

Advanced Systems Lab (Fall'16) – Third Milestone

Name: *Taivo Pungas*
Legi number: *15-928-336*

Grading

Section	Points
1	
2	
3	
4	
5	
Total	

Contents

1	System as One Unit	3
1.1	Model	3
1.2	Problems of the model	3
1.3	Data	3
1.4	Parameter estimation	3
1.5	Comparison of model and experiments	3
2	Analysis of System Based on Scalability Data	5
2.1	Model	5
2.2	Problems of the model	5
2.3	Parameter estimation	5
2.4	Data	5
2.5	Comparison of model and experiments	6
3	System as Network of Queues	7
3.1	Model	7
3.2	Problems of the model	8
3.3	Parameter estimation	8
3.4	Data	8
3.5	Comparison of model and experiments	8
3.5.1	Mean value analysis	8
3.5.2	Bottleneck analysis	9
4	Factorial Experiment	10
4.1	Experimental question and experiment design	10
4.2	Data	10
4.3	Results	10
4.3.1	Checking assumptions	10
4.3.2	Analysis of the model fit	12
4.3.3	Allocation of variation	12
5	Interactive Law Verification	14
5.1	Model	14
5.2	Data	14
5.3	Results	14
	Appendix A: Data for the factorial experiment	16

1 System as One Unit

In this section I will build a very simple black-box model of the system.

1.1 Model

The system under test (SUT) in this section includes the middleware, memcached servers and the network between them. It does *not* include clients or the network between clients and middleware.

In this section I create an M/M/1 model of the SUT. This means the following definitions and assumptions:

- The queues have infinite buffer capacity.
- The population size is infinite.
- The service discipline is FCFS.
- Interarrival times and the service times are exponentially distributed.
- We treat the SUT as a single server and as a black box.

1.2 Problems of the model

The assumptions above obviously do not hold for our actual system. Especially strong is the assumption of a single server; since we actually have multiple servers, this model is likely to predict the behaviour of the system very poorly. A second problem arises from my very indirect method of estimating parameters for the model (and an arbitrary choice of time window) which introduces inaccuracies.

1.3 Data

The experimental data used in this section comes from the updated trace experiment, found in [results/trace_rep3](#) (short names `trace_ms*`, `trace_mw` and `trace_req` in Milestone 1). For details, see Milestone 2, Appendix A. As a reminder, that experiment had $S = 3$, $R = 3$, $W = 1\%$, $T = 5$ and $C = 192$.

The first 2 minutes and last 2 minutes were dropped as warm-up and cool-down time similarly to previous milestones.

1.4 Parameter estimation

Using the available experimental data, it is not possible to directly calculate the mean arrival rate λ and mean service rate μ so we need to estimate them somehow. I estimated both using throughput of the system: I take $\lambda = 10294 \frac{\text{requests}}{s}$ to be the *mean* throughput over 1-second windows, and $\mu = 12900 \frac{\text{requests}}{s}$ to be the *maximum* throughput in any 1-second window, calculated from middleware logs. I chose a 1-second window because a too small window is highly susceptible to noise whereas a too large window size drowns out useful information.

1.5 Comparison of model and experiments

Table 1 shows a comparison of the predictions of the M/M/1 model with actual results from the trace experiment. The system is stable since $\rho < 1$.

It is clear that actual response time is much higher – about 40 times higher – than what the model predicts. This is because using M/M/1 for the SUT as a black box does not take into account internal parallelisation: we have $S \cdot T$ threads for GETs and S threads for SETs.

	metric	predicted	actual
1	response_time_mean	0.38	14.55
2	response_time_std	0.38	17.62
3	response_time_quantile50	0.27	12.00
4	response_time_quantile95	1.15	30.00
5	waiting_time_mean	0.31	13.22
6	waiting_time_std	0.38	17.06
7	utilisation	0.80	0.90
8	num_jobs_in_system_mean	3.95	149.79
9	num_jobs_in_system_std	4.42	
10	num_jobs_in_queue_mean	3.15	136.11
11	num_jobs_in_queue_std	4.26	
12	proportion_jobs_in_queue	0.80	0.91
13	num_jobs_served_in_busy_period_mean	4.95	
14	num_jobs_served_in_busy_period_std	13.19	
15	busy_period_duration_mean	0.38	
16	busy_period_duration_std	0.76	

Table 1: Comparison of experimental results ('actual') and predictions of the M/M/1 model ('predicted') for different metrics. Where the 'actual' column is empty, experimental data was not detailed enough to calculate the desired metric. All time units are milliseconds.

For a single thread to achieve the same throughput as K parallel threads, it needs to process each request K times faster. This doesn't explain the whole difference, though, given that in the trace experiment $K = 15$ (compared to the 40x difference in predicted and actual values we observe).

Another factor is queueing: since the M/M/1 model predicts a much lower response/service time, the queues in that model are also shorter. The number of jobs in each GET queue is $\frac{149.79}{5} = 9.07$ which is roughly 2 times more than what M/M/1 predicts. A longer queue means a longer waiting time.

The model does provide a reasonable estimate of the utilisation of the system and the proportion of jobs in queue (out of all jobs in the system). This is mostly because the inputs we gave to the model (arrival rate and service rate) were good estimates and not because the model itself is accurate.

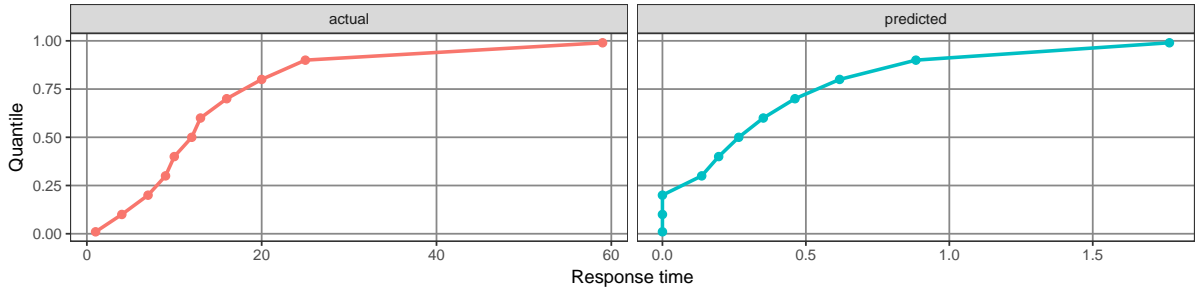


Figure 1: Quantiles of the response time distribution: experimental results and predictions of the M/M/1 model. Note the extreme difference in the response time scale.

The quantiles of the response time distribution are shown in Figure 1. M/M/1 appears to do a good job at predicting the shape of the distribution; however, the time scale is off by roughly 20x. As before, this is because M/M/1 does not account for parallelisation.

In summary, the model a) does not adequately take into account the internal structure of SUT, and b) does not predict empirical results with reasonable accuracy. For this reason, it is near worthless and we need to build more complex models in Sections 2 and 3.

2 Analysis of System Based on Scalability Data

TODO: Purpose of this section (1-2 sentences)

2.1 Model

The system under test (SUT) in this section includes the middleware, memcached servers and the network between them. It does *not* include clients or the network between clients and middleware.

The assumptions and definitions of the M/M/m model are the same as for the M/M/1 model laid out in Section 1.1 with the following modifications:

- We treat the SUT as a collection of m servers.
- All jobs waiting for service are held in one queue.
- If any server is idle, an arriving job is serviced immediately.
- If all servers are busy, an arriving job is added to the queue.

The model built here does not account for the effect of replication. For this reason I will only use data from experiments for which $R = 1$.

2.2 Problems of the model

TODO:

1. I actually have m queues (one for each server), not a single queue; each request is assigned to a server when [LoadBalancer](#) receives it.
2. I map requests to servers uniformly. M/M/m assumes that each server takes a request when it finishes with the previous one, but that is not true in my case – I take earlier

2.3 Parameter estimation

We need to determine three parameters: the number of servers m , the arrival rate λ and the service rate of each server μ .

Let us first find m . Since SUT has S [MiddlewareComponents](#), each of which has T read threads and 1 write thread, all of which ideally run in parallel (i.e. none starve for resources). Thus I take $m := S \cdot (T + 1)$.

To estimate μ we can calculate the service time of each worker as the time spent between dequeuing the request and sending it back to the client: $t_{service} := t_{returned} - t_{dequeued}$. From there we find $\mu := \frac{1}{t_{service}}$. Note that this calculation does not distinguish between write and read threads.

We can find λ as simply the mean throughput over 1-second windows, similarly to Section 1.1.

2.4 Data

The experimental data used in this section comes from Milestone 2 Section 2 and can be found in [results/replication](#). For this section, only data from one repetition (rep. no. 5) and $R = 1$ were used (short names `replication-S*-R1-r5`). As a reminder, that experiment had

TODO: $S = 3$, $R = 3$, $W = 1\%$, $T = 5$ and $C = 192$.

The first 2 minutes and last 2 minutes were dropped as warm-up and cool-down time similarly to previous milestones.

2.5 Comparison of model and experiments

Table 2 shows the results of modelling the system as M/M/m. Since $\rho < 1$ the model is stable for all cases. However, the table reveals an important shortcoming of the model: waiting times and the time spent in the queue are 0.

The reason becomes clear if we look at the formulas for calculating p_0 (the probability of 0 jobs in the system). As m goes to infinity, p_0 goes to zero (because the inverse of p_0 is a sum that goes to infinity). Even for finite values of $m \in [99, 231]$, p_0 is on the order of 10^{-32} to 10^{-77} (for $\rho=0.8$). Since the probability of queueing ϱ also depends on p_0 , this causes queueing to become nonexistent in the M/M/m model.

There are some aspects in which the model performs well, though. Figure 2 shows that the model only slightly underestimates response time, although the variance estimate is too low.

TODO:

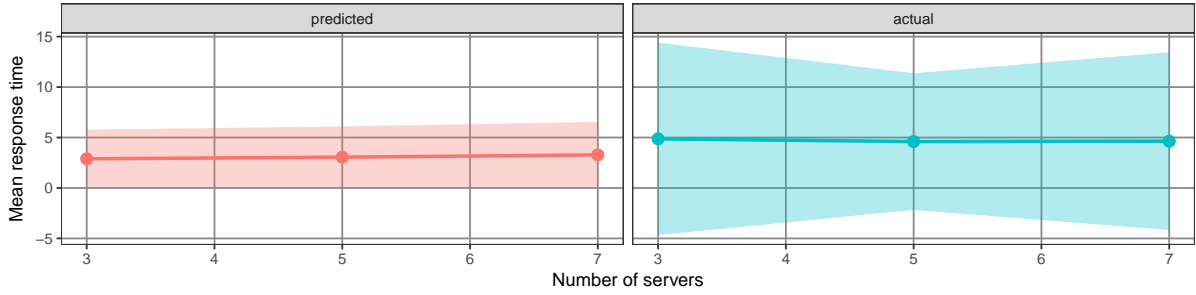


Figure 2: TODO: note difference in scale

	variable	predicted_3	actual_3	predicted_5	actual_5	predicted_7	actual_7
1	m	99.00	99.00	165.00	165.00	231.00	231.00
2	lambda	17242.28	17242.28	17676.91	17676.91	16885.81	16885.81
3	mu	346.12	346.12	327.62	327.62	305.28	305.28
4	response_time_mean	2.89	4.87	3.05	4.60	3.28	4.64
5	response_time_std	2.89	9.54	3.05	6.78	3.28	8.81
6	waiting_time_mean	0.00	1.98	0.00	1.54	0.00	1.36
7	waiting_time_std	0.00	5.80	0.00	4.51	0.00	4.55
8	num_jobs_in_system_mean	49.82	83.94	53.96	81.26	55.31	78.32
9	num_jobs_in_system_std	7.06		7.35		7.44	
10	num_jobs_in_queue_mean	0.00	34.12	0.00	27.31	0.00	23.01
11	num_jobs_in_queue_std	0.00		0.00		0.00	

Table 2: Comparison of experimental results and predictions of the M/M/m model, for $S \in \{3, 5, 7\}$. Where the 'actual' column is empty, experimental data was not detailed enough to calculate the desired metric. Variables 'm', 'lambda' and 'mu' were inputs to the model. All time units are milliseconds.

3 System as Network of Queues

TODO: Purpose of this section (1-2 sentences)

3.1 Model

TODO:

The system under test (SUT) in this section includes the clients, middleware, memcached servers and the network between them.

Thus we can model SUT as a *closed* queueing network: the total number of jobs in the system is constant and equal to the concurrency parameter C we give to memaslap – each concurrent job in memaslap sends one request and waits for a response before sending the next one.

I will use the following definitions and assumptions:

- The queues are defined as having infinite buffer capacity.
- The population size is infinite.
- The service discipline is FCFS.
- Interarrival times and the service times are exponentially distributed.
- The clients have $Z = 0$ think time.

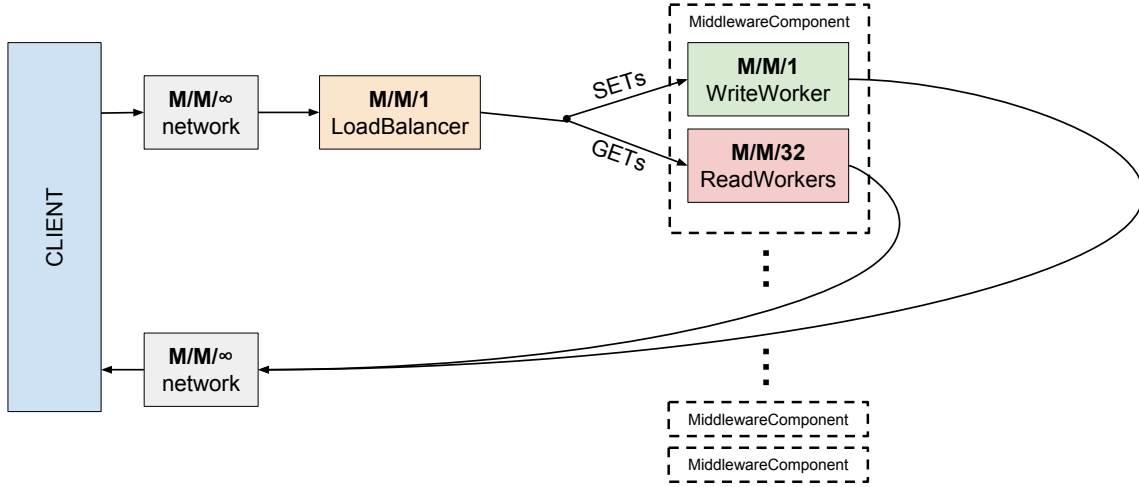


Figure 3: The queueing network model. Only one **MiddlewareComponent** out of S is shown. The network delay centers have identical service and are separated in the figure only for clarity.

Figure 3 shows the queueing network model used in this section. The client–middleware network time is modelled as a delay center. **LoadBalancer** – which consists of a single thread dequeuing from one queue – is modelled as an M/M/1 node.

There are two queues in each **MiddlewareComponent** and they are modelled as such: T **ReadWorkers** are dequeuing from a single queue of GETs and one **WriteWorker** is dequeuing from a queue of SETs. This gives rise to M/M/ T and M/M/1 architectures, respectively.

There is no explicit modelling of memcached servers: instead, the network round-trip to memcached, the waiting time in memcached and the service time of memcached servers is all counted into the service time of Workers.

GETs and SETs are modelled separately, i.e. this is a *multiclass* queueing network.

3.2 Problems of the model

One fundamental problem of the model is the simplification of not modelling memcached servers. This simplification makes sense for [ReadWorkers](#) because they block until a **GET** has been returned to the client, but since [WriteWorkers](#) are asynchronous, this is not an accurate model.

However, after dequeuing a **SET** and sending it to memcached, a [WriteWorker](#) waits for 1 millisecond (using `Selector.select(long timeout)`) before proceeding to dequeuing the next element. As a consequence, if the response from memcached arrives within 1ms and no other responses are ready to be `selected`, the behaviour is equivalent to blocking until a response is received. For a similar reason, if we increase S or R , [WriteWorkers](#) will get faster because there is less blocking on the `select()` method. (This also explains the behaviour seen in Milestone 2 Section 2).

For this reason, [WriteWorkers](#) exhibit a behaviour that is on average 'between' asynchronous and synchronous – and using a single M/M/1 model for the [WriteWorker](#) – memcached – [WriteWorker](#) part of SUT is a better approximation.

A consequence of the problem described above is that replication has no effect on the predictions – if we want to find parameters from one configuration and use them to predict the performance for varying R , we will get exactly the same results.

3.3 Parameter estimation

The inputs to the queueing network model are: S , T , C , W , and service times in each node of the network. The service times for each component are defined as follows (refer to Milestone 1 Figure 1 for timestamp definitions):

- $s_{network} = 0.5 \cdot (T_{ms} - T_{mw})$, where T_{ms} is the mean response time measured by memaslap, and $T_{mw} = \text{mean}(t_{returned} - t_{created})$ (mean response time measured by middleware)
- $s_{LoadBalancer} = \text{mean}(t_{enqueued} - t_{created})$
- $s_{WriteWorker} = \text{mean}(t_{returned} - t_{dequeued})$
- $s_{ReadWorker} = \text{mean}(t_{returned} - t_{dequeued})$

$s_{network}$ and $s_{LoadBalancer}$ are calculated separately for **GETs** and **SETs** (the names of Workers already imply that the service times are only calculated for one request type).

TODO: problem is that I include loadbalancer queueing time in network roundtrip, but this is not a problem – I just add a (tiny) constant amount of time to the network time because of this

MVA was performed using the Octave package [queueing](#).

3.4 Data

The experimental data used in this section comes from Milestone 2 Section 2 and can be found in [results/replication](#). For this section, only data from one repetition (rep. no. 5) and one configuration ($S = 5$, $R = 1$) were used (short name `replication-S5-R1-r5`). As a reminder, that experiment had $W = 5\%$, $T = 32$ and $C = 180$.

3.5 Comparison of model and experiments

3.5.1 Mean value analysis

TODO:

performance of writeworker *depends on queue length* because if there are no elements in queue then we check memcached responses every 1ms, whereas if there are elements in queue

	variable	predicted	actual
1	throughput	20444.53	17668.03
2	mean_response_time_get	8.71	9.68
3	mean_response_time_set	11.20	21.21
4	mean_response_time	8.83	10.29
5	throughput_readworkers	19640.99	16784.63
6	throughput_writeworkers	803.54	883.40
7	mean_response_time_readworker	3.05	4.02
8	mean_response_time_writeworker	5.48	15.49
9	util_lb_get	0.08	0.06
10	util_lb_set	0.00	0.00
11	util_readworker	0.37	0.32
12	util_writeworker	0.50	0.55
13	items_network_get	111.04	94.89
14	items_network_set	4.59	5.05
15	items_readworkers	59.88	67.54
16	items_writeworkers	4.40	13.69

Table 3: **TODO:** loadbalancer items and response times have been left out because they were extremely low

then we check always after dequeuing an element. This dependence violates **TODO:** what assumption?

if there is nothing in the queue and no responses from memcached then we will wait for 2ms!

3.5.2 Bottleneck analysis

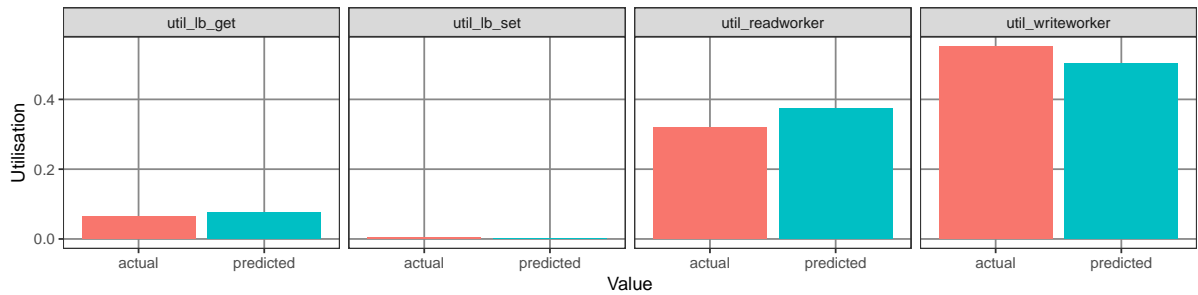


Figure 4: **TODO:**

TODO: Book 33-03 (33.6)
calculate bounds of throughput and response time

4 Factorial Experiment

In this section, I will use a $2^k r$ experiment to understand the system. Of special interest here is the proportion of SETs because the performance of SETs in SUT strongly depends on parameters, as explained in Section 3.2.

4.1 Experimental question and experiment design

The goal of this section is to find out the factors that influence throughput of the system. In particular, I will investigate the effect of $k = 3$ factors – S (number of servers), R (replication level) and W (percentage of SETs in workload) – on total throughput of the system. Everything else will be kept fixed: the number of threads $T = 32$ and the number of clients $C = 180$.

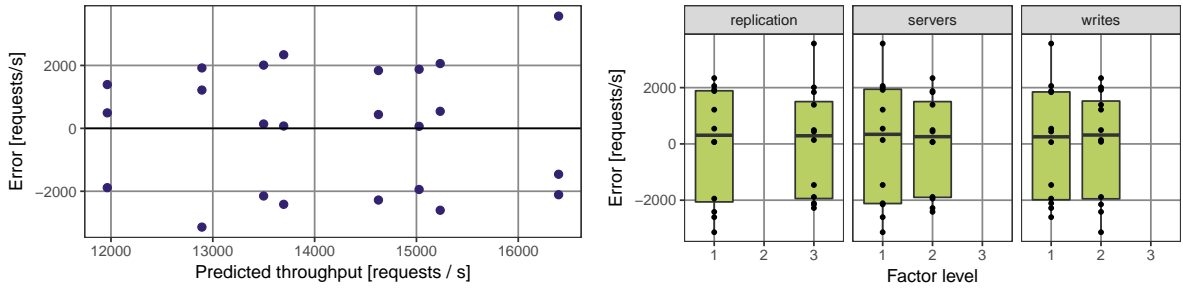
For each factor in $\{S, R, W\}$ we need to pick two levels. Since I expect throughput to monotonically increase with S , decrease with R , and decrease with W , we can use the minimum and maximum level for each factor in our $2^k r$ experiment: $S \in \{3, 7\}$, $R \in \{1, 5\}$, and $W \in \{1, 10\}$.

4.2 Data

The experimental data used in this section comes from Milestone 2 Section 3 and can be found in [results/writes](#) (short name `writes-S*-R*-W*-r*` in Milestone 2). For each combination of S , R , and W we have $r = 3$ repetitions. This means data from $2^k \cdot r = 24$ distinct runs are used in this section.

The data table is available in Appendix A.

4.3 Results



(a) Error of the $2^k r$ model as a function of predicted throughput. Each point is one repetition of all factors. The box shows the inter-quartile range (between 25th and 75th percentiles, IQR), with the median shown as a horizontal line. The top and bottom whisker show the highest and lowest value within $1.5IQR$ of the median. Experiment results are also plotted as single points. Note that each subplot contains all 24 data points.

Figure 5: Evaluation of the $2^k r$ model.

4.3.1 Checking assumptions

Before delving into the predictions of model we need to check whether assumptions we made in modelling actually hold.

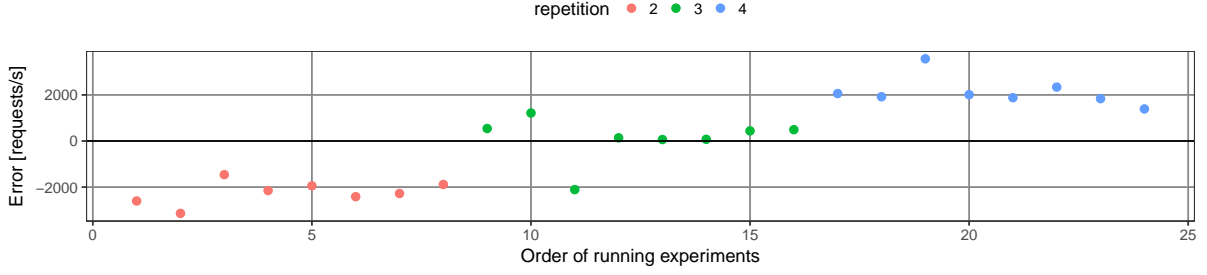


Figure 6: Error of the $2^k r$ model as a function of the order in which experiments were run. Color of the points shows the repetition number.

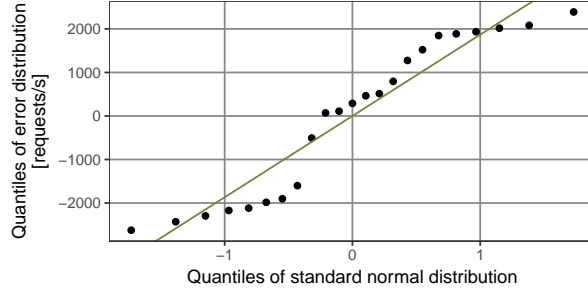


Figure 7: Quantiles of the residual distribution plotted against quantiles of the standard normal distribution. The straight line shows the best fit of a linear trendline through the points.

Independent errors The model assumes that errors are independently and identically distributed (IID). Figure 5a shows that error does not depend on the predicted throughput. Furthermore, errors do not depend on factor levels as shown in Figure 5b: the median and 25% and 75% percentiles of the error distribution are independent of the factor and the level.

Figure 6 shows that errors clearly depend on repetition. An obvious hypothesis here is that each successive repetition improved the throughput for some reason. However, repetition 2 was run on Nov 20, and both repetitions 3 and 4 were run on Nov 23 with a 6-hour difference; the resource group was hibernated and redeployed before each experiment. For this reason we can't accept the hypothesis that repetitions somehow affected each other.

Thus, the large variation in throughput between experiments can be explained in two ways: it could a) be caused by differing conditions in Azure between deployments, or b) be somehow inherent to the SUT. If b) were true, we would also see large variance *within* each deployment, which is not the case. This leaves us with option a): the conditions on Azure differed between employments – possibly due to different server or network allocation, or the total load in Azure, or some other factor.

Normally distributed errors The model assumes that errors are normally distributed. The quantile-quantile plot in Figure 7 shows that this is clearly not the case – the lowest quantiles are too low and medium-to-high quantiles too high for the distribution to be normal. This, however, is caused by the trimodality of the error distribution: the throughputs (and thus, errors) of each repetition individually are distributed much more closely to a normal distributions, but when we concatenate the repetitions, the result is trimodal.

Constant standard deviation As Figure 5b shows, the distribution looks similar at all factor levels, so the model assumption of a constant standard deviation holds.

4.3.2 Analysis of the model fit

Even though the deviation from the mean is high in our experiments, residuals are almost an order of magnitude smaller than throughput values, and there does not appear to be any definite trend in the mean or spread of the residuals, so the fit of the model is reasonable given the noisiness of the data. Given that the range of throughput values covered is small and the additive model is a reasonable fit, we don't need to use a multiplicative model.

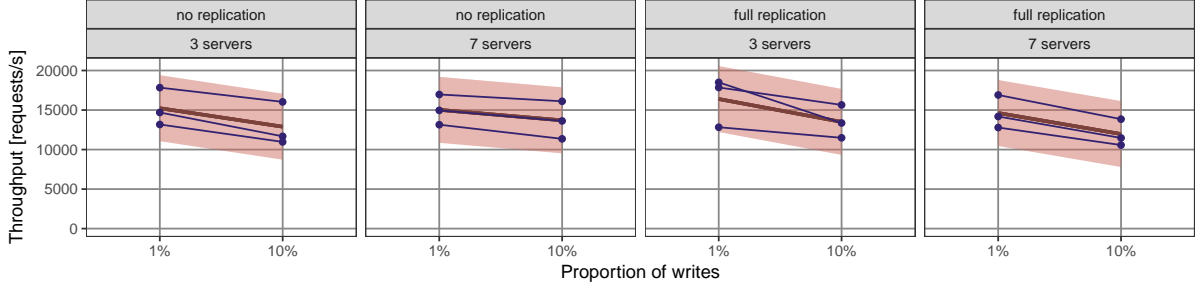


Figure 8: Throughput as a function of W . Each line with points shows the results of one repetition. The red line shows the throughput predicted by the 2^k model; the light red ribbon shows the standard deviation that we would expect to see according to the model, if we were to run 3 repetitions.

Figure 8 shows the predictions of the model and actual results. The results of all repetitions have the same trend as the predicted mean, and stay within 1 standard deviation of the mean. This again confirms that the fit is decent.

4.3.3 Allocation of variation

	variable	coefficient_value	variation
1	x0_constant	14166.7	
2	xa_servers	-338.6	0.024
3	xb_replication	-45.4	0.000
4	xc_writes	-1153.8	0.276
5	x_ab	-487.6	0.049
6	x_bc	-236.3	0.012
7	x_ac	155.5	0.005
8	x_abc	-97.1	0.002
9	error		0.672

Table 4: Values of coefficients and allocation of variation for all variables in the 2^k model.

In Table 4 we find the coefficients for each variable in our model, together with the allocated variation. By far the most variation is explained by the error. This is caused by the large variation in throughput in different deployments, as discussed above.

Looking at the variation explained by other components, we find that the percentage of SETs (x_c) explains the most at 27.6%. The combination of servers and replication (x_{ab}) explains about 5%; all other variables explain less than 1/10th of the variation of x_c and can thus be considered insignificant.

These results can be directly mapped to the system design. There are T times fewer [WriteWorkers](#) than [ReadWorkers](#). This causes SETs to have a significantly higher response time (see Milestone 2 for details); increasing the proportion of SETs therefore increases response time significantly.

The amount of servers S or replication R don't have a large effect individually; however, increasing both S and R from the minimum to the maximum increases the cost of SETs significantly ($S = 7, R = S$ means 7 writes are made to memcached servers for each SET, compared

to 1 write if $S = 3, R = 1$), and thus has a strong effect on throughput (via increased response time).

5 Interactive Law Verification

In this section, I will check the validity of experiments from Section 2 of Milestone 2 using the Interactive Response Time Law (IRTL).

5.1 Model

We are assuming a closed system, i.e. clients wait for a response from the server before sending another request. Under this assumption, IRTL should hold:

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

where R is mean response time, Z is waiting time in the client, N is the number of clients and X is throughput. In this section we test whether IRTL does in fact hold.

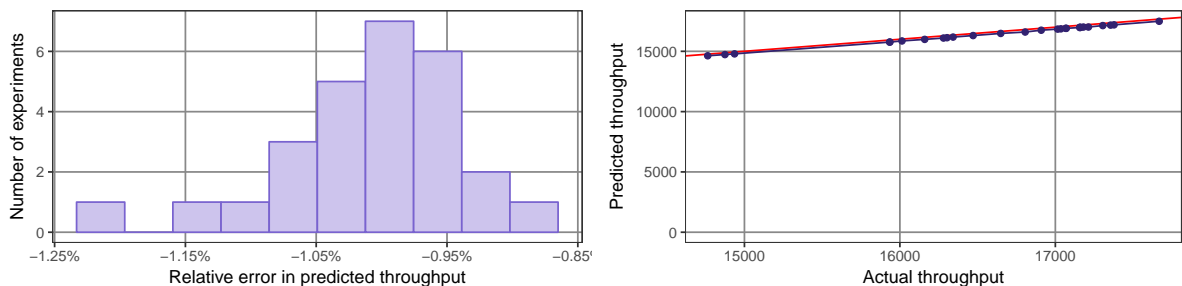
5.2 Data

The experimental data used in this section comes from Milestone 2, Section 2 (Effect of Replication) and can be found in [results/replication](#) (short name `replication-S*-R*-r*` in Milestone 2). This includes a total of 27 experiments in 9 different configurations.

The first 2 minutes and last 2 minutes were **not** dropped because IRTL should hold also in warm-up and cool-down periods. Repetitions at the same configuration were considered as separate experiments.

5.3 Results

Using IRTL, we can verify the validity of experiments by calculating the predicted throughput $X_{predicted}$ (given the number of clients C and mean response time R) and comparing it with actual throughput X_{actual} . This is precisely what I did for all experiments of Milestone 2, Section 2. $C = 180$ in all experiments, and both R and X_{actual} are aggregated results reported by the three memaslap instances generating load in that experiment.



(a) Histogram of the relative error of throughput predicted using IRTL, counting the number of experiments in a given error range. Note the horizontal scale does not include 0.

(b) Throughput predicted using IRTL (dark points), as a function of actual throughput calculated from experimental data. The red line shows hypothetical perfect predictions (the $x = y$ line). Note the horizontal scale does not include 0.

Figure 9: Evaluation of the validity of Milestone 2 Section 2 experiments

If we assume the wait time Z to be 0, we get a mean relative prediction error of -1.01% , defined as $\frac{X_{predicted} - X_{actual}}{X_{actual}}$. The distribution of these errors is shown in Figure 9a; the distribution looks reasonably symmetric. Figure 9b plots $X_{predicted}$ against X_{actual} and shows again that the predicted throughput is very close to actual throughput, but consistently smaller in all regions of the graph.

If we assume a nonzero Z and estimate it from the experiments, we get a mean estimated wait time of -0.111ms. Clearly this is impossible: wait time must be non-negative.

To explain these results, we need to answer the question: why is the actual throughput lower than the actual throughput? The most plausible hypothesis is that memaslap starts the clock for a new request before stopping the clock for the previous request – which would violate the closed system assumption. This could be caused by us using more virtual clients than CPU cores in each memaslap machine – a practice discouraged by the creators of memaslap.

Regardless of the exact reason of the deviation, the IRTL holds to a reasonably high accuracy. Perfect accuracy is impossible even without the middleware: in the baseline experiments (Milestone 1 Section 2), relative prediction error is 0.2%-0.5% in a selection of values of N that I checked.

Appendix A: Data for the factorial experiment

	servers	replication	writes	repetition_id	throughput
1	3	none	1	2	17836
2	3	none	1	3	14690
3	3	none	1	4	13173
4	3	none	10	2	16032
5	3	none	10	3	11676
6	3	none	10	4	10972
7	3	full	1	2	17855
8	3	full	1	3	18505
9	3	full	1	4	12828
10	3	full	10	2	15649
11	3	full	10	3	13360
12	3	full	10	4	11488
13	7	none	1	2	16969
14	7	none	1	3	14962
15	7	none	1	4	13147
16	7	none	10	2	16111
17	7	none	10	3	13622
18	7	none	10	4	11356
19	7	full	1	2	16905
20	7	full	1	3	14186
21	7	full	1	4	12789
22	7	full	10	2	13849
23	7	full	10	3	11471
24	7	full	10	4	10571

Table 5: Data used in building the 2^k model in Section 4.