Advanced Systems Lab (Fall'15) – Second Milestone

Name: *Sandro Huber*
Legi number: *10-924-777*

Grading

Section	**Points**
1	
2	
3.1	
3.2	
4	
5	
Total	

1 System as One Unit

Before coming to the M/M/1 modelling of the system, let's first clarify the notion of request used in this report, since this was not done in the first milestone. In short, a requests is equal to one atomic action of a client, like sending a message or reading all messages of a certain queue. The workload is designed such that, the number of messages in the database stays constant over time. Now that this is explained, let's move to the actual modelling.

The following numbers have been measured in a new stability experiment. For a detailed insight into the configuration and setup, please see section 5. The reason for running a new experiment was the lack of measurements which allow a decent modelling of the system. The following table summarises the measured input parameters for the M/M/1 model:

Measured Data

Throughput X	554.94 req/sec
Response Time R	216.5 ms/req
Queueing Time Q	142.6 ms/req
Sleep Time of clients Z	0.002 ms

When taking this data and putting it into the formulas of the M/M/1 model one ends up with this result:

Calculated Data

Arrival Rate λ	$\lambda = X$*	X
Service Time S	$S = R - Q$	71.91 ms/req
Service Rate μ	$\mu = \frac{1}{S}$	13.91 req/sec
Traffic Intensity ρ	$\rho = \frac{\lambda}{\mu}$	39.91

*This holds, because the system built is a closed one. This is known, because every client first sends a request into the system and waits until an answer in form of a request comes back. Only then the client proceeds with it's job based on the meaning of the answer. If an answer is lost during transmission or execution the system will automatically take care of this and closes the socket connection to this client, such that it knows an error occured and it has to reconnect if there are still messages not successfully sent yet. This guarantees that no messages are no false positives when logging the throughput.

Based on the following law:

$$\text{System is stable} \Leftrightarrow \text{Traffic Intensity} < 1, \tag{1}$$

the system should be heavily unstable. But multiple factors show evidence that this is not the case. First we can have a look at the plots displayed in figures 1 and 2. It's clearly visible that the system is stable. Another factor worth considering in view of this problem is the standard deviation of the throughput and response time which are 20.122 Req/sec and 7.85 ms/Req respectively. These two values are identical to 3.626% (throughput) and 3.625% (response time) variation with respect to the corresponding overall mean value, which again indicate a very stable system.

So how to explain then that the traffic intensity $\rho = 39.91 > 1$? The answer lies in the choice of the model. Since the system that was built, internally has a lot of parallelism and works with threads that execute along eachother, the simple M/M/1 model is not able to explain this complex structure and thus fails. The value of the traffic intensity is not random at all though. It is still somewhat connected to the inner structure of the system. Precisely I am talking about the number of database connections, which are constantly 40 during the whole experiment (20 per middleware). In practice this means that on the database, 40 concurrent

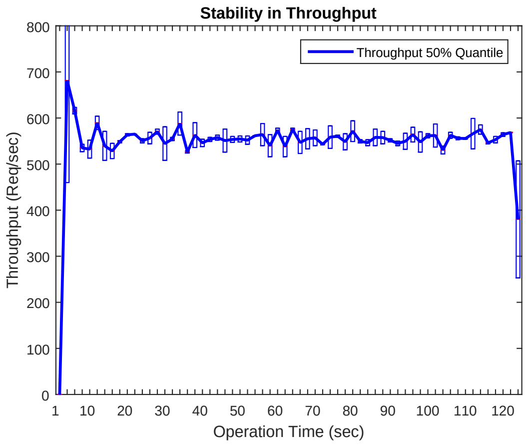

Figure 1: Throughput of the whole System

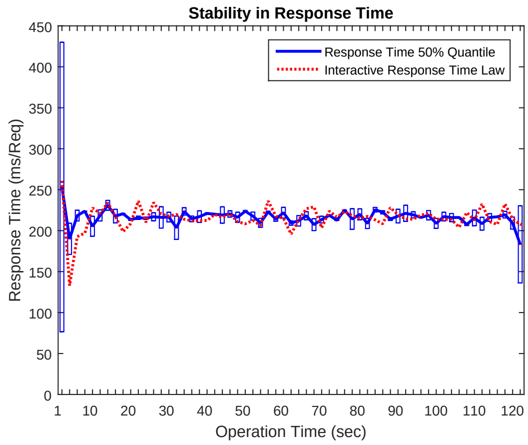

Figure 2: Response Time per Request

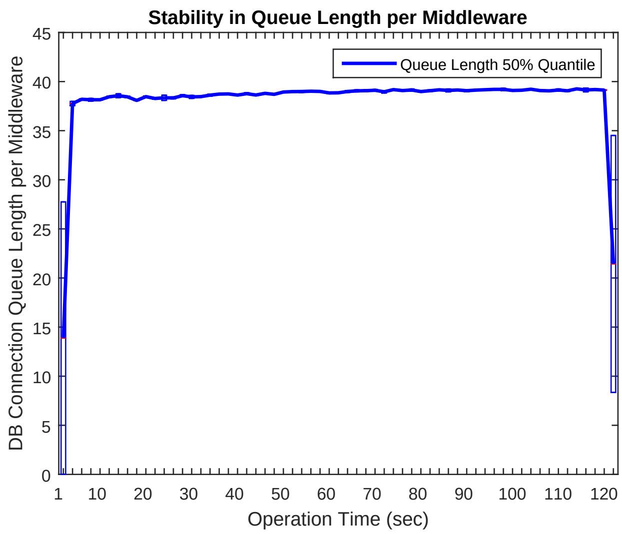

Figure 3: Queue length in front of DB Connection pool

threads were available which serve concurrently all queries sent from the middlewares. This would perfectly explain the traffic intensity, because when inserting these 40 concurrent threads into the equation: $\rho = \frac{\lambda}{40\mu} = \frac{554.94 \text{ sec/Req}}{40*13.91 \text{ Req/sec}} = \frac{554.94 \text{ sec/Req}}{556.4 \text{ Req/sec}} = 0.9973 < 1$. To be sure that all 40 threads on the database are really in use the whole time, we can see if there is queueing happening in the right part of the system. Each request has to go to the database somewhen. The connections are provided through a connection pool. If there are currently no connections available, the request has to wait. In figure 3 we can see that the queue length per middleware is on average of length 38.9, which makes visible that indeed the database is the bottleneck. These numbers do make sense, because we now can precisely say in which state all the requests are: 40 requests are processed on the database and $2 * 38.9 = 77.8$ are waiting for a database connection. Because there were exactly 120 clients online (each one having one open request in the system) we know that we are missing track of $120 - 40 - 77.8 = 2.2$ requests on average. But this makes sense, because there have to be some requests on the way from the database back to the clients and some new ones coming from the clients which did not yet reach the database connection queue.

2 Analysis of System Based on Scalability Data

The scalability of a system has two aspects: it describes the capability of the system to handle an increasing number of requests, but also the possibility of boosting the system performance by adding more modules. It's thus important to include both factors into the analysis. For a detailed description of the experiment setup, please refer to section 5.

Based on benchmarks evaluated in milestone one, we can assume that the database is the bottleneck in this configuration. As in the book of Raj Jain is written, an M/M/m queue is used to model a multi-server system where jobs for these servers are kept in one queue. In our case, the servers are equal to the threads on the database and the single queue is found as the waiting queue for a database connection on the middleware. It thus make sense to apply an M/M/40 queueing model, because as said, the database provides 40 concurrent connections, and thus, internally runs 40 threads.

Table 1: M/M/40 Model on Scalability Data for 1 Middleware

	Model Parameters		Computed and Measured Variables		
#Clients	λ (Req/sec)	μ (Req/sec)	ρ	ϱ^{*1}(%)	Queue Length*2
10	2481.5	366.15	0.1694	$2.5783 * 10^{-13}$	0.10
20	4773.5	341.82	0.3491	1.15	0.20
30	7015.5	339.42	0.5167	51.66	0.34
40	9311.5	332.88	0.6993	69.93	0.45
50	10346.5	303.02	0.8536	85.36	0.68
60	12302	317.06	0.9700	97.00	2.90
70	13152	329.90	0.9967	99.67	10.66
80	13005.5	326.25	0.9966	99.66	20.41
90	12321	308.14	0.9996	99.96	30.84
100	13383	335.71	0.9966	99.66	38.87
110	13013.5	326.28	0.9971	99.71	49.56
120	12642.5	316.59	0.9983	99.83	59.55

*1: Probability of Queueing *2: Measured and averaged per Request

Let's analyse the outcome of this modelling approach with respect to the previous M/M/1 model. The first improvement visible is the much better fit of the traffic intensity ρ. Indeed, the model does now also yield a stable system and thus now mirrors the reality. The model gives also reasonable insights into the probability of when queueing is happening. When we have values of the measured queue length which are < 1, it means that more requests had an empty queue than requests which had another request actually waiting infront of them already. But the model does also fail in some predictions: For example when wanting to compute the queue length yield by the model via $E[n_q] = \frac{\varrho}{\varrho} * (1 - \rho)$ the mismatch is immense:

#C	10	20	30	40	50	60	70	80	90	100	110	120
	0.10	0.20	0.34	0.45	0.68	2.90	10.66	20.41	30.84	38.87	49.56	59.55
	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.14	0.30	0.48	15.14	$5 * 10^{15}$

The first column shows the measured effective queue lengths on the middleware and the second, the computed one. It is evident that neither the point of the knee, nor the slope of the line does fit onto the data. This is due to the simple M/M/40 modelling approach which does not take into account, that there are effectively two separate components existing in this modelled unit. This leads to the conclusion that this M/M/40 model does fit already quite well onto some initial parameters, but does not lead to further insights afterwards.

The follow-up question to this is, if there are any other models worth trying for the one-middleware setup? The answer is no. Since the whole system is still treated as a black box, there is no other parameter indication besides the one yield by the bottleneck (in our case 40 connections on the database). Additionally it does not make sense to try other ones, when thinking about what context the M/M/40 model was presented: The scalability experiment was run with clients, which tried to send as much as possible. Thus a traffic intensity ρ of close to 1 does make sense. What happens when trying $m_1 < 40 < m_2$? The M/M/m_1 model will yield $\rho > 1$, because the parallelism of the database was not fully taken into account and thus the system seems to be unstable. A M/M/m_2 model will lead into $\rho < 1$, but that's also not optimal, because we already know that the system was operating at its limits. It is therefore not needed to apply another M/M/m model onto the data.

Seeing the resulting throughput of the scalability experiment in figure 20 in section 5 it's mentionable that having one or two middlewares in action does throughput-wise not make a difference. This should also be expressed in the modelling, when applying the M/M/40 model onto the data of two middlewares. For the sake of redundancy, only the traffic intensity ρ is calculated, to give a brief overview over the fitting quality.

Table 2: M/M/[40 and 42] Model on Scalability Data for 2 Middlewares

	Model Parameters		Computed Variables	
#Clients	λ (Req/sec)	μ (Req/sec)	ρ_{40}	ρ_{42}
10	2503	360.04	0.17	0.17
20	4794	342.45	0.35	0.33
30	7262.5	354.02	0.51	0.49
40	9423.5	337.38	0.70	0.67
50	11032.5	315.41	0.87	0.83
60	13147.5	330.46	0.99	0.95
70	12107.5	292.5	1.03	0.99
80	13144.5	316.52	1.04	0.99
90	13156	316.36	1.04	0.99
100	12625.5	304.13	1.04	0.99
110	13667.5	328.27	1.04	0.99
120	12937	311.13	1.04	0.99

The outcome of this analysis is somewhat surprising, since the M/M/40 model does - in contrast to what was expected and predicted - not quite fit the measured data. The solution for this modelling failure lies in the splitting of the queues. Having one or two middlewares does not influence the number of total requests currently waiting for a database connection, but it divides the length of the queue infront of every requests by two. This of course reduces the waiting time a bit and thus the overall throughput is a bit higher. This issue can be fixed when instead calculating with $m = 42$, but now there is no semantic connection between the choice of the m and the system configuration, and thus is not a good choice. What surely can be said is that, the M/M/40 model does a reasonable job when trying to model the whole black box containing middlewares and a database, but does not allow to foresee whole stories. To get a deeper understanding of the system, let's bring light into the black box and understand the components and then fit separate models onto each of them. This is done in the next section.

3 Modeling Components as Independent Units

3.1 Middleware

As already explained in the first Milestone, the middleware was built with maximal adaptivity to the current situation in mind. This means that a middleware is able to dynamically allocate resources on demand, when either more clients join the system, or in general more requests are generated. To realize this feature, a *cached thread pool* of the *ExecutorService* class has been instantiated to deal with the distribution of requests to threads. The magic behind this class is that it automatically creates and runs threads for each new job, in our case requests, entering the middleware. As shown in the benchmarks for the middleware, this setup allows a very high throughput of around 45'000 requests per second. Since the behaviour of the *cached thread pool* is not fully clear, because one just can't look into its source code, it's necessary to first understand the relationship between the two factors '#Clients' and '#Threads'. It's important to understand how the number of threads is evolving over time and load, because the later modelling will strongly depend on this evaluation. The number of requests per second is here not important, because it's not about the throughput, but about the number of requests currently in the system, which is directly correlated to the number of clients (system is a closed one).

Configuration: This analysis was done on the scalability data. There were one or two middlewares, one database with 40 concurrent connections and two client machines online which provided system access for clients. If two middlewares were online, the clients and database connections were evenly split among both. The clients sent as much as they could, following the load configuration mentioned in milestone 1. Each message had a content of length 200. The number of threads was measured by looping through all available threads and counting those which were in the state *RUNNABLE*.

Expectation: The middleware structure is based on a two-step approach: Whenever a new requests enters the middleware, a new thread gets launched which handles the reading, deserialization and database access. When this is completed, this thread shuts down and fires up a second thread that handles the serialization of the answer and sends it back to the corresponding client. So in average there is one thread online per request. Thus we can expect that the equation '#Requests in middleware or database' = '#Request not in clients' = '#Threads' should hold. But in section 1 we already saw that only a very small minority of requests is not found in the database or the middleware. With this reasoning we can expect the following behaviour for n clients: $\#Threads = n - \epsilon$, for $\epsilon > 0$ and small.

Reality (figure 4): It's obvious that the mismatch between the measured number of threads and the expectation is serious. What went wrong? The whole analysis was based on the idea of having one thread per request in the middleware or database. But one thing I forgot was the ability of threads to get reused. For example when a request waits for a database connection, its corresponding thread can meanwhile be reused by another request for e.g. deserialization. It therefore holds that '#Requests in middleware or database' = '#Request not in clients' $\neq$ '#Threads'. We still want to correlated the number of clients to the load (i.e. number of threads) of the middleware. Although the equation chain does not hold, we still can use its first part, namely '#Requests in middleware or database' = $n - \epsilon$ = '#Clients'$-\epsilon$, for $\epsilon > 0$ and small. By simply counting these requests which currently are processed by either the middleware or the database we end up with a much better fit onto the expectation (figure 5). We indeed can see that the relation is linear and fulfills the expected bahaviour of $n \approx$ '#Requests in middleware or database'.

The figure 5 does verify our assumption of having a linear load-dependent middleware behaviour. This is a problem for modelling, because fitting a single model won't be easy. So let's tackle this problem step by step. First, let's see if there is a model fitting onto every point of the middleware baseline, and in a second step, I try to unite these models into a single one. The first step is done, such that I get more insights into which models fit certain regions of the middleware behaviour and allow easier designing of the single M/M/m model. I quickly repeat the conditions of the middleware baseline, because the following analysis is based on it:

Configuration: The middleware was as isolated as possible, i.e. no database access was made. Two client machines were flooding either one or two middlewares with messages with content length 200. As service time, only the time within the middleware was measured. To apply an

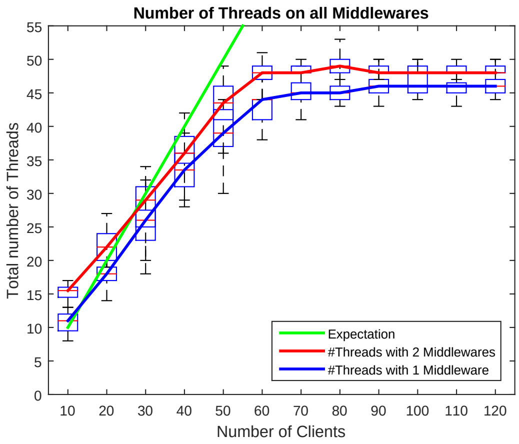

Figure 4: Number of threads over all middlewares under different configurations

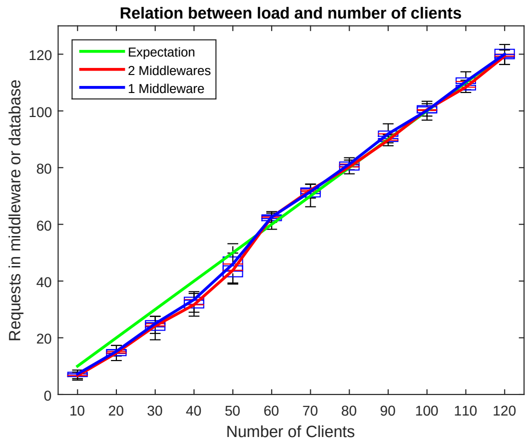

Figure 5: Refined load estimation on the middleware

M/M/m model, the throughput and service time of the corresponding experiment data were taken. To get an idea of how well different models fit onto the data I did the following: For every point i.e. throughput and response time for n clients, all M/M/m models with m going from 1 to 110 were applied. Because the system was stable in all phases of the experiment, the traffic intensity allows to see how well a model does fit.
Expectation: As seen in figure 5, the load factor seems to relate linearly to the number of clients, so I have no reason the expect anything else of the models.
Reality (figure 6): It's clearly visible that the linear trend could be verified also when applying the models. Since the system was stable, but still trying to get as much throughput as possible, a traffic intensity of just below 1 is expectable. The plot was generated with this assumption in mind. This leads to the basic formula of $|(\rho - 1)|^{-1}$, which models the closeness to 1. To have a visually more appealing plot, the final formula reads as $\log(|(\rho - 1)|^{-1} + 4)$. Additionally to the linear relationship, it's noticeable, that models with higher m's tend to fit more parts of the whole data. This insight is used for the second step, when trying to fit a single model onto the data.

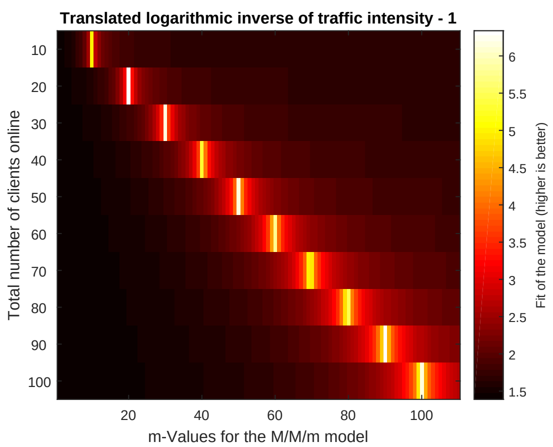

Figure 6: Overview of how well the M/M/[1-110] models fit onto the middleware data with respect to the traffic intensity

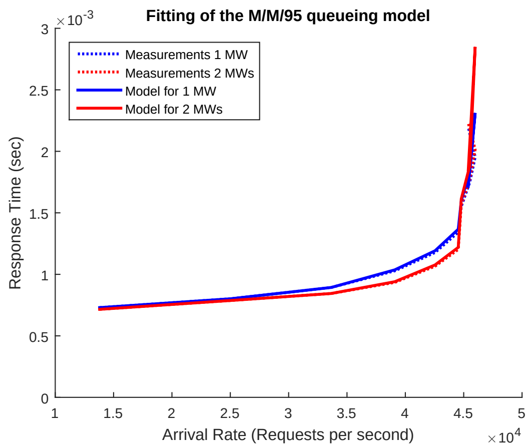

Figure 7: Applying the M/M/95 model onto the data of the middleware benchmark

To test a single model onto the middleware, the response time yield by the model and the measured one will be compared. This will give a detailed insight into if and where the model does fit the built system. The approach taken is to model the middleware best when being under maximal load, i.e. when the throughput stagnates, but the response time still grows. This is the case when having 60 clients or more. By examining different models, the best one was found to be the M/M/95 model, which is displayed in figure 7. The model is capable of not only representing the middleware when under maximal load, but also can closely predict the early stages when having a low request rate. The reason for not having the variance in the plot, is that I already worked with only the 50% quantile of the data, because the throughput and response time data were logged in different settings (per requests and per second). But to generate a plot like the one seen in figure 7, it's needed to have some sort of common structure. Coming back to the actual plot itself; why is it that exactly the M/M/95 model does the best job, and what is the relation to the real system? As mentioned in the in the first part of this section, threads are reused and thus can handle multiple requests. We saw in figure 4 that the number of threads when having the maximal throughput, is on average 47 (calculated from the mean values of both settings 1 and 2 MW's). The model does yield, that queueing happens with 83.88% on average, so we can assume that on average each thread can indeed handle multiple requests, because of the waiting induced by the queueing. Eyeing at the M/M/95 model we can see that a re-usage factor of $\frac{95}{47} = 2.02$ requests per thread would not only fit the model perfectly, but also does make sense with respect to the measured data.

3.2 Database

To find a queueing model for the database one has also to take the hardware as well as the software setup into consideration. Let's first start with the hardware specification of the machine used during the experiments, since some of these led to later decisions and reasoning.

The database was run on a r3.4xlarge machine. By providing 122GiB of RAM it was guaranteed to have reasonably big tables fit into the working memory and get good performance. The machine also contained 8 physical CPU's which allows to have 16 parallel threads with hyper threading. By the rule of thumb presented by PostgreSQL itself, it holds that the best optimal number of connections is $2 *$ '#physical cores' $+ C$, where C is a constant. In the benchmark of milestone one, it was found that the best performance is reached with 20 concurrent database connections (position of the knee), i.e. the moment before the throughput stops to increase as rapid as in the beginning. This means, that from the hardware perspective a model parameter m of 20 is yield. Optimality in this context means to have the highest throughput with the least response time. It can be easily viewed in the corresponding figure in milestone 1, that optimal is not equal to maximal. In our case, the maximal throughput is reached when having around 40 database connections. This finding was the reason to run the experiments with 40 database connections throughout most of the experiments. So from the software point of view we have 40 as an initial guess for the model parameter m. When searching through this model space ($m \in [20, 40]$), the best one to come across with respect to the mean squared error between the modelled and measured response time is M/M/30 with 0.0012 ms of deviation. See figure 8 for a comparison between the model and the effective measurements. How is this model connected to the real system? Obviously 30 is exactly between our initial numbers gotten from the hard- and software evaluation, so intuitively it seems to fit. Basically it means, that although 40 threads are running, the performance does not much change with respect to the scenario when having only 30 clients, flooding the database with requests. Eyeing at the throughput plot in milestone 1, we can see that this is indeed the case. The reason for not quite reaching the same throughput as with 40 clients, is because the model does also try to fit onto the the system when it's not fully loaded and thus favours a smaller m. It also shows that in our case the rule of thumb does underestimate the power of the system. This can be explained, because the general rule of thumb assumes a default amount of work to be done for every request. In the load used in the experiments we have only one costly (remove message) and four cheap requests. This allows the system to perform better than expected and thus also exceed the lower bound of 20. The M/M/30 model allows to get more insights into how the database performs, for example when the machine starts to reach its performance boundaries. In figure 9 it's shown how the

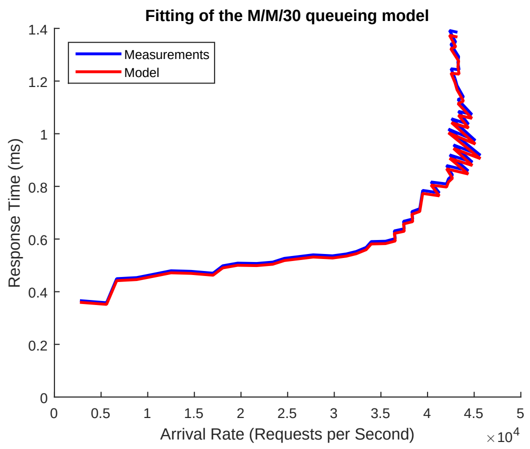

Figure 8: Applying the M/M/30 model onto the data of the database benchmark

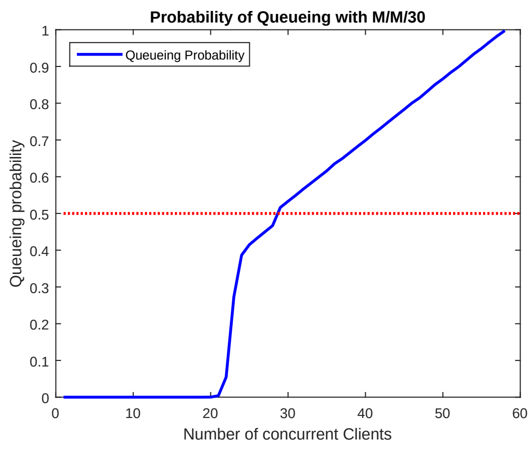

Figure 9: Visualisation of the probability of queueing predicted by the M/M/30 model

probability of queueing does behave when more clients are spamming requests to the database. The dashed red line indicates the 50% border. Having data points above this line means, that on average queueing does happen more often than not. Queueing in the context of a database can be interpreted as waiting for a connection, when all connections are currently used. We can not only see that the queueing probability exceeds 50% at around 30 clients and thus indicates that our choice of model is sound, but also that it's consistent with respect to the throughput measured in the first milestone, because it starts to decrease when having this many clients.

Both components have been thoroughly analysed and the best model for each of them was found. Let's use this information to evolve our model of the whole system in the next section.

4 System as Network of Queues

In the last three section, all important individual system parts have been modelled individually. We can now use this information to try to fit a model when combining the individual parts into a single system. To achieve this, a network of queues is built.
Configuration: To model the system I did choose to take one M/M/30 queue for modelling the database and a M/M/95 queue to simulate the behaviour of the middleware. The clients and the network delays are packed into a single unit, because neither does the service time change with different loads nor is it dependent on it's position in the system (e.g. before or after the middleware). All of these decision directly follow the results obtained in sections 3.1 and 3.2. The full setup is shown in figure 10. To proceed with the modelling, we need to find suitable input parameters for each of the devices. The findings are summarized in the following table:

Table 3: Fixed Model Parameters

Network delays	0.864723 ms/req
Sleep Time Z	0.0020 ms/req
Parallelisation factor for Middleware	95
Parallelisation factor for Database	30

Since we want to work with two load-dependent devices, we need to find a suitable service rate μ for each one. The problem is that, in contrast to a delay center, the amount of time

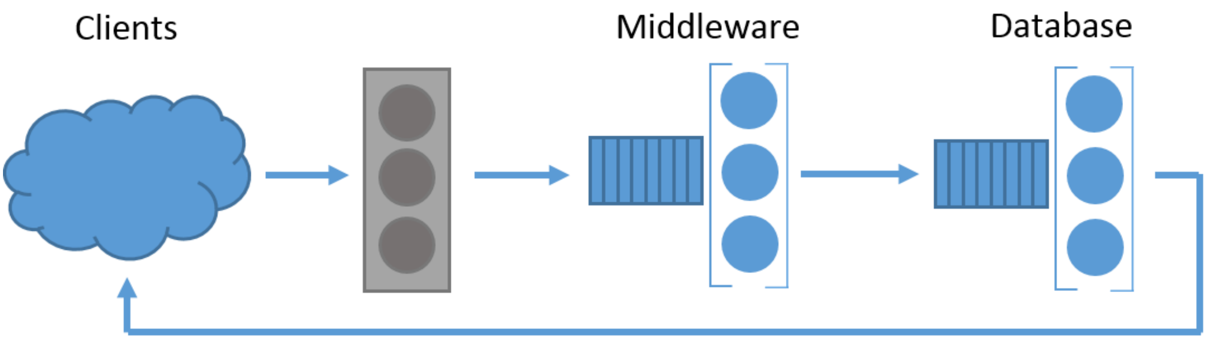

Figure 10: Schematic setup of the Network of queues to simulate the behaviour of the whole system. The grey box corresponds to the delay center and the other ones to it's corresponding devices.

spent in the device is - as the name suggests - dependent on the load. It thus holds that constant $C = \mu_{\text{delay denter}} \neq \mu_{\text{load-dependent device}} = f(\text{workload}) = f(\text{'\#Clients'})$. The last equation holds, because the system is closed. To have a suitable approximation of the service time, we used the measurements of the scalability experiment. There, the points for 10, 20, ... 120 clients were collected. It was thus possible to interpolate all missing values to get a fully defined function over the whole spectrum of clients. This was done in *MATLAB* with a call like *interp1(x_measurements, y_measurements, x_estimated, 'pchip');*. It is highly important to account for the parallelisation in each part of the system. Because if a request takes 100 ms to get processed, the MVA will never give a bigger throughput than $\frac{1}{0.1\text{sec}} = 10$ requests per second. It's a good idea to follow an approach proposed in the book of Raj Jain and multiply the number of processed requests by the number of active clients, until either a physical (e.g. physical cores) or software (e.g. number of database connections) boundary is reached. In our case these boundaries were already computed and listed in table 3. For example, the multiplication vector for the database looks like $[1, 2, 3, ..., 30, 30, 30, ..., 30]$, with length of 120. The parameter values listed in table 3 and the thoughts presented here, lead to the device definition shown in table 4.

Table 4: MVA Devices

System part	Service Time (ms/req)	Type	Parallelisation	#Visits/req
Client + Network	0.864723	DC*1	1	1
Middleware	0.844386*3	LDSC*2	95	1
Database	1.327447*3	LDSC	30	1

*1: Delay center *2: Load-dependent service center
*3: Example values when having 30 clients

Based on the device configuration presented in table 4, an MVA was performed. To have a nice and clean implementation and be able to evaluate the results properly, a JAVA program was written. The source code is found under $\backslash src \backslash org \backslash asl \backslash mva$.
Expectation: As seen in the first milestone, the bottleneck of the whole system was identified as the database. So I expect the MVA to at least give a hint on this behaviour through the utilization output. Since measurements of a real experiment were used as input of the algorithm, the predicted values should come pretty close to the original ones, even though the interpolation of missing values was only linear.

Table 5: Results of the Mean Value Analysis

	Computed Values				Measured Values	
#Clients	U_{MW}	U_{DB}	X (Req/sec)	R (ms)	X (Req/sec)	R (ms)
10	0.03	0.2	2505	3.98	2482	2.76
20	0.06	0.36	4496	4.44	4474	2.96
30	0.10	0.563	7004	4.28	7016	2.98
40	0.12	0.759	9322	4.29	9312	3.04
50	0.14	0.86	10453	4.78	10347	3.5
60	0.14	0.97	11180	5.36	12302	3.56
70	0.14	0.99	11124	6.29	13152	4.02
80	0.14	0.97	11034	7.24	13006	4.81
90	0.14	0.99	11043	8.14	12321	5.94
100	0.14	0.99	11010	9.1	13383	6.06
110	0.14	0.99	10980	10.00	13014	7.06
120	0.14	0.99	10952	10.95	12643	8.05

Reality (table 5): Let's first look at the both columns where the utilizations of the middleware and database are listed. It's clearly visible, that the database is fully saturated around 60 clients, whereas the middleware only has to show only a small portion of its capabilities. The position of the knee does indeed also match the one seen in the actual experiment displayed in the figures 20 and 21. From this perspective, the expectation is verified, that the MVA does also mirror the bottleneck behaviour of the database. It's a different story when talking about the matching of the measured and computed throughput and response time, though. The throughput does match very well the measured one until 60 clients are reached. After that point, the throughput does also stagnate, but in a lower region as the one actually measured. Since the system throughput is limited by the database it's possible that the network latency between the middleware and the database was underestimated, and thus the service time of the database was a bit too high, which led to a lower total throughput. It's therefore also not surprising that the overall response time is higher than expected.

The bottleneck is the core optimization point whenever a system is evaluated with respect to performance boosting. To get more insight into it, in the following a bottleneck analysis is done.

Configuration: The analysis was done on the scalability data, but there was only one middleware considered (because already one is enough to fully saturate the database). Because of the above mismatch of the response times, the database service time was corrected by 1ms. This is a big approximation of the real network latency happening between the middleware and the database, because simple pinging between two machines is always dependent on, for example the current network usage or machine proximity. This change was needed, because in the original measurements, the database service time was estimated from the middleware, because the inspection of the postgres logs was not doable in time anymore and thus this shortcut had to be taken. The analysis is done based on an asymptotic limit based on the demand of the bottleneck device in the context of the whole system. To compute this boundary, the formula (33.8) of the book was used, which reads as $X(N) \leq min\{\frac{1}{D_{max}}, \frac{N}{D+Z}\}$, where $D = \sum D_i$.

Expectation: The database is already in the early stages, i.e. when low numbers of clients are online, the limiting factor in the system. It is therefore reasonable to assume that the actual throughput does follow the asymptotic limit quite closely, when the system if fully saturated. We just saw that this happens at around 60 clients, so there should be a visible come-together of both lines.

Reality (figure 11): Despite the coarse approximation of the network, it's possible to get some valuable information out of the graph. First of all, the curves do show a very similar behaviour from the moment on, where the system is saturated, i.e. when the database starts to get overwhelmed by the amount of requests. When the system is only accessed by a low number of clients, the per-request service time is lower because there is no congestion happening between

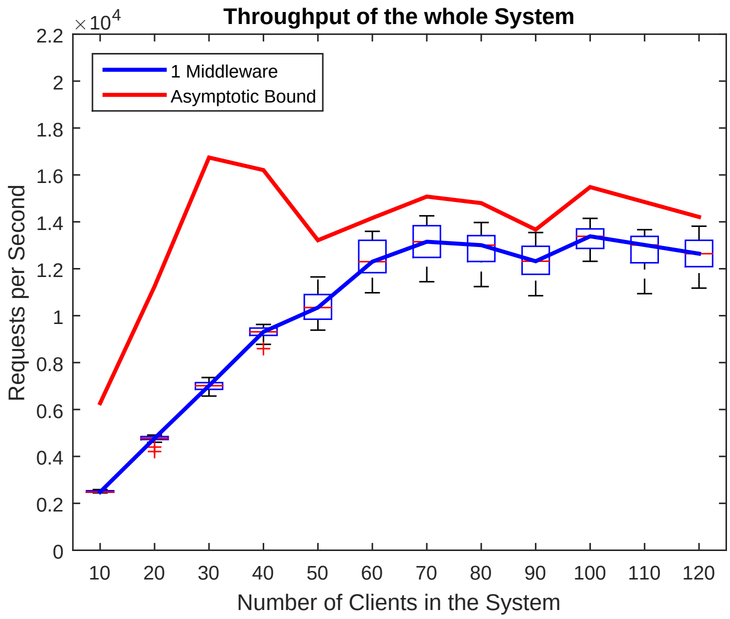

Figure 11: Asymptotic behaviour of the throughput when limited by the database as bottleneck.

the different database threads. At 30 clients, this behaviour suddenly changes and the database starts to get the limiting factor of the whole system, visible by the shrinking distance between the two lines. That this break happens at 30 clients is not surprising at all, since in section 3.2 we concluded that the database is best represented by a M/M/30 queue. The discrepancy between the curve in the saturated region is explained by the rough approximation of the network latency between the middleware and database, the middleware service time which also introduces some milliseconds and the whole network between the clients and the middleware. This reasoning thus allows to confirm that the database is indeed the bottleneck.

In case of wanting the system to perform better, the database would be the part where improvements are directly measurable. Having a better (i.e. more threads, more RAM, more CPU's, ect...) database the red line in figure 11 would not only be generally higher, but also the two points currently located at 30 and 60 clients could possibly move more towards the right. In practice this would mean that the overall system performance can handle more clients before it is saturated and the overall response time for every requests starts to increase (assuming it is still the database is still the limiting system part).

5 Interactive Law Verification

The interactive law is used to get a verification of the correctness of the experiments, the data and the plots itself. In this report, the interactive response time law of the form

$$R = \frac{N}{X} - Z$$

is used. All plots are always showing the matching between the measured and computed response time from the interactive law.

Let's follow the order of the first milestone and start with the database, followed by the baselines of the other two parts of the system.

Database baseline (figure 12): The plot shows a very good fit for the levels 0, 1 and 2 with indices. The discrepancy of the law and the measurements on the level 2 without indices is explained by the much bigger variance compared to the other levels.

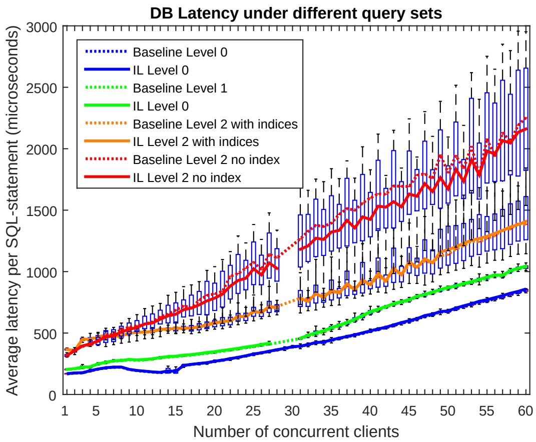

Figure 12: IRL on DB baseline

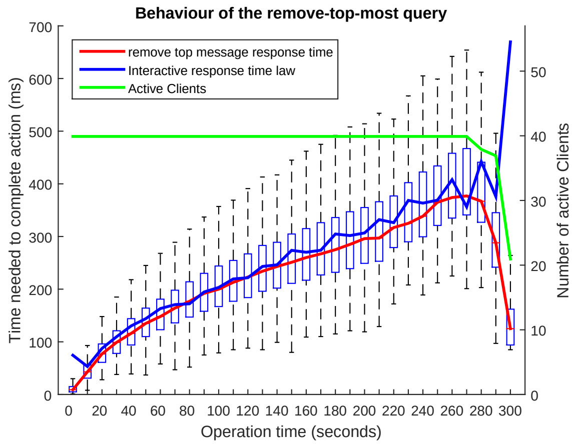

Figure 13: IRL on remove querey baseline

Remove query performance (figure 13): In the beginning the lines follow each other closely. The growing difference is explained by a) the growing variance and b) early client finishings as can bee seen by the green line in the plot.

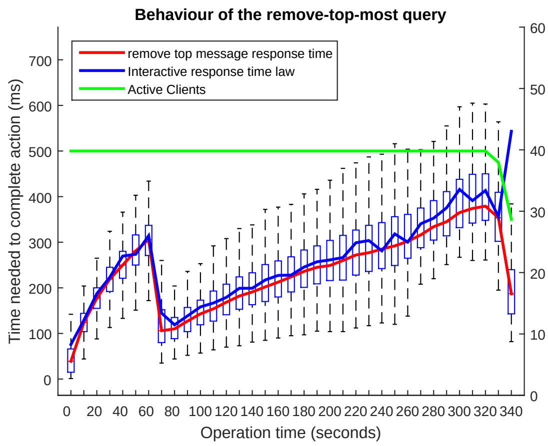

Figure 14: IRL on DB third index

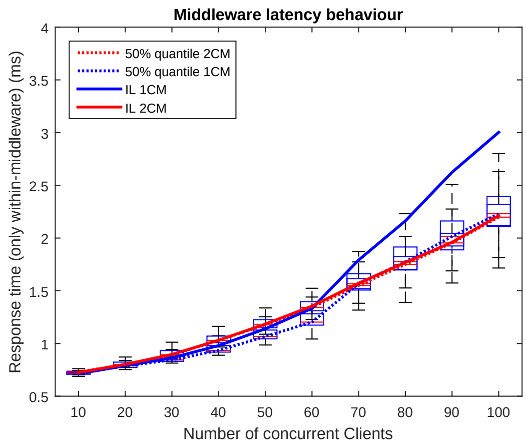

Figure 15: IRL on middleware baseline

Third index in database (figure 14): Same reasoning as for figure 13.
Middleware baseline (figure 15): The two aspects of the experiment when either having one or two client machines flooding requests are also visible in the interactive response time law. Having only one client machine was not enough when running more than 60 clients, because the resource limit on the client machine was reached. Thus, less throughput was registered and higher response times are expected, but not measured and therefore the difference between the blue lines.
Client baseline (figure 16): Everything is ok here.
Stability (figure 17): The y axis is intentionally not starting from 0, because then the information of how exact the lines do fit would be less, and in case of only having a plot as verification it certainly helps to have a more detailed view. The lines do also fit very well and mirror the measured behaviour.
Response Time Experiment (figures 18 and 19): Everything is ok.

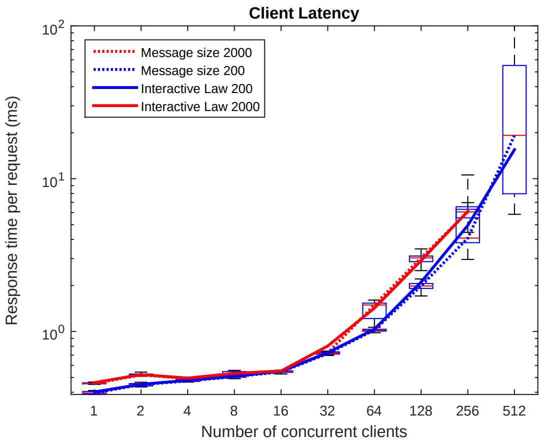

Figure 16: IL on client baseline

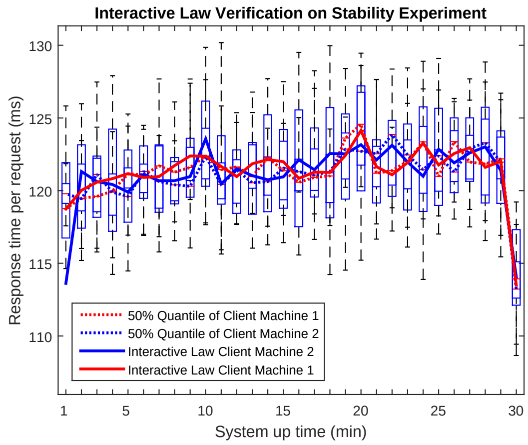

Figure 17: IL on Stability

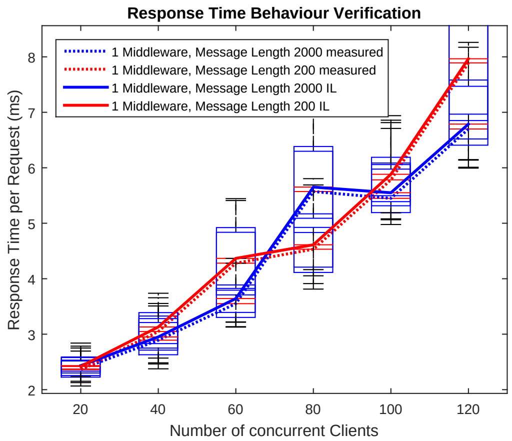

Figure 18: IRL on RT Experiment 1 MW

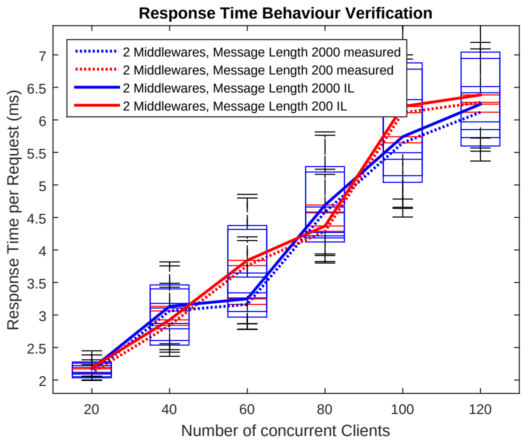

Figure 19: IRL on RT Experiment 1 MW

TP(Req/s)	RT(ms)	IRL(ms)	Abs. Difference(ms)	Percentual Difference(%)
12427	5.12	4.83	0.29	5.66
14707	3.87	4.08	0.21	5.43
11859	4.84	5.06	0.22	4.55
16133	3.66	3.72	0.06	1.64
12413	9.98	9.67	0.31	3.11
18233	7.07	6.58	0.49	6.93
13238	9.14	9.06	0.08	0.88
16428	6.97	7.30	0.33	4.73
13488	4.69	4.45	0.24	5.12
15954	3.87	3.76	0.11	2.84
12289	4.94	4.88	0.06	1.21
15855	4.19	3.78	0.41	9.76
13017	9.16	9.22	0.06	0.66
17701	6.57	6.78	0.21	3.20
13054	9.31	9.19	0.12	1.29
17619	6.73	6.81	0.08	1.19
			avg: 0.21	avg: 3.64

And last, but not least, let's quickly go through the 2^k verification via the interactive response time law. We can see, that the calculated and measured points are fitting well onto each other. On average we have $< 5\%$ of difference, which indicates that the law also holds here.

Appendix: Repeated Experiments

As told in section 1, the measured data from milestone one was not sufficient to do a modelling of the system. Thus, the two experiments stability and scalability were rerun. Let's first look at the stability experiment. Since this experiment did not change in setup, configuration or evaluation, here, just the main facts are repeated for locality and practicality reasons.
Configuration: The stability trace was run with one database, two middlewares and two client machines, each providing 60 clients which send requests with a content length of 200. In total, 40 database connections were distributed. Both of the middlewares got 20 each. The database was initially filled with 300'000 messages to simulate a more real-world behaviour. In contrast to the long stability trace done in the first milestone, this experiment was only run for 120 seconds. This is ok, because in the first experiment the system was already stable after 10 seconds and thus it seemed enough to just get 100 seconds of valid stable data (when ignoring the first and last 10 seconds because of warm-up and cooldown phases).
Expectation: The system was already confirmed to be able to run stably under this configuration, so there is no reason to expect something else.
Reality (figures 1 and 2): When comparing the results from this stability trace and the one done in milestone one, it's evident that there is a big mismatch with respect to throughput and response time. There was a constant shift down to 56% of the throughput. This can be explained by either more network usage by other customers or a bigger distance between the machines. The important fact here though is, that the system is indeed stable again and thus is valid as input data for this modelling task. Sadly because of limited time and money budget I was not able to run the experiment again and thus had to work with this data. For the sake of not repeating plots or data, the interactive law verification is already shown in figure 2.

As mentioned, the second experiment repeated for this milestone was the scalability run, let's go over its setup and outcome, too. The scalability experiment was not run explicitly enough in the first milestone and with the focus ond the wrong factors. This time the main factors numbers of middlewares and clients were closely analysed.
Configuration: The goal of this experiment is to find out the impact of how different system sizes do influence the overall performance. To achieve this, either one or two middlewares were active, serving in total 10, up to 120 clients. Each of these clients followed the same workload procedure and sent messages with a content length of 200. Since it was already clear that the best configuration for the system includes 40 concurrent database connections this was fixed throughout the whole experiment. If there were two middlewares running, both got 20 connections for their internal connection pool.
Expectation: In the first milestone the maximal throughput yield a throughput knee with 60 clients, after that the throughput stagnates and only the response time per requests grows. It thus can be expected, that this behaviour should also be visible in this experiment. Estimating the number of requests per second is a bit tricky, since in the previous experiment there were some inconsistencies with respect to data earlier measured. It's known, that the database as limiting system piece maximally allows $\approx$ 20'000 requests per second. Since the overall system performance seems to be lower, the expected throughput lies at around $\approx$ 11'000. Because of the database limitation there should be no difference in throughput between the one or two middlewares setup.
Reality (figures 20 and 21): It's great, that the expectation could be verified, that the number of middlewares is not important, because the database cuts off earlier. The predicted knee at 60 clients is also visible and makes the experiment a success. The maximal number of requests per second was underestimated though. In the previous experiment, the database was not empty

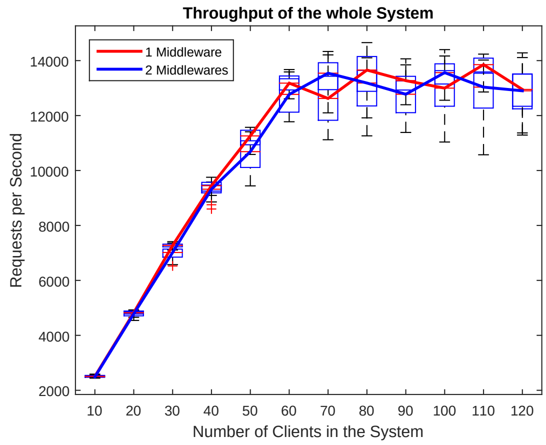

Figure 20: Throughput over the whole system.

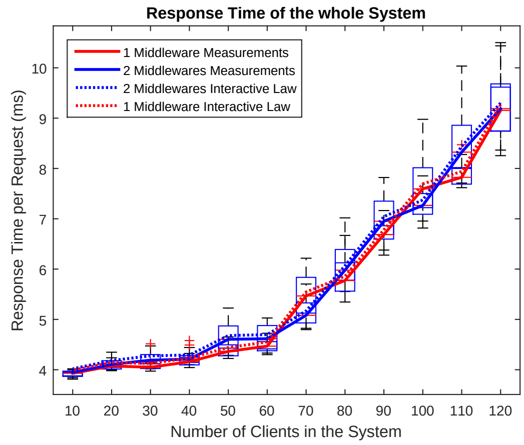

Figure 21: Response time over the whole system.

as in this one, so there are probably other hidden factors which influence this drop down. But overall the behaviour is the same as expected and is usable for the upcoming modelling tasks. For convenience, the interactive law is already plotted in figure **??**.

Because of incomplete logs in the maximum throughput experiment of the first milestone I was forced to redo the experiment.

Configuration and expectation: Because there was no change here, please refer to the first milestone.

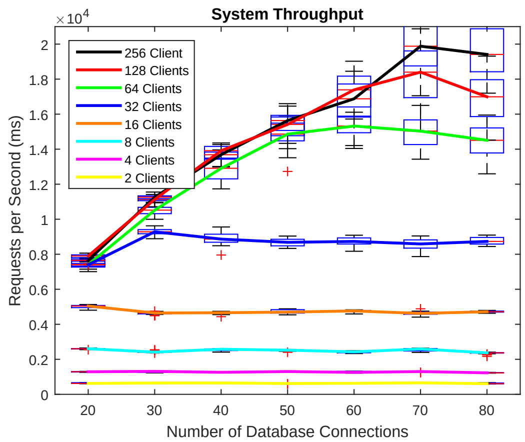

Figure 22: Throughput of the whole system

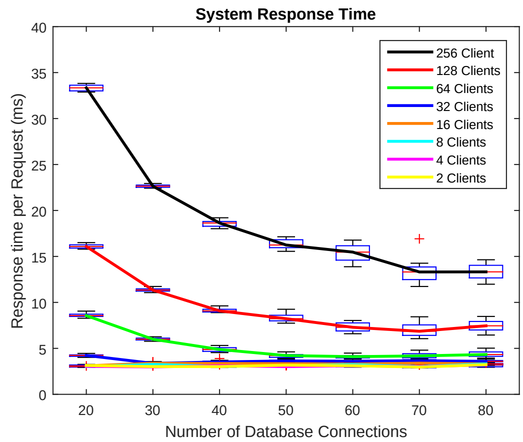

Figure 23: Response time of the whole system

Reality (figures 22 and 23): Nothing did change on the outcome.