Advanced Systems Lab (Fall'16) – Second Milestone

Name: *Eduardo Pedroni*
Legi number: *16-930-596*

Grading

Section	**Points**
1	
2	
3	
Total	

1 Maximum Throughput

This section explores how middleware throughput for read requests varies with different numbers of connected clients and middleware read threads.

1.1 Hypothesis

Middleware throughput for read operations is expected to increase with the number of connected clients, since client requests are generated in parallel by memaslap. Given enough clients, read throughput should begin to plateau as the middleware becomes saturated. Since the experiments are performed in a closed system, additional connected clients should not lead to significant performance degradation, as the number of requests in the system at any given time will never exceed the number of connected clients; pending requests simply wait longer in the read queue before being handled. Therefore, an increase in T_{queue} (the time spent by each request in its respective queue) as measured by the middleware is expected as the number of connected clients increases.

The read throughput is also expected to correlate with the number of read threads in the middleware. Increasing the number of read threads should enable the middleware to handle more requests per second, thereby raising the saturation point. However, additional read threads are not expected to increase the throughput indefinitely, as the efficiency of multi-threading is tied to the underlying hardware configuration and capacity.

1.2 Experimental Setup

1.2.1 Primary Factors

The experiment investigated the effect of varying the number of connected clients and middleware read threads over the following ranges:

Total connected clients	120 to 360 in steps of 40
Middleware read threads (per server)	10 to 40 in steps of 10

where total connected clients refers to the number of clients across all client machines and read threads refers to the number of threads handling read requests for each Memcached instance.

1.2.2 Secondary Factors

The following factors were held constant for all experiments:

Number of servers	5
Number of client machines	2
Workload	Key 16B, value 128B, reads 100%
Middleware replication	No replication ($R = 1$)
Memaslap sampling period	10s
Runs	$5 \times 200s$
Memcached threads	1
Logs	./reports/m2report/logs/section1-logs.tar.gz

Network congestion was assumed to remain constant throughout the experiment. Since it is impossible to control the geographical allocation of Microsoft Azure virtual machines, none of the virtual machine instances used were reallocated during the experiment. Virtual machines of constant size were used; Basic A2 for clients and servers, Basic A4 for the middleware. Experiments were performed for all primary factor level combinations using Bash scripts. Logs (stored in a tarball for compactness) consist of one directory for each run, named according to the following convention: "sec1-c{clients}-v{threads}-r{run}". Each run directory contains the middleware log ("middleware.log") and two memaslap logs, one for each machine used ("mema{0,1}.log").

1.3 Results

1.3.1 Throughput

Figure 1 shows the throughput of the system as measured by memaslap with increasing numbers of clients and threads. Raising the number of clients did not have a significant impact on throughput with only 10 threads; however, 20 or more threads yielded increasing throughput up to 320 clients, past which the throughput begins to plateau. At 320 clients, the difference in throughput between 20, 30 and 40 threads was well within the confidence interval of the measurements and thus negligible, as seen in Table 1. Therefore, the minimum number of clients and threads required to achieve maximum throughput was 320 and 20, respectively.

Read Threads	Throughput (operations/s)	95% Confidence Interval
10	17,417	309
20	20,915	261
30	21,076	328
40	20,762	440

Table 1: Middleware throughput with 320 clients and varying number of threads.

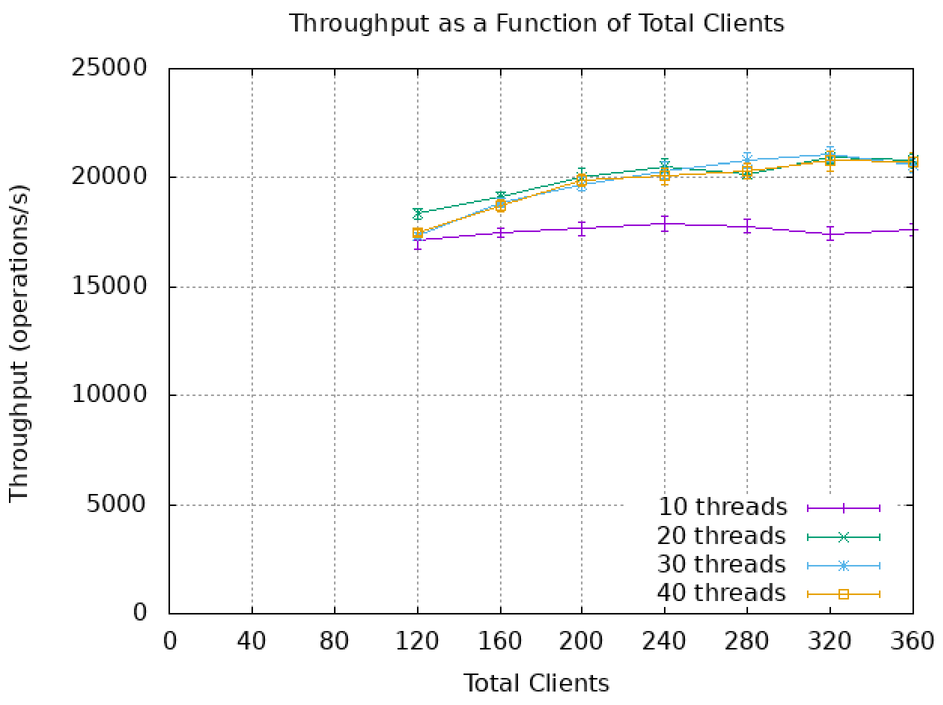


Figure 1: Middleware throughput with different combinations of read threads and total clients as measured by memaslap. Error bars show 95% confidence interval.

1.3.2 Response Time

Response time increased steadily with the number of clients and threads, as illustrated in Figure 2. Additional clients and threads also led to inconsistent response times, as evidenced by the increasingly wide distribution. The 10-thread configuration stood out with greater distribution range for all client values; and similarly to the throughput, there was no significant difference between the response times yielded by 20, 30 and 40 threads for any number of clients.

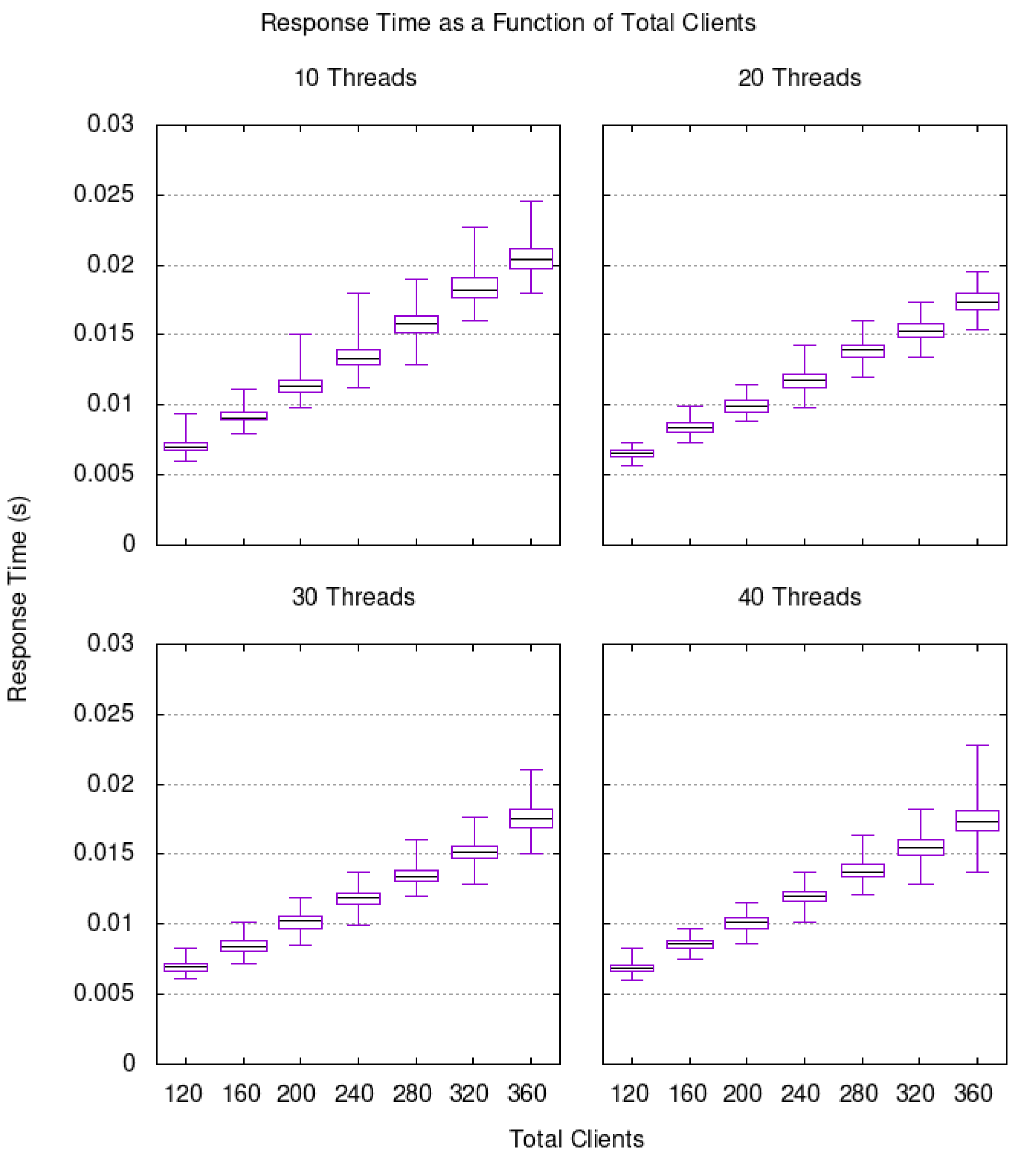


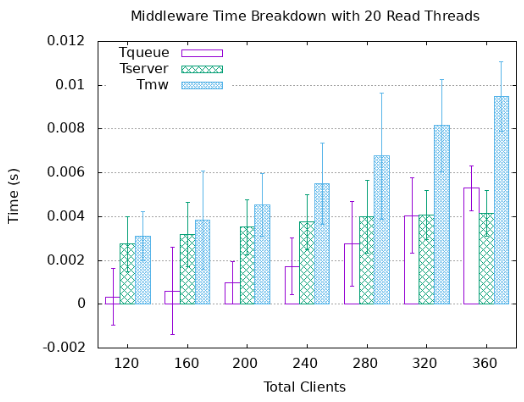
Figure 2: System response time with different combinations of read threads and total clients as measured by memaslap. Boxes show, from top to bottom: max, 75th percentile, median, 25th percentile and min.

1.3.3 Middleware Breakdown

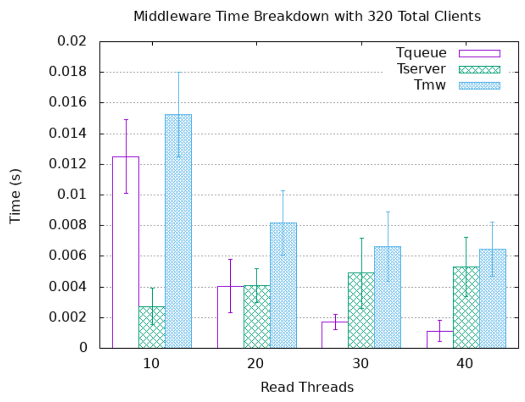
Figures 3 and 4 show a detailed breakdown of the internal middleware timings for the relevant combinations of the optimum parameters presented in Section 1.3.1: 320 threads and 20 clients.

Figure 3a shows a clear rising trend in T_{queue} as the number of clients increases with a fixed number of threads. A less accentuated increase in T_{server} is also noticeable, with an apparent tendency to plateau. Figure 3b shows the internal behaviour of the middleware for all explored thread counts given the optimum number of clients, 320. T_{queue} experienced a sharp drop with a tendency to level at around 1 ms above 40 threads, while T_{server} showed the same plateauing tendency seen in Figure 3a. In both figures, the total middleware time T_{mw} is roughly equal to the sum of T_{queue} and T_{server}.

Figure 4 paints a clearer picture of the middleware's total internal processing times. In the optimum case, the median time spent by read requests in the middleware was under 9 ms and 95% of the requests were processed in approximately 18 ms. The maximum observed time taken to process a request was over 1 s.



(a) Internal middleware timings across clients with the optimum number of threads.



(b) Internal middleware timings across threads with the optimum number of clients.

Figure 3: Middleware breakdown for optimum number of clients and threads. Error bars show 95% confidence interval.

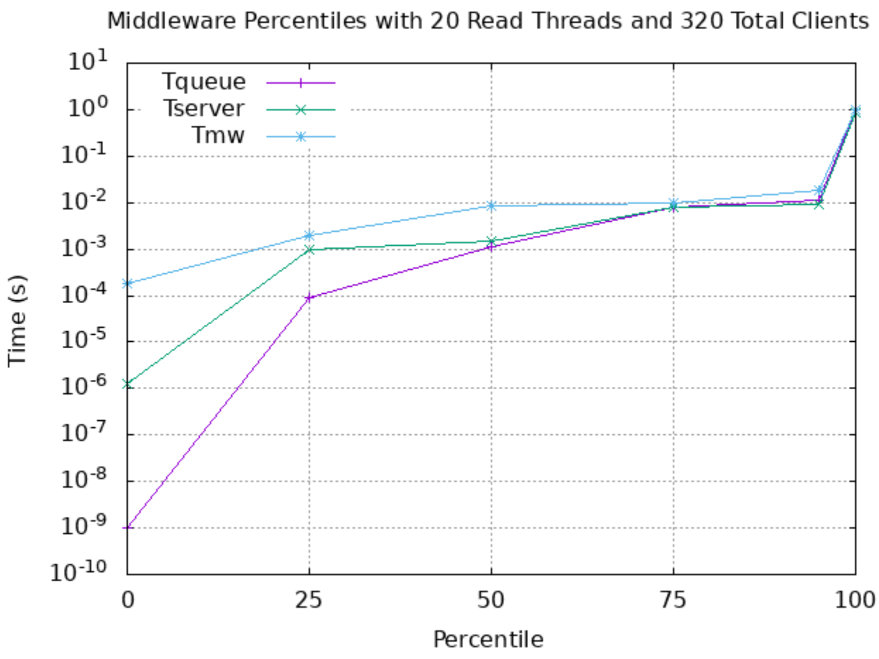


Figure 4: Internal middleware timings for the optimum combination of read threads and total clients. Error bars show 95% confidence interval.

1.4 Discussion

The results in Figure 1 agree with the predicted behaviour of the system; as expected, throughput increases with the number of clients and threads up to saturation, at which point the throughput plateaus with increasing threads and clients. This supports the hypothesis that the middleware reaches a saturation point given a large enough workload. Furthermore, the sustained throughput observed over the investigated client range with 10 read threads confirms that no significant throughput degradation occurs past the saturation point. It was hypothesised that an increase in T_{queue} would be observed instead; this is indeed the case as shown in Figure 3a. However, contrary to the predicted behaviour, T_{server} also rises with increased load. This may be explained by Memcached's execution parameters; running with a single thread, it is limited to processing requests synchronously. Additional load must wait before being handled, causing the observed increase in T_{server}.

Throughput alone does not fully characterise system performance, as response time consistency is often desirable in software performance. Figure 4 shows that the middleware processed the majority of requests in under 20 ms, which may be sufficiently fast depending on the application. However, Figure 2 shows a mean response time of approximately 15 ms with extremely high standard deviation; in certain applications, it may be more desirable to achieve more consistent response times at the expense of throughput. The general rising trend in Figure 2 suggests that the middleware may already be beginning to saturate even at 120 clients and 10 threads. In order to determine the maximum throughput with consistent response times, more experiments would need to be performed, investigating lower ranges such as 20 to 100 total clients and three to 15 read threads per server.

2 Effect of Replication

This section investigates changes in middleware performance due to variations in the replication factor for different numbers of server backends.

2.1 Hypothesis

Varying the number of Memcached instances and the replication factor is expected to affect get and set performance differently.

2.1.1 Get Performance

Get requests are always forwarded exclusively to the primary server regardless of replication factor, and the primary server is selected in constant time (assuming a constant key size). Therefore, a higher replication factor should not significantly impact get throughput and response time. However, additional servers are expected to cause an increase in the throughput and a decrease in the response time of get requests, since the total number of threads is the product of the number of servers and the middleware's read thread parameter, and additional threads have been shown to have a generally positive effect on performance in Section 1. This hypothesis assumes that Memcached is not significantly overloaded by the middleware and relies on the fact that the middleware hash distributes load approximately evenly (as shown in milestone 1).

2.1.2 Set Performance

Set requests are forwarded to a number of servers equal to the replication factor, so both throughput and response time should be affected by different replication factors and numbers of servers. For a fixed number of servers, increasing the replication factor is expected to cause a throughput drop, since the middleware aggregates all responses to set requests before replying to the client. It should also lead to a significant response time increase due to the only partially asynchronous design of the middleware write workers (all server responses must be received before the worker begins handling the next request).

For set requests, additional servers potentially mean additional requests to be sent and additional responses to be received, depending on replications. With replication factors greater than one, increasing the server count should impact set requests in the same way as increasing the replication factor, though the two effects are expected to compound. With a replication factor of one, increasing the number of servers should affect set requests in the same way as get requests (i.e. no effect), since the behaviour of the write workers with single replication is the same as that of the read workers.

2.2 Experimental Setup

2.2.1 Primary Factors

The experiment explored the effect of varying the number of Memcached server instances and middleware replication factor over the following ranges:

Number of servers (S)	3, 5, 7
Middleware replication factor	1, $\lceil \frac{S}{2} \rceil$, S

2.2.2 Secondary Factors

The following factors were held constant for all experiments:

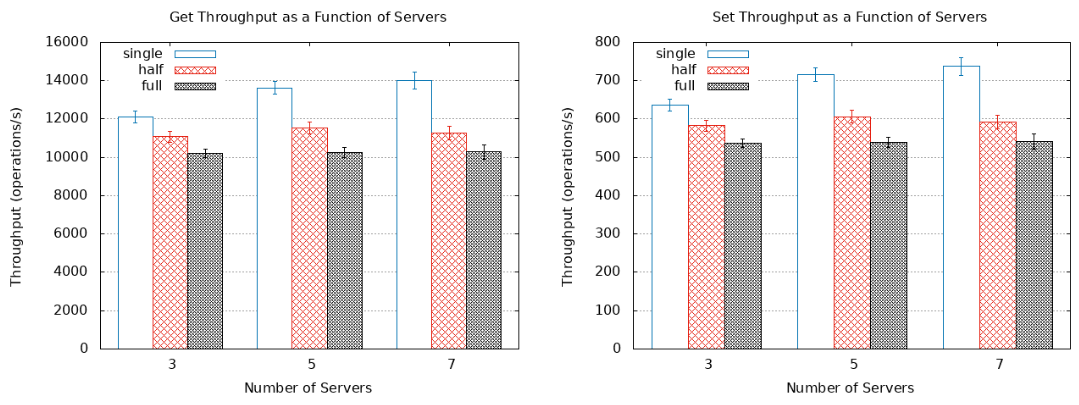
Number of client machines	2
Total number of clients	320 (160 per machine)
Workload	Key 16B, value 128B, reads 95%, writes 5%
Middleware read threads	20 per server
Memaslap sampling period	10s
Runs	5 × 200s
Memcached threads	1
Logs	./reports/m2report/logs/section2-logs.tar.gz

All limitations listed in Section 1 regarding Azure virtual machines apply. Client and middleware read thread numbers were chosen based on the optimum results from Section 1. Logs (stored in a tarball for compactness) consist of one directory for each run, named according to the following convention: "sec2-s{servers}-v{replications}-r{run}". Each run directory contains the middleware log ("middleware.log") and two memaslap logs, one for each machine used ("mema{0,1}.log").

2.3 Results

2.3.1 Throughput

Figure 5 shows the throughput as measured by memaslap for both get and set requests with all combinations of factors. Although get requests achieved much higher throughput than set requests, both request types suffered a proportional decrease in throughput as the replication factor was increased. Increasing the number of servers led to a minor throughput increase for both request types with no replication, and no significant throughput change for replication factors greater than one.



(a) Get request throughput.

(b) Set request throughput.

Figure 5: Comparison of get and set request throughput behaviour for all combinations of factors. Error bars show 95% confidence interval.

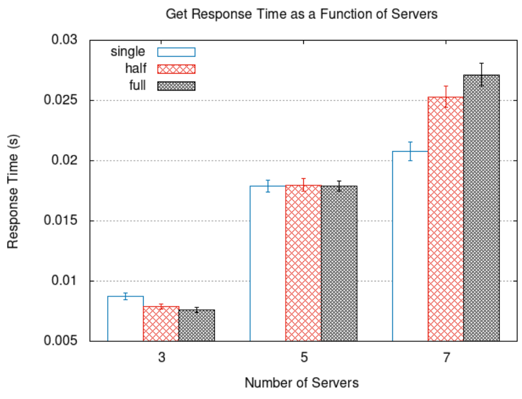
For each combination of factors, the ratio of measured set throughput to combined throughput is consistent with the workload write proportion (5%), as shown in Table 2:

Servers	Replications	Combined Throughput	Set Throughput	Ratio
3	single	12,745	636	4.99%
3	half	11,665	582	4.99%
3	full	10,765	537	4.99%
5	single	14,349	717	4.99%
5	half	12,145	606	4.99%
5	full	10,797	539	4.99%
7	single	14,759	737	4.99%
7	half	11,864	592	4.99%
7	full	10,829	541	4.99%

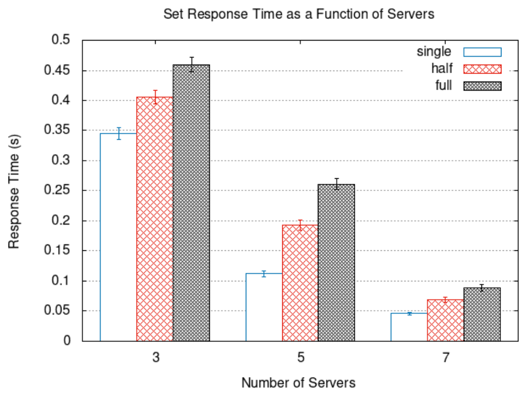
Table 2: Ratio of set throughput to combined throughput for all factor combinations.

2.3.2 Response Time

As shown in Figure 6, the response time of get requests was observed to increase slightly with the number of servers, with little impact from increased replication factors. Set request response time displayed a rising trend associated with increasing replication factors for all server configurations. However, increasing the number of servers caused a uniform reduction in response time independent of replication factor.



(a) Get request response time.



(b) Set request response time.

Figure 6: Comparison of get and set request response time behaviour for all combinations of factors. Error bars show 95% confidence interval.

2.3.3 Middleware Breakdown

Figures 7 and 8 illustrate the internal behaviour of the middleware for get and set requests respectively, over all combinations of factors. Get requests were largely unaffected by increasing replication factors in the three- and five-server cases, although raising the number of servers also increased the request processing time. With seven servers, raising the replication factor caused an increase in response time. T_{queue} was generally shorter than T_{server} across all factor combinations.

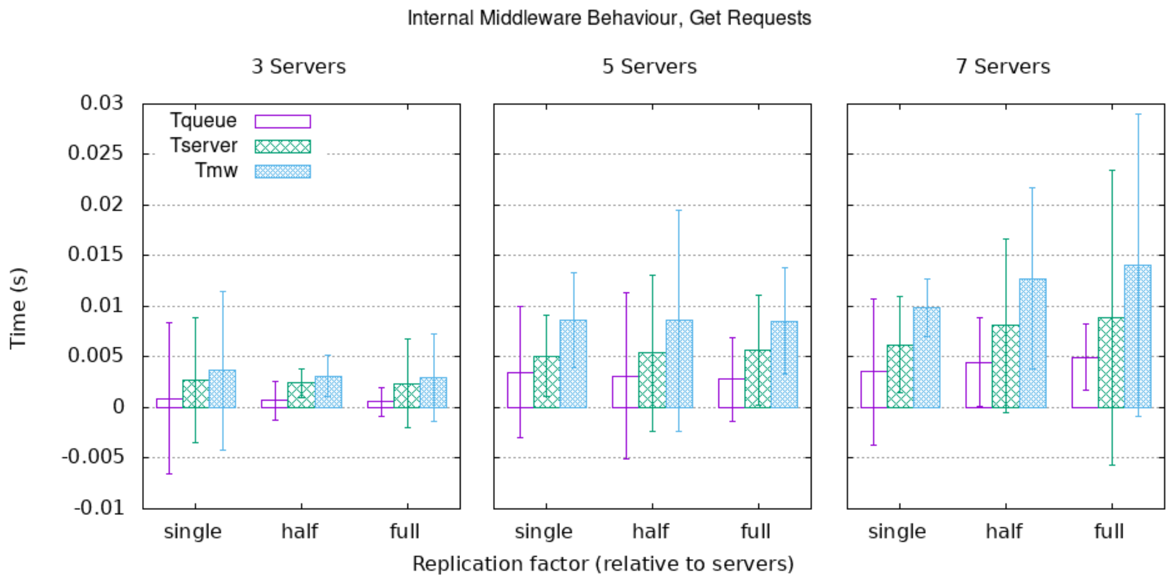


Figure 7: Internal middleware timings for get requests with all combinations of replications and servers. Error bars show 95% confidence interval.

Set requests were significantly affected by the explored factor combinations. Across all server configurations, increasing the replication factor led to a clear increase in T_{queue}, which always accounted for the majority of T_{mw}. However, a dramatic drop in T_{queue} was observed as the number of servers was increased. An approximately tenfold reduction was observed between three and seven servers with no replication (from 0.33 ms to 0.03 ms.)

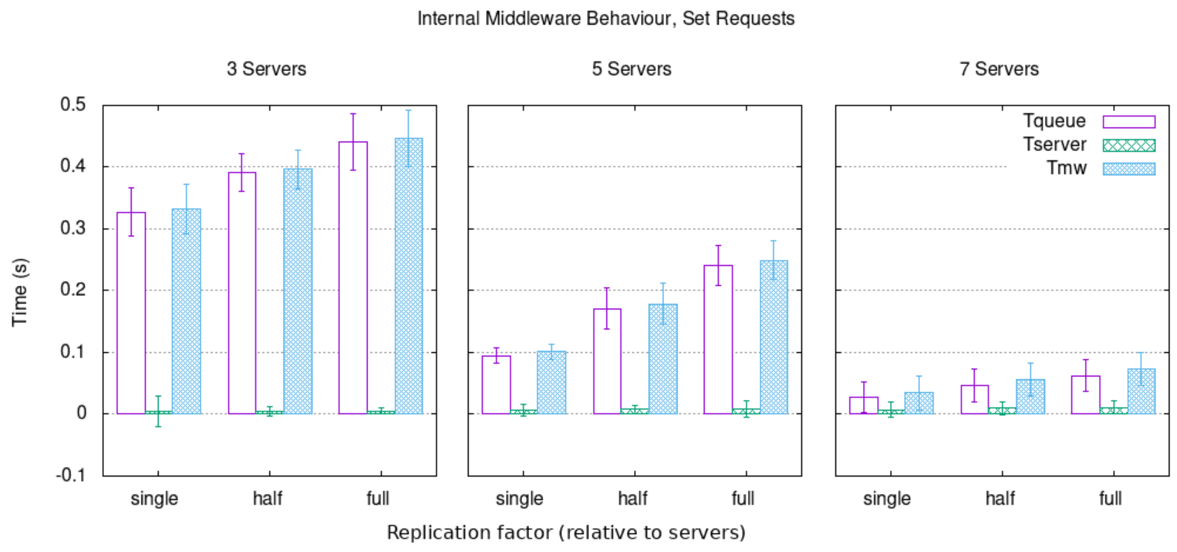


Figure 8: Internal middleware timings for set requests with all combinations of replications and servers. Error bars show 95% confidence interval.

2.4 Discussion

2.4.1 Get and Set Performance

Figures 5 and 6 partially support the predicted set throughput behaviour. Increasing the replication factor has a clear negative impact on set throughput, but increasing the number of servers does not (outside of the no-replication special case); this implies that the benefit of increased load distribution approximately counteracts the cost of replicating to additional servers. This conclusion is further supported by the increase in throughput seen by increasing the number of servers without replication (which shows only the effects of load distribution).

Get throughput behaves very differently to the prediction, matching set throughput exactly instead of remaining constant. The observed fluctuation is due to a throttling effect evident in the collected data. Table 2 suggests that memaslap limits the throughput of each request type according to the workload parameter. If set requests take longer to be processed (for instance due to additional replications), memaslap is forced to reduce the throughput of get requests as well, to ensure that it only accounts for its allocated percentage of the combined throughput. Different results would likely be observed if the experiments were instead repeated in an open system without the memaslap's read-write limiting factor.

Additional servers were expected to cause a response time decrease for get requests, however Figure 6a shows this not to be the case. Such instability is likely caused by the competition for server time between set and get requests, as evidenced by the increased deviation towards higher replication factors and numbers of servers. Response time for set operations with different replication factors, on the other hand, behaves as expected, reflecting the significant impact of the partially asynchronous set worker design; processing one request until completion before moving on to the next means requests spend longer waiting in the queue. The uniform decrease in response time associated with additional servers is caused by the improved load distribution, since servers have to deal with less requests per second.

2.4.2 Middleware Behaviour

Internal middleware measurements for get requests (Figure 7) suggest that, with three and five servers, replication has no effect on T_{queue} and T_{server}. In the seven-server configuration, replication appears to have a minor effect on the middleware's internal behaviour, though the measurements are not precise enough to draw conclusions. Nevertheless, the trends in Figure 7 are consistent with the response time results shown in Figure 6a, and likely also due to competition between read and write threads for server time.

Set request metrics, on the other hand, provide a very clear picture with obvious trends, as seen in Figure 8. T_{server} is only slightly affected by the number of servers, most likely due to the minor overhead associated with waiting for responses from all servers. T_{queue}, on the other hand, clearly rises as the number of replications is increased. This is consistent with the trend depicted in Figure 6b, and can similarly be justified by the particular design of the read worker, as discussed previously. In addition, T_{queue} improves dramatically as more servers are added. As discussed previously, this is due to the better load distribution enabled by additional servers.

2.4.3 Scalability

An ideal implementation of a middleware such as the system under test should speed up linearly with the number of servers, since ideally it should distribute load perfectly. It should also replicate to any number of hosts with no performance overhead, assuming a perfectly parallel implementation of the write worker and no hardware limitations. The system under test, however, performs quite differently to the ideal case. While it speeds up relatively well as more servers are added (for instance as shown in Figure 7), it is very significantly affected by replication factors greater than one due to the limitations of the write worker (also visible in Figure 7).

3 Effect of Writes

This section investigates the performance impact of increased write workloads for a variety of server and replication configurations.

3.1 Hypothesis

Increasing the proportion of write requests generated by memaslap is expected to initially increase the write request throughput. However, since the middleware only uses a single write thread for each server, additional workload should quickly overwhelm the middleware. This effect should be visible as a sharp set request response time increase, since requests will wait longer in the queue until they are handled. Such behaviour is expected for both full and no replication, though the set request throughput with full replication should be lower than that with no replication. Given additional servers, the middleware should be able to handle larger write workloads before the write workers are saturated, since the number of write workers is equal to the number of servers.

As observed in Section 2, get request throughput appears to be throttled by memaslap according to the workload configuration. Therefore, the throughput of get requests in this section are expected to match the behaviour of set request throughput. In addition, the ratio of get throughput to total throughput should be equal to the percentage of get requests in the workload configuration. Response time for get requests should decrease as the throughput is throttled by memaslap, since less requests per second will arrive at the read queues.

3.2 Experimental Setup

3.2.1 Primary Factors

The experiment explored the effect of varying the number of Memcached server instances, middleware replication factor and memaslap workload over the following ranges:

Number of servers (S)	3, 5, 7
Middleware replication factor	1, S
Workload write proportion	1%, 2.5%, 5%, 7.5%, 10%

3.2.2 Secondary Factors

The following factors were held constant for all experiments:

Number of client machines	2
Total number of clients	320 (160 per machine)
Workload request size	Key 16B, value 128B
Middleware read threads	20 per server
Memaslap sampling period	$10s$
Runs	$5 \times 200s$
Memcached threads	1
Logs	./reports/m2report/logs/section3-logs.tar.gz

All limitations listed in Section 1 regarding Azure virtual machines apply. Client and middleware read thread numbers were once again chosen based on the optimum results from Section 1. Logs (stored in a tarball for compactness) consist of one directory for each run, named according to the following convention: "sec3-rep{replications}-s{servers}-v{workload}-r{run}". Each run directory contains the middleware log ("middleware.log") and two memaslap logs, one for each machine used ("mema{0,1}.log").

3.3 Results

3.3.1 Get Requests

Figure 9 shows how get request throughput reacts with different combinations of workloads, servers and replications. Increasing the proportion of set requests in the workload caused a decrease in get throughput. This reduction occurred across all server configurations and was consistently observed for both single and full replication. Throughput with seven servers and higher workloads was generally slightly higher than that of its neighbouring configurations.

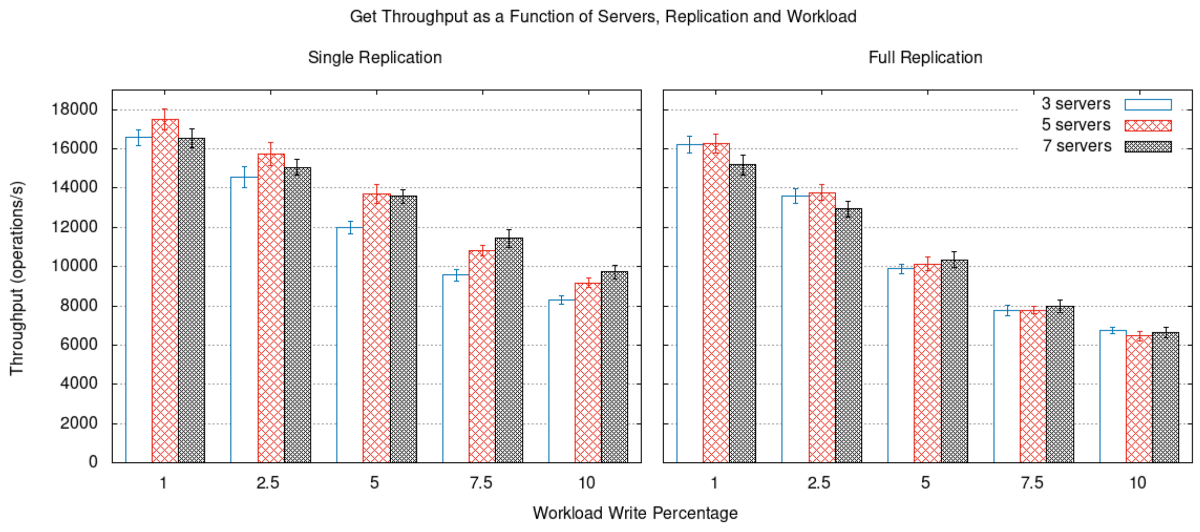


Figure 9: Get request throughput for all combinations of factors. Error bars show 95% confidence interval.

Get response time appears erratic upon first inspection of Figure 10, although trends can be identified. All three server configurations suffered a peak in response time followed by a relatively sharp drop, particularly in the case of three servers. With greater workloads, response time appears to correlate with the number of servers; configurations with less servers also displayed shorter response time in this case. With the smallest workload, all combinations of servers and replications seem to be approximately equivalent.

Complementing Figures 9 and 10, Table 3 shows the ratio of get throughput to combined throughput for all investigated combinations of factors. The ratio of the throughputs produced by memaslap is shown to be precisely equal to $100 - WritePercentage$.

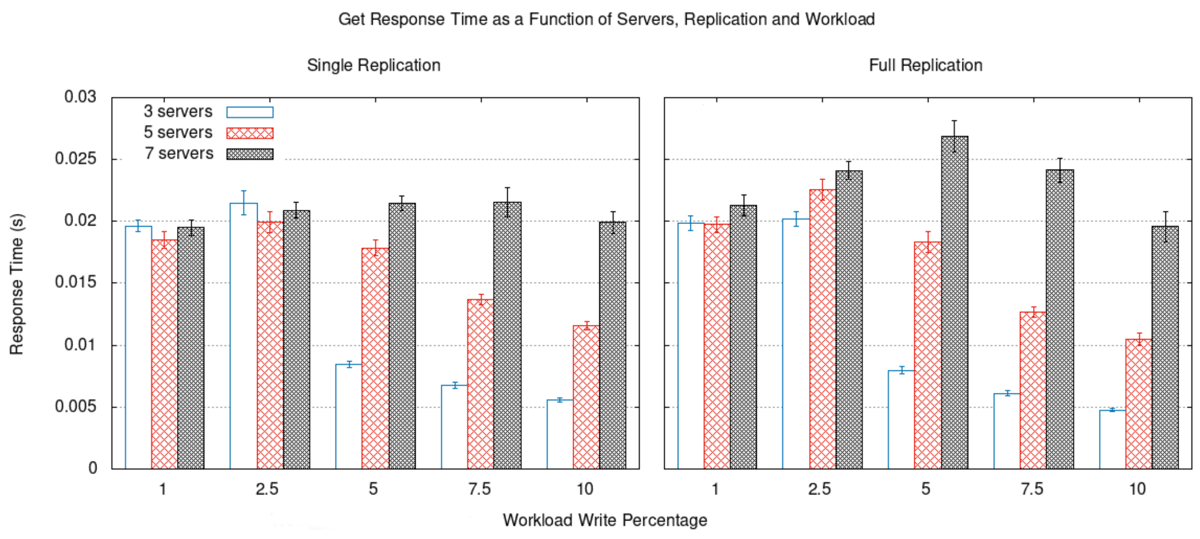


Figure 10: Get request response time for all combinations of factors. Error bars show 95% confidence interval.

Write Percentage	Servers	Replications	Combined Throughput	Get Throughput	Ratio
1	3	single	16,737	16,569	99.00%
1	3	full	16,393	16,229	99.00%
1	5	single	17,680	17,504	99.00%
1	5	full	16,452	16,287	99.00%
1	7	single	16,697	16,530	99.00%
1	7	full	15,339	15,185	99.00%
2.5	3	single	14,916	14,543	97.50%
2.5	3	full	13,946	13,597	97.50%
2.5	5	single	16,133	15,730	97.50%
2.5	5	full	14,133	13,779	97.50%
2.5	7	single	15,443	15,056	97.50%
2.5	7	full	13,262	12,930	97.50%
5	3	single	12,616	11,985	95.00%
5	3	full	10,409	9,888	95.00%
5	5	single	14,425	13,704	95.00%
5	5	full	10,666	10,133	95.00%
5	7	single	14,295	13,580	95.00%
5	7	full	10,888	10,343	95.00%
7.5	3	single	10,348	9,571	92.50%
7.5	3	full	8,378	7,750	92.50%
7.5	5	single	11,671	10,795	92.50%
7.5	5	full	8,410	7,779	92.50%
7.5	7	single	12,357	11,430	92.50%
7.5	7	full	8,624	7,977	92.50%
10	3	single	9,218	8,296	90.00%
10	3	full	7,502	6,752	90.00%
10	5	single	10,183	9,165	90.00%
10	5	full	7,171	6,454	90.00%
10	7	single	10,803	9,723	90.00%
10	7	full	7,359	6,623	90.00%

Table 3: Ratio of get throughput to combined throughput for all factor combinations.

3.3.2 Set Requests

Set requests experienced a constant increase in throughput associated with additional workload, though increasing the replication factor from single to full led to a peaking trend rather than the constant increase observed with no replication. The number of servers had little effect on set throughput, only becoming significant with the higher workload configurations and no replication. This is shown in more detail in Figure 11.

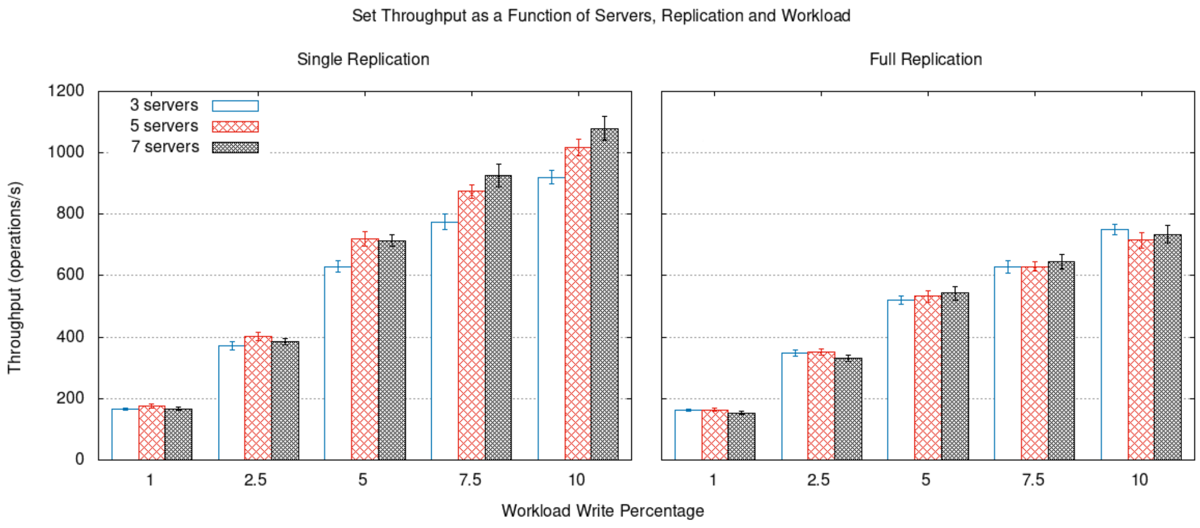


Figure 11: Set request throughput for all combinations of factors. Error bars show 95% confidence interval.

Response time for set requests varied in a complementary manner in relation to get requests. As shown in Figure 12, set response time with three servers peaks at a workload of 5% writes for both replication configurations and slowly decays thereafter. Five- and seven-server configurations show a similar, though delayed, trend, tending to peak at higher workload configurations.

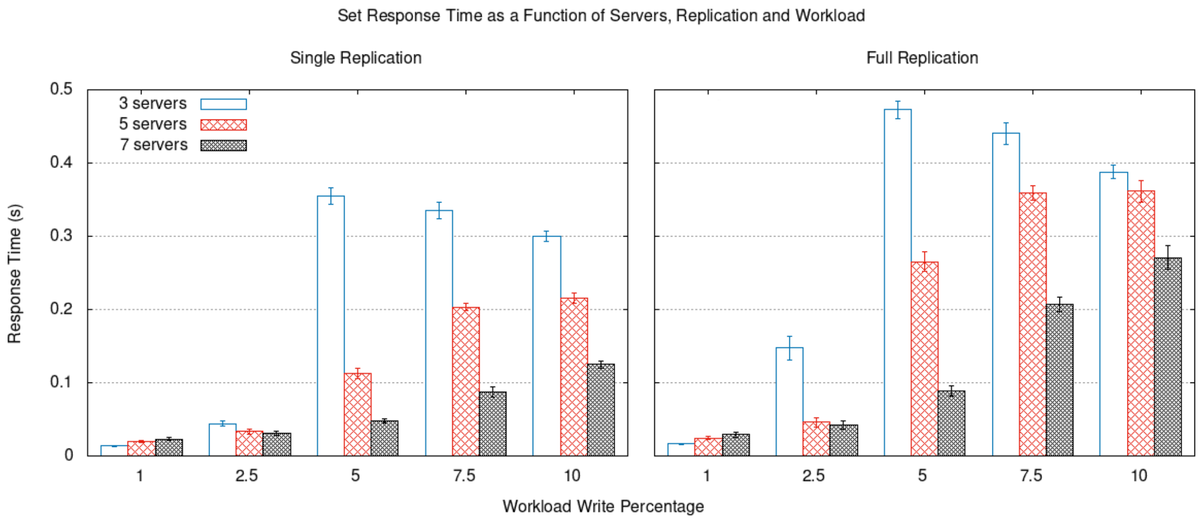


Figure 12: Set request response time for all combinations of factors. Error bars show 95% confidence interval.

3.4 Discussion

The biggest performance impact was observed with three servers, a 5% write workload and full replication. The exact figures are shown in Table 4

This is considered to be the point of greatest performance degradation due to the extreme increase in set request response time and considerable drop in get request throughput, which outweigh the comparatively minor improvements in set throughput and get response time.

	Base Case	Impacted
Get Throughput (op/s)	16569	9888
Get Response Time (s)	0.019	0.008
Set Throughput (op/s)	166	520
Set Response Time (s)	0.014	0.473

Table 4: Performance impact on get and set performance compared with base case (S = 3, 1% workload, no replication).

The unusual behaviour in Figure 12 is caused in part by the middleware's write worker design, but also by the throttling behaviour observed in memaslap in Sections 2.4.1 and 3.3.1. With smaller workloads such as 1% and 2.5% writes, the writer workers are capable of handling the load with no significant spike in response time. At 2.5% with three servers, the first signs of write worker saturation being to appear in the form of increasing set response time with full replication. Conversely, get response times actually drop when the writer threads begin to saturate. This is due to memaslap's throttling effects; as long as set requests must wait in their queue for write workers to become available, memaslap must ensure that the workload ratio is not disrespected by excessive get requests. With less get requests arriving per second, the read threads are able to respond promptly, leading to lower response time and throughput.

Additional servers have the effect of delaying the saturation of the write worker. As discussed in Section 2.4, the benefit of load distribution to additional servers outweighs the overhead of replication; in Figure 12, this effect is visible in the forward shift of the saturation peaks.