Above reasons affect the $E[r]$ of all models and these values behave opposite to TPS / λ behavior.
Regarding $E[w]$ it is possible to say that for 24 clients all models assign negligible value to it. In contrast, our measurement reports 2ms of waiting time. Difference is due to fact that we have more than 1 queue in the system, and T_{left}, e.g., is served by one thread only. There are also 5 GET queues and each queue is assigned to 16 threads, while the model assumes that there is 1 big queue and 80 workers. For 192 clients both predicted and fitted $E[w]$ are close to measured ones, however we assume that $E[w]$ implicitly includes network waiting time, while measured values do not include it, so it's still hard to compare them.
In general, TPS predictions are close to the ones observed, however this is only 1 aspect of the model. Discussions above show that despite performance pattern is somewhat similar, general concepts and architecture of model and our system are different and it makes hard to compare model output to observed values.

2.2 Scalability with increase in resources

In this subsection I will analyze scalability of M/M/m model with increase in server count and map it to behavior of our system. For that I will be using results of MS2 experiment 2, using measurements for 3, 5 and 7 servers with no replication (MS2 log file **exp_2_total**).
First, I define a base case, for which I will initialize a model. As my base case I'm taking 3 server configuration to analyze predictions for increasing server count. Following is the table with models' values and measurements, explanation follows.

Table 3: Values for increase in # of servers

	3 servers		5 servers		
Parameter	Base model	Measurements	Prediction model	Fitted model	Measurements
λ or TPS (ops/s)	14172.1	14172.1	23612.71	12089.85	12089.85
ρ	0.997	-	0.997	0.997	-
$E[s]$ or T_{server} (μs)	3587.73	2199.81	3587.73	7007.21	2571.24
$E[n]$	372	374.57[i]	372	372	376.69[i]
Std. $E[n]$	330.25	-	298.94	298.93	-
$E[n_q]$	321.18	-	287.32	287.32	-
Std. $E[n_q]$	330.09	-	298.66	298.65	-
$E[r]$ or Res. time (μs)	26250.77	26429.92	15755.88	30772.49	31157.39
$E[w]$ or $T_{queue} + T_{left}$ (μs)	22663.04	18237.99	12168.15	23765.28	18969.63

As an input for base model I am using λ (TPS for that experiment) = 14172.1 ops/s and $E[n]$ (number of clients) = 372 (124*3). Following last subsection, I choose m = 51 (3 servers * (16 read + 1 write thread). These inputs allow me to obtain μ = 278.73 ops/s. Having μ and knowing and $E[n]$ allows me to make predictions for different number of servers. To make predictions for 5 servers configuration, I use μ obtained above, same $E[n]$ and updated m, which equals to 85 (3 servers * (16 read + 1 write thread). From the above table it is seen, that predicted λ for 5 servers equals to 23612.71 ops/s.

In general, this prediction follows the idea that increasing m with fixed μ will increase "capability" of the system / model. Predicted TPS demonstrates what TPS should be equal to, in order to obtain same number of jobs in the system. Comparing this value with real measurements from above table we can see that system behaves in exactly opposite way – actual TPS of the system decreases. As explained in MS2 report section 2, this comes from the fact that middleware (MiddlewareServer class employing single thread) gets slower with higher number of threads. Model, however, doesn't assume that increase in m may cause degrading in capability to serve jobs.

Results in Table 3 show that $E[s]$ for base model is bigger than T_{server} for that configuration – reason for that was explained in previous subsection. $E[w]$ is also different from waiting time in the middleware, and it is probably bigger because it implicitly includes network waiting time. Predictions for 5 server configuration report smaller both $E[r]$ and $E[w]$ - as a consequence of higher system capability.

The model fitted for 5 servers configuration was initialized using λ as actual TPS of that configuration, $E[n]$ = 372 and m = 85. Table 3 clearly demonstrates that $E[s]$ for that model is ~2 times bigger than $E[s]$ from prediction model (~7ms vs ~3.6ms). This proves that our real service time gets bigger with increase in number of servers due to threads context switch. Same as is 3 servers case, $E[w]$ is larger than waiting time in middleware due to including network times.

Since model is not capable to relate change in service time to change in m, we can conclude that M/M/m model is not applicable to analyze scalability with increase in resources. Also, same as in previous subsection, it can't provide outputs comparable to real measurements due to difference in architecture of system and model. To conclude this section, I provide predictions, fitted model and real measurements for 7 servers configuration (m = 119) which follow all the above discussions and show same trends:

	7 servers		
Parameter	Prediction model	Fitted model	Measurements
λ or TPS (ops/s)	33044.62	9277.6	9277.6
ρ	0.997	0.997	-
$E[s]$ or T_{server} (μs)	3587.73	12778.65	3459.47
$E[n]$	372	372	373.89[i]
Std. $E[n]$	267.15	267.15	-
$E[n_q]$	253.49	253.48	-
Std. $E[n_q]$	266.72	266.72	-
$E[r]$ or Res. time (μs)	11258.78	40100.87	40300.55
$E[w]$ or $T_{queue} + T_{left}$ (μs)	7671.05	27322.22	23605.02

Table 4: M/M/m models' outputs vs. actual data for 7 servers (m = 119)

3 System as Network of Queues

In this section I will be modelling my system as a network of queues, and for that purpose, I will be using results of MS2 experiment 3 (effect of writes). In order to define the architecture of queueing network, I will select 5 server configuration. It's worth to mention that 3 instances of experiment will be analyzed: 1% writes, 5% writes and 10% writes for replication factor equal to 1.
Following is the schematic of my queueing network's architecture, explanations follows:

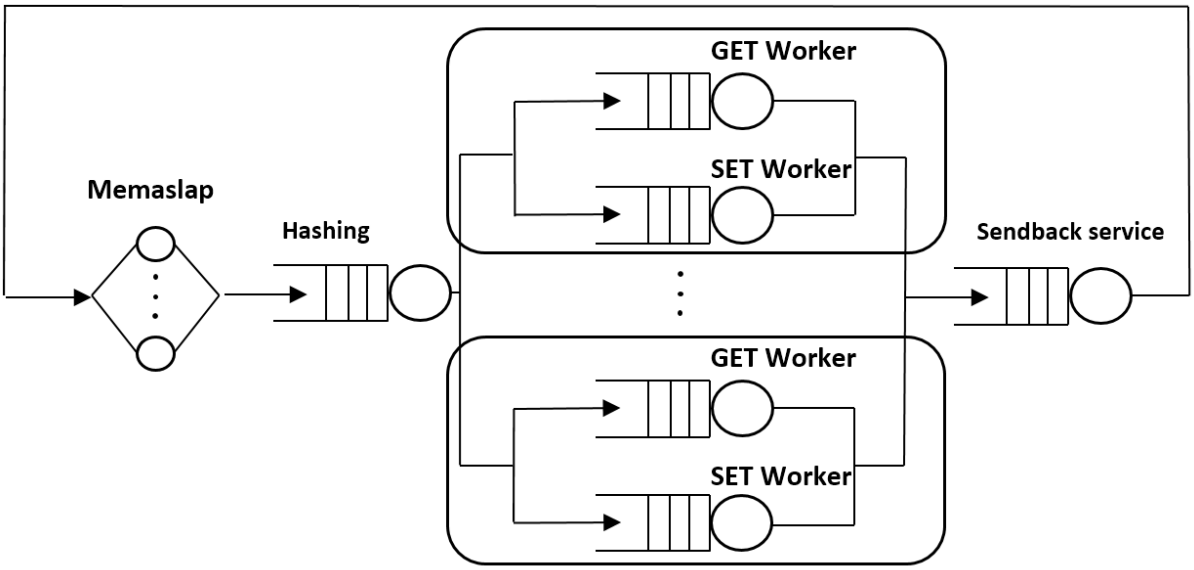


Figure 2: Queueing network architecture

We are dealing with closed queueing network which always assumes job flow balance. On the figure above, memaslap represents client side and is not explicitly modeled (doesn't represent a device). First actual device in the architecture is hashing device. It parses the request, hashes it's key, and based on the hash value determines where the request goes next. After that, we have 2 worker devices per memcached server – GET and SET workers, which are doing work in parallel. Since we will be dealing with 5 memcached servers in this section, we will have 10 of these devices. The last device in the architecture is a sendback device. As was mentioned several times in milestone 2 report, after middleware receives response by memcached, it forwards it to our MiddlewareServer class, where response is waiting to be sent back in a big loop iterating over SelectedKeys (typical for Java NIO). Ideally, total response time of a network should be equal to middleware measurement T_{total}. But apart from T_{total}, there also exists some lag in between memaslap and MW. This lag consists of network times (memaslap to MW and MW to memaslap) and time, that request spends waiting to be read by middleware. We can't split that lag time into corresponding categories not due to incorrect measurement, but because there just exists no way of defining it. Due to such a reasoning, in our architecture we assume that requests are falling into hashing queue as soon as they are sent by memaslap. We then use lag time $T_{memaslap} - T_{total}$ as "think time", to compensate for it. Later, when we will analyze the bottlenecks we will see, why not including separate device for reading requests doesn't prevent us from defining what is a bottleneck in my system.
All devices are modeled as M/M/1. Using M/M/m for gets could be difficult, since we do not know

exact number of effective threads (m). Apart from that, memcached server employs 1 thread and serves requests in M/M/1 style. Using M/M/1 for other devices apart from GET worker seems logical. For the analysis of this network, I will use MVA algorithm for closed networks. As an input I need the following: number of jobs in the system (number of memaslap virtual clients – for us it's 372), visit ratio and service time for each device, and optionally, think time.

Visit ratios are equal to 1 for hashing and sendback devices. For worker devices, they are equal to $\frac{1}{5}(for\ 5\ memcached\ servers) * ratio\ of\ op.type$: e.g., with 10% writes, each SET worker will have visit ratio of 1/5 * 0.1 = 0.02. With visit ratios we can calculate arrival rate for each device.

Using service times from middleware measurement created difficulties. For sendback device we don't know service time: writing back itself might be superfast but we have long waiting times due to context switches, as described in MS2 report. Attempts have been made to use T_{server} times as service times for worker devices, but they didn't give meaningful results. Explanation is the following: T_{server} doesn't contain effect of context switch overhead in itself, while T_{queue} does include it. Using T_{server} as service time then will yield much lower waiting time for the device. However, we know total response time for each device. So, for obtaining proper estimates of service times for workers and sendback devices, I decided to estimate $E[s]$ using response time and arrival rate for each device. For workers, total response time of a device is $T_{server} + T_{queue}$ (for gets and sets correspondingly), while for sendback device it is equal to T_{left}. The only exact service time we have is for hashing (MS2 log file **hash_test)**. Mean values for hashing were corrupted by outliers, so I used 75th percentile as estimation of hashing time and obtained stable result of 1 μs (across all logs). After estimating it, I also subtract 1 μs from T_{left} since T_{left} included hashing time. After estimating $E[s]$ for each device, we are able to use MVA to obtain throughput, response time, utilization and queue length for each device. Following table contains estimates and measures from our middleware logs, as well as the outputs of MVA algorithm for 5 servers, 5% writes configuration:

Devices:	Hashing	SET workers	GET workers	Sendback
Inputs and other derived parameters				
Device's visit ratio	1	0.01	0.19	1
λ_i (ops/s)	**13036.875**	**130.36875**	**2477.00625**	**13036.875**
$E[r]_i$	1.01321E-06	**0.00669517**	**0.008223275**	**0.011260788**
$E[n]_i$	0.01320908	0.872841007	20.36910344	146.8054807
$E[s]_i$ (s)	**0.000001**	0.003574874	0.000384821	7.61865E-05
ρ_i ($E[s]_i * \lambda_i$)	0.013036875	0.466051845	0.953203465	0.993234351
Think time (Z)	0.009447723			
MVA outputs				
Device's TPS (ops/s)	13115.55894	131.1555894	2491.956199	13115.55894
Device's RT (s)	1.0133E-06	0.006730401	0.00919407	0.009843637
# of jobs at device	0.013289859	0.882729753	22.91121955	129.1047957
ρ_i	0.013115559	0.468864707	0.958957077	0.999228532

Table 5: Derived parameters for MVA and MVA outputs

In table above bold values were used as input to estimate other values in the table's upper part (apart from think time, which equals to lag time). We can clearly see that MVA outputs for TPS and RT are

almost same as λ_i and $E[r]_i$ from the inputs, which is not surprising, since we used that values to derive all service times (except for hashing's). There is small 1.4ms difference in $E[r]_i$ and device's RT for sendback, which is due to high utilization of device. Same can be told about calculated $E[n]$ and MVA's number of jobs. MVA output, thus, proves the validity of our network, inputs and derived values. Device with highest utilization is **sendback device**, which means that it is our bottleneck. To prove that sendback is a consistent bottleneck, I ran MVA for other configurations (5%, 10% writes). I will not provide TPS, RT and # of jobs outputs since they consistently following the actual values. Following is the total table with utilizations per device for all configurations:

MVA outputs	Config.	Hashing	SET workers	GET workers	Sendback
ρ_i	1% writes	0.012150063	0.122228153	0.962251325	0.99897212
	5% writes	0.013115559	0.468864707	0.958957077	0.999228532
	10% writes	0.013959796	0.685683966	0.954626418	0.999443227

Table 6: MVA utilizations for each device / configuration

Above table demonstrates some clear trends. Utilization for SET workers is increasing with increase in writes percentage, and decreases for GETS workers, which is totally reasonable since we add more load to *sets* and decrease load for *gets*. This increase is more intensive for *sets* since when going from 1% to 10% we increase load 10 times, while for *gets* decrease is really slow – going from 99% to 90% is just decrease of 1/11 portion of the load. As utilizations of *sets* and *gets* balance, we see higher utilization in hashing, meaning that we are able to squeeze more out of it. It follows the general description in MS2 report section 3, where with higher writes percentage we were seeing higher performance. Sendback device is our consistent bottleneck with highest utilization across all configurations, and its utilization increases with write percentage (and performance) increase.
Despite we identified bottleneck device as sendback, looking from the prospective of my system, it would be more proper to identify bottleneck resource. Earlier in this section I was mentioning that not explicitly including "reading device" in the queueing network doesn't prevent us to identify bottleneck. Reason for that is the following: reading requests, sending back responses and hashing – all these 3 operations are done by a single thread of my MiddlewareServer class. Hashing's time is very small and is not affected by context switch (otherwise, it would be much higher), so explicitly including hashing device allowed us to define that it is the least source of problems for MiddlewareServer's thread. However, we are not able to split reading times for sending back times – due to the design of MiddlewareServer class. After subtracting hashing time from T_{left}, it consists of waiting time, and it includes waiting while other requests are read and hashed, waiting while other responses are sent back – it is all done in one big loop and serving discipline is not FCFS.
However, above analysis allows to tell us that both reads from memaslap and writes back cause MiddlewareServer's thread to be our bottleneck resource. Getting rid of that bottleneck would involve either allocating more resources (e.g., more threads), or a complete change of architecture where requests would be sent back to memaslap directly from GET/SET worker devices.
Despite it is not possible to model each aspect of my system due to architectural limitations, network of queues gives enough insight into performance of each important component of my MW, and proved "bottleneck assumptions" made in MS2 report sections 2 and 3.

4 Factorial Experiment

I have done series of new experiments in order to design $2^k r$ factorial experiment. The setup of these experiments is up to the following table:

Memcached VM's	5
Memaslap VM's	3
Replication factor	Full replication
Number of threads PTP	16
Virtual clients / VM	64, 128
Value size	128B (small), 512B (large)
Writes proportion	1%, 10%
Runtime x repetitions	90s x 5
Warm-up, cool-down period	20s
Sampling ratio	1 of 100 requests
Memaslap output	once in 5s
Log files	2kr

In my $2^k r$ factorial experiment I have chosen number of factors k = 3, and number of repetitions r = 5. Factors are: writes proportion, value size and virtual clients / VM. The performance metric for this analysis will the Throughput (ops/s). Following discussions from milestone 2, we know that effect of write proportion and number of clients is monotonic within the range selected (for #clients we know this for 5 memcached's and thread pool size = 16, so we use same setup here). Hypothesis regarding value size is that performance will monotonically decrease with higher value size. Full replication is used to better see impact of writes. Last 5 rows of setup table conform to methodology of all 3 experiments of MS2. Following table provides TPS metrics for each configuration / repetition:

Configuration	run #1	run #2	run #3	run #4	run #5	Avg. TPS
64cl / w1% / 128B	10011.4	9693.6	9959.5	9787.4	10129.5	9916.28
64cl / w10% / 128B	10417.3	9954.6	10106.2	10086.7	9922.1	10097.38
64cl / w1% / 512B	10165.1	10190.9	9925.9	9680.8	9513.2	9895.18
64cl / w10% / 512B	9948.3	9673.7	10133.3	9804.6	10296.9	9971.36
128cl / w1% / 128B	11754.2	11073.9	11293.5	11011	11705.8	11367.68
128cl / w10% / 128B	11693.4	12589.9	12105.5	11774.7	11563.9	11945.48
128cl / w1% / 512B	10977.9	10962.6	10827.2	10713.7	10783.9	10853.06
128cl / w10% / 512B	11861.3	10906.5	11700.8	11017.4	10942.2	11285.64

Table 7: Measured TPS for each configuration / repetition

Above table shows that expected effect on performance is observed, and best performance is given by configuration "128cl / w10% / 128B". To proceed with analysis using $2^k r$ experiment, I assign 3 variables (one per factor) with two levels for each:

Writes proportion (A)	Value size (B)	Virtual clients / VM (C)
A = -1 if writes proportion = 1%	B = -1 if value size = 128B	C = -1 if virtual clients / VM = 64
A = 1 if writes proportion = 10%	B = 1 if value size = 512B	C = 1 if virtual clients / VM = 128

Table 8: Definition of variables and their levels for factors

Response variable is defined as Y and will depict our Throughput.

After defining the variables, it is possible to use nonlinear regression to obtain coefficients "q" for each of the factors. This is done with the help of "sign table", which is provided next with corresponding coefficients:

I	A	B	C	A*B	A*C	B*C	A*B*C	Y (TPS)
1	-1	-1	-1	1	1	1	-1	9916.28
1	1	-1	-1	-1	-1	1	1	10097.38
1	-1	1	-1	-1	1	-1	1	9895.18
1	1	1	-1	1	-1	-1	-1	9971.36
1	-1	-1	1	1	-1	-1	1	11367.68
1	1	-1	1	-1	1	-1	-1	11945.48
1	-1	1	1	-1	-1	1	-1	10853.06
1	1	1	1	1	1	1	1	11285.64
q_0	q_A	q_B	q_C	q_{A*B}	q_{A*C}	q_{B*C}	q_{A*B*C}	
10667	158.5	-165.2	696.5	-31.27	94.14	-128.4	-5.038	

Table 9: Sign table and estimated coefficients

Last row of the above table contains estimated coefficients. q_0 is mean TPS across all experiments. The sign of other coefficient determines if going from -1 to 1 within that factor yields positive or negative impact on the response (TPS). Columns 4 to 8 are used to determine impact of interaction terms – e.g., varying only one factor may give no impact but varying two of them at the same time may impact performance.

From the estimated coefficients we may infer that increasing number of virtual clients / VM (factor C) has the largest coefficient, thus gives biggest positive impact to the performance (follows up with behavior in max. TPS experiment of MS2). Value size (B) has negative impact on performance when going from 128B to 512B, which proves our hypothesis. Writes proportion (A) impacts positively (exactly as in MS2 experiment 3) on TPS, but it's impact is less than the one of number of clients. We also may observe that interaction term between value size and clients number (B*C) has negative impact on performance. In our context we can interpret it as "negative impact of value size gets bigger when going from 64 to 128 clients per VM". Other interaction terms seem to have small effect on performance, so their significance still is to be tested.

$2^k r$ experiment also allows us to compute sum of squared error for each of the factors (and their interaction terms) and SSE – unexplained variation due to fluctuations of TPS across repetitions. Ratio of these values to SST (total sum of squares) defines percentage of total variation, explained by corresponding terms. These percentages are presented in the table below:

A	B	C	A*B	A*C	B*C	A*B*C	Experimental error
3.94%	4.28%	76.14%	0.15%	1.39%	2.59%	0.00%	11.50%

Table 10: Percentage of explained variation for factors and repetitions

It is seen from the above table that factor C (clients / VM) explains 76.14% of total variation. Next biggest number – 11.5% - is due to experimental error. Looking at the above table we observe that differences in TPS across repetitions are larger than differences due to change in value size or writes proportion. But this doesn't allow us to say that these factors are insignificant. To judge about significance, we should obtain confidence intervals for all coefficients q.

Using formulas from the book I was able to compute std. dev. for all effects q which is equal to 47.85.

95% confidence intervals for all coefficients are then calculated and shown in the below table:

q_0	q_A	q_B	q_C	q_{A*B}	q_{A*C}	q_{B*C}	q_{A*B*C}
(10585.3, 10747.7)	(77.3, 239.7)	(-246.4, -84)	(615.3, 777.7)	*(-112.4, 49.9)*	(12.9, 175.3)	(-209.6, -47.2)	*(-86.2, 76.2)*

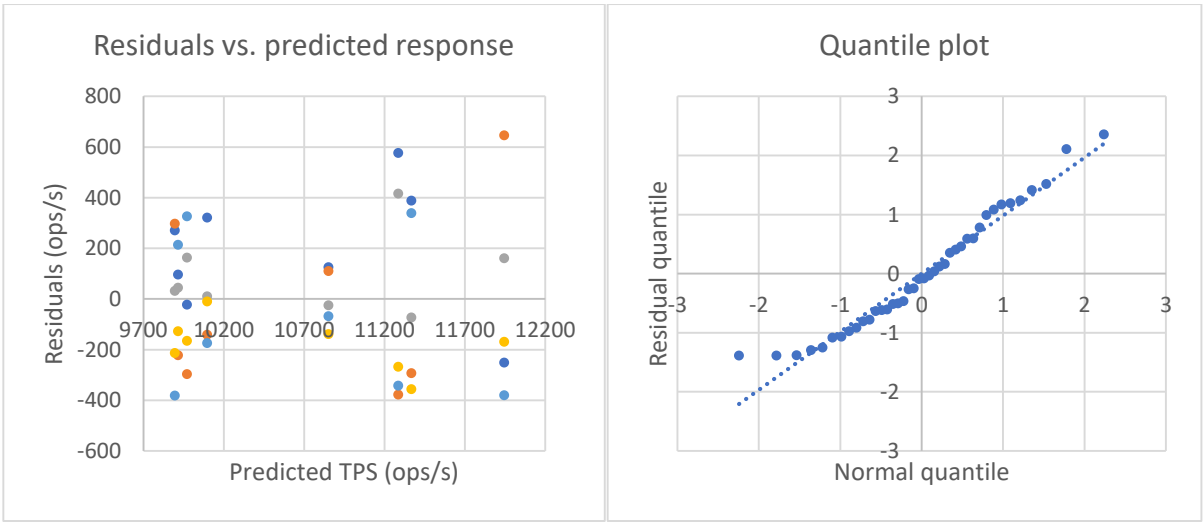
Table 11: 95% confidence intervals for coefficients q

The table shows that 2 coefficients - q_{A*B} and q_{A*B*C} include 0 in their confidence intervals, thus are insignificant. Also, interaction term of writes proportion and clients per VM (A*C) and value size and clients per VM (B*C) are significant.

First term makes sense because higher writes proportion leaded to lower reads proportion (MS2 section 3) and was speeding up gets – e.g., decreasing their queue time. With 64 clients / VM we are not yet saturated so decrease in queue time for gets would be not so big. This explains why going from 1% to 10% for writes has bigger impact with 128 clients / VM.

Value size, on the other hand, should not affect speed of operations inside the middleware, since actual reading from and writing to buffers (when buffers already contain data) are very fast operations. What should be affected is a network performance. Since network is exposed to higher load with higher number of clients, second interaction term also seems reasonable. With higher load of the network, transferring 4 times bigger value affects the performance more than with lower load.

Interesting takeaway from these is that, despite we observe larger fluctuations in TPS due to experimental randomness than due to many other factors, proper 2^kr analysis allows to state that majority of factors still provides significant impact on performance. To show that experimental errors are not correlated and are normally distributed, I provide a scatter plot of residuals versus predicted TPS and quantile plot:



Plots 3 and 4: Residuals vs. predicted TPS and Residual quantiles vs. normal quantiles

No correlation or pattern in residuals with increase in predicted TPS is observed on the first plot, which means that residuals are indeed random. Quantile plot shows that residuals are normally distributed, and thus proves that additive model was a proper choice for 2^kr factorial experiment design.

5 Interactive Law Verification

I will be checking the validity of my experiments with Interactive Response Time Law in this section. The experiment of milestone 2 I chose to validate is the third one – Effect of Writes. It contains 18 different configurations: number of servers is 3, 5, 7; replication factor is R = 1 and R = full; writes proportion is 1%, 5%, 10%. For details about setup of the experiment refer to setup table in milestone 2 repost, section 3, page 16.

Interactive Response Time Law implies that Response Time = Number of Clients / Throughput – Think time. Since we have no information about think time and we know, that memaslap should send a new request as soon as it gets response for the previous one, we will assume that think time Z = 0. Later, if discrepancies will be observed, we may proceed with estimation of the think time. Following table contains measured response times from memaslap as well as calculated response times from IRTL:

	Writes proportion	3 servers		5 servers		7 servers	
		R = 1	R = full	R = 1	R = full	R = 1	R = full
Measured RT (μs)	1%	24446.60	24989.69	31028.71	31426.03	39719.83	39543.88
	5%	23330.02	23738.18	28855.70	29234.29	37071.93	38529.55
	10%	22711.57	24518.10	27124.81	28237.70	35360.78	36843.02
Calculated RT (μs)	1%	24220.41	24751.94	30797.89	31033.36	39548.59	39317.34
	5%	22957.79	23554.33	28534.45	28929.49	36838.98	38246.72
	10%	22480.74	24266.22	26814.34	28059.85	35147.14	36649.52

Table 12: Measured and calculated res. times for all configurations

We can observe from the above table that calculated values are pretty close to observed ones, however small discrepancies still exist. Considering that it might be a think time, we can compute what a think should be to compensate for these discrepancies. The formula mentioned above suggests that we can compute think time as Z = Calculated RT – Measured RT. Following table then demonstrates computed think times for all 18 configuration of the experiment:

	Writes proportion	3 servers		5 servers		7 servers	
		R = 1	R = full	R = 1	R = full	R = 1	R = full
Assumed think times (μs)	1%	-226.19	-237.75	-230.82	-392.67	-171.24	-226.54
	5%	-372.22	-183.85	-321.26	-304.80	-232.95	-282.83
	10%	-230.83	-251.87	-310.47	-177.85	-213.64	-193.49

Table 13: Assumed think times for all configurations

The numbers computed are all negative. In reality, think time cannot be negative. It may imply that virtual client sends a new request before he gets response for the previous, which can't happen while we are in the closed system context. More proper assumption is that think time in fact equals to 0, and these discrepancies are just due to memaslap measurement errors. To show that it is just negligible error in measurement I compute ratio of this "error" to measured response time. Following table contains percentages for each of 18 configurations of the experiment, showing within which percentage of measured response time is the corresponding error:

	Writes proportion	3 servers		5 servers		7 servers	
		R = 1	R = full	R = 1	R = full	R = 1	R = full
Percentage	1%	0.93%	0.95%	0.74%	1.25%	0.43%	0.57%
	5%	**1.60%**	0.77%	1.11%	1.04%	0.63%	0.73%
	10%	1.02%	1.03%	1.14%	0.63%	0.60%	0.53%

Table 14: Ratio of errors to measured RT for all configurations

We see that for 7 out of 18 experiments ratio of error to measured response time is within 1%, for 15 out of 18 experiments ratio is less than 1.3% and the one last highest value report 1.6%. Assuming such a small error, we can state that these discrepancies are due to some internal inconsistencies in memaslap measurement style.

Despite it is impossible to state the clear reason, since memaslap is 3rd party software, several possible explanations for that may exist. One of the reasons could be that each memaslap process on one of three VM's employs 124 threads, while VM itself has only two cores. In reality, only two threads may happen exactly in parallel, and for other ones, CPU has to switch to them. It might be possible that due to thread switch, memaslap reads response later than it actually arrives to the client, and this small difference causes this error. Another reason could be that after receiving response, memaslap first sends new request, and only then fixes the time of arrival of previous response, which conceptually might be equal to "sending new request earlier than previous arrived", despite still conforming to closed system context.

Unfortunately, it is not possible to verify any of the assumptions so we will just assume that this is some "generic measurement error".

To conclude, I would also like to mention that these small discrepancies are the exact reason why number of jobs in the system, calculated through Little's Law for our measurements in section 1 and 2 is slightly higher than actual number of clients.

Logfile listing

Short name	Location
mmm_tps	https://gitlab.inf.ethz.ch/babayevt/asl-fall16-project/blob/master/milestone_3_logs/mmm_tps.7z
2kr	https://gitlab.inf.ethz.ch/babayevt/asl-fall16-project/blob/master/milestone_3_logs/2kr.7z

[i] Refer to last paragraph of section 5 (IRTL checks)