Advanced Systems Lab (Fall'16) – Second Milestone

Name: *Iveri Prangishvili*
Legi number: *15-926-140*

Grading

Section	**Points**
1	
2	
3	
Total	

1 Maximum Throughput

In this section I look for the highest throughout of the system with number of Memcached servers fixed to 5, no replication and **read** only workload configuration. I conduct a series of experiments varying number of virtual clients and threads in the thread pool to achieve maximum throughput.

- After each experiment and each repetition, the Middleware and Memcached servers are restarted.
- The first and last 30 seconds of Memaslap data for an experiment is discarded to account for a warm up/cool down phase.
- Throughput (requests per second) is averaged across iterations and aggregated across 3 Memaslap Client Machine data.
- Response time and variance (Micro Seconds) is averaged across iterations and 3 Memaslap Client Machine data.
- Every 100th **get** and **set** request is logged inside the Middleware and the data is appended across iterations.

My hypothesis, for the behavior of the system while varying number of virtual clients and the number of threads in the thread pool is the following.

- As the number of virtual clients is increased I expect an increase in the Throughput (Successfully Completed Requests/Second) and an increase in the Response time (Micro Seconds) as well. The logic behind this assumption is that increasing number of virtual clients, results in an increase of the workload and if a system can handle the workload, it will have a higher throughput and higher response time.
- As the number of threads in the thread pool is increased I expect an increase in the throughput and decrease in the response time till a specific number of threads. The logic behind this assumption is that, increasing the threads in the thread pool will result in the decrease in the **T_queue** - time in Micro Seconds a **get** request spends inside a queue waiting to be processed - consequently decreasing the total time a request spends inside the middleware (**T_mw**) and thus decreasing the response time. However, I expect a stagnation or decrease in the throughput after a certain number of threads in the thread pool for each number of virtual clients. This is due to the fact, that **read** threads inside the thread pool will compete for the requests waiting to be processed inside the associated **read** queue. Also since, there will be more **Read** threads in the middleware, this will result in the CPU context switching overhead - CPU will have to switch from executing one thread to executing another thread more frequently. Therefore, the threads in the Middleware processes will get less CPU time, meaning the system won't be able to utilize the additional threads and will not improve the performance of the system, moreover it can slow it down resulting in decrease in the throughput.

This section is divided into two sub-sections. First sub-section explores the effect of varying virtual clients as the number of threads in the thread pool is fixed. The results from this sub-section will provide an insight to the way system handles an increase in the number of virtual clients. The second sub-section explores the effect of varying the threads in the thread pool, while also varying the number of virtual clients.

1.1 Exploring the effect of Virtual Clients

In the series of experiments described in the Table 1, I fix number of threads in thread pool to 16 and vary the number of virtual clients. The motivation for this setup is to explore the behavior of the system as number of virtual clients increases and identify the point where the middleware reaches the highest throughput.

TABLE 1: EXPERIMENT SUMMARY

Number of Memcached Servers (S)	5
Number of Client Machines	3
Virtual clients / Machine	**32, 48, 64, 80, 96, 112, 128, 144, 160, 196**
Workload	Key 16B, Value 128B, Writes/Reads Ratio: 0/1
Replication (R)	No Replication (R=1)
Number of Threads in thread pool	16
Runtime x repetitions	300s x 10
Total Number of Experiments conducted:	10 * repetition (10)
Log files	Section1_Experiment_Series1

Figure 1, presented below, visualizes the changes in the throughput (Successfully Completed number of Requests/Second) as the number of virtual clients are varied. The estimated standard deviation, calculated across the **10** repetitions for each setup, is shown as error bars on the plot. Based on the estimated standard deviation, I calculate a 95% confidence interval for each number of virtual clients and compare it to the 5% interval around the associated mean. The Table 2, compares the margin of error with 95% confidence to the value of 5% of the mean, for each number of virtual clients. It can be seen from the table that the margin of error with 95% confidence is always smaller then 5% of the mean. This indicates, that I can be at least 95% confident that the throughput (Requests/Second) values I observer, will lie within the 5% margin around the mean.

TABLE 2: COMPARING MARGIN OF ERROR WITH 95% CONFIDENCE TO THE VALUE OF 5% OF THE MEAN BASED ON SECTION1_EXPERIMENT_SERIES1 DATA

	Number of Virtual Clients									
	32	**48**	**64**	**80**	**96**	**112**	**128**	**144**	**160**	**196**
Margin of error with 95% confidence (Req/Sec)	162	251	212	259	200	210	161	274	224	350
5% of the mean (Req/Sec)	505	579	588	601	613	656	670	605	564	539

Based on Figure 1, it can be seen that the throughout is continuously increasing from 32 to 128 virtual clients reaching the maximum throughput of 13400 Requests/Second. After, 128 virtual clients the throughput begins to decrease. This means, that the Middleware with 16 threads in the thread pool can't handle the workload generated by more than 128 virtual clients. The Figure 2, depicts the average response time in Micro seconds versus the number of virtual clients. The curve is mostly linear with a slight increase in the slope from 144 to 160 virtual clients.

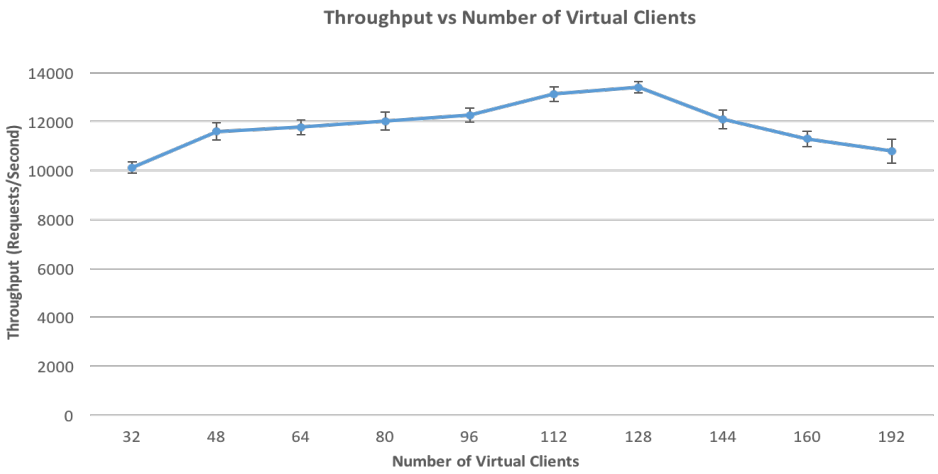


FIGURE 1: THROUGHPUT VS NUMBER OF VIRTUAL CLIENTS BASED ON SECTION1_EXPERIMENT_SERIES1 DATA

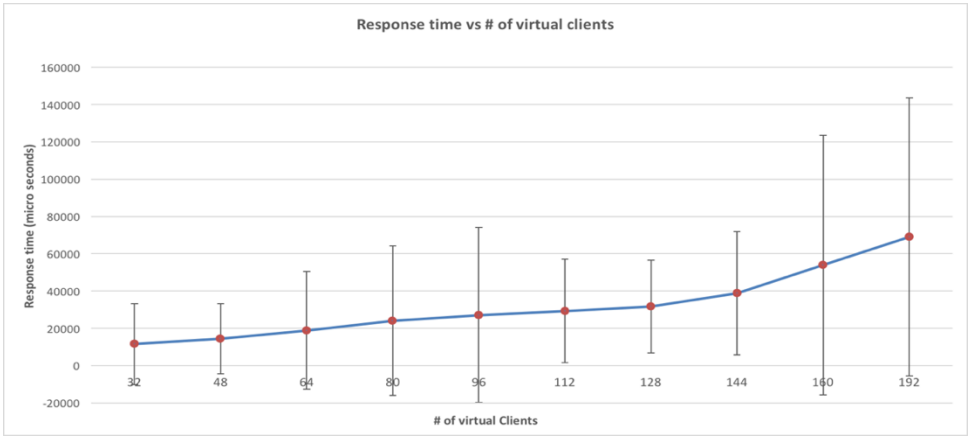


FIGURE 2: AVERAGE RESPONSE TIME VS NUMBER OF VIRTUAL CLIENTS BASED ON SECTION1_EXPERIMENT_SERIES1 DATA

However, this plot isn't informative enough since the standard deviation represented by error bars, don't provide any insight to the behavior of the system. Therefore, I present an empirical Cumulative Distribution Function plot for each of the number of virtual clients, depicted in the Figure 3.
As it can be seen from the Figure 3 plot **32, 48, 64, 80** and **96** virtual clients, all complete more than 90% of the total number of Requests within 32768 (2^{15}) Micro Seconds. For **112** and **128** virtual

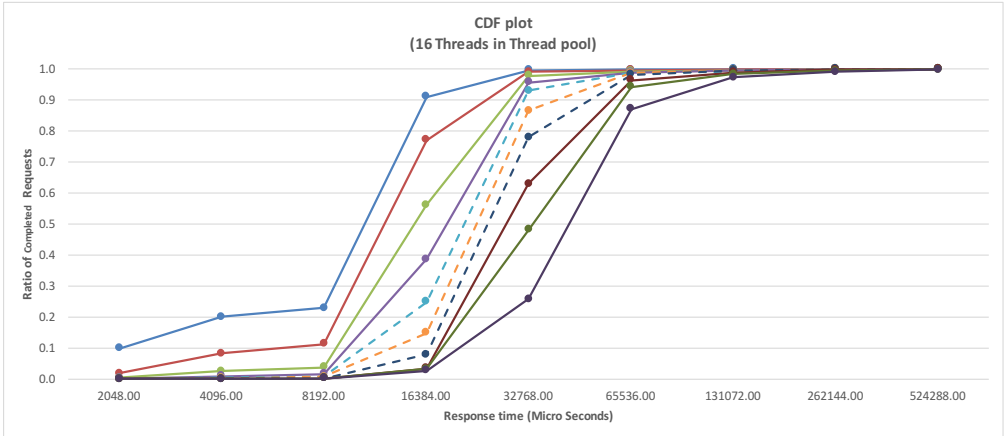


FIGURE 3: CUMULATIVE DISTRIBUTION FUNCTION PLOT BASED ON SECTION1_EXPERIMENT_SERIES1 DATA

clients within 32768 (2^{15}) Micro Seconds the percentage of completed requests drops below 90%, but stays within the 80%. However, the drop in the percentage of request completed after 128 virtual clients for within 32768 (2^{15}) Micro Seconds, keeps decreasing significantly, falling to 25% for 192 virtual clients. Based, on the Throughput and CDF plots from Figure 1 and Figure 3, the *saturation* range is between **96** and **128** virtual clients, while 144 virtual clients generate workload that my system with 16 threads cannot handle anymore. The observed behavior of the system based on the experiments described above is in accordance to the **hypothesis** I made in the beginning of the section, relating to the effect of increasing number of virtual clients.

1.2 Exploring the effect of threads in thread pool

The series of experiments described in Table 1, explored the effect of increasing the number of virtual clients with 16 threads in the thread pool. The series of experiments conducted for this sub-section, explore the effect of different number of threads in the thread pool and find the minimum configuration to reach the maximum throughput.

I introduce the following 6 factors for number of threads in the thread pool: **2, 4, 8, 16, 24, 32** that I will explore through the experiments**.** Varying the number of threads will also affect the number of virtual clients that my system can handle. Based on the experiments in the previous sub-section I know that **16 threads** can handle workload generated by at most **128** virtual clients per Memaslap Machine (in total 3 Memaslap Machines). Therefore, I can conclude that **2, 4,** and **8** threads won't be able to handle workload generated by more than **128** virtual clients. However, I can't make the same conclusion for the **24** and **32** threads, since they may be able to handle workload generated by more than **128** virtual clients. Therefore, I select a following range of number of virtual clients to explore the effect of threads in the thread pool: **32, 64, 96, 112, 128, 144**.

The Tables 3, 4, and 5 summarize the experiments conducted for this section. Notice that I reduced the run time and number of repetitions for each experiment to 3 minutes and 5 iterations. I reduced run time to 3 minutes, because based on the previous series of experiment described in Table 1, 3 minutes is enough time to generate a stable data and running for more than that is futile. As for the decrease in the number of repetitions, from the previous experiment data even with 5 repetitions, 95% confidence interval is still within the interval of 5% around the mean.

TABLE 3: EXPERIMENT SUMMARY

Number of Memcached Servers	5
Number of Client Machines	3
Virtual clients / Machine	**32, 64**
Workload	Key 16B, Value 128B, Writes/Reads Ratio: 0/1
Replication	No Replicate (R=1)
Threads in Thread Pool	**2, 4, 8, 16, 24, 32**
Runtime x repetitions	180s x 5
Total Number of Experiments conducted:	12 * repetitions (5)
Log files	Section1_Experiment_Series2

TABLE 4: EXPERIMENT SUMMARY

Number of Memcached Servers	5
Number of Client Machines	3
Virtual clients / Machine	**96, 112, 128, 144**
Workload	Key 16B, Value 128B, Writes/Reads Ratio: 0/1
Replication	No Replicate (R=1)
Threads in Thread Pool	**2, 4**
Runtime x repetitions	180s x 5

Total Number of Experiments conducted:	8* repetitions (5)
Log files	Section1_Experiment_Series3

TABLE 5: EXPERIMENT SUMMARY

Number of Memcached Servers	5
Number of Client Machines	3
Virtual clients / Machine	**96, 112, 128, 144**
Workload	Key 16B, Value 128B, Writes/Reads Ratio: 0/1
Replication	No Replicate (R=1)
Threads in Thread Pool	**8, 16, 24, 32**
Runtime x repetitions	180s x 5
Total Number of Experiments conducted:	16* repetitions (5)
Log files	Section1_Experiment_Series4

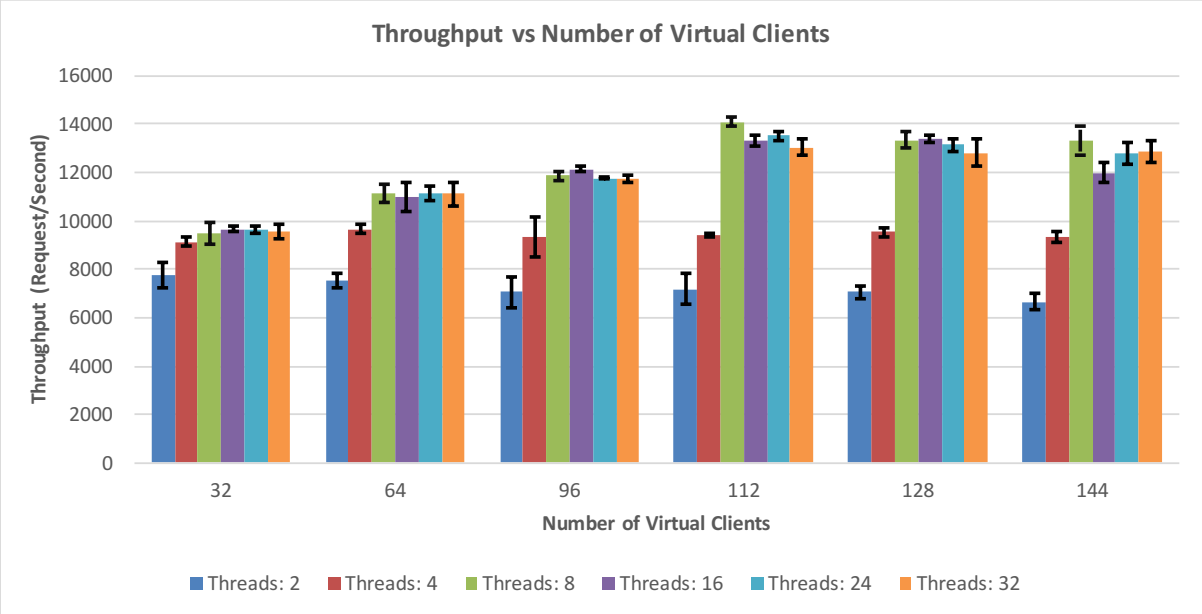


FIGURE 4: THROUGHPUT VS NUMBER OF VIRTUAL CLIENTS BASED ON THE DATA: SECTION1_EXPERIMENT_SERIES2 , SECTION1_EXPERIMENT_SERIES3, SECTION1_EXPERIMENT_SERIES4

As it can be seen from the Figure 4, the **2** and **4** threads have consistently lower throughout (Request/second) compared to the **8, 16, 24** and **32** threads. Furthermore, the number of Completed Requests/Second for **2 threads** decreases when the number of virtual clients is more than **32** and similar behavior is observed for **4 threads** after **64** virtual clients. This means, that **2** and **4** number of threads can't support the workload generated by more than **32** and **64** virtual clients respectively. However, it can be seen that the **8, 16, 24** and **32** threads can handle the workload generated by higher number of virtual clients. It can be noticed that the **8, 16, 24** and **32** threads have same value for **32**, **64**, and **96** virtual clients, however after that they begin to differ. Since, the difference is very small, it can be accounted to the existing variation between the iterations. The standard deviation is represented by the respective error bars on the plot, and if taken into consideration will results in the same number of request/second across the **8, 16, 24** and **32** threads for each number of virtual clients. Hence, increasing the number of threads in the thread pool to more than 8 threads doesn't result in any increase in the throughput. This means that having more than 8 threads in the thread pool causes threads to compete for the CPU resources, thus resulting in no performance improvement. On the

other hand, having less than 8 threads will not fully utilize the CPU resources available in the middleware, thus resulting in less throughput as seen on Figure 4 for **2** and **4** threads. One can also notice, that throughput (Requests/Second) for **8, 16, 24** and **32** threads taking into consideration standard deviations, stop increasing after **112** virtual clients, meaning that the *saturation* range for these range of threads starts from **112** clients. Since, the goal of this section is to select a minimum configuration that achieves the maximum throughout, based on the Figure 4, I choose **8 threads** and **112 virtual clients** reaching 14107 Successfully completed Request/Second. The Figure 5, 6, and 7

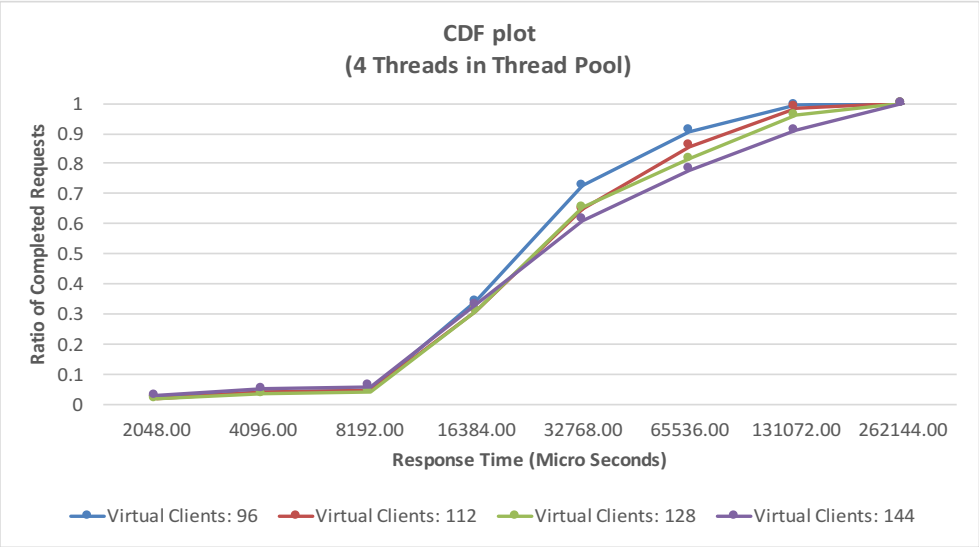


FIGURE 5: CUMULATIVE DISTRIBUTION FUNCTION PLOT BASED ON SECTION1_EXPERIMENT_SERIES3

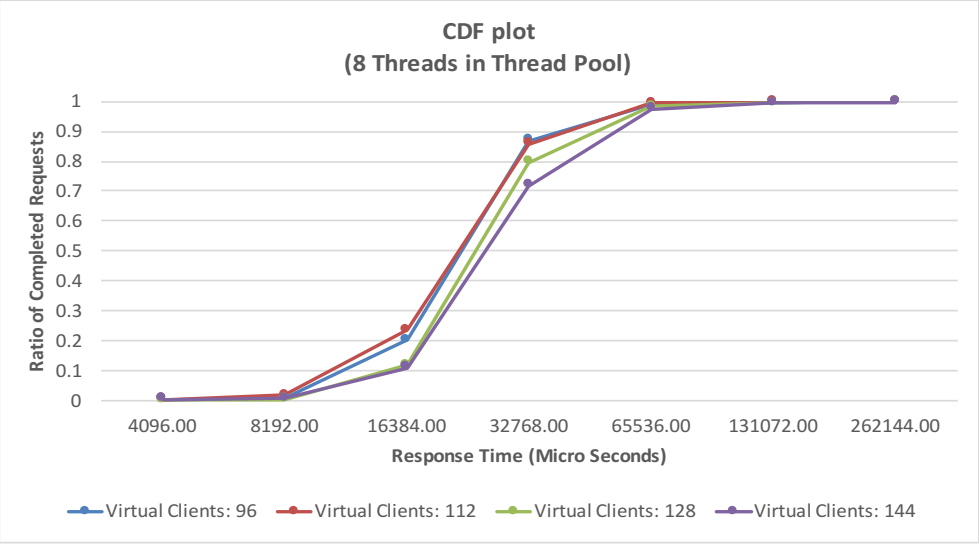


FIGURE 6: CUMULATIVE DISTRIBUTION FUNCTION PLOT BASED ON SECTION1_EXPERIMENT_SERIES4

show a CDF plot for **4, 8,** and **16** threads in the thread pool respectively for **96, 112, 128,** and **144** virtual clients. Comparing the Figure 5 and 6, it can be seen that they have different shapes, and that using **4** threads in the thread pool results in approximately 20% drop in the percentage of request completed within 32768 (2^{15}) Micro Seconds for **96, 112, 128** Virtual Clients. Supporting the argument that **4** threads can't deal with the workload generated by **96, 112, 128,** and **144** virtual clients. Figure 6 and 7, are similar to each other with minor differences in percentage of request completed within each time bucket, meaning that increasing number of threads to more than **8,** doesn't result in any decrease in the response time. The Figure 6, also shows that the lines for the **96** and **112** virtual clients are superimposed, meaning there is no significant increase in the response time, while for **128** and **144** clients the percentage of Request completed drops by approximately 10%

and 20%. Therefore, this plot further supports the choice of **112 virtual clients** for **8** threads, achieving the 14107 throughout (Requests/second).

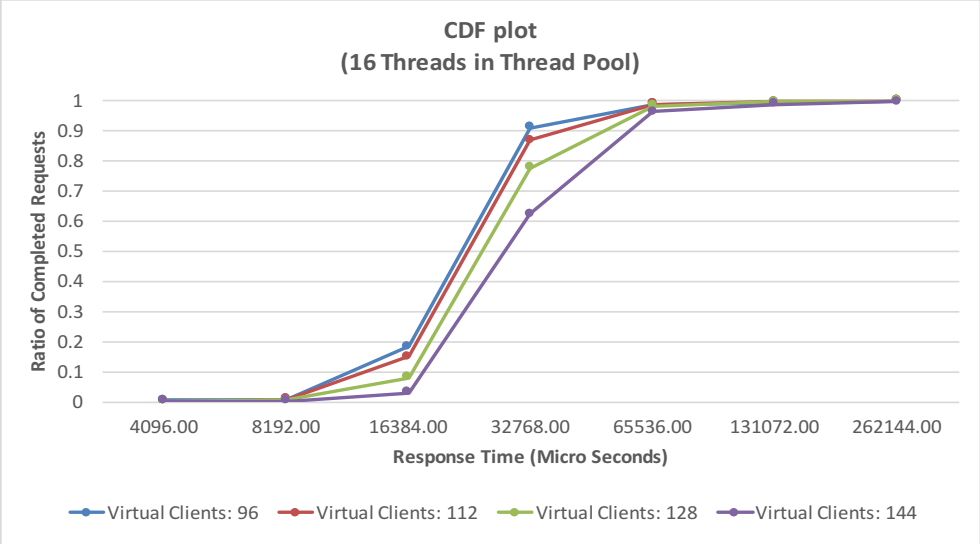


FIGURE 7: CUMULATIVE DISTRIBUTION FUNCTION PLOT BASED ON SECTION1_EXPERIMENT_SERIES4

The Table 6 provides a detailed breakdown of time spent in Middleware for **Read** operation type for selected configuration: **8 threads** and **112** virtual clients.

TABLE 6: MIDDLEWARE INFORMATION FOR READ OPERATIONS BASED ON SECTION1_EXPERIMENT_SERIES4 DATA

Percentile	**Read** Operations		
	T_queue (Micro Seconds)	**T_server** (Micro Seconds)	**T_mw** (Micro Seconds)
25th	1054	439	10149
50th	4927	634	14072
75th	11290	1074	20652
95th	22356	6718	32672

In conclusion, the behavior of the system based on the experimental results agree with the expectations listed in the **hypothesis.**

- For a fixed number of threads (**8** threads), as the number of Virtual Clients increases so does throughput and response time, till **112** virtual clients. After that the TPS stays around the same value and the response time increases, meaning it is inside the *saturation* range.
- As the thread number is increased from **2** to **8,** the throughput (Completed Requests/Second) increases as well. However, using more than **8 threads** in the thread pool doesn't increase the throughput (Requests/Second) and the response time is not affected significantly either. This means, that using more than **8 threads** results in threads fighting for CPU resources, hence no increase in the throughput.

2 Effect of Replications

To explore the effects of replication on the middleware, I conduct series of experiments. Specifically, to investigate the behavior of the middleware for a 5%-write workload with 3, 5, and 7 Memcached servers and three replication factors. **Each experiment in this section was run for 3 minutes and repeated 5 times. After each experiment and each repetition, the Middleware and Memcached servers are restarted.** The first and last 30 seconds of Memaslap data for an experiment is discarded to account for a warm up/cool down phase. Throughput (operations per second) is averaged across 5 iterations and aggregated across 3 Memaslap Client Machine data. Response time and variance (Micro Seconds) is averaged across 5 iterations and 3 Memaslap Client Machine data. The Middleware is logging every 100th get and set request. The final middleware logging data is the produced by appending logging data across 5 iterations.

TABLE 7: EXPERIMENT SUMMARY

Number of Memcached Servers (S)	**3, 5, 7**
Number of Client Machines	3
Virtual clients / Machine	96
Workload	Key 16B, Value 128B, Writes/Reads Ratio: 0.05/0.95
Replication (R)	**No Replication (R=1),** **Half Replication (R=S/2),** **Full Replication (R=S)**
Number of Threads in Thread Pool	8
Runtime x repetitions	180s x 5
Total Number of Experiments conducted:	9 * repetitions (5)
Log files	Section2_Experiment_Series1

The number of virtual clients used for each Memaslap machine in all the experiments described in the Table 7 is 96, which is 85% of 112 virtual clients that reaches maximum throughout (investigated in the section 1). I chose this number of virtual clients to account for the unforeseen consequences that may be caused by the change in the number of Memcached Servers.
My **hypothesis** for the behavior of my system based on the experiments described in the Table 7 is the following. First of all, **get** and **set** requests will be impacted in different ways by different setups.

- For **read** operations I expect the three replication factors: R=1, R=half, R=all to have **no impact**, since replication only affects the way **write** operations are handled inside the Middleware. However, **get** requests will be affected by different number of Memcached Servers. As number of Memcached servers increases the **T_queue** time for **get** requests will decrease and **T_server** time won't be affected, resulting in the decrease of **T_mw –** total time spent in the middleware – for read requests. The logic behind this is that, since the workload is distributed uniformly across the Memcached servers (because of Consistent Hashing), increasing the number of servers will result in less workload for each of them. Consequently, there will be less amount of request pushed to an associated **read** queue. This will result in decease of time a request spends inside the queue, since there are less number of requests to be processed by threads in the thread pool. This behavior is expected, because of the fact that **read** operation is implemented in synchronous manner, where the time a request spends inside the queue, waiting to be processed, relies on the number of threads in thread pool. I expect the throughput of the system to increase up to a certain point. This is due to the fact that 95% of the requests are **read** operations, and as already mentioned increasing the number of servers will decrease the **T_queue** time, which is the most expensive operation for **read** request.

- For the different number of Memcached servers I expect no impact on **write** operations. Even though, having more Memcached servers would result in less **write** requests being added to an associated **set** queue, this won't impact the time a **write** request spends inside a queue, since **set** operation is implemented in an asynchronous manner. For **write** requests I expect replication factors to have an effect, as the replication level increases, I expect the **T_server** time (Most expensive operation for **Set** requests) to increase, being minimum at No Replication (R=1) and reaching maximum at full replication (R=Full). On the other hand, **T_queue** time will remain the same. The logic behind the expected increase in the **T_server** time is that, during replication **T_server** measures a time spent waiting for the specific number of replications to complete. Hence, across replication factors I expect the throughout to decrease and response time to increase, since the most expensive operation for Set request the **T_server** time will increase.
- In an ideal system, I believe an increase in the number of servers would lead to the increase in the throughput and decrease in the response time, since more servers would result in better load distribution among Memcached Servers.

2.1 Throughput

The Figure 8 presents the throughput (Successfully Completed Requests per second) as the replication factor is varied. Three boxes on the plot depict regions with different number of Memcached servers used. The estimated standard deviation, calculated across the **5** repetitions for each setup, is shown as error bars on the plot. Based on the estimated standard deviation, I calculate a 95% confidence interval for each number of Memcached servers and three Replication factors and compare it to the 5% interval around the associated mean. The Table 8 compares the margin of error with 95% confidence to the value of 5% of the mean. It can be seen from the Table 8, that the margin of error with 95% confidence is always smaller then 5% of the mean.

TABLE 8: COMPARING MARGIN OF ERROR WITH 95% CONFIDENCE TO THE VALUE OF 5% OF THE MEAN BASED ON SECTION2_EXPERIMENT_SERIES1 DATA

	Memcached Server: 3			**Memcached Server: 5**			**Memcached Server: 7**		
	R=1	**R=2**	**R=3**	**R=1**	**R=3**	**R=5**	**R=1**	**R=4**	**R=7**
Margin of error with 95% confidence (Req/Sec)	368	453	364	578	333	288	86	431	299
5% of the mean (Req/Sec)	572	555	552	617	607	608	597	585	579

It can be seen from the Figure 8 that, for a fixed number of Memcached servers the throughput decreases as replication number is increased, however the decrease in the number of Requests/Second is not significant (at most 300 Requests/Second). This plot supports the **hypothesis,** that the throughput (Requests/Second) should decrease as the replication number is increased for a fixed number of Memcached servers. Also to note, the throughput increases when increasing the number of Memcached servers from 3 to 5, which is in accordance with logic, that as I add more servers

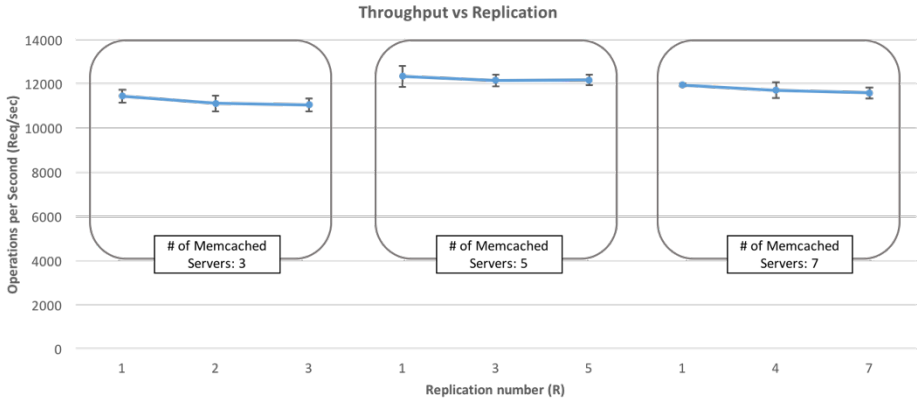


FIGURE 8: THROUGHPUT VS REPLICATION BASED ON SECTION2_EXPERIMENT_SERIES1 DATA

(resources) the throughout should increase. However, when increasing the number of Memcached Servers from 5 to 7, the throughput stays around the same value (Requests/Second), if taken standard deviation into consideration. I suspect this is due to the limitation in the availability of the CPU resources in the middleware, since as number of servers increases so does the number of threads in the system. I will explore this assumption in the next sub-section, by analyzing the Memaslap CDF plot based on *LogDist* information.

2.2 Response time

Figure 9 presents the CDF plots separately for **Read** (left) and **Write** (right) operations based on the Memaslap data from Section2_Experiment_Series1. The reason for the separate plots is to get a better idea how each operation behaves as system configuration is changed. As seen from CDF plot for

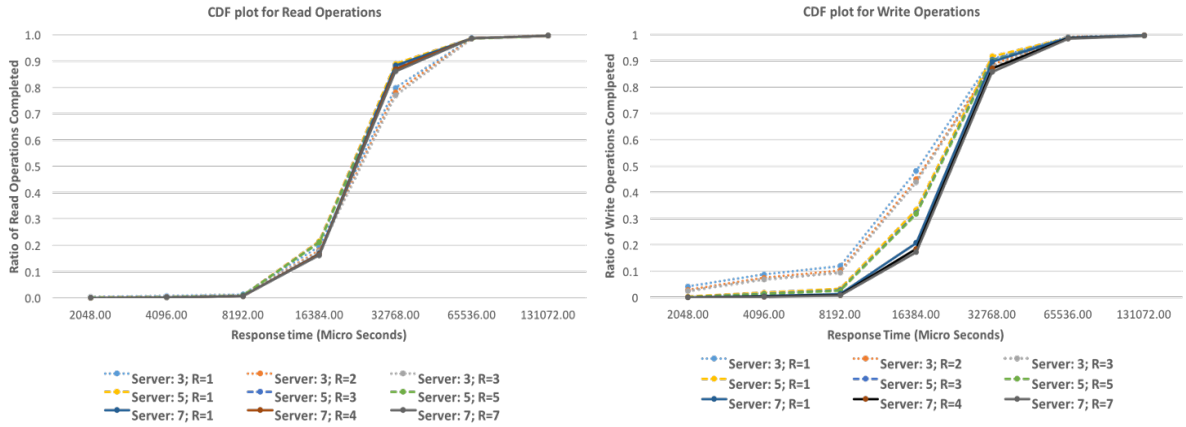
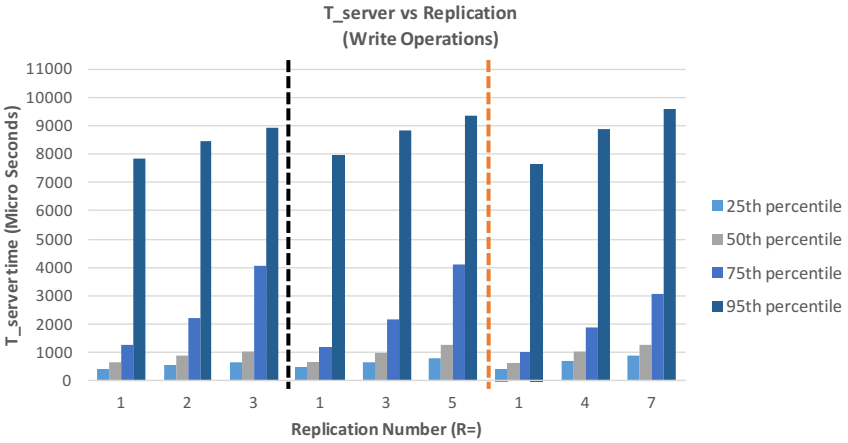


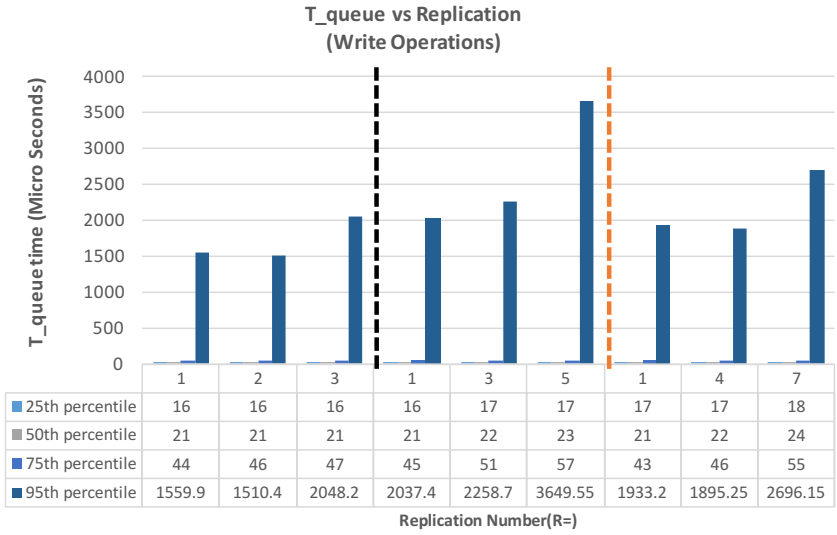
FIGURE 9: CDF PLOT FOR READ OPERATION (LEFT), CDF PLOT FOR WRITE OPERATION (RIGHT) BASED ON SECTION2_EXPERIMENT_SERIES1

Read operations (Figure 9 left), the lines for three replication factors for each of the Memcached Servers are superimposed, meaning that replication has no effect on the **get** requests. However, it is noticeable that the three lines, corresponding to three replication factors, for the 3 Memcached servers complete 10% less operations (**Get** requests) within 32768 Micro Seconds then does the setup corresponding with the 5 and 7 Memcached servers. This point is partially in accordance with the

hypothesis, as the number of Memcached servers increases from 3 to 5 the response time decreases. However, as the number of Memcached servers are increased from 5 to 7, there is no change in the corresponding curves. This can be due to the fact that, the resources – total number of **Read** threads (5*8) - allocated by using 5 Memcached Servers is enough to cope with the current workload and introducing any more resources (Increasing Total number of Threads), in terms of increasing the Memcached servers to 7, will introduce a CPU context switching overhead. An overhead introduced in terms of maintaining and managing more **Get** and **Set** threads is balanced by a reduced **T_queue** time for **read** operations, resulting in no significant change in the **Read** operation response time.
The CDF plot for the **Write** requests (Figure 9 right), shows no evident effect of replications. Since, the lines representing three replication factors for each of the number of Memcached servers are nearly superimposed on each other. Also, there is an unexpected variation in the percentage of **Write** operations completed within 16384 Micro Seconds between the different number of Memcached servers. Since, Memaslap data doesn't provide buckets between 16384 and 32768 Micro Seconds, this plot is not enough to explain this behavior. Therefore, to have a comprehensive understanding of the way different setups in these experiments affect **Read** and **Write** operations, I present a series of plots based on the Middleware Logs (Figure 10, 11). These plots depict **T_server** and **T_queue** time in Micro Seconds for **Read** and **Write** Operations across different number of replications and number of Memcached Servers. Each plot is split into three parts by the black and red vertical dotted lines representing the setup of using 3, 5, and 7 Memcached Servers respectively (Region till black dotted line – 3 Memcached Servers, region between black and red dotted line– 5 Memcached Server, region after red dotted line – 7 Memcached Servers).

Let's take a look at the Middleware log plots for the **Set** requests (Figure 10), and see how it is affected by different configurations. The plot for the **T_server** (Micro Seconds) vs Replication (Figure 10), depicts an increase (for all percentiles) in the **T_server** time as the number of replication is increased for 3, 5, and 7 Memcached Servers. The fact that **T_server** increases for 25th, 50th, 75th and 95th percentile, means that replication has an effect on the **Set** requests, making the operation more expensive, this behavior is expected and is mentioned in the **hypothesis.** The increase in



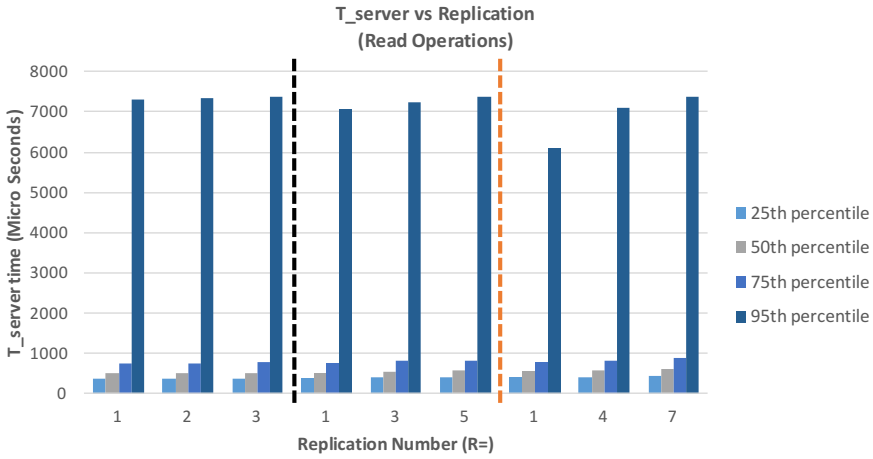


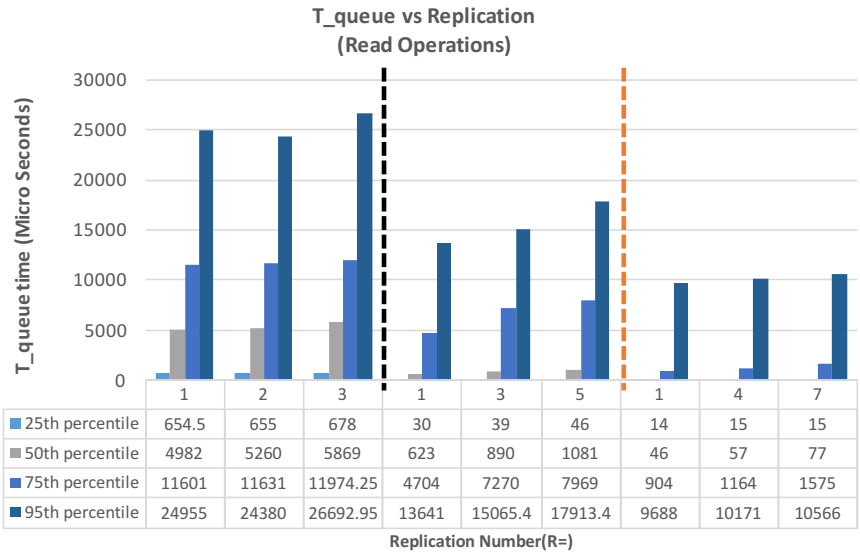
	1	2	3	1	3	5	1	4	7
25th percentile	16	16	16	16	17	17	17	17	18
50th percentile	21	21	21	21	22	23	21	22	24
75th percentile	44	46	47	45	51	57	43	46	55
95th percentile	1559.9	1510.4	2048.2	2037.4	2258.7	3649.55	1933.2	1895.25	2696.15

FIGURE 10: T_SERVER VS REPLICATION NUMBER AND T_QUEUE VS REPLICATION NUMBER PLOTS FOR WRITE OPERATIONS BASED ON SECTION2_EXPERIMENT_SERIES1

T_server across replications can be due to the fact that, during replication **Write** thread has to wait until it receives all responses from replicated servers, and only then can it send back the response to Memaslap. When a request is completely replicated, a **Write** thread in addition to sending the response back to Memaslap Client also has to perform an operation to update the mapping of Memcached Responses to according requests, thus slowing down the **Write** thread**.** the Figure 10 presents the **T_queue** time vs Replication number, as expected since **Write** thread is asynchronous the requests don't spend a lot of time inside the queue and thus the 25th, 50th, and 75th percentiles have a small value of less than 100 Micro seconds and stay approximately the same across the replications, meaning the number of replications has no effect on **set** requests' **T_queue** time. However, the **T_queue** time for outliers presented by the 95th percentile increases across replication, but doesn't follow a linear trend. The biggest increases in the **T_queue** time are noticeable when there is a full replication case. The reason for the increase in the **T_queue** time for outliers can be due to the fact, during replication **Write** thread has to parse multiple responses sequentially blocking the thread for a brief moment, which might **Write** thread to pull new requests from the associated **write** queue,

resulting in the anomaly described by 95th percentile. In conclusion, since **T_mw** time is an aggregation of **T_server** and **T_queue** time, the total time for the **Set** requests is dominated by the **T_server** time, while **T_queue** time constitutes a small portion. Meaning that for **Set** Requests most expensive operation is **T_server,** and this claim is also supported by the **hypothesis**.
The Middleware Log plots for **Read** requests, are depicted on the Figure 11. The plot for **T_server** vs Replication, shows no effect on **T_server** time for 25th, 50th, and 75th percentiles, as the





	1	2	3	1	3	5	1	4	7
25th percentile	654.5	655	678	30	39	46	14	15	15
50th percentile	4982	5260	5869	623	890	1081	46	57	77
75th percentile	11601	11631	11974.25	4704	7270	7969	904	1164	1575
95th percentile	24955	24380	26692.95	13641	15065.4	17913.4	9688	10171	10566

FIGURE 11: T_SERVER VS REPLICATION NUMBER AND T_QUEUE VS REPLICATION NUMBER PLOTS FOR READ OPERATIONS BASED ON SECTION2_EXPERIMENT_SERIES1

number of replication is increased for 3, 5, and 7 Memcached Servers, as described in the **hypothesis.** The **T_server** time for 25th, 50th, and 75th percentiles is not affected by the change in the number of Memcached Servers eithers, which is also expected since **T_server** for **Get** requests is a time Memcached Server takes to process the **Read** request and send back the Response. The plot for **T_queue** vs Replication, shows that as number of Memcached Servers increases the **T_queue** time for across all percentiles decreases, which is an expected behavior described in the **hypothesis.** Since, requests are distributed uniformly among Memcached Servers, each one will receive less amount of **Read** request, resulting in less requests in each **Read** queue and thus decrease in **T_queue** time. However, across replications it can be seen that for the case of 3 Memcached servers 50th percentile time increases, for the case of 5 and 7 Memcached Servers 75th percentile time increases. This behavior was not expected in the **hypothesis,** but the reason may be due to the fact, during replication Asynchronous **Write** threads are more expensive and hence more CPU time is spent on them. Thus

taking CPU time from **Read** threads, consequently resulting in **get** request spending more time inside the associated **read** queue waiting to be processed. In conclusion, the behavior described by the Figure 11 plots are in accordance with the **hypothesis. T_queue** is the most expensive operation in the middleware for the **Get** requests and it decreases significantly as the number of Memcached servers is increased. The interesting thing to note is that the **T_queue** time decrease significantly from 5 to 7 Memcached Servers, however there is no gain in the throughput as seen in Figure 8. This can be explained, by the fact that this gain is offset by the limitation in the availability of the CPU resources in the middleware.

Next I compare the scalability of my system to that of an ideal implementation. I assume that, in an ideal implementation of my Middleware, different threads don't compete for the CPU time and thus adding more resources, such as increasing the number of Memcached Servers, which consequently increases the total number of **Read** and **Write** threads, would result in the increase in the performance (Increase in the throughput and decrease in the Response time).
Figure 12, depicts the comparison of the throughput of my Middleware system for three replication factors: No replication, Half replication and Full replication and for 3, 5, and 7 Memcached servers to that of an ideal implementation. As it can be seen in the ideal implantation (bars on the right) I expect as the number of Memcached servers is increased the throughput to also increase for all replication factors. However, that is not the case for my implementation of the middleware, where the throughout increases from 3 to 5 Memcached Servers for all replication factors, but doesn't increase

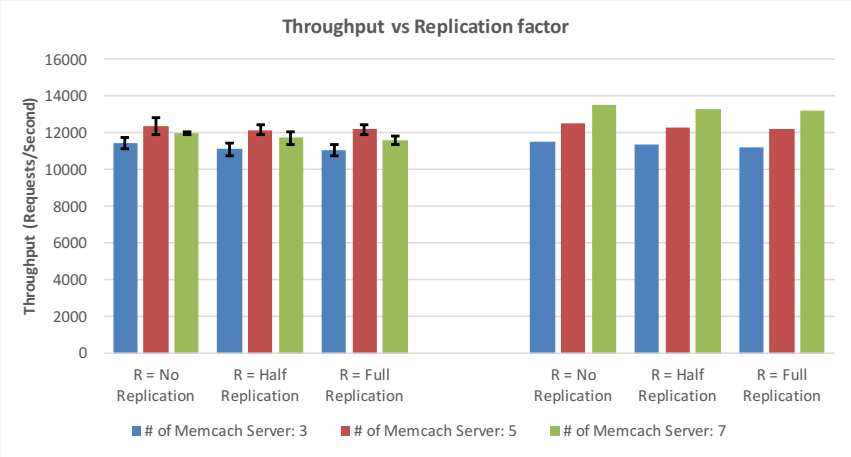


FIGURE 12: THROUGHPUT VS REPLICATION NUMBER; MY MIDDLEWARE (LEFT) BASED ON SECTION2_EXPERIMENT_SERIES1 DATA, IDEAL IMPLEMENTATION(RIGHT)

from 5 to 7 Memcached servers. The table 9, shows a calculated speedup for 3, 5, and 7 Memcached servers based on the 3 Memcached server (baseline) for three replication factors. I am using the average response time in Micro Seconds from the Memaslap data, for each of the setup described above. In an ideal implementation speed up should be linear and more than 1 for each of the replication factors, as the number of Memcached servers is increased. However, that is not the case for my Middleware implementation, where speed up as number of Memcached Servers is increased is not linear. It can be seen that for No Replication as number of Memcached servers is increased the speed up is more than 1, however it is not a linear trend. For Half Replication and Full Replication as number of Memcached Servers increases the speed up is not consistently more than 1.

TABLE 9: SPEED UP BASED ON SECTION2_EXPERIMENT_SERIES1 DATA

	# Memcached Servers: 3	# Memcached Servers: 5	# Memcached Servers: 7
No Replication	1	1.028599621	1.049587627
Half Replication	1	0.96644976	1.064547225
Full Replication	1	1.084853892	0.973198298

In Conclusion,

- **Get** and **Set** requests are impacted in different ways by different setups.
- **T_server** operation is the most expensive one for the **Write** requests. As replication number increases the **T_server** time (in Micro Seconds) for **Write** operation increases, while for **Read** operation it stays the same (has no impact).
- **T_queue** operation is the most expensive one for the **Read** requests. As number of Memcached servers increase the **T_queue** time for **Read** operation decreases, while for **Write** operation it stays the same (has no impact).
- In an ideal implementation adding more Memcached Servers should result in the increase in the throughput. However, that is not the case for my system when increasing Memcached servers from 5 to 7, doesn't result in the increase in the throughout, due to the limitation in the CPU resources in the Middleware.

3 Effect of Writes

In this section I conduct a series of experiments to investigate the changes in throughput and response time as the percentage of write operations are varied. I use a combination of 3 to 7 Memcached servers and carry out experiments for two replication factors: R = 1, R = all. The parameter description of experiments conducted in this section is listed in **Table 10**. **Each experiment in this section was run for 3 minutes and repeated 5 times. After each experiment and each repetition, the middleware and Memcached servers are restarted.** The first and last 30 seconds of Memaslap data for an experiment is discarded to account for a warm up/cool down phase.

TABLE 10: EXPERIMENT SUMMARY

Number of Memcached Servers (S)	**3, 5, 7**
Number of Client Machines	3
Virtual clients / Machine	96
Workload	Key 16B, Value 128B, **Writes/Reads Ratio:** **0.01/0.99,** **0.05/0.95,** **0.10/0.90**
Replication (R)	**No Replication (R=1),** **Full Replication (R=S)**
Number of Threads in Thread Pool	8
Runtime x repetitions	180s x 5
Total Number of Experiments conducted:	18 * repetitions (5)
Log files	Section3_Experiment_Series1

My **hypothesis** for the behavior of my system based on the experimental setup described in the Table 10, is the following.

- For a fixed number of Memcached servers and no replication (R=1), as the percentage of **write** operations is increased, I expect to see an increase in the **T_server** and **T_queue** time for the **Set** requests. The logic behind this as that, as the percentage of **write** operations is increased, each **Set** thread will get more requests in the associated **set** queue. **Write** thread is implemented in asynchronous manner, and it continuously pulls requests from associated **set** queue and sends it to the Memcached server without waiting for the response. **Write** thread uses one socket channel for communication to each Memcached server, hence it can receive multiple responses for different requests, that it has sent, at once. Receiving and processing multiple response at once for different requests will result in the increase of the **T_server**

time. Since, before reading a response for a request from Memcached Server, the system sends another request and then reads responses for several requests at once, thus increasing the **T_server** time. Furthermore, I process multiple responses in a sequential manner in the for loop. Thus, during the multiple response processing, the **Write** thread is busy and doesn't pull from the queue. As the percentage of **Write** requests increase, so will the amount of individual responses that is received at once in a ByteBuffer, resulting in the increase of **T_queue** time as well. However, for most of the request it is expected the **T_queue** time to stay the same, and it is expected to increase only for small percentage of the requests.

- For the case, of the Full replication I expect the same behavior as with No replication case described above, but with more significant effect. Since, during full replication **Write** thread maintains a socket channel connection to each of the Memcached Servers and processes the requests from each one of them. Furthermore, sending the response back to Memaslap only after receiving the response from all Memcached Servers, thus increasing the **T_server** time.
- Based, on the description above, as the percentage of write operations increase I expect a decrease in the throughput and an increase in the response time. I expect the biggest impact on performance to happen for the case of **7 Memcached servers**, Full Replication and 10% write operations. Since, for that setup the increase in the **T_server** and **T_queue** time will be most significant, resulting in the increase of the response time and decrease in the throughput.

3.1 Throughput

The Figure 13, depicts the throughput vs number of Memcached Servers plot for **No** Replication (R=1) case with three write percentage factors: 1%, 5%, and 10%. As it can be seen from the Figure, for each number of Memcached servers the three setups for different **write** percentages, all achieve around the same throughout, if the standard deviation - represented by the black error bars – is taken into consideration. However, the throughput for the three write percentage factors are still different. For the case of 3 Memcached Servers as percentage of write operations increases so does the throughout. Based on the analysis from the **Section 2**, and results from **Section 1,** the setup with 5 Memcached Servers reaches higher throughput. For 3 Memcached Servers, there is not enough **Read** threads to deal with the workload generated by the **96** virtual clients (Per Memaslap Machine), thus

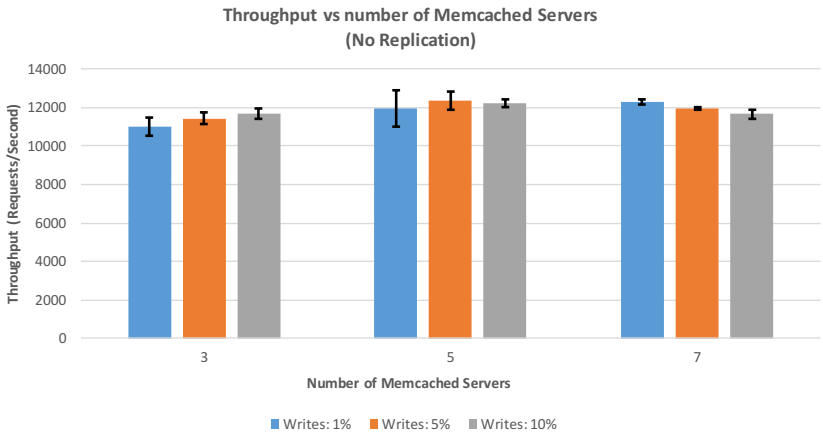


FIGURE 13: THROUGHPUT VS NUMBER OF MEMCACHED SERVERS FOR NO REPLICATION BASED ON SECTION3_EXPERIMENT_SERIES1 DATA

Read operation is more expensive **Write** operation (since it is Asynchronous). As the percentage of **Write** operations is increased, percentage of **Read** operations decreases resulting in the higher throughput. However, that is not the case for 5 and 7 Memcached Servers, where it can be observed that as percentage of **Write** operations is increase the throughput decreases, which is in accordance to the **hypothesis** made earlier in this section. The Figure 14, depicts the same plot but for the case of full replication. As it can be seen the setup with higher **write** percentage achieves lower throughput (Requests/Second). This behavior is noticeable across all number of Memcached server and it is

expected, since during the full replication the **Write** operation is more expensive, due to the overhead introduced inside each of the **Set** threads in terms of maintaining additional socket channels to several

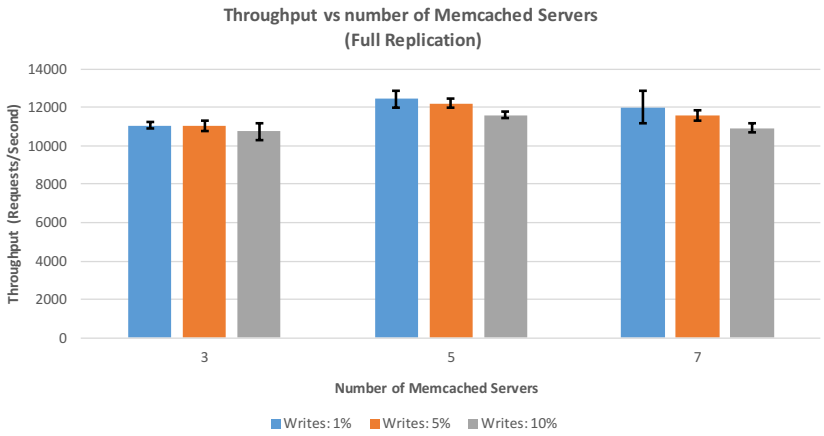


FIGURE 14: THROUGHPUT VS NUMBER OF MEMCACHED SERVERS FOR FULL REPLICATION BASED ON SECTION3_EXPERIMENT_SERIES1 DATA

Memcached servers for replication, parsing multiple responses from each Memcached server and waiting for the response from all Memcached Servers. To investigate the effect of writes further and in more details in the next section I investigate the Memaslap response times and the Middleware logs.

3.2 Response Time

To investigate the effect of the **Writes** on my system, I look at the Cumulative Distribution Function plots from Memaslap data (based on Section3_Experiment_Series1 data) for **Set** requests with three factors for percentage of write requests (1%, 5%, 10%), for different number of Memcached Servers and for the case of No replication and Full replication. I don't examine the CDF plot for the **Get** request in the report, since varying percentage of Write operations and varying replication factors won't have an effect on the **Read** operations. Moreover, I checked the CDF plot for **Get** requests and the curves for different write percentage overlapped, meaning there is no effect on it. For the **Set** request, I plotted the total of 6 graphs for each setup (number of Memcached Servers and Replication factor), and the results were similar across different setups. The points of interest in this graphs are the percentage of **Set** requests completed within 16384 Micro Seconds and within 32768 Micro Seconds. For the sake of saving the space instead of plotting these graphs, I make a table showing the percentage of **Set** requests completed within 16384 and 32768 Micro Seconds for the setups mentioned above, however the plots are available on gitlab, the short name of file is: supplementary_plots. The Table 11, depicts the **Percentage of Successful Set Requests Completed** for the case of No replication.

TABLE 11: PERCENTAGE OF SUCCESSFULLY COMPLETED SET REQUESTS FOR WITHIN SPECIFIC TIME FOR NO REPLICATION, BASED ON SECTION3_EXPERIMENT_SERIES1 DATA

	Within 16384 Micro Seconds			Within 32768 Micro Seconds		
# Memcached Servers	1% Write	5% Write	10% Write	1% Write	5% Write	10% Write
3	56%	48%	43%	93%	91%	90%
5	43%	33%	26%	92%	92%	90%
7	28%	21%	16%	91%	90%	87%

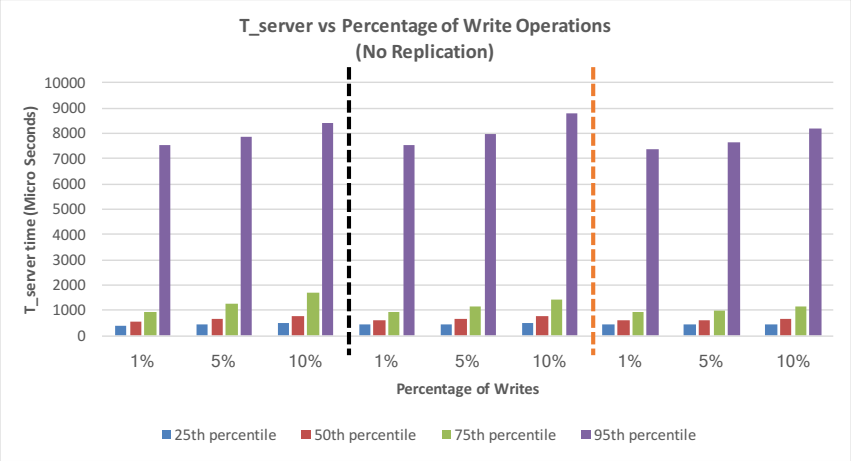
As it can be seen from the table 11, within 16384 Micro Seconds, for each number of Memcached Servers there is an evident decrease in the percentage of **Set** Requests Completed as the setup for the percentage of Write operations is increased from 1% to 10%. Within 32768 Micro Seconds for each number of Memcached servers the decrease in the percentage of **Set** Requests completed as the setup for the percentage of Write operations is increased from 1% to 10% is very small (at most 4% for the case of 7 Memcached Servers resulting in the decrease by the factor of 91%/87% = 1.045). Based on this table 11, for the case of No replication, increasing the percentage of Write requests results in the increase in the response time, since the percentage of completed **Set** requests decreases for within a fixed time in Micro Seconds. This behavior is expected and the reason behind it is described in the **hypothesis** in the beginning of this section. The Table 12, presents the same setup, but for the case of the Full Replication.

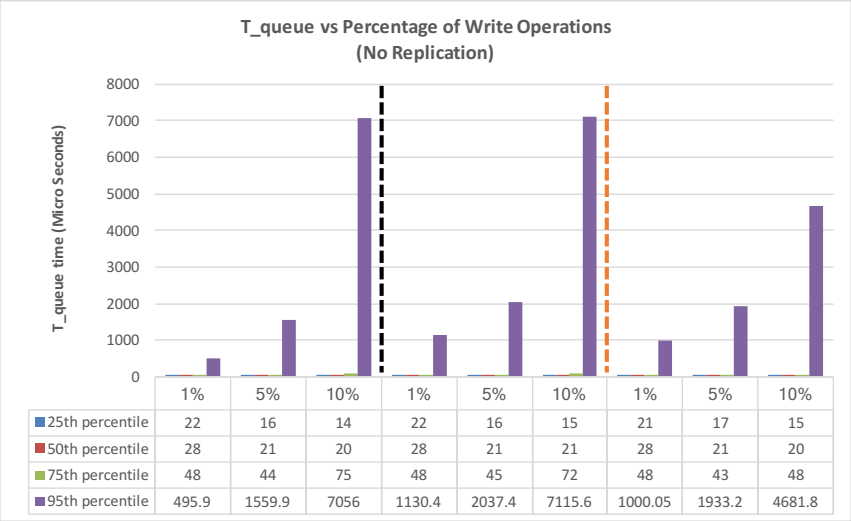
TABLE 12:PERCENTAGE OF SUCCESSFULLY COMPLETED SET REQUESTS FOR WITHIN SPECIFIC TIME FOR FULL REPLICATION BASED ON SECTION3_EXPERIMENT_SERIES1 DATA

	Within 16384 Micro Seconds			Within 32768 Micro Seconds		
# Memcached Servers	1% Write	5% Write	10% Write	1% Write	5% Write	10% Write
3	51%	44%	39%	92%	88%	85%
5	40%	32%	24%	92%	90%	86%
7	29%	17%	13%	90%	86%	81%

As it can be seen from the table 12, within 32768 Micro Seconds, for each number of Memcached Servers the decrease in the percentage of **Set** Requests Completed as the setup for the percentage of Write operations is increased from 1% to 10% is significant compared to the No replication case (within 32768 Micro Seconds) described in table above. This means that for the case of Full replication, the increase in the percentage of **Write** operations results in bigger impact. The largest decrease in the percentage of **set** requests completed within 32768 Micro Seconds is observed for the case of 7 Memcached Servers, decreasing by 9% for the case of increasing the **Write** percentage from 1% to 10% write, in another words it decreases by the factor of 90%/81% = 1.111, which within 32768 Micro Seconds is the largest decrease. The biggest impact on the system for the percentage of **Set** request completed within 32768 Micro Seconds, relative to the base case (3 Memcached servers, No replication, 1% write) happens for the 7 Memcached Servers for the case of full replication and 10% writes, resulting in the decrease of the 12% or by the factor of 93%/81% = 1.148. Thus supporting the **hypothesis** that largest impact on the performance should be observed for the case of Full replication, 7 Memcached Servers and 10% Write operations.

To get a more detailed overview of the impact of the writes on the middleware system and investigate the main reason for reduced performance, I plot the **T_server** and **T_queue** time for the **Set** requests for different setups. Each plot is split into three parts by the black and red vertical dotted lines representing the setup of using 3, 5, and 7 Memcached Servers respectively (Region till black dotted line – 3 Memcached Servers, region between black and red dotted line– 5 Memcached Server, region after red dotted line – 7 Memcached Servers). The Figure 15, depicts the **T_server** time for the **Set** requests for the case of No Replication, for different percentage of Write operations and number of Memcached Servers. As it can be seen, as the percentage of Writes increases the **T_server** time





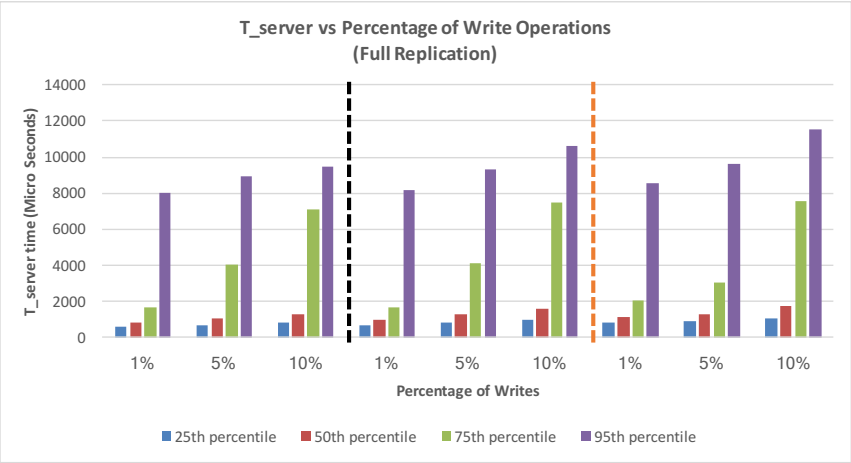
	1%	5%	10%	1%	5%	10%	1%	5%	10%
25th percentile	22	16	14	22	16	15	21	17	15
50th percentile	28	21	20	28	21	21	28	21	20
75th percentile	48	44	75	48	45	72	48	43	48
95th percentile	495.9	1559.9	7056	1130.4	2037.4	7115.6	1000.05	1933.2	4681.8

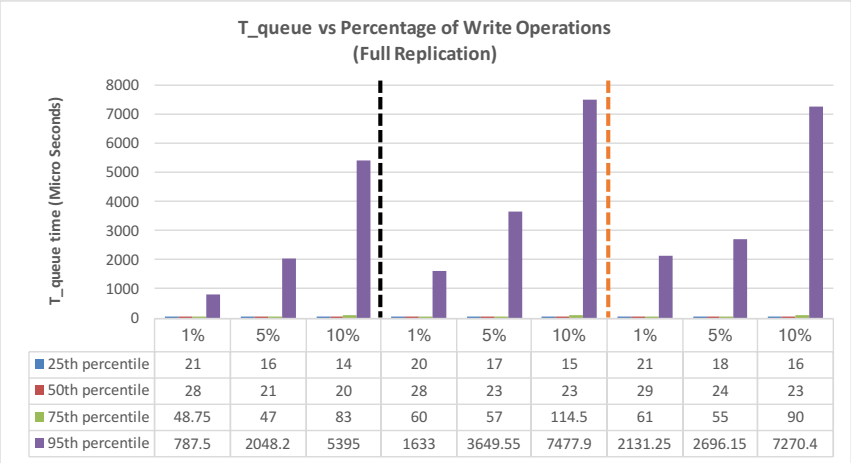
FIGURE 15: T_SERVER VS PERCENTAGE OF WRITE OPERATIONS (TOP), T_QUEUE VS PERCENTAGE OF WRITE OPERATIONS (BOTTOM), FOR THE CASE OF NO REPLICATION. BASED ON SECTION3_EXPERIMENT_SERIES1 DATA.

increases across all percentiles, meaning that **Set** requests spends more time in the middleware, thus increasing the response time. The plot of the **T_queue** vs percentage of Write operations depicts an increase in the 95th percentile for the **T_queue** time as the percentage of writes increases, which also increases the total time **Set** requests spends in middleware and increases the response time. The behavior observed from the figure 15, is in accordance with the expectations made in the **hypothesis** for this section and can be accounted to the reasons presented there.

The Figure 16, presents **T_server** and **T_queue** time for the **Set** requests for different setups for Full replication case. As expected for the full replication the impact of the writes is more significant. Based on the **T_server** vs Percentage of Write operations plot, it can be observed that as the percentage of writes increases the 25th, 50th, 75th, and 95th percentiles also increase in value (Micro Seconds). For the case of No Replication (figure 15) **T_server** time for the 75th percentile stays under 2000 Micro Seconds, while in the case of full replication (Figure 16) the 75th percentile increase till

approximately 7000 Micro Seconds. It should also be noted, that **T_server** time is increase the most for the case of 7 Memcached Server for 75th and 95th percentiles.





	1%	5%	10%	1%	5%	10%	1%	5%	10%
25th percentile	21	16	14	20	17	15	21	18	16
50th percentile	28	21	20	28	23	23	29	24	23
75th percentile	48.75	47	83	60	57	114.5	61	55	90
95th percentile	787.5	2048.2	5395	1633	3649.55	7477.9	2131.25	2696.15	7270.4

FIGURE 16: T_SERVER VS PERCENTAGE OF WRITE OPERATIONS (TOP), T_QUEUE VS PERCENTAGE OF WRITE OPERATIONS (BOTTOM), FOR THE CASE OF FULL REPLICATION. BASED ON SECTION3_EXPERIMENT_SERIES1 DATA.

The **T_server** and **T_queue** plots for the case of No Replication and Full replication (Figure 15, 16), depict a behavior which was expected in the **hypothesis.** Based on the middleware plots presented above, the biggest impact compared to the base case of 3 Memcached Servers for the case No replication and 1% writes is the setup of 7 Memcached Servers, 10% writes and Full Replication.

In, Conclusion

- I see the biggest impact on the performance of the system relative to the base case (3 Memcached Servers, no replication, 1% write) for the **7** Memcached Servers, Full replication and 10% writes. The reason for it is that, increasing number of Memcached Servers, makes the **Write** operation more expensive for the full replication case. Also, increasing percentage of Write requests, means that percentage of Read requests is decreased, consequently replacing a less expensive operation with more expensive one.

Log file listing

Short Name	Location
Section1_experiment_series1	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section1_Experiment_Series1.zip
Section1_experiment_series2	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section1_Experiment_Series2.zip
Section1_experiment_series3	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section1_Experiment_Series3.zip
Section1_experiment_series4	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section1_Experiment_Series4.zip
Section2_experiment_series1	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section2_Experiment_Series1.zip
Section3_experiment_series1	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/Section3_Experiment_Series1.zip
supplementary_plots	https://gitlab.inf.ethz.ch/iverip/asl-fall16-project/blob/master/Logs/Milestone2_Data/supplementary_plots/supplementary_plots.png