Advanced Systems Lab (Fall'16) – Third Milestone

Name: ***Georgios Touloupas***
Legi number: ***16-932-550***

Grading

Section	Points
1	
2	
3	
4	
5	
Total	

1 System as One Unit

In this section we build an M/M/1 model for the entire system using the data collected from the experiments of Section 3 in Milestone 1, Stablility Trace (log files `stabilitytrace`). This model assumes that the interarrival times and the service times are exponentially distributed and there is only one server. There are no buffer or population size limitations and the service discipline is FCFS.

The M/M/1 queuing model we built includes the whole system (clients, middleware, memcached servers and the network among them) for both `get` and `set` requests. To analyze this model we need to know only the mean arrival rate λ and the mean service rate μ. Since we have a closed system we can assume job flow balance and use the average measured total aggregate throughput as the arrival rate. For the service rate we use the maximum total aggregate throughput measured during the experiment. We present the model parameters λ and μ in Table 1.

Parameter	**Value (sec^{-1})**
λ	15350
μ	21056

Table 1: M/M/1 model parameters

In order to compare the model with out experimental data, we calculate the mean response time as well as the standard deviation of the response time using the following formulas.

$$\rho = \frac{\lambda}{\mu}$$

$$E[r] = \frac{1/\mu}{(1-\rho)}$$

$$Std[r] = \sqrt{\frac{1/\mu^2}{(1-\rho)^2}}$$

The M/M/1 model does not take into account the parallel handling of `get` requests per server per thread pool inside the middleware, which account for the biggest part (99%) of the requests in the experiment, for the calculation of the mean response time. The total number of threads handling the `get` requests (`ReadHandler` threads) among the thread pools are $T = 3 \cdot 64 = 192$. To be able to compare the response time results with the experimental data we need to multiply them by T, in order for the model to take into account this parallelism. The results derived from the model are compared to the values measured in the experiment in Table 2.

Metric	**M/M/1 Model**	**Experiment**
$E[r]$ (msec)	33.649	7.121
$Std[r]$ (msec)	33.649	26.99

Table 2: M/M/1 model comparison with experimental data
(model metrics are multiplied by the number of virtual clients)

The comparison shows that the M/M/1 model gives a very rough estimation of the response time of the system. While the derived metrics are of the same order of magnitude as the experimental results and both have high variance (large standard deviation compared to the mean), the derived mean response time is 4.725 times greater than the measured mean response time.

The overestimation of the response time can be attributed to the fact that we used a large factor T to remedy the fact that the model does not take into account any parallelization inside the system (multiple memcached servers, write threads and read thread pools). This factor is large because not all 64 `ReadHandler` threads are utilized at any time per thread pool, since in Section 1 of Milestone 2 we found that increasing the number of threads per thread pool over 24 does not yield any significant benefit. Another factor that may have also contributed to the overestimation of the response time is a probable underestimation of the service rate μ, since we choose it as the maximum total aggregate throughput measured during the experiment, which is susceptible to the variations of the performance of the VMs in the Azure cloud.

The M/M/1 queuing model not being a good fit for our system was expected, since it is too simple and does not take into account many characteristics of the system. The model includes the whole system (clients, middleware, memcached servers and the network among them) and cannot capture all the interactions between its components. Furthermore `get` and `set` requests are not differentiated by the model, even though they are handled in different ways in our system (asynchronous replicated writes, thread pools for synchronous reads). Finally, as we noted previously, the M/M/1 model does not take into account any parallelization inside the system (multiple memcached servers, write threads and read thread pools).

The most important metrics that can be derived from the M/M/1 model are presented in Table 3. The notation as well as the formulas of the course book[1] were followed for the calculations (Chapter 31.2). For $E[r]$ and $E[w]$ we multiply the results with the total number of `ReadHandler` threads $T = 192$ for the reason explained previously.

Metric	**Model value**
ρ	0.729
U	0.729
$E[n]$	2.690
$E[n_q]$	1.961
$E[r]$ (msec)	33.649
$E[w]$ (msec)	24.530

Table 3: M/M/1 model metrics

[1] "Art of Computer Systems Performance Analysis" - Raj Jain

2 Analysis of System Based on Scalability Data

In this section we build M/M/m queuing models of the system as a whole using the data collected from the experiments of Section 3 in Milestone 2, Effect of Writes (log files `writes`). The M/M/m queue can be used to model systems that have several identical servers and all jobs waiting for these servers are kept in one queue. There are no buffer or population size limitations and the service discipline is FCFS.

The M/M/m queuing models we built for different configurations include the whole system (clients, middleware, memcached servers and the network among them) for both `get` and `set` requests. As we want to analyse the system based on scalability data, we choose the number of memcached servers as the scalability factor and build models for $m = 3$, 5 and 7 memcached servers. We also choose the corresponding experiments with full replication and 10% writes workload to accentuate the effects of these parameters.

To analyze each model we need to know only the mean arrival rate λ, the mean service rate μ and the number of the model's servers m. Since we have a closed system we can assume job flow balance and use the average measured total aggregate throughput as the arrival rate. For the service rate we use the maximum total aggregate throughput measured during one of the repetitions for all of the experiments (20328 operations/sec) divided by the number of servers m. We present the model parameters λ μ and m in Table 4.

Parameter	**M/M/3 Model**	**M/M/5 Model**	**M/M/7 Model**
m	3	5	7
λ (sec^{-1})	14993	13185	12747
μ (sec^{-1})	6776	4066	2904

Table 4: M/M/m model parameters

In order to compare the model with out experimental data, we calculate the mean response time as well as the standard deviation of the response time using the following formulas.

$$\rho = \frac{\lambda}{m\mu}$$

$$p_0 = \left[1 + \frac{(m\rho)^m)}{m!(1-\rho)} + \sum_{n=1}^{m-1} \frac{(m\rho)^n}{n!}\right]^{-1}$$

$$\varrho = \frac{(m\rho)^m}{m!(1-\rho)} p_0$$

$$E[r] = \frac{1}{\mu}\left[1 + \frac{\varrho}{m(1-\rho)}\right]$$

$$Std[r] = \sqrt{\frac{1}{\mu^2}\left[1 + \frac{\varrho(2-\varrho)}{m^2(1-\rho)^2}\right]}$$

The M/M/m models do not take into account the parallel handling of `get` requests per thread pool inside the middleware, which account for the biggest part (90%) of the requests in the experiments, for the calculation of the mean response time. The number of threads handling the `get` requests (`ReadHandler` threads) among the thread pools are $T = 24$. To be able to compare the response time results with the experimental data we need to multiply them by T, in order for the models to take into account this parallelism. The results derived from the models are compared to the values measured in the corresponding experiments in Table 5.

Metric	**M/M/3 Model**	**3 Server Experiment**	**M/M/5 Model**	**5 Server Experiment**	**M/M/7 Model**	**7 Server Experiment**
$E[r]$ (msec)	6.010	18.014	6.913	20.164	8.888	21.027
$Std[r]$ (msec)	5.353	37.350	6.373	24.203	8.477	31.284

Table 5: M/M/m models comparison with experimental data
(model metrics are multiplied by the number of virtual clients)

The comparison shows that all M/M/m models give a relatively rough estimation of the response time of the system. The derived metrics for all the models are of the same order of magnitude as the experimental results and both have high variance (large standard deviation compared to the mean). The derived mean response time is 2.997, 2.917 and 2.366 times smaller than the measured mean response time for the M/M/3, M/M/5 and M/M/7 models respectively. However the models capture the scalability of the system, following the trend of increasing response time when increasing the number of memcached servers.

The underestimation of the response times can be attributed to the fact that the models does not take into account the replication for the `set` requests. Taking into consideration the effect of replication, the mean response time increases, however this cannot be captured by the M/M/m models. This effect is accentuated by the fact that we chose to model the configurations with the highest writes workload (10%). Another factor that may have also contributed to the underestimation of the response time is a probable overestimation of the service rates μ, since we choose them as the maximum total aggregate throughput measured during one of the repetitions for all of the experiments divided by the number of servers m. This means that they are susceptible to the variations of the performance of the VMs in the Azure cloud.

The M/M/m queuing models not being a perfect fit for our system was expected, since they do not take into account many characteristics of the system. Each model includes the whole system (clients, middleware, memcached servers and the network among them) and cannot capture all the interactions between its components. Furthermore `get` and `set` requests are not differentiated by the model, even though they are handled in different ways in our system (asynchronous replicated writes, thread pools for synchronous reads). Finally, as we noted previously, the M/M/m model does not take into account the thread pool parallelization inside the system.

The most important metrics that can be derived from the M/M/m models are presented in Table 6. The notation as well as the formulas of the course book were followed for the calculations (Chapter 31.3). For $E[r]$ and $E[w]$ we multiply the results with the number of virtual clients $T = 24$ for the reason explained previously.

Metric	**M/M/3 Model**	**M/M/5 Model**	**M/M/7 Model**
ρ	0.738	0.649	0.627
U	0.738	0.649	0.627
ϱ	0.549	0.301	0.197
$E[n]$	3.754	3.798	4.721
$E[n_q]$	0.106	0.069	0.046
$E[r]$ (msec)	6.010	6.913	8.888
$E[w]$ (msec)	0.170	0.125	0.087

Table 6: M/M/1 model metrics

3 System as Network of Queues

In this section we build a network of queues model for the whole system. We then use Mean-Value Analysis (MVA) to analyse the models.

In order to have all the metrics needed to perform MVA to the network of queues models we built, the middleware is modified to measure the following additional metrics for each request:

- `tHash`: Time spent processing a request in the hashing thread (`NIOServer`[2]).
- `tWrite`: Time spent processing a `set` request in a `WriteHandler`[3]. This time does not include the processing time of the request (time between sending the request to the memcached servers and receiving the answers). Applies only to `set` requests.

The configuration used to collect the experimental data for this section is chosen to have 3 memcached servers, in order to restrict the number of devices of the network of queues. We also choose full replication and 10% writes workload to accentuate the effects of these parameters. We use 3 memaslap clients with 87 virtual clients each (261 virtual clients in total) and 24 threads per read thread pool in the middleware. Memaslap clients use the `smallvalue_10.cfg`[4] workload configuration (Key 16B, Value 128B, Writes 10%). Each experiment runs for 150 seconds, we discard the first and last 30 seconds of the experiment as the warm up and cool down phases respectively and consider the 90 seconds left as 3 repetitions of 30 seconds each. We use one Basic_A4 Azure machine for the middleware, 3 Basic_A2 Azure machines for the 3 memcached servers and 3 Basic_A2 Azure machines the 3 memaslap clients.

Number of servers	3
Number of client machines	3
Virtual clients / machine	87
Workload	Key: 16B, Value: 128B, Writes: 10%
Middleware	Replication: Full, Threads: 24
Runtime	150 sec
Log files	queuing

The closed network of queues model we built for the get requests includes 11 devices and is presented in Figure 1.

- The network between the memaslap clients and the middleware is included as a delay center, since we assume that on average it only adds a constant delay.
- The hashing thread is included as a load-independent service center, since the service time per request does not depend upon the queue length.
- We include 3 load-dependent service centers, each corresponding to a `ReadHandler` thread pool and its corresponding memcached server for handling `get` requests. We model the thread pool and the memcached server in the same device, since `get` requests are performed synchronously. We use load-dependent service centers since we have 24 threats per pool, thus the service time per `get` request in the thread depends on the queue length.
- We include 3 load-independent service centers, each corresponding to a `WriteHandler` thread. We model the thread and the memcached servers behavior on different devices, since `set` requests are performed asynchronously. We use load-independent service centers since the service time per `set` request in the thread does not depend upon the queue length.

[2]https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/src/ch/ethz/gtouloup/NIOServer.java
[3]https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/src/ch/ethz/gtouloup/WriteHandler.java
[4]https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/tree/master/workloads/smallvalue_10.cfg

- We include 3 delay centers, each corresponding to the total memcached server delay of the replicated `set` requests for each `WriteHandler` thread. We assume that on average the processing of a replicated `set` request by all memcched servers only adds a constant delay.

We should also note that the model considers the 10% writes workload, since we set the aggregate ratio of visits to the `set` request path to 10% and the aggregate ratio of visits to the `get` request path to 90%. The visit ratios for the corresponding devices are divided evenly by the number of servers 3, since we use a hashing scheme that uniformly distributes the requests.

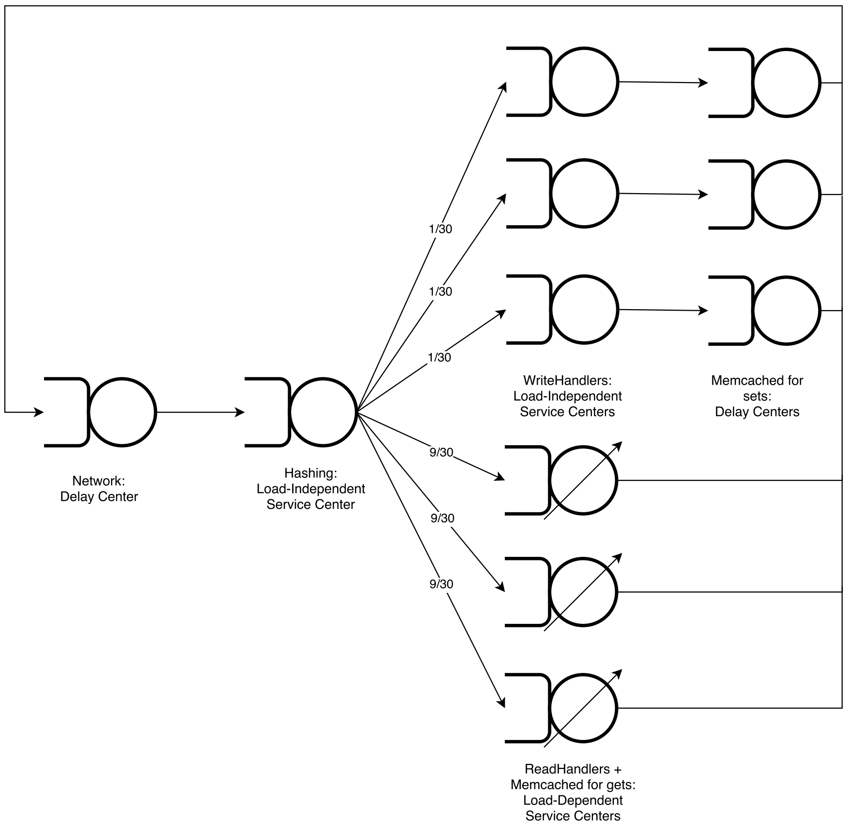
1/30
1/30
1/30
9/30
9/30
9/30
WriteHandlers: Load-Independent Service Centers
Memcached for sets: Delay Centers
Network: Delay Center
Hashing: Load-Independent Service Center
ReadHandlers + Memcached for gets: Load-Dependent Service Centers

Figure 1: Network of queues model

We perform MVA with the help of the Java Modelling Tools[5] suite using the service times for the devices from the experiment, presented in Table 7. The network device service time is calculated as the difference of the memaslap reported and the middleware reported response time. For the load-dependent devices n is the number of the requests in the queue.

Device type	**Service time (msec)**
Network	9.897
Hashing	0.063
WriteHandler	0.187
Memcached for sets	5.568
ReadHandler + Memcached for gets	$\frac{2.774}{\min(24,n)}$

Table 7: Network of queues model parameters

The results derived from the MVA on the network of queues model are compared to the

[5]http://jmt.sourceforge.net

values measured in the experiment in Table 8.

Metric	**Network of Queues Model**	**Experiment**
Throughput (operations/sec)	15976	12457
$E[r]$ (msec)	32.484	21.329

Table 8: Network of queues model comparison with experimental data

The comparison shows that the network of queues model gives a good estimation of the throughput and the response time of the system. The derived total aggregate throughput is 1.282 times bigger than the measured throughput. The mean response time is 1.523 times bigger than the measured mean response time.

The network of queues model is a better fit for our system compared to the previously examined M/M/1 and M/M/m models, since it takes into account more characteristics of the system. The model, instead of treating the system as one queue, splits it into its basic components, thus being able to capture the interactions between them. The model differentiates between `get` and `set` requests by using different devices for their excecution paths, while also modeling the their behavior closer to the real system. More specifically the writes workload, the asynchronous `set` requests, the `ReadHandler` thread pools, as well as the synchronous `get` requests are all characteristics of the system that are taken into consideration by the network of queues model. However the effect of the replication factor is not captured by the model, since we only use delay centers to model this factor.

The most important metrics that can be derived from the network of queues model using MVA are presented in Table 9.

Device type	**Number of Jobs**	**Residence Time (msec)**	**Utilization**
Network	158.117	9.897	1.000
Hashing	53.740	3.363	1.000
WriteHandler	0.111	0.007	0.100
Memcached for sets	2.965	5.568	0.948
ReadHandler + Memcached for gets	13.304	0.832	1.000

Table 9: Network of queues model metrics

To find the bottleneck of our system we examine the utilization of the devices in the network of queues. A delay center can not be the bottleneck, so we focus on the rest of the devices. We see that the hashing thread as well as the `ReadHandler` threads have an utilization of 1. The hashing thread does not perform any time consuming operations as we measured from the service time in the experiment, therefore the bottleneck must be the `ReadHandler` threads. This can be explained by the fact that `ReadHandler` threads are blocking waiting for the synchronous reply from the memcached server. For increased load this leads to an increased queue time for `get` requests. In contrast to that, the `WriteHandler` threads are performing the sets asynchronously, thus leading to a constant queue time for increased load, which prevents them from being the bottleneck of the system.

4 Factorial Experiment

In this section we design a 2^k factorial experiment using the data collected from the experiments of Section 3 in Milestone 2, Effect of Writes (log files `writes`).

In this experiment we will use $k = 3$ factors, namely the number of memcached servers S, the replication factor R as well as the writes workload W. Since we have run the experiments for $r = 3$ repetitions, we will perform $2^k r$ analysis. We use the configurations with the extreme values as the two levels for our factors, presented in Table 10.

Factor	**Level -1**	**Level 1**
Memcached servers, S	3	7
Replication factor, R	No	Full
Writes workload, W	1%	10%

Table 10: $2^3 3$ design factors and levels

Using the data from the experiments we compute that the ratio $\frac{y_{max}}{y_{min}} = 1.384$ is small, where y is the measured total aggregate throughput among repetitions, therefore we use an additive model for the $2^k r$ analysis. The sign table for the $2^3 3$ design is presented in Table 11, where the following effects are calculated:

$$q_0 = 15000, q_S = -931, q_R = -339, q_W = -703, q_{SR} = -31, q_{SW} = 81, q_{RW} = -87, q_{SRW} = -242$$

I	**S**	**R**	**W**	**SR**	**SW**	**RW**	**SRW**	y			$\bar{y}$
1	-1	-1	-1	1	1	1	-1	17198	17448	16890	17179
1	-1	-1	1	1	-1	-1	1	16400	14738	14759	15299
1	-1	1	-1	-1	1	-1	1	16409	16451	15895	16252
1	-1	1	1	-1	-1	1	-1	15345	14647	14987	14993
1	1	-1	-1	-1	-1	1	1	14825	14819	14550	14731
1	1	-1	1	-1	1	-1	-1	14166	13984	14291	14147
1	1	1	-1	1	-1	-1	-1	14544	14800	14605	14650
1	1	1	1	1	1	1	1	12603	12848	12791	12747
119998	-7447	-2715	-5625	-248	651	-697	-1939				Total
15000	-931	-339	-703	-31	81	-87	-242				Total/8

Table 11: $2^3 3$ design sign table

Using the data from Table 11 we compute the following sums of squares which will be useful for the analysis.

$$SSY = \sum_{i=1}^{2^k} \sum_{j=1}^{r} y_{ij}^2 = 5439586949$$

$$SS0 = 2^k r q_0^2 = 5399808002$$

$$SST = SSY - SS0 = 39778947$$

$$SSj = 2^k r q_j^2$$

$$SSS = 20799409, SSR = 2764164, SSW = 11865984, SSSR = 23035, SSSW = 158980,$$

$$SSRW = 182016, SSSRW = 1409798$$

$$SSE = SSY - \sum_j SSj = 2575561$$

Using the sums of squares we can calculate the percentage of variation explained by each factor using the formula $\frac{SSj}{SST}100\%$.

Factor	Variation %
S	52.287%
R	6.949%
W	29.830%
SR	0.058%
SW	0.400%
RW	0.458%
SRW	3.544%

Table 12: Allocation of variation

The results presented in Table 12 show that the number of servers S has the biggest impact, contributing 52.287% of the variation of throughput, followed by the writes workload W contributing 29.830%, while the replication factor R has a smaller impact contributing 6.949%. The interaction SRW contributes 3.544%, while the rest of the interactions have negligible impact. The results agree with our observations in Sections 2 and 3 of Milestone 2 considering the significance of the impact of the factors S, R and W, as well as their interaction SRW. Increasing the number of servers, the replication factor or the writes workload leads to decreased total aggregate throughput. One would expect that the interaction between the replication factor and the writes workload RW should have a bigger impact, however as we concluded in Subsection 3.1 of Milestone 2 this interaction has a significant effect only if the number of servers is also increases, hence the significance of the interaction SRW.

To compute the confidence intervals for the effects we first need to calculate the standard deviation of errors and the standard deviation of effects.

$$s_e = \sqrt{\frac{SSE}{2^k(r-1)}} = 401$$

$$s_{q_j} = \sqrt{\frac{s_e}{2^k r}} = 82$$

Then the confidence intervals for the effects can be calculated using $q_j \mp t_{[1-a/2;2^k(r-1)]} s_{q_j}$, where the t-value at 16 degrees of freedom and 90% confidence is 1.746. The results are presented in Table 13.

Effect	Low CI value	High CI value
q_S	-1074	-788
q_R	-482	-196
q_W	-846	-560
q_{SR}	-174	112
q_{SW}	-62	224
q_{RW}	-230	56
q_{SRW}	-385	-99

Table 13: 90% confidence intervals for effects

The results presented in Table 13 confirm that at this confidence level We note that the 90% confidence interval for the effects of the interactions SR, SW and RW include zero, therefore we can conclude that these interactions do not have a significant impact. This conclusion agrees the results of the allocation of variation for these interactions.

5 Interactive Law Verification

In this section we check the validity of all the experiments from Section 2 in Milestone 2, Effect of Replication (log files `replication`), using the interactive law. We perform the verification by comparing the average response time reported by the memaslap clients with the response time calculated using the interactive law.

According to the interactive response time law $R = \frac{N}{X} - Z$, where R is the system response time, N are the clients, X is the aggregate throughput and Z is the think time of the clients. Since memaslap clients do not wait explicitly between receiving a response and sending the next request, we can assume that Z is in the order of nanoseconds since it only includes a limited amount of CPU cycles. R on the other hand is in the order of milliseconds, which means that we can safely ignore Z from the calculations ($Z \simeq 0$) since R is orders of magnitude greater than Z. Therefore we use the formula $R \simeq \frac{N}{X}$ to derive the response time values in Table 14. The values used for the measured response time are the average response time reported by memaslap clients.

Servers	**Replication**	**Throughput (operations/sec)**	**Measured Response time (msec)**	**Measured Response time (msec)**
3	No	16535	15.785	16.067
3	Half	15021	17.375	17.836
3	Full	15746	16.576	16.788
5	No	15148	17.230	17.514
5	Half	14387	18.142	18.431
5	Full	14307	18.242	18.500
7	No	14156	18.438	18.988
7	Half	13931	18.735	19.063
7	Full	13237	19.718	20.375

Table 14: Interactive response time law verification (the number of clients N is 261 for all experiments)

As we can see in Table 14 the interactive law holds for the replication experiments. The derived values are close to the measured values of the response time (their difference varies between 1.25% and 3.25%).

One thing worth noting is that the measured response time is always greater than the response time derived using the interactive law. As we showed in Subsection 1.1 of Milestone 2 (Choosing the Response Time Metric) the response time distribution reported by memaslap clients is a right-handed tail distribution. Therefore average response time is not a good metric for representing the overall behavior since it is affected by large individual values, however we have to use it because the interactive law works only with the average (not with percentiles). The average is overestimating the response time which explains why the measured response time is always greater than the response time derived using the interactive law.

Logfile listing

Short name	**Location**
baseline	https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/logfiles/baseline
stabilitytrace	https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/logfiles/stabilitytrace
maxthroughput	https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/logfiles/maxthroughput.zip
replication	https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/logfiles/replication.zip
writes	https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/logfiles/writes.zip
queuing	https://gitlab.inf.ethz.ch/gtouloup/asl-fall16-project/blob/master/logfiles/queuing