Advanced Systems Lab (Fall'16) – Third Milestone

Name: *Rami Khalil*
Legi number: *16-932-568*

Grading

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

1 System as One Unit

In this section we build an M/M/1 queue model that represents our middleware system as an individual device. We use the data from the stability trace we ran in the first milestone (Section 3) to populate the parameters of this model.

1.1 Introduction

An M/M/1 queue denotes a queue model whereby: **1)** The interarrival times and service times of jobs are exponentially distributed. **2)** There exists only one server that processes jobs. **3)** The buffer size of the server is assumed to be infinite. **4)** The population size is assumed to be infinite. **5)** The service discipline employed is First Come, First Serve (FCFCS).

We assume a Birth-Death process model, whereby jobs arrive individually, and not in batches. In designing this model, We make extensive use of the fact that we operate on a closed system. Therefore, we make the following notable simplifications: **1)** No jobs are lost in the system. Jobs exit the system only when serviced. **2)** The total number of jobs in the system is constant. **3)** The flow of jobs re-entering the system after being serviced defines the throughput.

Moreover, since we model the system as a single device in a closed loop, and assume that it maintains job flow balance, the operational laws governing our system are simplified according to the Forced Flow Law as follows: **1)** $A = C$. The number of arrivals A is equal to the number of completions C. **2)** $\lambda = X$. The arrival rate λ is equal to the throughput X.

Given all this, we do not expect the M/M/1 to at all accurately model our system, as there are many details for which it does not account for, given that a large portion of our system is parallelized. This will lead to the model making bad predictions about our system's behavior.

1.2 Methods

We will extract data from middleware and memaslap logs produced by the trace[1], ignoring the first and last 300 seconds of data to avoid warmup and cooldown.

1: $\lambda = \bar{X}$. The mean throughput $\bar{X}$ is extracted from the memaslap logs and used as the mean arrival rate.

2: $E[s]_{Serialized} = \frac{99}{100} * \frac{1}{30} E[T^{GET}_{Processing}] + \frac{1}{100} * \frac{1}{9} E[T^{SET}_{Processing}]$ In order to extract the service rate parameter for the model μ, we will estimate the serialized mean service time from $T_{Processing}$ in the middleware logs. Since our middleware system is in reality not composed of a single queue and one server, we transform the data we observe to fit a single queue model such that any times measured for processes done in parallel are virtually serialized: **GET Jobs:** These jobs are split across 3 queues, each of which has 10 reader threads. The mean service time for these jobs should be divided by 30, in order to vritually serialize the operations done in parallel by all the reader threads at once. **SET Jobs:** These jobs are split across 3 servers, and then processed asynchronously with full replication. For each SET job, three operations are performed in parallel by a writer thread. To virtually serialize this, we divide their mean service time by a factor of 9. The total mean service time is the weighted average of mean GET service time and mean SET service time, where the weights are the ratios of the request types (99% GET, 1% SET)

3: $\mu = \frac{1}{E[s]_{Serialized}}$. We will use the above estimated mean **serialized** service time E[s] to calculate the service rate of the model as follows:

We extract the mean response time $\bar{R}$ and its variance σ^2_R from the memaslap logs, and the mean queueing time $\bar{T_{Queue}}$ and its variance σ^2_Q from the middleware logs, and combine them with Little's Law, $Q_i = X_i R_i$, to derive the following:

4: $E[r]_{Measured} = \bar{R}$ **5:** $E[n]_{Measured} = E[r]_{Measured} * \lambda$

[1]trace.logs/*

6: $E[w]_{Measured} = \bar{T}_{Queue}$ **7:** $E[n_q]_{Measured} = E[w]_{Measured} * \lambda$

We choose to only deal with a subset of the descriptive variables associated with the theoretical M/M/1 model, as to create a succint, meaningful discussion about the signficant areas this section is meant to tackle.

1.3 Results

Metric/Parameter	**Symbol**	**M/M/1 Equation**	**Model**	**Measurement**
Throughput/Arrival Rate	λ	$\lambda = \bar{X}$	10012	
Service Rate	μ	$\mu = \frac{1}{E[s]}$	13710	-
Traffic Intensity	ρ	$\frac{\lambda}{\mu}$	0.73	-
Mean # of jobs in system	$E[n]$	$\frac{\rho}{(1-\rho)}$	2.71	196.1
Mean # of jobs in queue	$E[n_q]$	$\frac{\rho^2}{(1-\rho)}$	1.98	129.6
Mean response time	$E[r]$	$\frac{1}{\mu(1-\rho)}$	270.14us	19.61ms
Var. of the response time	$V[r]$	$\frac{1}{\mu^2(1-\rho)^2}$	0.07us	64.30ms
Mean waiting time	$E[w]$	$\rho\frac{1/\mu}{1-\rho}$	197.46us	12.96ms
Var. of waiting time	$V[w]$	$\frac{\rho(2-\rho)}{\mu^2(1-\rho)^2}$	0.07us	270.21ms

Table 1: M/M/1 Queue Model Metrics

1) The Model column is the prediction resulting from substituting the parameters (λ, μ) into the M/M/1 model equations in the equation column. **2)** The Estimate column values are derived as mentioned in the Methods subsection (1.2).

1.4 Analysis

We see a stark difference between the orders of magnitude of the values predicted by the model, and those we measured and derived. This affirms our previously stated expectation of the failure of this model to accurately describe our system, given that the model takes only two parameters (λ, μ), describing only the rates of input and output, it tries to build a primitive, singular processing queue which would perform as efficiently as our complicated system, if all requests were processed purely sequentially.

1. ρ**:** We see that the system is stable. As our trace had run for an hour, without system instability, we therefore accept this prediction to be true.

2. $E[r], V[r]$**:** The forced serialization means the flow of jobs in and out of the system is quicker, and thus has significantly lower response (E[r]) time than in the real-life system, such that the theoretical single queue matches the throughput performance of the multiple servers in the implemented system. Multiplying the model's E[r] by a factor of ≈ 33 would bring its values closer to those of the real life system by removing the forced serialization. However, there would still be an offset that is roughly equal to the network latency, which the model doesn't account for. The simplicity of the model also leads it to predict very low variance values for response time, which in contrast to the real life values are minuscle.

3. $E[w], V[w]$**:** The model's queueing time is also much smaller and more stable than that measured in the experiment[2]. This behavior is also due to the fact that the model forces the single server to clear out the queue exceptionally fast to match the service rate.

4. $E[n], E[n_q]$**:** The differences between the estimated and measured values for these quantities [3] is explained by the differences in E[r] and E[w], and the consequences of Little's Law.

[2]We attribute the exceptionally high E[w] and V[w] values in this experiment to the very unstable Azure environment.

[3]E[n] is actually estimated to be larger than the population size because mean response time is an over estimator of the "average" of the distribution (See Section 5).

2 Analysis of System Based on Scalability Data

In this section we build a series of M/M/m queue models that attempt to approximate our middleware system in different configurations. We use the data from the experiments conducted to investigate the effect of replication in the second milestone (Section 2) to populate the parameters of these models.

2.1 Introduction

We hold all of the definitions and assumptions previously stated in section 1.1, with the following exceptions: **1)** In an M/M/m queue, there is not one server, but m servers that service requests. Therefore, we expect time estimates by the model to be closer to real life measurements than those predicted by the M/M/1. **2)** The M/M/m model is still not expected to properly depict the behavior of our system, as we expect the scalability of the model to more representative of an ideal system implementation, and to be unable to accurately predict the scaling behavior of our real-life implementation.

Additionally, the reader should review and keep in mind the hypotheses and conclusions made about our system in section 2 of the second milestone.

2.2 Methods

We will extract data from middleware and memaslap logs produced by effects of replication experiments[4], ignoring the first and last 15 seconds of data to avoid warmup and cooldown.

1: $\lambda = \bar{X}$. As in the previous section.

2: $E[s]_{Serialized} = \frac{95}{100} * \frac{1}{10} E[T^{GET}_{Processing}] + \frac{5}{100} * \frac{1}{P} E[T^{SET}_{Processing}]$ The reasoning is similar as in the previous section, but since we have m servers, we make minor adjustments: **GET Jobs:** We only divide the service time by the number of reader threads, since the number of servers is represented in the model's calculations by the variable m. **SET Jobs:** We only divide by the replication factor, denoted by P in the equation.

3: $\mu = \frac{1}{E[s]_{Serialized}}$. Same as before.

We extract the mean response and queueing times as in the previous sections, and also combine them with Little's Law as before. Again, We will only deal with a subset of the descriptive variables associated with the model to create a meaningful discussion. When presenting the results and analysing them, we will refer to the metrics by their shorthand symbols, as shown in table 10, for brevity.

We then use the throughput data from milestone 2, experiment 1, to compare the response time predictions of one of the models with those measured by the system.

2.3 Results

We present the results using tables 2, 3 and 4. The replication factor of the configuration is denoted by R at the top of each table. For each configuration of servers (m), three replication factor configurations (R) are presented, as in the experiment in milestone 2, section 2.

Each model column contains the value calculated according to the equations in table 10. The Experiment column contains values directly measured from logs or derived using operational laws, as specified in section 2.2.

[4]effects/*

Metric	R = 1		R = 2		R = 3	
	Model	**Experiment**	**Model**	**Experiment**	**Model**	**Experiment**
λ	7934		8259		8044	
μ	3328	-	3537	-	3670	-
ρ	0.79	-	0.78	-	0.73	-
ϱ	0.64	1	0.61	1	0.54	1
$E[n]$	4.86	43.17	4.48	43.75	3.65	43.55
$E[n_q]$	2.47	4.20	2.15	4.40	1.46	3.40
$E[r]$	612.19us	5440.61us	542.87us	5297.39us	454.04us	5414.38us
$V[r]$	0.30us	29.88us	0.23us	2.28us	0.16us	29.53us
$E[w]$	311.69us	529.29us	260.17us	532.66us	181.53us	422.93us
$V[w]$	0.21us	2961.65us	0.15us	2434.49us	0.09us	1805.21us

Table 2: M/M/3 Queue Models

Metric	R = 1		R = 3		R = 5	
	Model	**Experiment**	**Model**	**Experiment**	**Model**	**Experiment**
λ	9188		8774		8697	
μ	3615	-	4195	-	4360	-
ρ	0.51	-	0.42	-	0.40	-
ϱ	0.14	1	0.07	1	0.06	1
$E[n]$	2.68	45.45	2.14	45.11	2.03	44.66
$E[n_q]$	0.14	1.96	0.05	1.69	0.04	1.72
$E[r]$	292.17us	4946.35us	244.10us	5141.68us	233.84us	5135.38us
$V[r]$	0.08us	7.26us	0.06us	28.78us	0.05us	2.22us
$E[w]$	15.52us	212.99us	5.75us	192.92us	4.51us	197.51us
$V[w]$	0.00us	992.03us	0.00us	744.19us	0.00us	700.25us

Table 3: M/M/5 Queue Models

Metric	R = 1		R = 4		R = 7	
	Model	**Experiment**	**Model**	**Experiment**	**Model**	**Experiment**
λ	9299		8923		8795	
μ	3972	-	4868	-	4900	-
ρ	0.33	-	0.26	-	0.26	-
ϱ	0.01	1	0.00	1	0.00	1
$E[n]$	2.35	45.63	1.83	45.27	1.80	45.20
$E[n_q]$	0.01	1.68	0.00	1.50	0.00	1.59
$E[r]$	252.38us	4906.49us	205.53us	5073.57us	204.19us	5139.19us
$V[r]$	0.06us	9.53us	0.04us	17.01us	0.04us	23.76us
$E[w]$	0.60us	180.81us	0.12us	168.35us	0.10us	180.42us
$V[w]$	0.00us	830.52us	0.00us	630.86us	0.00us	732.34us

Table 4: M/M/7 Queue Models

Notably, the reader should keep in mind the volatility of performance on Azure due to inconsistent allocation of virtual machines at different times. Therefore, the results presented in this section should not be expected to be similar to those in section 1.3, neither in terms of model predictions nor in terms of measured performance under similar configurations, as the experiments done to generate their data were carried out at different times on different machine allocations.

We choose to fix the value of ϱ to be equal to one in our measurements since we know the behavior of our system forces requests to enter a queue, even when the system is underloaded.

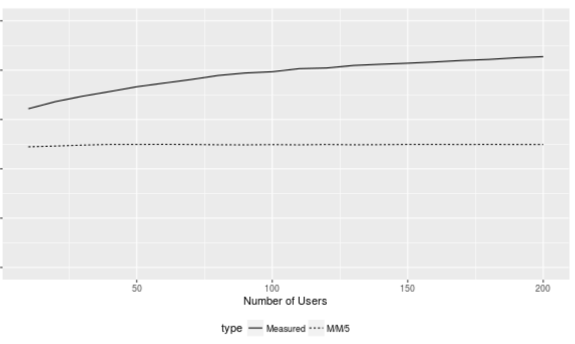


Figure 1: Load vs Response Time (Log base 10) of M/M/5 model (R=1) and measured system.

2.4 Analysis

Our main points of interest are how accurately the model approximates the system under test behavior in different configurations, and how the model's predicted scalability compares to the measured scalability of the system[5] as the parameters of number of servers and replication are varied. In the the following subsections we will refer to the replication factor as the variable R, and to the number of servers as the variable m for brevity. All trends mentioned here are from the observations in tables 2, 3 and 4.

Service Rate Per Server:

1. As m increases, μ increases.

2. As R increases, μ increases.

These two trends occur because we derive μ from E[s], and because E[s] is dependent on the number of servers[6] and the replication factor[7]. This isn't a problem of the serialization method as much as it is a problem of how the M/M/m model misrepresents the asynchronous behavior of the middleware.

Traffic Intensity and Stability:

1. As m increases, ρ decreases. The **linear** trends as m increases would make sense for an ideal implementation that suffers no overhead of adding servers, but only gains performance ($\approx$ 400 higher service per server added). However, from the trends in mean throughput, we see that the real system's gains from an increased number of servers are not so perfectly linear, and quickly decay as more servers are added.

2. As R increases, ρ decreases. This is due to the associated increases in μ explained above.

These two trends indicate that the model is a bad estimator for traffic intensity. In any case, however, the model and the measurements agree that the system is stable, since $\rho < 1$ and the system did not crash during testing.

Jobs in System, Response Time and Speed-up:

1. As m increases, $E[r]_{Model}$, $E[r]_{Measured}$, and $E[n]_{Model}$ decrease, but $E[n]_{Measured}$ increases. The decrease in $E[r]_{Model}$ is relatively more significant than that in $E[r]_{Measured}$ due to the design of the M/M/m model, which has speed-up behavior closer to an ideal system than the implemented system does. $E[n]_{Model}$ decreases while $E[n]_{Measured}$ increases because the model does not account for any overhead associated with maintaining multiple servers and assumes that adding more will only lead to higher values of μ, while this is not the case for the real-life system, which incurs an overhead associated with queue and connection maintenance to multiple servers. We see that the overhead is minuscle as the increase in $E[n]_{Measured}$ is relatively small.

[5]as established in milestone 2, section 2

[6]Because $T_{Processing}$ decreases as m increases as established in Milestone 2

[7]Because of the replication factor denominator in the $E[s]_{Serialized}$ equation

2. As R increases, $E[r]_{Model}$, $E[n]_{Model}$ and $E[n]_{Measured}$ decrease, but $E[r]_{Measured}$ increases. The decrease in $E[r]_{Model}$ and $E[n]_{Model}$ follows from the increase in μ, combined with the decrease in λ. The discrepancy between $E[n]_{Measured}$ and $E[r]_{Measured}$ trends is because of the complexity of the implementation relative to the model; $E[r]_{Measured}$ increases because the work done per SET job increases when more replication is employed. $E[n]_{Measured}$ decreases, however, because the increased work per SET job means an overall decreased throughput, as observed by the trends in λ, and an decrease in service time/increase in service rate.

Probability of Queueing, Jobs in Queues and Waiting Time:

1. As m increases, ϱ_{Model}, $E[n_q]$ and $E[w]$ decrease. $\varrho_{Measured}$ is constant by definition. The decreases happen in the model since it predicts that with more servers, the main queue gets cleared faster, and therefore a job is less likely to have to wait for others to finish. Less jobs will be waiting on average in the queue, and consequentally, by Little's Law, jobs exhibit less waiting time, as the arrival rate is kept constant. The decreases happen in the real system, however, because each server's queue is less congested, as jobs are spread out across more queuest when more servers are added. The significant difference in changes is because the real life queueing implementation does not perfectly scale out as in the model.

2. As R increases, ϱ_{Model}, $E[n_q]$ and $E[w]$ decrease. $\varrho_{Measured}$ is constant by definition. The decreases happen in the model mainly due to the increases in the model service rate that were previously discussed. The decreases in the system, on the other hand, occur due to the decrease in throughput, making queues less congested.

Model Scaling vs System Scaling:

As m increases, we see that the Speed-Up of the model and the server are very different. For the range of server counts we have measured, $SpeedUp_{Model} \approx 2.4$. While for the system $SpeedUp_{Measured} \approx 1.06$. This is because the M/M/m model is ignorant of overhead incurred to add new servers, while also being optimistic about the benefits of adding a new server in comparison to our system, since in our implementation there is a queue for each server. Therefore, an underloaded server cannot fetch jobs to process from another server's queue in the real system, while in the M/M/m, all servers fetch jobs from the same queue. Moreover, a "server" in the model encapsulates many components of the real life system, including middleware worker threads and the memcached servers backing them up. This means that the service time includes the actual service time at a memcached server plus the roundtrip network latency.

In general, we would like to point out that as m increases or as R increases, the predicted behavior by the model deviates more from the measured behavior of the system. Since we have scaled the service time in order to simulate sequential processing, we will also need to scale back the model outputs if we are to use them as viable predictions.

Predicted Response Times:

We chose to display the y-axis values in figure 1 in log base 10 as to highlight the differences in order of magnitude between the predictions and the actual time. This major discrepancy between the model and the measured system is because the model only makes its predictions based on the incoming and outgoing flows (λ and μ) while a multitude of effects, such as network times and thinking times affect the real system.

Limitations:

Generally, the M/M/m model, like the M/M/1 model, is very simple compared to the real middleware. For example, we do not even have a notion of the number of users in the system when constructing the models. Moreover, the network delays are completely unaccounted for by the model and thus the process of job circulation throughout the system becomes very under-represented.

We hope to alleviate some of these limitations and better approximate our system in the next section.

3 System as Network of Queues

In this section we will attempt to build a more elaborate approximate model of the system, and use that model to identify key components that are causing performance bottlenecks.

3.1 Introduction

We will be leveraging a subset of the data from the experiments carried out to investigate the effects of writes in milestone two. We will model the system as the following closed queuing network:

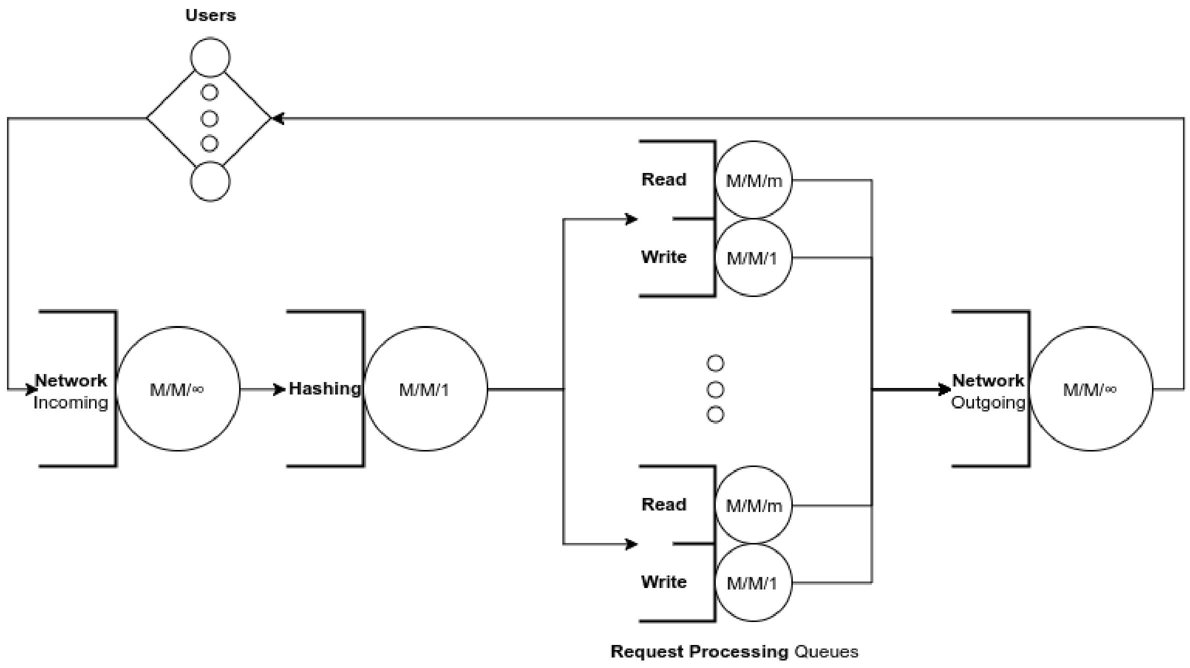


Figure 2: The network of queues model.

Our model is ignorant of any connection establishment or termination processes by clients, and is designed to emulate the system in a steady-state. We will also leverage all of the consequences of having a closed system mentioned in previous sections.

The model is composed of 2 network queues, one hashing queue, pairs of processing queues for each memcached server and a fixed delay center representing the memaslap clients. The network queues are design to model network traffic between the users and the middleware. The hashing queue is made to represent accepting incoming data and mapping it to one of the processing queues. Each pair of processing queues is composed of a read queue for GET requests and a write queue for SET requests. Read queues are M/M/m models, where m is the number of reader threads. Write queues are M/M/1 models. While the M/M/1 model is a gross approximation for the asynchronous behavior, we will only use the model to approximate the behavior of a system with no replication such that the error resulting from this model discrepancy is minimized.

3.2 Methods

In this subsection we will detail how we derive the parameters of each queue in the model, starting with the middleware queues, and then the networking queues, from logging data. Then we will explain which parts of the data we will be analyzing.

Hashing M/M/1 Queue: First we note that the output stream of this queue is split equally across all processing queues, due to the normally distributed hashing implementation. λ will be equal to the mean throughput $\bar{X}$ of the system, as it is closed. μ will be derived from the hashing time measured in the middleware as follows: $\mu = \frac{1}{\bar{T}_{Hashing}}$.

Processing M/M/1 WRITE Queues: λ for each write queue will be equal to the throughput of the system, divided by the number of servers used, and multiplied by the proportion of SET requests: $\lambda = \frac{0.05 * \bar{X}}{5}$. μ will be derived as done in the previous section: $E[s]^{SET}_{Serialized} = \frac{1}{R} E[T^{SET}_{Processing}]$, $\mu = \frac{1}{E[s]^{SET}_{Serialized}}$.

Processing M/M/m READ Queues: λ for each read queue will be equal to the throughput of the system, divided by the number of servers used, and multiplied by the proportion of GET requests: $\lambda = \frac{0.95 * \bar{X}}{5}$. μ will be derived as done in the previous section: $E[s]^{GET}_{Serialized} = \frac{1}{10} E[T^{GET}_{Processing}]$, $\mu = \frac{1}{E[s]^{GET}_{Serialized}}$.

Networking M/M/∞ Incoming/Outgoing Queues: λ for each Networking queue will be equal to the throughput of the system. E[s] will be derived from the estimated network latency $\bar{T}_{Network} = \bar{T}_{Response} - \bar{T}_{Total}$, where $\bar{T}_{Response}$ is the mean response time as measured by memaslap, and $\bar{T}_{Total}$ is the total time spent in the middleware as recorded in the middleware logs. E[s] for each networking component will be equal to $\frac{\bar{T}_{Network}}{2}$, assuming the network is symmetric and the latency should be the same to and from the memaslap clients. Networking queues are for traffic between the middleware and the clients, and do not involve the memcached server in this model.

Users: This component is only a delay center with a fixed thinking time Z, which we estimate to be equal to 0.5ms, a minuscle quantity relative to the rest of the delays in the system, and therefore it will not be a main point of discussion.

Data: We will be looking at the data resulting from experimenting with a configuration of 5 servers, no replication and 10% writes used in the third section of milestone two to derive the service times of our model devices.

Using Mean-Value Analysis, we will extrapolate the mean throughputs and response times of each device in our model, so that we are able to calculate device utilization and perform bottleneck analysis, for N = 40 users.

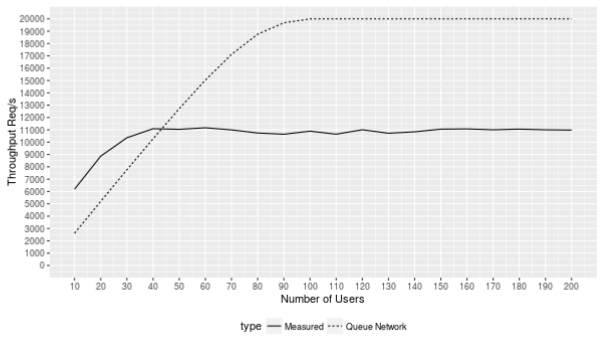
3.3 Results

Device	**Type**	$E[s_i]$	$U_i \mid \rho_i$	$E[r_i]$	$\bar{X}_i$	D_i
User Terminals	Delay	0.50ms	5.14	0.5ms	10271	500us
Network Incoming/Outgoing	Delay	1.50ms	15.41	1.5ms	10271	1500us
Hashing Queue	Load-Independent	0.05ms	0.51	0.10ms	10271	50us
Read Processing Queue	Load-Independent	0.18ms	0.03	0.18ms	1848	3us
Write Processing Queue	Load-Independent	1.06ms	0.22	1.34ms	205	21us

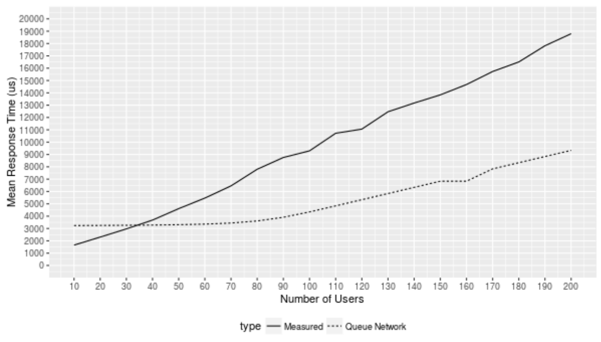
Table 5: Calculated device statistics for the model in figure 2

The data in table 5 is the result of running MVA on the network of queues using the service time estimates in the $E[s_i]$ column. $U_i|\rho_i$ denotes the Utilization or traffic intensity. $E[r_i]$ is the mean response time. $\bar{X}_i$ is the mean throughput. D_i is the device service demand (per server for M/M/m queues).

The measured system throughput for this configuration, in milestone 2 section 3, was 8879 requests per second, 7991 of which were GETs and 888 SETs. By memaslap, the measured total mean response time was 5.4ms, GET mean response time was 5ms and SET mean response time was 8.4ms.



(a) Mean Throughput



(b) Mean Response Time

Figure 3: Predictions of Network of Queues VS Measured Data

3.4 Analysis

Predictions: The models' predictions (table 5) are a bit far from the measured metrics. The total mean throughput was over-estimated by 1392 requests per second, which is an error of $\approx 15\%$. The mean response-time ($\sum D_i - Z$) was under-estimated by 2.12ms ($\approx 39\%$) for GETS and 3.96ms ($\approx 47\%$) for SETS. We see that it is a significant improvement over the predictions of the previous models, which where orders of magnitude away from the measured values for the response time, since the previous models under-represented the system complexity. Forming a network of queues allowed us to more precisely place where each delay in the network was coming from.

Bottlenecks: The devices with the maximum service demands are the network queues. Since we can't do much about improving them, and we are mainly interested in optimizing the middleware, we see that the device with the maximum service demand inside our middleware is the Hashing Queue. This is surprising as we expected the processing queues to be the bottlenecks. However, it makes sense for the hashing queue to be a highly utilized device even though all its operations are local and straightforward since all incoming traffic passes through it, unlike the processing queues which have traffic divided among them with equal probability. If we had only one pair of processing queues in the system instead of five, on the other hand, we would expect the reader queue to be the bottleneck, as it would have $V_i = 0.9$ and consequently a $D_i = 162us$ under the same service time and workload conditions.

Predictions: Relative to those the previous single queue models, the predictions of this network (figure 3) are much more in line with the measured data. Albeit there is still a large margin of error, the predictions are at least on the same order of magnitude as the measurements. Moreover, it should be kept in mind that this network was constructed with a 10% write workload in mind while the measured data is from a read-only workload. We acknowledge the mere fact that the plot of the predictions at least exhibits the same behavior as that of the measurements.

Analysis Limitations: Consequently, even though this model is better than the previous ones, we must acknowledge its design limitations. The processing queues are the main points of concern as their models include the time taken to contact the memcached servers as part of their service time. This simplified device could have been replaced by a set of devices that model the network times and the memcached servers as independent queues, perhaps improving accuracy. We must also acknowledge the limitations of the queueing models themselves, which do not take into account implementation details such as multi-threading contention in the queues, and the inter-dependence of jobs that occurs on the memcached servers. Furthermore, our estimate of the think time Z could be a significant source of error. If Z were in reality higher than our estimate, then the throughput predictions would have a lower error rate.

4 Factorial Experiment

4.1 Introduction

In this section we will design a $2^k r$ factorial experiment to investigate the extent of the effects some of the configuration parameters have on the performance of our system in terms of mean throughput $\bar{X}$.

We initially assume that the errors in our observations/measurements in the experiment were independent and normally distributed with a constant variance. This allows us to use an additive model to describe the effects of the parameters.

From the conclusions drawn in Milestone 2, and the reader thread bottleneck concluded in the previous section, we hypothesize that: **1.** The factor accounting for the main source of variation in throughput will be the number of servers in use. **2.** The factor accounting for the least amount of variation in throughput will be the replication factor. **3.** The highest amount of interaction variation will be between number of servers and replication factor. **4.** The least amount of interaction variation will be between the Number of Servers and the Workload Write-Percentage.

Furthermore we assume that at least the following factor(s) will have a **significant** effect on throughput: **1.** Number of Servers

4.2 Methods

We will be exploring three configuration parameters as our factors:

Factor	**Option 1**	**Option 2**
S: Number of Servers	3	7
R: Replication Factor	None	Full
W: Workload Write-Precentage	1%	10%

No additional experiments are needed to be carried out as all the data[8] needed to explore the above factors has already been generated from the experiments carried out in Milestone 2, Section 3. Each configuration was tested with 5 repititions, therefore we will be fitting a $2^3 * 5$ experiment design to the data.

Prior to the full $2^k r$ analysis, to confirm that an additive model can indeed be applied to our measurements, we will validate the previously stated assumptions through confirming that the measured errors are independently and identically distributed, with a normal distribution, and that their standard deviations are constant.

4.3 Results

i	**I**	**S**	**R**	**W**	**SR**	**SW**	**RW**	**SRW**	$\bar{X}_i$ **Req/s**
1	1	-1	-1	-1	1	1	1	-1	8053
2	1	-1	-1	1	1	-1	-1	1	8465
3	1	-1	1	-1	-1	1	-1	1	8044
4	1	-1	1	1	-1	-1	1	-1	7818
5	1	1	-1	-1	-1	-1	1	1	9096
6	1	1	-1	1	-1	1	-1	-1	9142
7	1	1	1	-1	1	-1	-1	-1	9167
8	1	1	1	1	1	1	1	1	8079
-	67869	3103	-1649	-857	-335	-1227	-1771	-493	Total
-	8483	387	-206	-107	-41	-153	-221	-61	Total/8

Table 6: Estimates of $2^3 5$ Design. Nearest whole numbers presented.

[8]effects/*

i	**Estimated Response**	**Measured Responses**					**Errors**				
	$\hat{y}_i$	y_{i1}	y_{i2}	y_{i3}	y_{i4}	y_{i5}	e_{i1}	e_{i2}	e_{i3}	e_{i4}	e_{i5}
1	8055	8992	8454	7783	7330	7709	937	399	-272	-725	-346
2	8467	7952	8820	8820	8364	8372	-514	353	352	-103	-95
3	8045	7235	8305	8250	7656	8778	-810	260	205	-389	733
4	7817	7369	7783	7926	8267	7746	-448	-34	109	450	-71
5	9095	9499	8317	9241	9369	9057	404	-778	146	274	-38
6	9139	9175	9459	8812	9137	9129	36	320	-327	-2	-10
7	9165	9309	9133	9214	9218	8963	144	-32	49	53	-202
8	8081	7920	8205	8082	8170	8021	-161	124	1	89	-60

Table 7: Measurements and errors of 2^35 Design. Nearest whole numbers presented.

Tables 6 and 7 contain the results of applying a 2^kr analysis to the data. Table 8 contains a set of calculations involving the sums of squares of parts of the data, whereby each sum SS (sum of square) corresponds to its expression, to account for the variation by error, measurements and effects of factors and interactions.

Sum	**Expression**	**Result**	**SST-%**
SS0: Sum of Squares of the Mean	$\sum_{i,j} q_0^2$	**2878805595**	N/A
SSY: Sum of Squares of Measurements	$\sum_{i,j} y_{i,j}^2$	**2895475488**	N/A
SST: Total Variation	$\sum_{i,j}(y_{ij} - \bar{y}_{..})^2$	**16669893**	N/A
SSS: Variation by factor S	$\sum_{i,j}(q_S x_{Si})^2$	**5990760**	36%
SSE: Squared Sum of Error	$\sum_{i,j} e_{ij}^2$	**5363030**	32%
SSR: Variation by factor R	$\sum_{i,j}(q_R x_{Ri})^2$	**1697440**	10%
SSRW: Variation by interaction RW	$\sum_{i,j}(q_{RW} x_{Ri} x_{Wi})^2$	**1953640**	12%
SSSW: Variation by interaction SW	$\sum_{i,j}(q_{SW} x_{Si} x_{Wi})^2$	**936360**	6%
SSW: Variation by factor W	$\sum_{i,j}(q_W x_{Wi})^2$	**457960**	3%
SSSR: Variation by interaction SR	$\sum_{i,j}(q_{SR} x_{Si} x_{Ri})^2$	**67240**	$< 1\%$
SSSRW: Variation by interaction SRW	$\sum_{i,j}(q_{SRW} x_{Si} x_{Ri} x_{Wi})^2$	**148840**	$< 1\%$

Table 8: Calculated Variations.

$$s_e^2 = \frac{SSE}{2^k(r-1)} = \frac{5362894}{8 * 4} = 167590.4375 \tag{1}$$

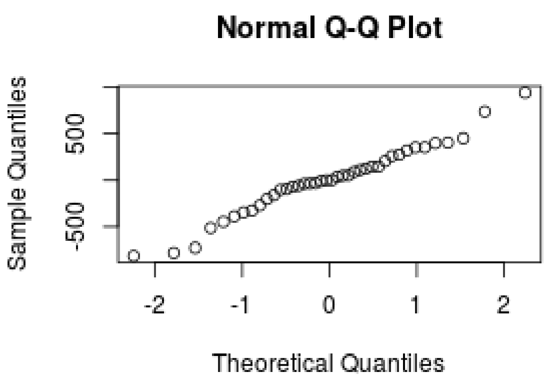
$$s_{q_i} = \frac{s_e}{\sqrt{2^k r}} = \frac{167590.4375}{\sqrt{40}} \approx 26498 \tag{2}$$

$$t_{[1-\alpha/2;2^k(r-1)}s_{q_i} = t_{[1-0.05/2;32]}s_{q_i} = 2.03693334 * 26498 \approx 53974 \tag{3}$$

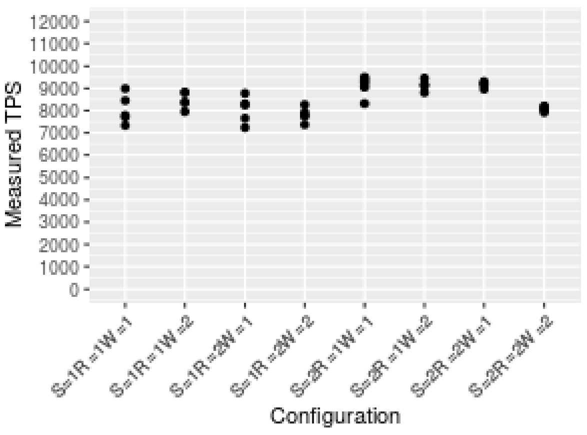
Equation 1 calculates the variance of errors, which is used in equation 2 to estimate the variance of the effects. This is used in equation 3 to calculate the 95% confidence interval size with 32 degrees of freedom for all effects. The results of applying the interval size calculation to each effect is in table 9.

I	**S**	**R**	**W**
(-45491, 62457)	(-53587, 54361)	(-54180, 53768)	(-54081, 53867)
SR	**SW**	**RW**	**SRW**
(-54015, 53933)	(-54127,53821)	(-54195, 53753)	(-54035, 53913)

Table 9: Confidence intervals of effects and interactions.



(a) Q-Q Plot of $2^k r$ model residuals.



(b) Q-Q Plot of $2^k r$ model residuals.

4.4 Analysis

First, we validate the underlying assumptions of our $2^k r$ model. We conclude that the errors are independent since the residuals and the values in table 7 are an order of magnitude apart[1]. From figure 4a we see that the errors are normally distributed. From figure 4b we can see that there are no significant changes in the spread as the measurements are within the same orders of magnitude. Therefore we choose to continue our analysis.

From table 8 we see that the factor with highest variation is the number of servers as predicted. However, the factor accounting for the least variation is the write percentage rather than our prediction of replication factor. The interaction with most replication is that between replication and write load contrary to our prediction of server count and replication. Furthermore, our prediction that the interaction between sever count and write percentage will have the least variation amongst interactions was also wrong, as it accounts for 6% of the variation, more than that between server count and replication.

Unfortunately, from table 9 we see that none of the effects are significant, since all of their confidence intervals contain zeroes. This is due to the high values of s_e^2. Had we increased the number of repititions in our previous experiments, perhaps we would have attained significant results.

5 Interactive Law Verification

In this section, we validate, using the interactive response time law, the results of the experiments carried out to investigate the effects of replication (Section 2) in the previous milestone.

5.1 Introduction

Our sytem is a closed system, where a single user does not send more than one request at once, but sends requests sequentially, and waits for each request to be serviced before sending the next one. This also makes our system interactive, whereby the interactive response time law[1] states that:

$$R = \frac{N}{X} - Z \tag{4}$$

Where R is the system response time, N is the number of users, X is the system throughput, and Z is the think time, which is an amount of time a user spends in between receiving a response from the server and sending their next request. We will be verifying the validity of the data collected in a previous experiment, under the assumption that our system is indeed interactive, whereby equation 4 should hold.

Our hypothesis is that for **a fixed *number of clients N***, and **varying *replication factors* and *number of memcached servers***, the difference between the **measured *mean response time R*** and the **total *cycle time of requests*** $C = \frac{N}{X}$, where **X** is the measured system throughput, is equal to a small constant think time **Z**. This ensures that equation 4 holds, such that $R = C - Z$.

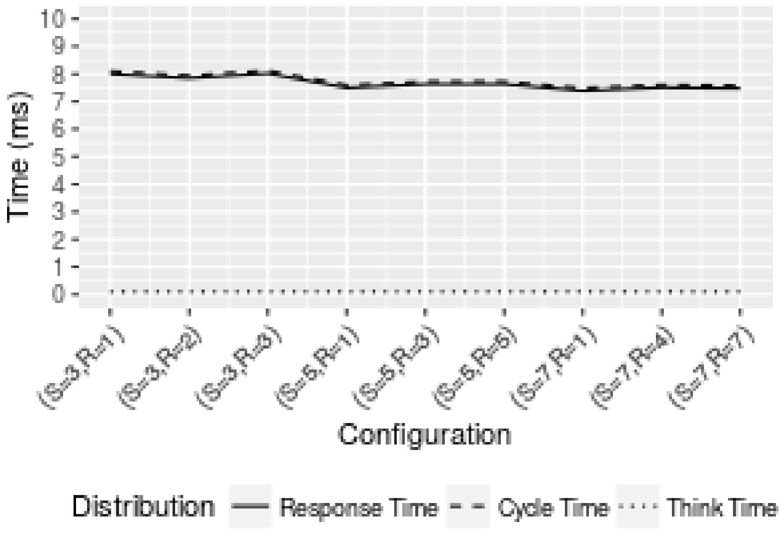


Figure 5: Hypothesized Response Time Values and Cycle Time Values. X-axis denotes configurations where S is the number of servers and R is the replication factor, *not to be confused with Response time.* The think time, Z, is estimated to be $\approx 0.1ms$.

.

5.2 Methods

We reused the memaslap log files from the 2nd experiment in Milestone 2. We provide the table below as a quick experiment reference. For more details about the experiment, please refer to the milestone 2 report.

Number of servers	3,5,7
Replication	R=1,R=$\lceil\frac{S}{2}\rceil$,R=all
Total Client Count	40
Reader Threads	10
Workload	Key 16B, Value 128B, 5%-writes
Runtime x repetitions m	1m x 5
Log files	effects/*

We will substitute N = 40 in equation 4 to derive the cycle time for each of the 9 configurations. We will carry out averaging of X and R over the 5 repititions of each datapoint in order to approximate their actual values.

5.3 Results

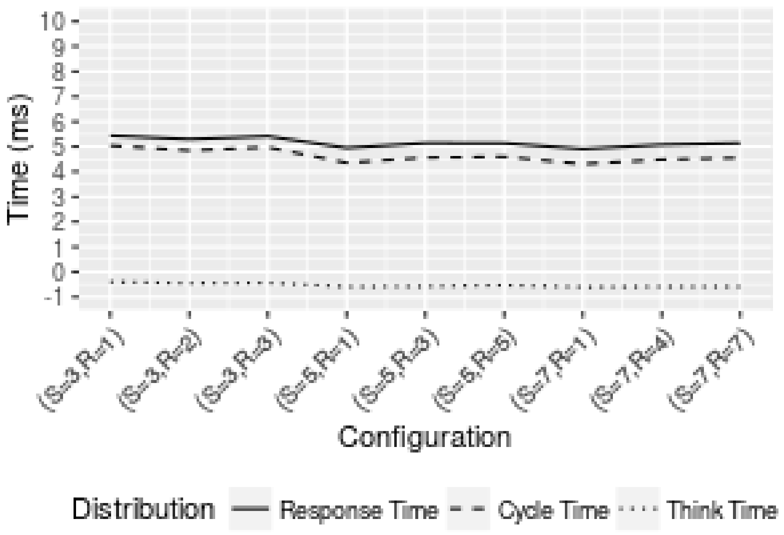


Figure 6: Resulting Response Time and Cycle Time Values. S is the number of servers and R is the replication factor. The think time, $Z \approx 0.5ms$.

.

5.4 Analysis

Figure 6 shows that the difference between Response time and Cycle time remains relatively constant as the configuration changes. However, unexpectedly, this difference is negative rather than positive, since the measured mean response time seems to be higher than that estimated by the throughput equation. Nonetheless, this confirms our hypothesis and re-affirms that the generated data is valid and adheres to the interactive response time law as stated in equation 4, since the order of magnitude of the think time is negligible compared to the measurements.

The relative independence of the think time from the middleware configuration is because that value is particular to the memaslap client. We expect that the think time Z would change as the number of clients changes or as the client machine setup changes, but such analysis is out of our scope, as we have no statistics about memaslap itself and assume our system to be closed, which implies that each virtual client is independent from the other.

In conclusion, we estimate the mean think time to be $\bar{Z} \approx -532us$ with a standard deviation equal to $\sigma = 79$. The negative estimation is due to the nature of the distribution of response times, which has a long tail. This leads to the mean-statistic becoming an over-estimator of where the 'middle' of the distribution lies.1

References

[1] Bukh, Per Nikolaj D., and Raj Jain. ”The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling.” (1992).

Logfile listing

Short name	**Location**
effects/*	`https://gitlab.inf.ethz.ch/rkhalil/asl-fall16-project/tree/master/milestone2/`
trace.logs/*	`https://gitlab.inf.ethz.ch/rkhalil/asl-fall16-project/tree/master/report/trac`

A M/M/m Model Equations

Metric/Parameter	**Symbol**	**Model Equation**
Throughput/Arrival Rate	λ	$\lambda = X$
Service Rate **per server**	μ	$\mu = \frac{1}{E[s]}$
Traffic Intensity	ρ	$\frac{\lambda}{m\mu}$
Probability of zero jobs in the system	p_0	$\left[1 + \frac{(m\rho)^m}{m!(1-\rho)} + \sum_{n=1}^{m-1} \frac{(m\rho)^n}{n!}\right]^{-1}$
Probability of queueing	ϱ	$\frac{(m\rho)^m}{m!(1-\rho)} p_0$
Mean # of jobs in the system	$E[n]$	$\rho(m + \frac{\varrho}{1-\rho})$
Mean # of jobs in the queue	$E[n_q]$	$\rho\varrho/(1-\rho)$
Mean response time	$E[r]$	$\frac{1}{\mu}\left(1 + \frac{\varrho}{m(1-\rho)}\right)$
Var. of the response time	$V[r]$	$\frac{1}{\mu^2}\left(1 + \frac{\varrho(2-\varrho)}{m^2(1-\rho)^2}\right)$
Mean waiting time	$E[w]$	$E[n_q]/\lambda$
Var. of waiting time	$V[w]$	$E[w]^2 \frac{(2-\varrho)}{\varrho}$

Table 10: M/M/m Queue Model Metric Equations