# Advanced Systems Lab (Fall'16) – Third Milestone

Name: *Tom Sydney Kerckhove*
Legi number: *15-908-064*

**Grading**

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

# 1 Overview

This report is the third installment in a series of three reports for the Advanced System Lab course. The two previous reports considered the building of a middleware and experimentation. This report will focus on modelling the system.
An illustration of the situation can be found in figure 1.

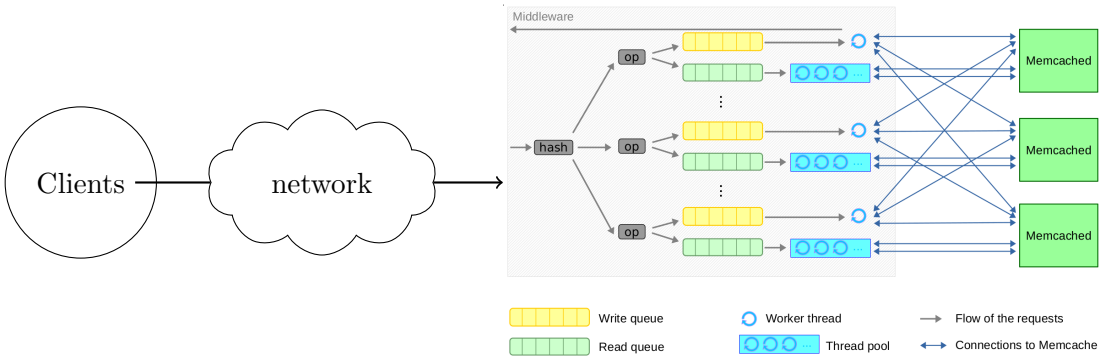


Figure 1: Complete view of the system

In this report I will use the same definitions as in the last report. Section 3 of the previous report lists them clearly.

# 2 System as One Unit

## 2.1 Boundaries of M/M/1 model

To make a M/M/1 model of the entire system, we have to treat the system as a black box. The black box is drawn from the middleware, across the network on the server side and across the servers.
This black box seemed the most appropriate. Adding the network on the client-side to the black box would add more network-traveling to the service that we are modelling. On the other hand, drawing the black box around the middleware only makes it hard to define what the service of the model is.
In figure 2, there is an illustration of the black box used for this model.

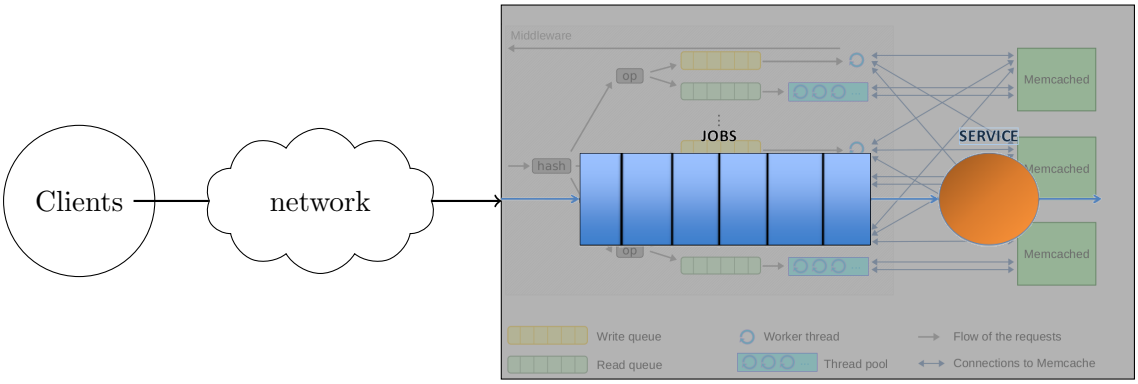


Figure 2: Black box system for M/M/1 model

The 'service' that the black box is providing is the same service as what a single Memcache instance would provide. In the end, this is precisely the point of using a middleware like this. The black box abstracts away from a lot of details, so while it's relatively easy to define what

the 'service' is, we cannot really point to a component of the real system and call it the 'queue' of this model.

## 2.2 Parameter estimation

Next we pretend that everything in this black box corresponds to a system as modeled by an M/M/1 model, and try to estimate the parameters $\lambda$ and $\mu$ of the M/M/1 model.
The model parameters will be estimated on a stability trace experiment that was run in the first part of the project. However, due to updates to the system, this experiment has had to be re-run. The details of the configuration can be found in figure 3.

| | |
|---|---|
| Number of server machines | 3 |
| Number of client machines | 3 |
| Virtual clients per machine | 64 |
| Workload | Key 16B, Value 128B |
| Write percentage | 1% |
| Replication | 1 |
| Middleware threads per read pool | 16 |
| Runtime x repetitions | 1h x 1 |
| Log files | stability-trace-client-logs |
| | stability-trace-middle-traces |

Figure 3: Stability trace

### 2.2.1 Estimation of the arrival rate $\lambda$

The arrival rate $\lambda$ is estimated by the average[1] throughput as seen by the clients. Indeed, because we are dealing with a closed system, the rate at which requests arrive at the black box is exactly equal to the rate at which requests are processed. To justify this way of estimating $\lambda$, we rely on the fact that we are dealing with a closed system. Indeed, the number of requests that have been processed in a given period is equal to the number of requests that arrive in that period.

### 2.2.2 Estimation of the service rate $\mu$

The service rate $\mu$ is estimated by the maximum throughput throughput as seen by the clients over all the 1 second intervals of the experiment. This can be justified by realising that the service rate is the maximum speed at which the system can process requests. If a given throughput is realised in a one-second interval, that means the system can process requests at least that quickly. This means that taking the maximum throughput will yield a lower bound for the service rate. Because we consider many one second intervals, this estimation should give us a relatively accurate lower-bound for the service rate.

## 2.3 Comparison with experimental results

In figure 4, we can see the M/M/1 model that was built according to guidelines of the previous subsection and box 31.1 of the book [1].

[1] over the 1 second intervals of the trace

| Measure | Model | Unit |
|---|---|---|
| Arrival rate | 12055.919 | (transactions / second) |
| Service rate | 21871.000 | (transactions / second) |
| Traffic intensity | 0.551 | |
| Mean response time | 101.884 | $\mu s$ |
| Std Dev response time | 101.884 | $\mu s$ |
| Mean waiting time | 56.161 | $\mu s$ |
| Std Dev waiting time | 91.048 | $\mu s$ |
| Jobs in the queue | 0.677 | Jobs |
| Jobs in the system | 1.228 | Jobs |
| Jobs in queue/jobs in system | 0.551 | |

Figure 4: M/M/1 model

It is important to note that 'response time' means the total time before the black box replies. It has little to do with what the clients call the 'response time'. The mean waiting time means the time that a request spends in the M/M/1 model queue before being processed.

### 2.4 Comparison and mapping between comparison and system components

Because the real system is much more complex than the system in an $M/M/1$ model, we cannot map measurements onto predictions perfectly. The data in figure 5 are the most relevant data available.

| Measure | Measurement | Unit |
|---|---|---|
| Mean time in middleware | 2329.660 | $\mu s$ |
| Std Dev time in middleware | 5155.201 | $\mu s$ |
| Mean time in queue | 863.859 | $\mu s$ |
| Std Dev time in queue | 3629.069 | $\mu s$ |
| Jobs in the system | 105 | Jobs |
| Total nr of workers | $(16 + 1) \times 3 = 51$ | Jobs |
| Jobs in the queue | $(105 - 51) = 54$ | Jobs |
| Jobs in queue/jobs in system | 0.514 | |

Figure 5: M/M/1 model

The mean response time predicted by the model is compared to the average total time between receiving a request and sending a response (in the middleware). This is called the mean time in middleware or the response time of the middleware. The mean waiting time is compared to the time between being enqueued and being dequeued for any request.
In both cases we see that the prediction is not even close to the measurements. The ratio between the measurements and the predictions are 41.48 and 15.38 for response time and waiting time respectively.
This ratio is so large because the model does not take parallelisation in account. A system with only one queue and one server that achieved a service rate as high as 21871.000 with an arrival rate of 12055.919 would have to have a much shorter response time and waiting time than the real system.
Indeed, the ratio of the measured response time to the predicted response time is close to the total number of workers in the real system. The ratio is still smaller, of course, because not every worker in the system is necessarily constantly busy.
The predicted number of jobs in the system is compared to the total number of clients in the system, an over-approximation for the real number of jobs in the system. The predicted number

of jobs in the queue is compared to this approximation minus the total number of workers in the real system, an approximation of the total number of jobs in any queues in the middleware. Once again we see that the predictions are a lot smaller than than the measured values. This is for the same reason: A system with one queue and one server, that can process jobs with the given service rate at the given arrival rate, would have much fewer jobs in the system and in its queue than the real system with parallelisation.
In fact, this can be shown by looking at the ratio between the number of jobs in the queue and the number of jobs in the system. These ratios are very close, precisely as we would expect.

# 3 Analysis of System Based on Scalability Data

## 3.1 Boundaries of M/M/m models

For these model, we use the same black box as in the last section. In figure 6, there is an illustration of the black box used for these models.

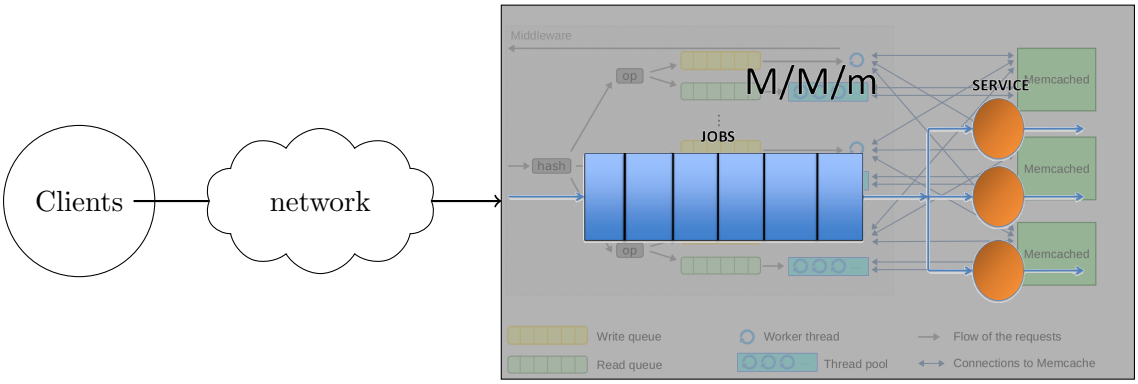


Figure 6: Black box system for M/M/m model

## 3.2 Parameter estimation

The arrival rate $\lambda$ is estimated just like for the M/M/1 model in the last section: as the average throughput. The reasoning behind this estimation is exactly the same as in the previous section.

### 3.2.1 Estimation of the number of servers

To estimate the number of servers $m$ in the $M/M/m$ model, it is tempting to use the number of Memcache instances. However, because all the workers in the middleware can work almost entirely in parallel, it makes more sense to use the total number of workers in the middleware instead. This total number of workers is equal to the number of Memcache instances times the number of threads in each reader thread pool plus one for the worker thread per server.

### 3.2.2 Estimation of the service rate $\mu$

The service rate $\mu$ is estimated using the maximum throughput, just like in the previous section, but unlike in the previous section, the service rate is estimate as this maximum throughput divided by the number of servers in this $M/M/m$ model.

### 3.2.3 Experiment

The data from which the $M/M/m$ models are estimated comes from the 'effect on replication experiment' from the previous report. Due to updates in the system, this experiment was re-run. The details of the setups can be found in figure 7.

| | |
|---|---|
| Number of server machines | [3, 5, 7] |
| Number of client machines | 2 |
| Virtual clients per machine | 35 |
| Workload | Key 16B, Value 128B |
| Write percentage | 5% |
| Replication | [1, $\lceil S/2 \rceil$, $S$] |
| Middleware threads per read pool | 16 |
| Runtime x repetitions | 1m x 3 |
| Log files | replication-effect-client-logs |
| | replication-effect-middle-traces |

Figure 7: Experiment setup

### 3.3 Characteristics of the models

For each different setup, a seperate $M/M/m$ model was built according to the estimation techniques above. The resulting models can be found in figure 8.

| $S$ | $R$ | $\lambda$ | $\mu$ | $m$ | $\rho$ | E[r] ($\mu s$) |
|---|---|---|---|---|---|---|
| 3 | 1 | 10321 | 245.84 | 51 | 0.823 | 4439 |
| 3 | 2 | 10026 | 229.14 | 51 | 0.858 | 4881 |
| 3 | 3 | 9879 | 226.53 | 51 | 0.855 | 4925 |
| 5 | 1 | 10937 | 150.87 | 85 | 0.853 | 7080 |
| 5 | 3 | 10384 | 144.24 | 85 | 0.847 | 7384 |
| 5 | 5 | 9718 | 137.12 | 85 | 0.834 | 7723 |
| 7 | 1 | 10563 | 103.25 | 119 | 0.860 | 10184 |
| 7 | 4 | 9679 | 99.04 | 119 | 0.821 | 10487 |
| 7 | 7 | 8968 | 94.38 | 119 | 0.799 | 10948 |

Figure 8: M/M/m models

In this table $S$ is the number of memcache instances, $R$ is the replication factor, $\rho$ is the traffic intensity and $E[r]$ is the predicted average response time.
The traffic intensity and response time are predicted according to the formulas in Box 31.2 of the book [1].

### 3.4 Comparison with real data and analysis

In figure 9 and 12, response times are plotted. The 'Real' response times are the 'response times' as measured by the middleware. This is the total time for an average request being received and a response being sent. The 'Model' response times are the average responses of the model system predicted by each $M/M/m$ model.
In figure 9, the bars are first grouped by replication coefficient, and then by number of servers.

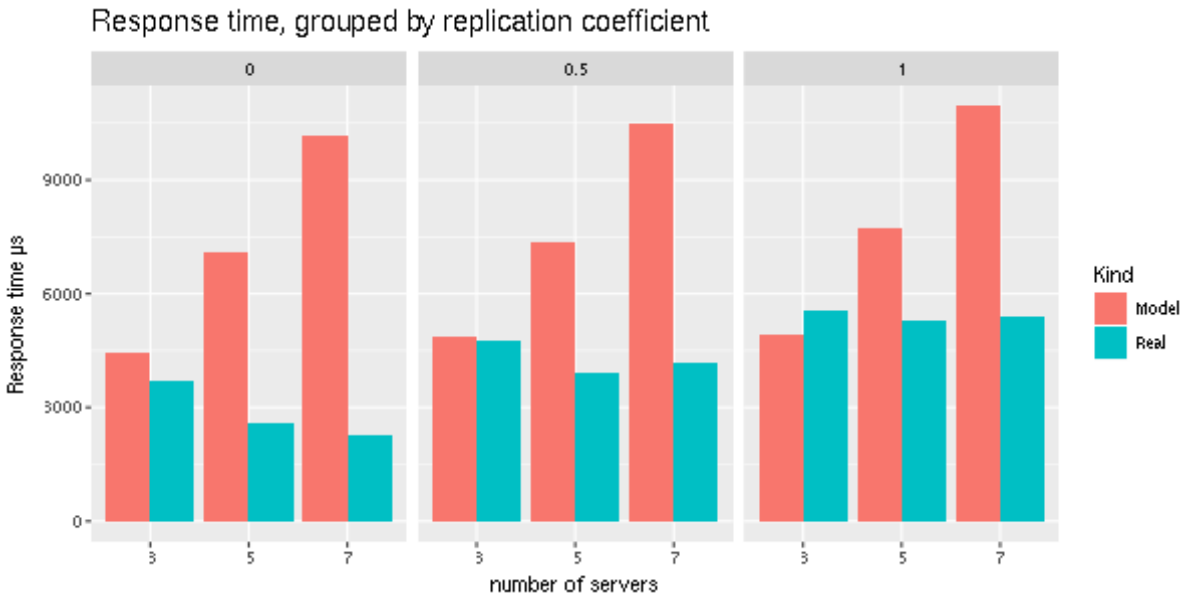


Figure 9: Response time by Replication Coefficient

A number of observations can be made about this plot.

- The models predict a severe increase in response time for an increased number of servers.

  This is because of the way the models are constructed from both the throughput and number of workers. To explain, we have to look at a similar plot, but this time plotting absolute average throughput on the y axis as in figure 10.

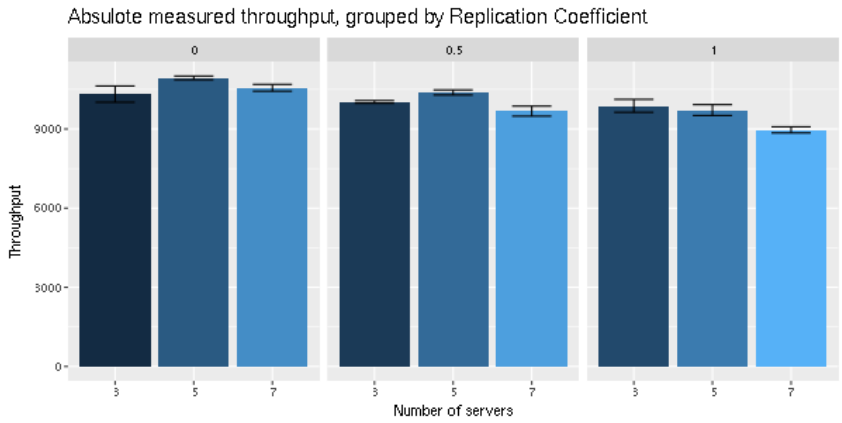


Figure 10: Response time by Replication Coefficient

As we can see, for fixed replication coefficients 0 and 0.5 and an increasing number of Memcache instances, throughput first increases and then decreases. This is because the number of worker threads in the middleware increases for every new server. This increase in number of worker threads helps throughput when going from 3 to 5 servers, but when going from 5 to 7 servers, the extra workers don't help anymore because of the way they need to contend for CPU time. This initial increase weakens as the replication coefficient increases because of how much more expensive writes become, and for full replication it is completely gone. Overall throughput decreases (slightly) as the number of servers increases.

However, the number of workers increases rapidly (linearly) as more servers are added. This means that that the service rates of the models decrease as the number of servers grow, mainly because the number of servers in the model increases so quickly. When service rates drop in the models, the predicted response time increases, which is why this observation concerning response times makes sense, even if it does not model reality well.

- The severity of this increase is independent of the replication coefficient.

  After the previous explanation, this makes perfect sense as well. As the increase in predicted response times is mainly due to the increase in the number of workers in the middleware, and this increase in number of workers is independent of the replication coefficient, it makes sense that the increase in predicted response times is also independent of the replication coefficient.

- The real response times decrease for an increasing number of servers.

  This makes sense because more servers allow for a better distribution of the load. With more servers, queues will more often be empty and as a result, requests spend less time in queues. As a result, the response time decreases.

  Keep in mind that this is the response time as measured by the middleware. The response time as measured by the clients does in fact behave exactly as the throughput would suggest in a closed system. This can be seen in figure 11

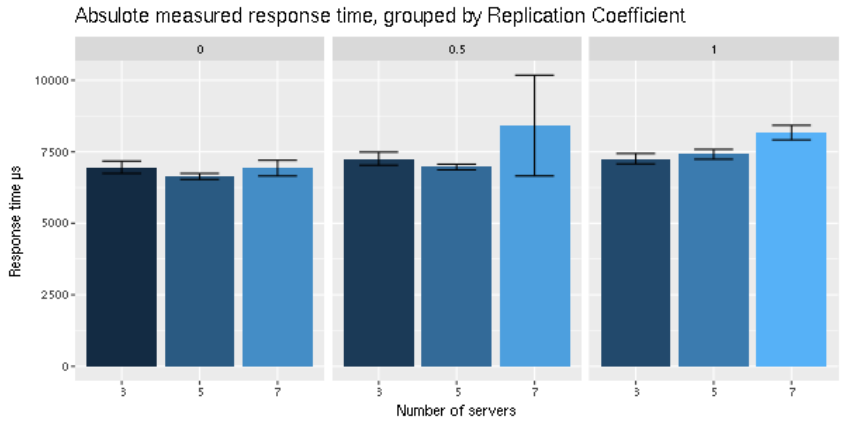


Figure 11: Response time by Replication Coefficient

The reason behind this difference in the response time as measured by the middleware and the response time as measured by the client is due to the following phenomenon. As the total number of workers increases with the number of servers, the workers have to contend more and more for CPU time. That means that the thread that receives in the requests gets an increasingly smaller share of time to do so. As long as requests haven't been received, their 'timer' for response time doesn't start yet.

- The decrease in real response times (as measured by the middleware) is dependent on the replication coefficient.

  The same situation as described in the previous point happens for all different replication coefficients. However, as the replication coefficient increases, the replication factor increases and, as a result, write requests become a lot more expensive. This can be seen in both the response times as measured by the clients (figure 9) and the response times as measured by the middleware (figure 11). This effect of more expensive write requests on the response time somewhat cancels out the effect of an increased number of servers on the

response time (as measured by the middleware) that was described in the previous point. As a result the response time decreases more slowly the higher the replication coefficient.

Figure 12 shows the same data as figure 9, but first grouped by the number of servers and then by the replication factor.


Figure 12: Response time by Number of Servers

On this plot we additionally see:

- Both in the models and in the real data, the response time increases as the replication factor increases.

  The reason why the real response times increase as the replication factor increases was already discussed previously: because write requests become more expensive. The reason that the model models this trend well, is because of the decrease in maximum throughput as a result of an increased replication factor on throughput. This can be seen in figure 13.

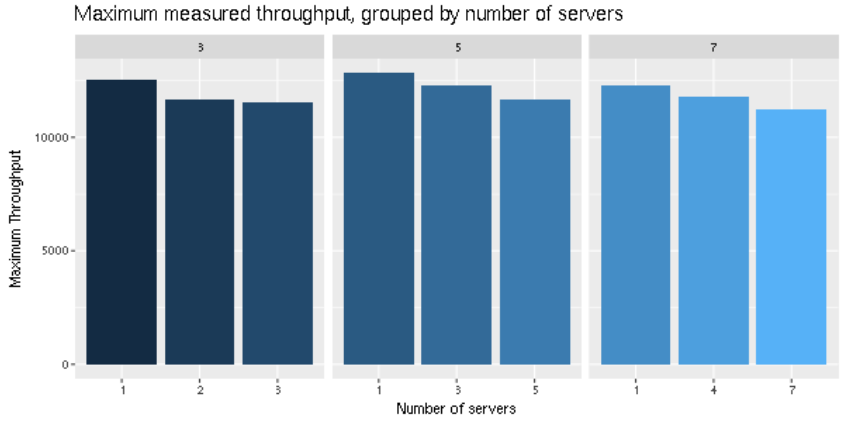


Figure 13: Response time by Replication Coefficient

A decrease in maximum throughput, for a fixed number of servers, directly results in a decreased service rate of the model. As a result it makes sense that the response time predicted by the models increases.

# 4 System as Network of Queues

## 4.1 Definition of the network

The same black box is used as in the previous section. The modeled network is an open queuing network and it is defined as follows.
For a given setup with $S$ servers and $T$ threads per reader threadpool, the queueing network consists of $1+4 \cdot S$ service centers. The first network is an $M/M/1$ service center and it represents the part of the middleware that handles incoming requests. Its queue is not explicitly visible in the code, because it is defined in Java's Asynchronous IO framework, but it is there nonetheless. Next, for each server, there is an $M/M/T$ service center that represents the $T$ threads in the reader threadpool. The 'service' of this service center includes the synchronous sending of read requests, the network delay, the Memcache read operation, and sending back the response to the appropriate client. For each server, there are three more service centers that represent the parts of the middleware that handle writes. First there is an $M/M/1$ service center that represents the asynchronous sending of write requests. Directly behind that, there is an $M/M/\infty$ delay center, that represents the network delay and Memcache's write operations. Lastly, behind that, there is another $M/M/1$ service center that represents sending back the response to the appropriate client if the replication is done. [2]
A detailed illustration of this queueing network can be found in figure 14

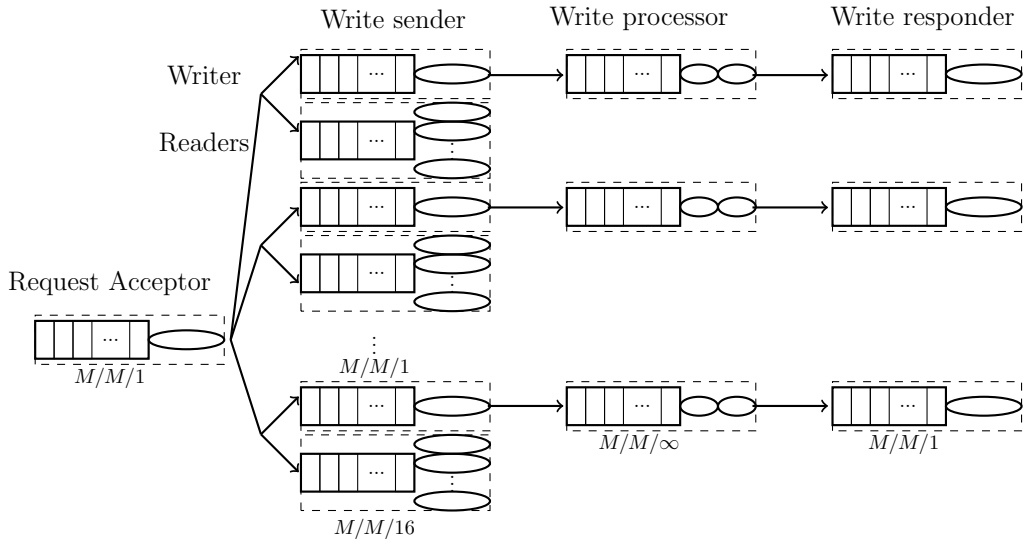


Figure 14: Comprehensive network of queues

The reasoning behind choosing an open queueing network as opposed to a closed network is twofold.

- An open queueing network is more generally applicable than just a memaslap workload.
- Modelling a closed queueing network would require additional estimates for network times and client think times. While think times are easy to estimate (see section 6.1), estimating network times accurately, while not interfering with acceptor queue times, is much harder and would require additional measurements.

[2] For details on how write requests are handled, see the first report.

### 4.2 Experiment setup

A single model of this kind was built using the data of a new experiment. This experiment was designed so as to exhibit as many different aspect of the system under test. For this reason, many clients and servers were used, the middleware used many threads per reader queue, and the replication coefficient was set to 0.5.
The details of this experiment can be found in figure 15.

| | |
|---|---|
| Number of server machines | 7 |
| Number of client machines | 3 |
| Virtual clients per machine | 35 |
| Workload | Key 16B, Value 128B |
| Write percentage | 10% |
| Replication | 4 |
| Middleware threads per read pool | 16 |
| Runtime x repetitions | 3m x 5 |
| Log files | extreme-client-logs |
| | extreme-middle-traces |

Figure 15: Experiment setup

### 4.3 Parameter estimation

The network of queues as modelled above has 7 parameters and they were estimated as follows. Given that the system that we are measuring data in is a closed system, we can estimate the overall arrival rate as the average throughput. The service time of the acceptor service center was estimated as the time between receiving a request and enqueuing the request. The service time of the read workers' service center was estimated as the time between dequeuing a read request and responding to the request's client. The number of servers of the read worker's service center was estimated as the number threads per read thread pool. The service rate of the first write service center was estimated as the time between dequeueing a write request and forwarding the request to the first server. The service rate of the second write service center was estimated as the time between forwarding a write request to the first server and receiving the last response from the servers. The service rate of the third write service center was estimated as the time between receiving the last response from the servers and and sending the response back to the client.
The model that was derived from the data gathered from the experiment as detailed in figure 15 can be found in figure 16.

| Measure | Value | Unit |
|---|---|---|
| Arrival rate | 8652 | transactions / second |
| Acceptor service time | 17 | $\mu s$ |
| Read worker service time | 1977 | $\mu s$ |
| Nr of reader servers | 16 | Servers |
| Write sender service time | 14 | $\mu s$ |
| Write processor service time | 7509 | $\mu s$ |
| Write responder service time | 8 | $\mu s$ |

Figure 16: Network of queues model parameters

The most important characteristics of the model can be found in figure 17. These numbers were computed using Octave's queueing package [3].

| Modeled measure | Value | Unit |
|---|---|---|
| Response time | 2552 | $\mu s$ |
| Acceptor utilisation | 0.14796 | |
| Reader utilisation | 0.13743 | |
| Write sender utilisation | 0.00174 | |
| Write processor utilisation | 0.92814 | |
| Write responder utilisation | 0.00102 | |
| Reader waiting time | 0.000 | $\mu s$ |
| Write sender waiting time | 0.024 | $\mu s$ |

Figure 17: Network of queues characteristics

### 4.4 Identification of bottlenecks

The writer receiver service centers have the highest utilisation, but because they are delay centers, they cannot be identified as the bottleneck of the system theoretically. The service center with the highest utilisation that is not a delay center is the acceptor service center. We conclude that this is the bottleneck of the system.
It makes sense that this is the bottleneck of the system because it is the only part of the middleware that does not scale with the number of servers that are available. In fact, as more servers are added, this part even shrinks in relative terms because of CPU contention between the threads.

### 4.5 Analysis of the model

In figure 18, there are measurements of the real system.

| Measure | Value | Unit |
|---|---|---|
| Response time | 3263 | $\mu s$ |
| Read queue waiting time | 794 | $\mu s$ |
| Write queue waiting time | 5 | $\mu s$ |

Figure 18: Network of queues measurements

The response time in this table corresponds to the response time as measured by the middleware. The prediction in figure 17 is relatively close to the real response time, but the distance is still significant.
The real queue waiting times are much larger than the predcted queue times. In fact, if we take the weighted average of the real queue waiting times, then we get a value that is extremely close to the difference between the real response time and the predicted response time.

$$0.9 \cdot 794 + 0.1 \cdot 5 = 751 \approx 711 = 3263 - 2552$$

This suggest that queue times are the most important mis-prediction of this model However, the extreme under-estimation of the queue times by the model is entirely reasonable. This problem occurs because the model does not take into account the fact that all parallelisation in the middleware happens on the same machine. As such, the different threads have to contend with eachother for CPU time. This means that adding a new thread does not just create more capacity, it just divides up the capacity among more threads. Whenever a thread is scheduled to do work, it can work at the given service rate, but as the total number of workers in the middleware increases, each thread gets scheduled less frequently. If all middleware threads could

work at their respective service rate all of the time, the model would predict queueing times more accurately.[3]

# 5 Factorial Experiment

A full factorial experiment with replication has been run to explore the impact of three factors $A$, $B$, and $C$ on a response variable $y$; as listed in figure 19.

| | |
|---|---|
| A | Replication coefficient |
| B | Request Value Size |
| C | Write Percentage |
| y | Throughput (transactions / second) |

Figure 19: Symbol definitions

Three repetitions were performed for each setup. For each of these factors, two different configurations were specified. All the elements of the Carthesian product of these configurations are tested. The details of this experiment's setups can be found in figure 20.

| | |
|---|---|
| Number of server machines | 3 |
| Number of client machines | 3 |
| Virtual clients per machine | 35 |
| Workload | Key 16B, Value [128, 1024]B |
| Write percentage | [5%, 50%] |
| Replication | [1, $S$] |
| Middleware threads per read pool | 16 |
| Runtime x repetitions | 3m x 3 |

Figure 20: Factorial experiment

The results of this experiment can be found in figure 21.

| A | B | C | $y$ | $\bar{y}$ |
|---|---|---|---|---|
| -1 | -1 | -1 | [12900,12959,12806] | 12888.37 |
| 1 | -1 | -1 | [11955,11808,11995] | 11919.74 |
| -1 | 1 | -1 | [12307,12299,12201] | 12268.91 |
| 1 | 1 | -1 | [11524,11617,11477] | 11539.34 |
| -1 | -1 | 1 | [12759,12890,12555] | 12734.86 |
| 1 | -1 | 1 | [9759,10158,9766] | 9894.13 |
| -1 | 1 | 1 | [11211,11529,11424] | 11387.97 |
| 1 | 1 | 1 | [9358,9175,9312] | 9281.52 |

Figure 21: Throughput results

In figure 22, we can find a sign table of an additive model of this $2^3 \cdot 3$ experiment.

[3] An extra experiment I ran with only one server and 6 middleware thread confirms this.

| I | A | B | C | AB | AC | BC | ABC | $\bar{y}$ | Err |
|---|---|---|---|---|---|---|---|---|---|
| 1 | -1 | -1 | -1 | 1 | 1 | 1 | -1 | 12888 | [12,71,-83] |
| 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 11920 | [36,-111,76] |
| 1 | -1 | 1 | -1 | -1 | 1 | -1 | 1 | 12269 | [38,30,-68] |
| 1 | 1 | 1 | -1 | 1 | -1 | -1 | -1 | 11539 | [-15,78,-63] |
| 1 | -1 | -1 | 1 | 1 | -1 | -1 | 1 | 12735 | [24,155,-179] |
| 1 | 1 | -1 | 1 | -1 | 1 | -1 | -1 | 9894 | [-135,263,-128] |
| 1 | -1 | 1 | 1 | -1 | -1 | 1 | -1 | 11388 | [-177,141,36] |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 9282 | [76,-106,30] |
| 91915 | -6645 | -2959 | -5318 | 973 | -3249 | -960 | 495 | Total | |
| 11489 | -831 | -370 | -665 | 122 | -406 | -120 | 62 | Total/8 | |

Figure 22: Throughput Sign Table (additive model)

| Factor | Variation | Percentage |
|---|---|---|
| A | 16560392 | 46.67 |
| B | 3284178 | 9.26 |
| C | 10604919 | 29.89 |
| AB | 355272 | 1.00 |
| AC | 3958437 | 11.16 |
| BC | 345339 | 0.97 |
| ABC | 91968 | 0.26 |
| Error | 280110 | 0.79 |

Figure 23: Throughput Variation Table (additive model)

In figure 23 variation explained by each factor is laid out, as well as the relative variation explained by each factor.
This table suggests that the replication coefficient has the highest impact on throughput, out of the factors that were considered. A higher replication coefficient means that every write request needs to hit additional Memcache servers. This means that a significant ratio of the requests that passes through the middleware, the write requests, are suddenly a lot more expensive. As a result, the average request will take longer to be processed and consequently fewer requests can be processed in a given amount of time. The fact that this is the most important factor is consistent with the common knowledge [2] that the network is one of the most expensive parts of a distributed system.
The next most important factor is the write percentage. This makes sense because write requests only have one thread to deal with them per server, as opposed to the entire thread pool that read requests get. This matters because while forwarding requests happens asynchronously, sending responses back to clients happens synchronously. In addition to this, write requests are more expensive than read requests as soon as any replication is used.
The third most important factor is, interestingly, an interaction factor. The interaction between the replication coefficient and the write percentage is the third most important factor. This means that the impact of the replication coefficient on the throughput is even greater in the case of a greater write percentage and vice versa. This is because writes become significantly more expensive if replication is used, but writes need not be much more expensive than reads if no replication is used at all.
The last factor that has a significant impact is the size of requests. This is because larger requests are more expensive for Memcache to process, but also because the middleware needs to do more work in order to process a larger request. Every byte of a request is copied twice.

The first time to parse it from the contents of a ByteBuffer and the second time to render back the response as a ByteBuffer. As a result, it makes sense that throughput decreases as request value increases.
There are two more factors that have little impact on throughput: The combination of request value size and the write percentage, and the combination of all three individual factors.
Lastly, almost none of the variation is explained by experimental error, which tells us that the experiments were rather internally consistent.
For completeness: A multiplicative model was made as well, but because the difference between the maximum $y$ and the minimum $y$ is small, it gave similar results and the additive model already has little variation explained by error.

# 6 Interactive Law Verification

To check that the experiment data makes sense using the Interactive Response Time Law, we need the following data:

- Number of users of the system
- Response time
- Throughput
- Think time

We know the number of users of the system from the total number of virtual clients, and we can measure the response time and the throughput of the system. The only data that is missing is the think time.

## 6.1 Think time

To estimate the think time of an average client, the following benchmark was performed. An experiment was configured with one server, one middleware, one client. The middleware was configured to log every request instead of sampling requests to log. The client was configured to only use one virtual client. Most importantly: all of these were run on the same machine and this machine is of the same kind as the usual client machines. The details of this experiment can be found in figure 24.

| | |
|---|---|
| Number of server machines | 1 |
| Number of client machines | 1 |
| Virtual clients per machine | 1 |
| Workload | Key 16B, Value 128B |
| Write percentage | 1% |
| Replication | 1 |
| Middleware threads per read pool | 1 |
| Runtime x repetitions | 30s x 3 |
| Log files | think-time-client-logs |
| | think-time-middle-traces |

Figure 24: Think time benchmark

The middleware logs timestamps for the arrival of every request and for the time when the middleware responds. The difference between the moment in time when the middleware responds to one request, and the moment in time where it receives the next request is an estimate of

the think time of the client. This estimate is an over-approximation, but it should be relatively tight as the client, middleware and server are run on the same machine.
Using this method, the average think time of a client was estimated to be $Z = 50.43$ microseconds on average. This will be used in the next section.

## 6.2 Verification

The data to be verified comes from the experiments on the topic of the effects of the replication effect as listed in figure 7.
To verify that the experimental data makes sense, we predict the response time $R$ from the throughput $X$, number of users $N$ and the estimated think time $Z$. The results can be found in figure 25.

| Users | Avg Throughput | Estimated Response Time | Measured Response Time ($\mu s$) | Difference |
|---|---|---|---|---|
| 70 | 10321.48 | 6731.54 | 6958.27 | 226.73 |
| 70 | 10026.45 | 6931.10 | 7258.74 | 327.63 |
| 70 | 9879.29 | 7035.10 | 7257.77 | 222.67 |
| 70 | 10937.27 | 6349.71 | 6637.54 | 287.84 |
| 70 | 10384.07 | 6690.67 | 6967.15 | 276.49 |
| 70 | 9717.61 | 7152.99 | 7412.68 | 259.69 |
| 70 | 10562.82 | 6576.59 | 6929.92 | 353.33 |
| 70 | 9678.63 | 7182.00 | 8417.23 | 1235.23 |
| 70 | 8968.38 | 7754.77 | 8171.69 | 416.92 |

Figure 25: Interactive response time verification

As we can see, the estimated response time is smaller than the actual response time. The error is small and it makes sense. The average response time is not perfectly representative for the response time of a typical request, because of the fact that the response times comes from a skewed distribution with a long tail, and the average is heavily influenced by outliers. This can be seen in figure 26

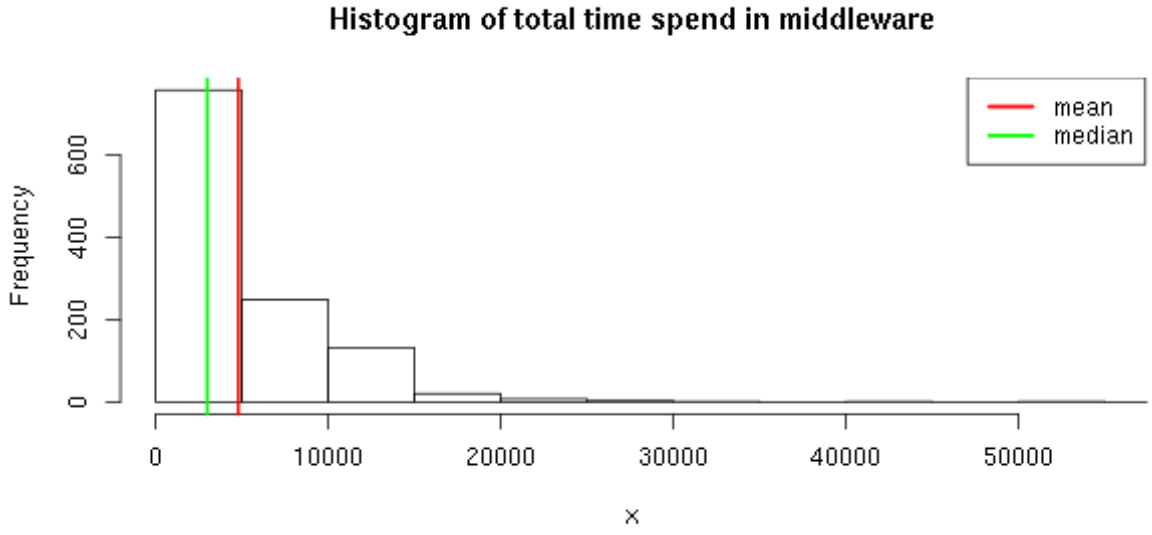


Figure 26: Histogram of Response time

In conclusion, the data gathered during the performed experiments looks valid.

# 7 Log file listing

| Short name | Location |
|---|---|
| stability-trace-client-logs | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-stability-trace/local-client-logs |
| stability-trace-middle-traces | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-stability-trace/traces |
| think-time-client-logs | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-think-time/local-client-logs |
| think-time-middle-traces | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-think-time/traces |
| replication-effect-client-logs | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-replication-effect/local-client-logs |
| replication-effect-middle-traces | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-replication-effect/traces |
| 2k-factorial-client-logs | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-2k-factorial/local-client-logs |
| 2k-factorial-middle-traces | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-2k-factorial/traces |
| extreme-client-logs | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-extreme/local-client-logs |
| extreme-middle-traces | https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-extreme/traces |

# References


[1] The Art of Computer Systems Performance Analysis: Techniques, Jain, Raj and Menasce, Daniel and Dowdy, Lawrence W. and Almeida, Virgilio AF and Smith, Connie U. and Williams, Lloyd G. 2010, John Wiley & Sons.

[2] Latency Numbers Every Programmer Should Know https://gist.github.com/jboner/2841832

[3] The 'queuing' package for the Octave programming language http://www.moreno.marzolla.name/software/queueing/