Advanced Systems Lab (Fall'16) – Second Milestone

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

Grading

Section	**Points**
1	
2	
3	
Total	

1 Introduction

This is the second report in a series of three reports for the the "Advanced Systems Lab". In this course we implement and investigate the performance of a middleware for Memcache. This part focuses mostly on investigating the performance of the system that was built for the previous part.

2 Fixes and upgrades

The middleware has changed since the last part to improve its efficiency.
Previously the middleware was able to perform very verbose logging for debugging purposes. This logging could be turned off at runtime. Sadly, due due Java's strict evaluation semantics, the logging messages were still constructed even if they weren't printed. As a result, the middleware would perform very poorly and achieve a very low throughput.
The middleware has been adapted not to include this logging functionality anymore. It is now much more performant.

3 Background, internals and definitions

The exact internal workings of the middleware system are described at length in the first report of this series. The most relevant bits are repeated here for reference.

3.1 Instrumentation

Each request is instrumented with different timestamps that indicate certain important points in the processing of the request.
Two of these timestamps indicate the time that a first request to any server was sent, and the time that the last response was received from all servers. The time between these two timestamps is called the 'interaction time' because that is when it is interacting with the servers.
In the case of read requests, there is always only one server to contact, so this interaction time signifies the time it takes for the server to respond. In the case of write requests, the interaction time signifies the time it takes for the middleware to receive a response from all relevant servers. In the case of no replication that is just one server, but in the case of full replication that is all the servers.

3.2 Threads

The middleware uses separate queues and workers for the two kinds of requests. Read request are executed synchronously by a pool of read workers per server. Write requests are executed asynchronously by a single write worker per server. The number of threads in the read worker pool will be referred to as the 'thread count' of the middleware.

3.3 Replication

The replication factor is an absolute number between 1 and the number of servers. The so-called replication coefficient R_c is a fraction that represents the ratio of redundant replication to the total number of backend servers. For example: For a setup with a total of 7 backend servers and replication factor 4, the replication coefficient is $\frac{1}{2}$ It can be computed with the formula $R_c = \frac{R-1}{S-1}$.

3.4 Workload

The pretend clients that are used in the experiments use a predefined distribution of requests. One fraction of the requests will be read requests and the rest will be write requests. The percentage of the workload that consists of write requests is called the write percentage.

4 Maximum Throughput

4.1 System under test

In the first experiment, the middleware is in charge of five servers with no replication and a read-only workload configuration. The aim is to investigate which combination of the total number of clients and the number of threads in each server's thread pool for reads achieves the highest throughput. The details of the explored configurations can be found in figure 1.

Number of server machines	5
Number of client machines	2
Virtual clients per machine	[1 .. 96]
Workload	Key 16B, Value 128B
Write percentage	0%
Replication	No replication ($R = 1$)
Middleware threads per read pool	[1 .. 21]
Runtime x repetitions	1m x 1
Log files	maximum-throughput-client-logs
	maximum-throughput-middle-traces

Figure 1: Maximum throughput experiments

4.2 Hypothesis

Making an accurate estimate of the exact configuration that achieves the highest throughput is hard if not unnecessary. What is more important is to estimate what will happen as we increase the total number of virtual clients or the number of threads in each server's thread pool for reads.

4.2.1 Increasing the total number of virtual clients

As the middleware operates with an unbounded queue of requests to process, it never drops requests even if the load increases. [1]
As the total number of virtual clients increases, we expect to see throughput increase linearly up to the point where request queues are no longer always empty. From then on we expect to see throughput increase sub-linearly up to the point where the request queues are no longer ever empty. After that the throughput should not increase any further as subsequent will just be punt onto the queue and requests will not be processed any quicker.

4.2.2 Increasing the number of threads in each server's thread pool for reads

The fact that reads are executed in a synchronous manner means that each read thread can perform at most one request per round-trip time. Using more threads to processes read requests should somewhat solve this problem, but context switches are relatively expensive. As a result,

[1] We assume that the limits of memory are not reached. This assumption is reasonable as the middleware machine that will be used has 14 GiB of main memory.

we expect that using multiple threads will increase the throughput up to a certain point. At that point the cost of using multiple threads will outweigh the benefit.
In general we expect response time to remain constant up to the point where queues are no longer constantly empty. After this point, any more work gets put onto the queue and we then expect response times to increase linearly. [2]

4.3 Results and analysis

In figures 3, 4 and 5, the results of the maximum throughput experiment are plotted. For each thread count under test there are four plots.
The first plot shows the throughput in function of the total number of virtual clients and the second shows the total response time. Both of these first plots are derived from the data as viewed by the clients.
The next two plots show data gathered in the middleware. They show a detailed breakdown of the time that an average request spends in the middleware.

4.3.1 Maximum throughput configuration

An exact number for either the maximum throughput or the number of virtual clients at which this throughput is achieved is hard to compute. Uncertainties in measurement, along with a vague definition of 'maximum throughput' make it hard to give an exact number.
The estimated maximum throughput and the number of virtual clients at which this throughput is achieved can be found in figure 2. The maximum throughput estimate is given with an error of plus-minus 500 transactions per second. The estimate for the number of virtual clients is given with an error of plus-minus 10.

Number of threads	Maximum throughput (TPS)	Number of virtual clients
1	2000 ± 500	25
5	5000 ± 500	30
9	10000 ± 500	50
13	11000 ± 500	60
17	11500 ± 500	70
21	11750 ± 500	80

Figure 2: Maximum throughput estimates

[2] This is only true because the system under test is a closed system. The same argument would not necessarily hold for open systems.

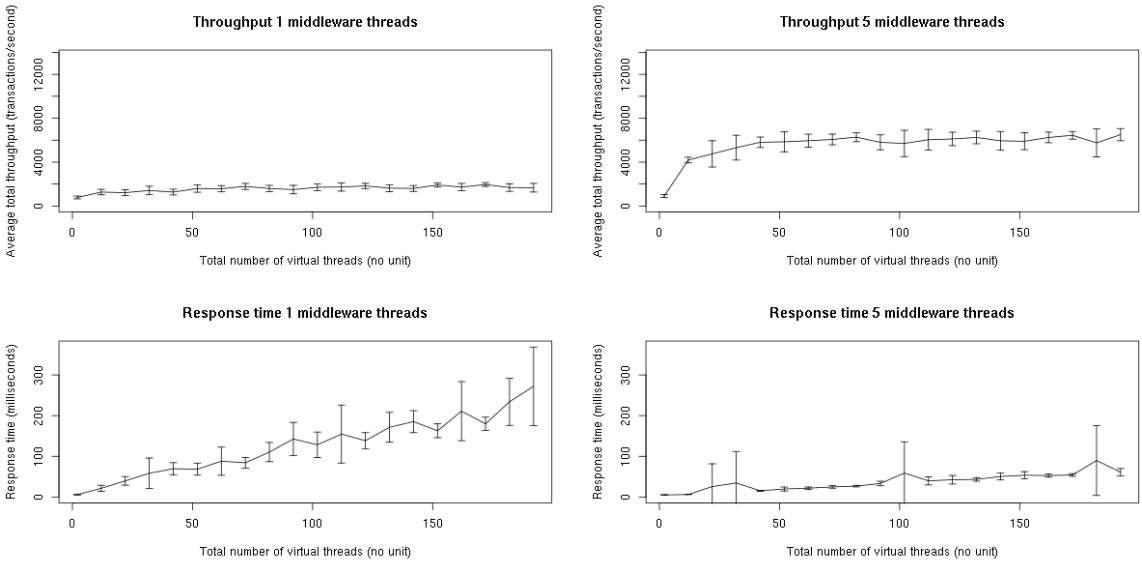

Figure 3: Maximum Throughput

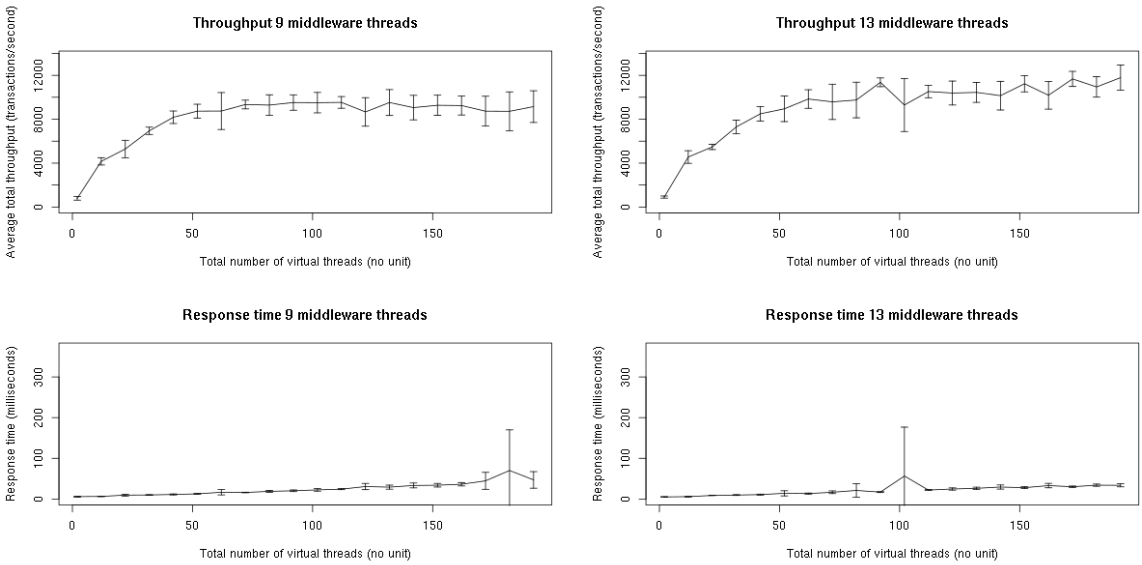

Figure 4: Maximum Throughput

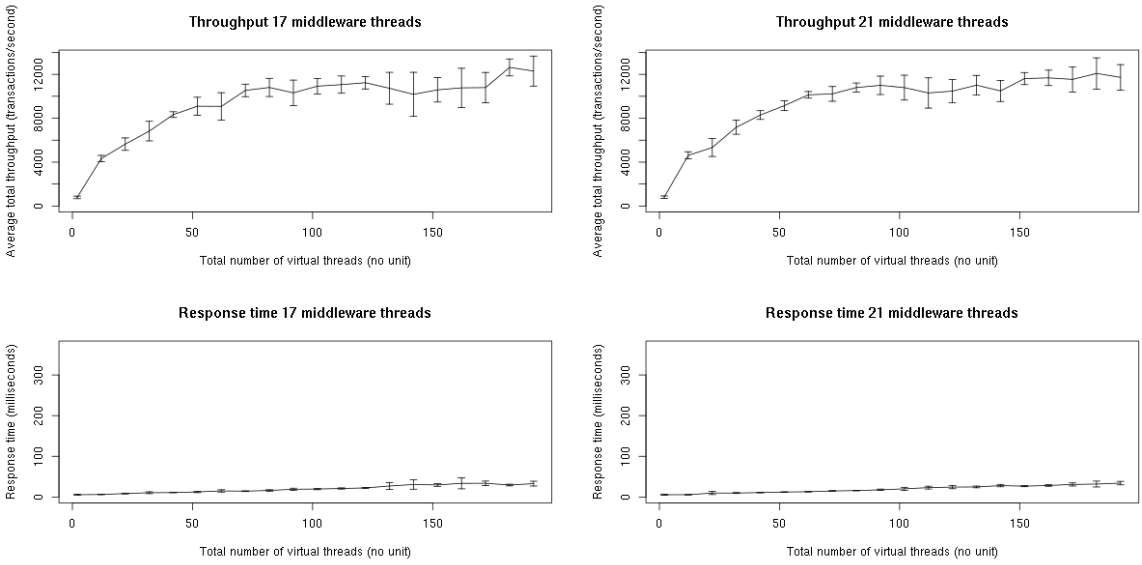

Figure 5: Maximum Throughput

4.3.2 Comparison of results with hypothesis

In each throughput plot we see that throughput first increases linearly, then increases sub-linearly and finally levels off. This corresponds exactly to the hypothesis.
In each response time plot, we see a linear increase in response time. There is no phase of constant response time to see in these plots as hypothesized. It is possible that the constant response time phase was too short to show up on the plots.

4.3.3 Breakdown of time spent in middleware

Figures 6, 7 and 8 show a detailed breakdown of where an average request spends time in the middleware. The top plots show the absolute time spent, broken down by operation. The bottom plots show the same data, but the time is show in percentages relative to the total amount of time spent for an average request.
Note that we see that these response time plots are very similar in shapes to the same response times as seen by the client.
In the case of one thread per read pool, we can see that the majority of an average request's time is spent on the queue, waiting to be processed. Using more threads per read pool results in the average request spending more time interacting with the server. This is because the threads start contending for for both access to the network and for CPU time. As the number of threads per pool increases further, the interaction time takes over most of the time spent. This corresponds to the predictions

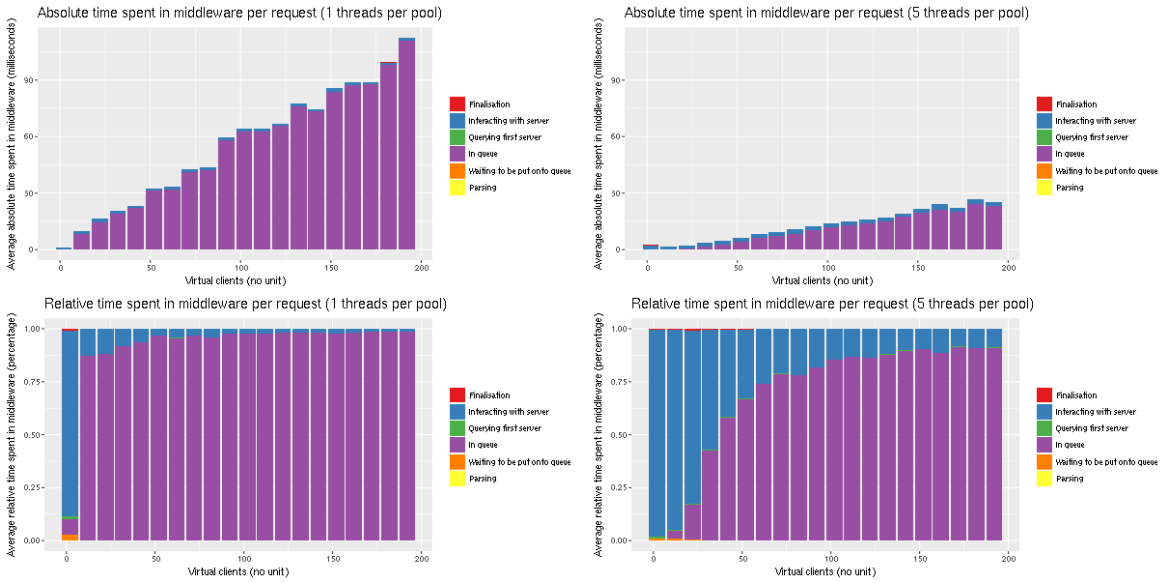

Figure 6: Maximum Throughput

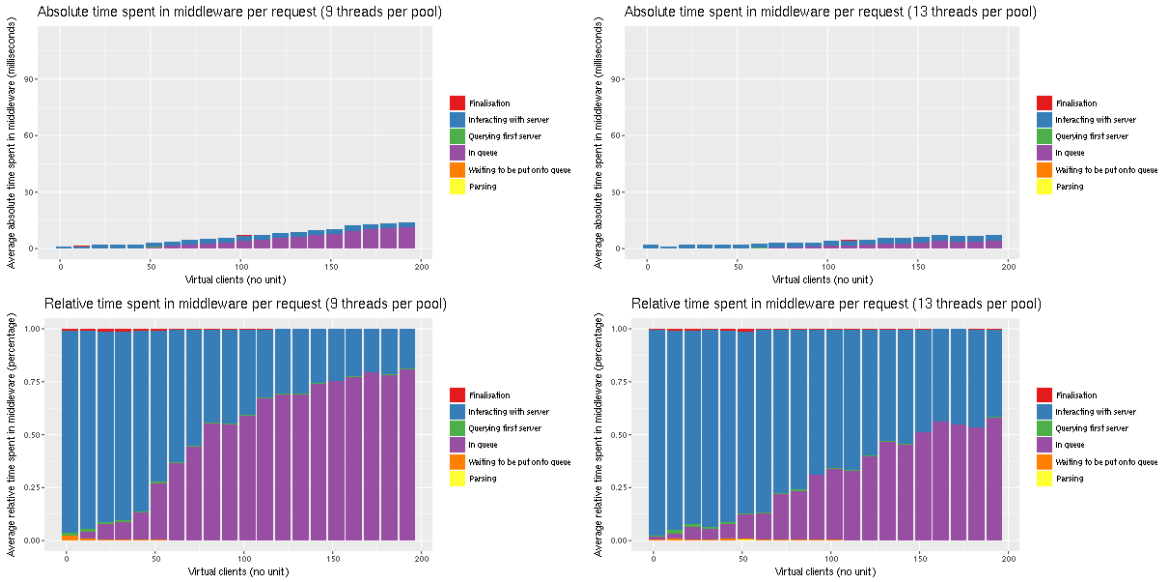

Figure 7: Maximum Throughput

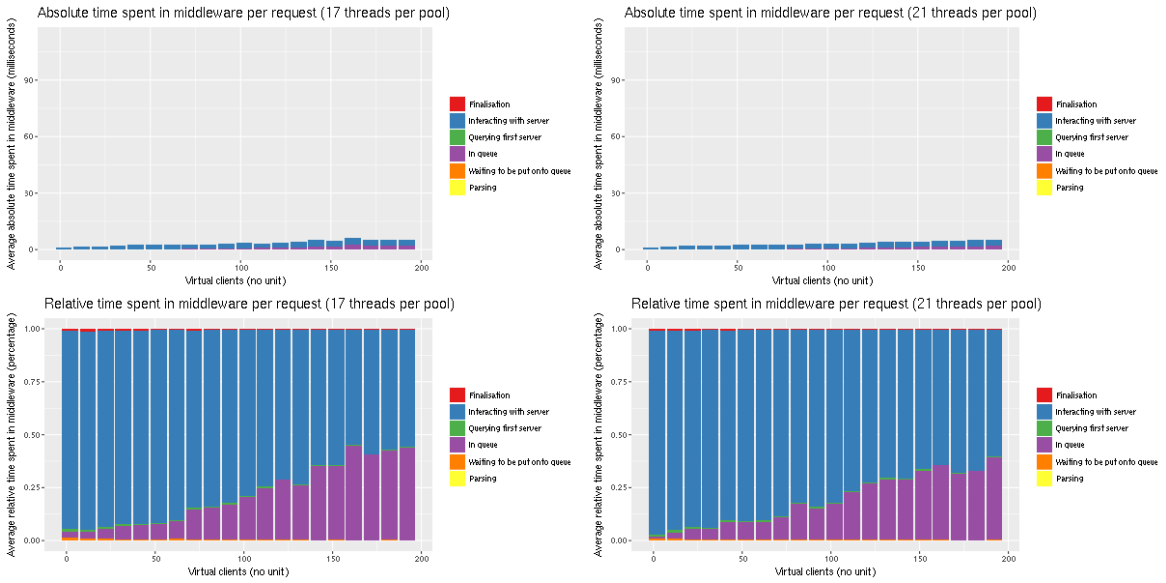

Figure 8: Maximum Throughput

5 Effect of Replication

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 1
Log files	replication-effect-client-logs
	replication-effect-middle-traces

Figure 9: Replication effect experiments

5.1 System under test

The second experiment aims to explore the effect of using more replication on performance. The clients will use a 5%-write workload. There will be 3, 5 or 7 server backends. Three configurations of the replication factor will be explored: No replication ($R = 1$), half replication ($R = \lceil \frac{S}{2} \rceil$) and full replication ($R = S$).
The details of this experiments' setups can be found in Figure 9.

5.2 Hypothesis

For any fixed number of backend servers, we expect the interaction time for write requests to increase when the replication factor is increased. This is because write requests need to be sent to more servers as the replication factor is increases and the middleware needs to wait for all the responses. As such we expect the total response time for write requests to increase. Naturally, for a greater number of backend servers, we expect this increase to be even larger.
For read requests, we don't expect the interaction time, nor the total response time, to increase when the replication factor is increased. This is because read requests only ever get sent to one

server no matter the replication factor.
For any fixed replication coefficient, we expect the interaction time for write requests to increase when the number of backend servers is increased. This is because, for a fixed replication factor, increasing the number of backend servers also increases the replication factor. Again we expect the total response time for write requests to increase as a result of this. However, for a greater replication coefficient, we expect this increase to be larger. In particular we expect there to not be a difference in the case of $R = 1$.
For read request, we don't expect the interaction time, nor the total response time, to increase when the number of servers is increased. Again this is because read requests only ever get sent to one server no matter the replication factor.

5.3 Results and analysis

In figure 10, there are grouped bar plots of response times [3], grouped by the replication factor, for each fixed number of servers.. On each plot we see that write requests incur a greater response time than read request. Moreover we see that response times for read requests stay constant as the replication factor increases. Write requests, on the other hand, become more expensive as the replication factor increases. This increase is larger for a greater number of backend servers.
These results exactly match the hypothesis

[3] as measured by memaslap

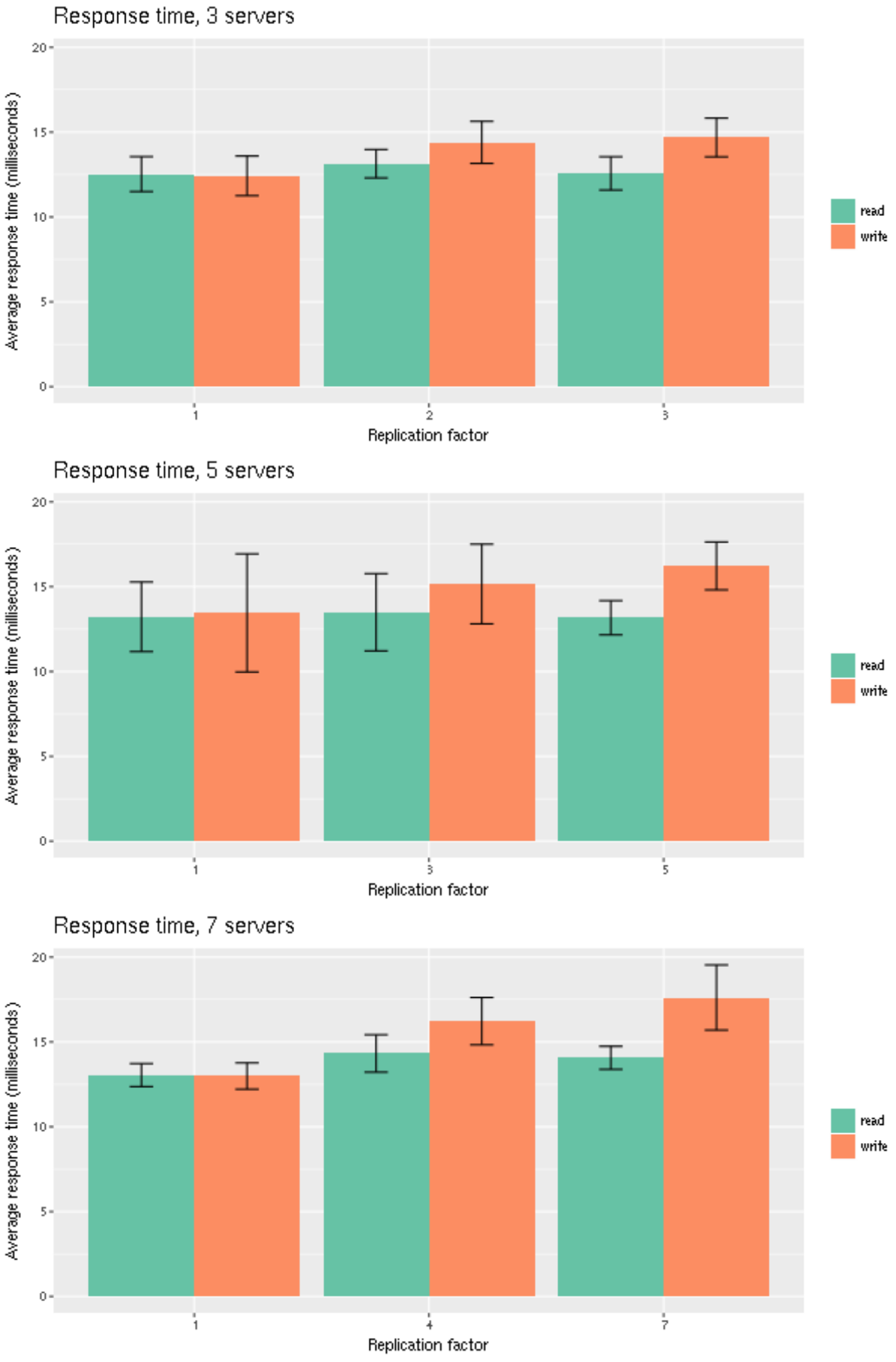

Figure 10: Effect of replication factor

In figure 11, there are grouped bar plots of response time, grouped by the number of backend servers, for each fixed replication coefficient. On each plot we again see that write requests incur a greater response time than read requests.
In the case of a replication coefficient of 0.0, we see that response times for both kinds of requests increase ever so slightly as the number of backend servers increases. This result differs from the expected result. It happens because there is more bookkeeping to do in total if there are more backend servers so there is more processing to contend with. This is where the real system differs from an ideal system.
In the cases of replication coefficients 0.5 and 1.0, we see that response times for read requests become only slightly longer while there is a big increase in the response times for write requests. Moreover, this increase is larger for the situation with replication coefficient 1.0 than for the situation with replication coefficient 0.5. These results match the expected results.

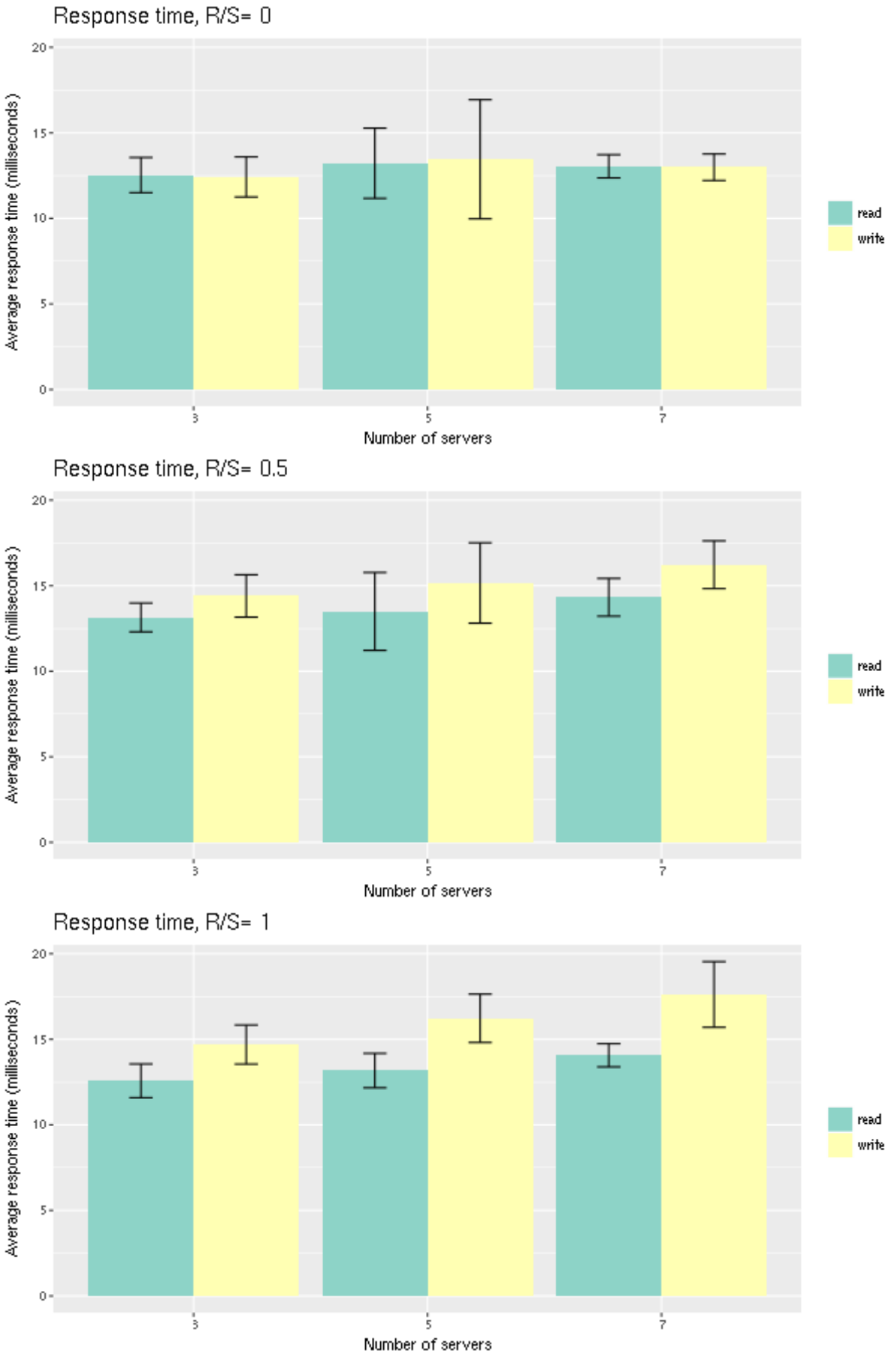

Figure 11: Effect of replication factor

In figure 12, there is a detailed breakdown of the time that an average requests spends in the middleware. On the left are the plots for read request, on the right are plots for write requests. Each horizontal pair of plots accounts for a fixed number of servers, with the replication factor on the x-axis.
First note that the shapes of these time[4] plots correspond to the shape of the plots of the response times as measured by the clients.
We can see that in both cases, the middleware spends most of a request's time interacting with the servers. As hypothesized, this time stays constant for read requests as the replication factor increases.
For write requests, the time spent interacting with the servers increases severely when the replication factor increases. Moreover, this effect is larger when the total number of backend servers is greater. This result also matches expectations.

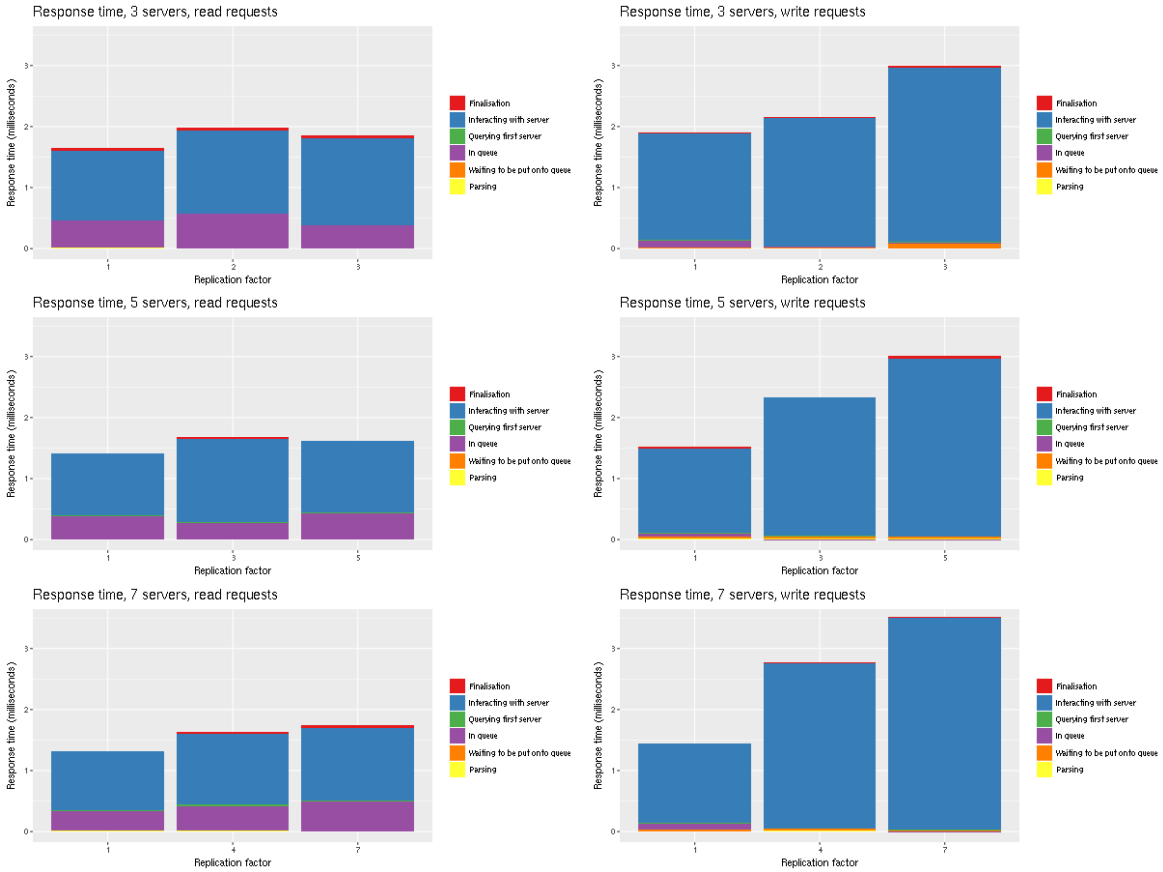

Figure 12: Effect of replication factor

[4]As measured by the middleware

6 Effect of Writes

Number of server machines	[3, 5, 7]
Number of client machines	2
Virtual clients per machine	35
Workload	Key 16B, Value 128B
Write percentage	[1%, 5%, 10%]
Replication	[1, S]
Middleware threads per read pool	16
Runtime x repetitions	1m x 1
Log files	write-effect-client-logs
	write-effect-middle-traces

Figure 13: Write effect experiments

6.1 System under test

In this last experiment, we investigate the results of increasing the size of the fraction of the workload that consists of write requests on performance. Between 3 and 7 servers will be used with either no replication or full replication. The percentage of the requests that consists of writes will be varied between 1% and 10%.
The details of the experiments' setups can be found in Figure 13.

6.2 Hypothesis

As investigated in the previous section, writes are much more expensive in a setup with full replication than in a setup with no replication. This means that we can expect there to be a difference in performance between the no-replication situation and the full-replication situation for any given non-zero write percentage. We expect that the system with full replication will perform worse and that this difference will grow larger as the write percentage increases. Moreover, we expect this difference to be even greater if more servers are used because full replication is even more expensive when more servers are used. Concretely we expect a lower throughput and higher response times in the case of full replication relative to the case of no replication. We also expect this difference to be larger for a setup with more servers and/or a greater write percentage.

6.3 Results and analysis

In Figure 14, the throughput and response time of the middleware is plotted in function of the write percentages of the clients' workloads for fixed numbers of backend servers.
The plots show that throughput and response time both stay constant as the write percentage increases in the case of no replication. In the case of full replication we see that throughput decreases and response time increases as the write percentage increases. Moreover, this decrease in throughput and increase in response time are greater for greater numbers of backend servers. These results match up with the hypotheses exactly.

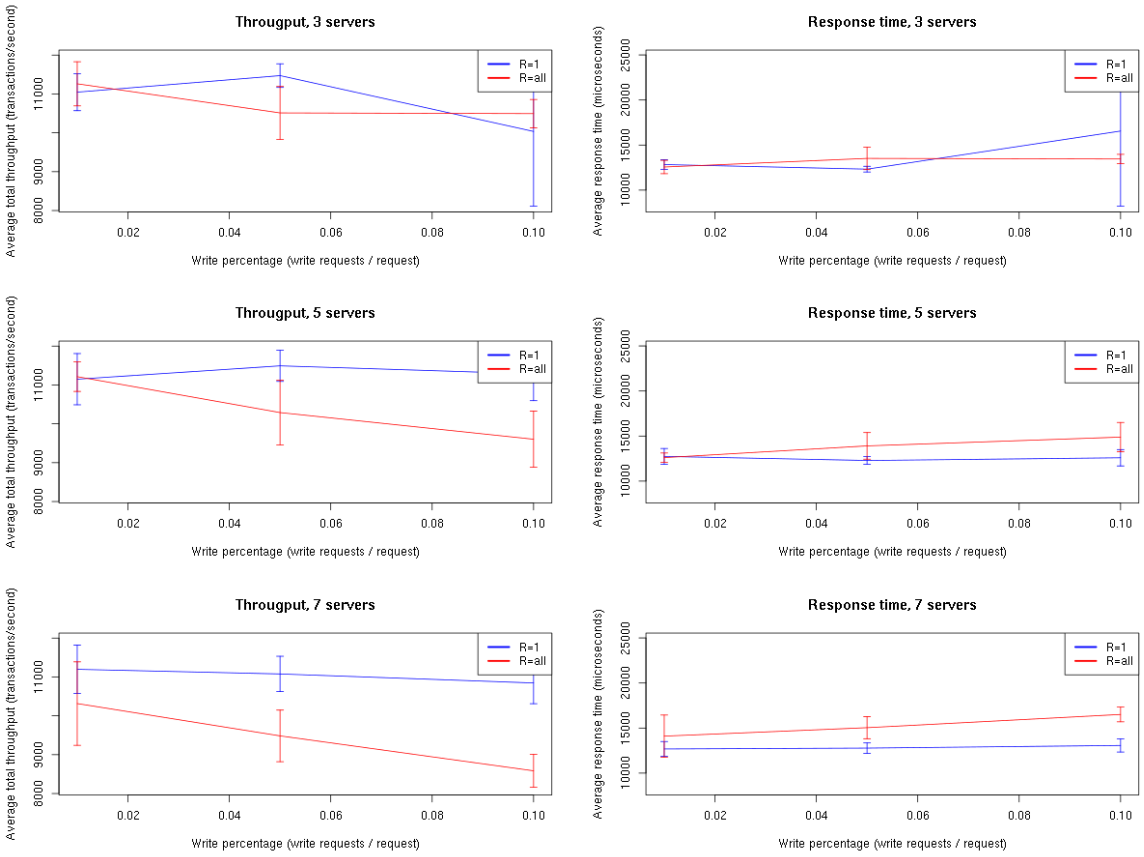

Figure 14: Effect of write percentage: Throughput

7 Log file listing

Short name	Location
maximum-throughput-client-logs	https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-maximum-throughput/local-client-logs
maximum-throughput-middle-traces	https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-maximum-throughput/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
write-effect-client-logs	https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-write-effect/local-client-logs
write-effect-middle-traces	https://gitlab.inf.ethz.ch/tomk/asl-fall16-project/blob/master/results/remote-write-effect/traces