Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Analytical Queuing Model for Telesto

Report Group 32

Dominic Langenegger

dominicl@student.ethz.ch

Advanced Systems Lab 2013
Systems Group
ETH Zürich

Supervisors:
Markus Pilman
Prof. Dr. Gustavo Alonso

December 20, 2013

Abstract

This report summarizes the work of analytically modeling and evaluating *Telesto*, the Distributed Message Passing System built by Simon Marti and Dominic Langenegger during the first milestone of the course project of the *Advanced Systems Lab* course 2013 at ETH Zurich. All relevant information about the first milestone can be found in [1].

Used concepts, mathematical formulas and theoretical aspects are heavily based on [2] and the lecture slides.

Contents

Chapter 1

Analytical Queuing Model

This chapter introduces the analytical queuing model built to model the characteristics of *Telesto*. It also includes explanations about simplifications and assumptions that were made.

We recall Table 5.3 in [1], showing in which parts of the system a request spends how much time. Table 1.1 shows very similar data (taken from an other experiment) and includes dispatching time.

Phase	Time	Relative
Dispatching	12 µs	0.14%
Parsing request	14 µs	0.17%
Waiting for database	8.059 µs	99.40%
Responding	22 µs	0.27%

Table 1.1: Time spent on various tasks by middleware workers

Based on this data we can safely say, that the time a request is handled by the middleware is minimal in comparison to the time spent to actually handle it on the database tier. But still we know, that the tasks the worker threads fullfil is also load dependent and should be modeled too.

Figure 1.1 shows the network model of *Telesto* including all important parts. As explained below, the clients themself are not modeled as separate delay centers but directly influence the network think time z.

1.1 Workload & Model

As seen in figure 1.1, we use a closed queuing network to model *Telesto*. Since all conducted tests during milestone 1 (see [1] for more details) operate with clients that only send new requests as soon as a previous on is completed, the number of jobs in the system at any time is directly corresponding to the number of clients and therefore fixed.

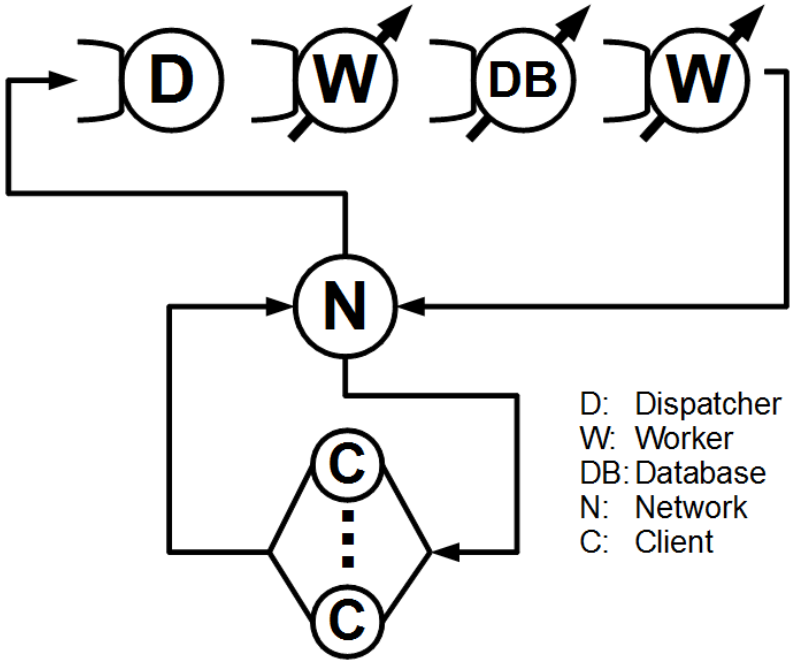

Figure 1.1: Network model of the closed queuing system for *Telesto*

It is important to note, that some clients may query the middleware to retrieve a message when actually no message is waiting for them. Based on the experiments from milestone 1, we know that this is the case for roughly one sixth of all message requests. Notice also, that the network is only a delay center and therefore does not have a queue in the figure.

1.1.1 Clients

Our queuing network does not include any modeling of the clients. This is because the time a job actually spends at the client is minimal since a client has only the following very simple tasks (that take almost no time):

(a) Send a message to a middleware

(b) Wait for a message

(c) Receive the message

(d) Start from (a); possibly using some information of the received message

The actual time a job is being processed at the client is in steps (c), (d) and (a), which is comparable in workload to the tasks of the middleware for message processing as seen in table 1.1, excluding the part where the middleware waits for the database. Since this takes very little time compared to the time spent processing a job at the database, it is save to simplify the model by not modeling the clients as separate load centers.

However the think time of the network can be determined by including the client retry delay of 100 ms into the calculation. If we assume that (as stated above) roughly one sixth of all client queries end up with no message as a result (and therefore the clients waits those 100 ms), we get a think time T of about $\frac{100ms}{6} \approx 17ms$.

1.1.2 Network

In *Telesto*each client request actually passes the network in four different phases:

1. Client → Middleware
2. Middleware → Database
3. Database → Middleware
4. Middleware → Client

Since we cannot measure the time spent in the network for the connections to the database individually, we don't model the network in between the middleware and the database separately.

For the connection from the clients to the middlewares we can assume that all messages have about the same size since all conducted tests use messages that are never longer than about 10 characters. This means, that we can assume the network delay is constant for every phase and not load dependent. Therefore it can simply be modeled as a delay center in the network model.

1.1.3 Middleware

The middleware consists of a single dispatcher thread that handles all incoming connections and puts incoming data into a queue. This can be modeled as a simple fixed-capacity server.

The many worker threads in a *Telesto* middleware instance, are modeled as load-dependent servers that then pass the job into a queue to the database connection pool. Since the thread pool scales up to the number of threads it is trivial to see, that this is load-dependent because multiple jobs can be processed in parallel.

The response from the database is then again packed into a *Telesto* network packet to be sent to the client. This again happens by the same worker thread that handled the database call. Therefore this can also be modeled load-dependent with the same properties as the parsing part of the workers.

1.1.4 Database

Every worker thread in the middleware retrieves a database connection from a theoretically independent database connection pool and uses it to execute a query on the database. Each of these connections is modeled as a separate load dependent server.

Notice that the database itself may not scale very well with increasing number of connections since its parallelism is very limited. However we think that it is very hard to model this by directly relating e.g. the number of cores of the machine to the number of service centers since the database may very well scale to many more connections than cores for several reasons like idle time and partial occupancy or internal parallelization optimization.

We modeled some overhead (10%) with high number of connections and capped the parallelization effect at 25 connections, the point where adding more connections didn't improve throughput anymore during tests in milestone 1. Because it is very hard to actually determine such effects in real systems, these are values chosen by good faith in order to optimize the simulation results in the right direction.

The following formula was used to model the order of parallelism of the database based on the number of database connections c:

$$\text{Order of Parallelism}(c) = \begin{cases} c & c \leq 15 \\ 0.9 \cdot c & c > 15 \wedge c \leq 25 \\ 0.9 \cdot 25 & c > 25 \end{cases}$$

This formula is used to get the value for the order of parallelism in the μ function for the service rate in the load dependent center for the database. For details about this service rate function are listed in section 1.3.1.

1.2 Notation

This section provides a short overview of the used notation as seen in [2].

Arrival Process

τ_i is the **interarrival time** between two arrivals at t_{i-1} and t_i.

Service Time Distribution
The distribution of time spent by a job in a specific service.

Number of Servers
The number of identical and independent servers for one particular service

System Capacity
The maximum number of jobs allowed in the system. This is usually always ∞.

Population Size
The maximum number of potential jobs that can ever enter the system. This is usually always ∞.

Service Discipline
The order jobs are served from a queue like *First In, First Out* (FIFO), *Round Robin* (RR) or *Last In, First Out* (LIFO).

1.3 Parameters

Every device and the whole system modeled in the analytical queuing network for *Telesto* is specified by some parameters. These are (based on [2]):

Service Type t
The type of the device. I.e. fixed capacity, delay center or load dependent

Service Time s
The time needed to complete one job by a server

Visits v
The number of visits to a device

Service Rate μ
The service rate of a device given the number of jobs in it. This measure is used for load dependent servers only.

Number of Clients n
The number of clients in the system sending requests.

Think Time z
The additional delay between two requests. In *Telesto* a client only waits for a certain *client retry delay* of usually $100ms$ if a query for a message didn't return any message, which is the case in roughly 25.8% of all queries as seen in table 1.2.

$$\text{Think Time } z := \text{Retry Delay} \cdot 25.8\%$$

To determine these values for all components of the network model, we used the data from our benchmarks during milestone 1 as shown in table 1.2.

Measure	Value
Mean Packet Throughput	2498 s^{-1}
Mean Message Throughput	927 s^{-1}
Mean Response Time	10.16 ms
Mean Packets With Wait Time	$2498 - 2 \cdot 927 = 644 s^{-1} \approx 25.8\%$
Mean Client Wait Time (Think time)	$100ms \cdot 25.8\% = 25.8ms$

Table 1.2: Important measurement data of *Telesto*

Table 1.3 shows the values for each of these parameters.

	Service Type t	**Service Time** s	**Visits** v	**Think Time** z
Client				25.8ms
Network	Delay Center	1.30ms	2	
Dispatcher	Fixed Capacity	0.01ms	1	
Worker In	Load-Dependent	0.01ms	1	
Worker Out	Load-Dependent	0.02ms	1	
Database	Load-Dependent	8.01ms	1	

Table 1.3: Parameter values for devices in the *Telesto* network model

1.3.1 Service Rate μ

The service rate has to be treated specially for the load dependent centers at the database and the middleware. For this we used the simple μ function introduced in the literature [2] modeling a direct and overhead-less parallelization of multiple executions up to some order of maximum parallelism m. The following function directly calculates the service rate μ for some number of jobs n in the load dependent center, an order of parallelism m and a mean service time S.

$$\mu(n) = \begin{cases} n/S & n < m \\ m/S & n \geq m \end{cases}$$

As mentioned above, we use a special values for the order of parallelism m in the load-dependent center modeling the database in order to simulate parallelization overhead and a maximum order of parallelism.

For the worker threads in *Telesto* we directly use the number of worker threads as order of parallelism since the high percentage of idle time makes parallelization have less overhead and the very low execution times anyway lead to low impact on the whole simulation.

Chapter 2

Analytical Analysis

Based on the parameters and model introduced in the previous chapter this chapter gives details about the executed performance analysis.

2.1 Algorithm

To perform the analysis of the whole closed queuing network of *Telesto* we use Mean-Value Analysis (MVA) as introduced in section 34.2 of [2]. Since we have multiple load-dependent centers within our model, we in particular use the *MVA Including Load-dependent Centers* algorithm from Box 36.1 in [2].

For the implementation of the load dependent MVA algorithm, we used a small Java application that simulated the same datapoints we measured during various executions of the real *Telesto* system in milestone 1. Refer to table 1.3 for details about the used model parameters.

2.2 Queuing Network

The following subsections show the results of our simulations in comparison to some of the measurements and benchmarks we did on *Telesto* during milestone 1. The graphs were produced with `gnuplot`[1] as those for milestone 1.

For simplification and readability, we omitted the error bars of the measurements. Please refer to [1] for further details about the measuring methods and result data in milestone 1. The throughput is always given in messages per second.

Also the following sections only contain a analysis of the simulated results in comparison to the measured values and a detailed discussion about why the differences exist and how they might be reduced is given in chapter 3.

[1] Gnuplot Website
Available at: http://www.gnuplot.info/ [Accessed December 20, 2013]

2.2.1 Combined Database Connection and Worker Thread count

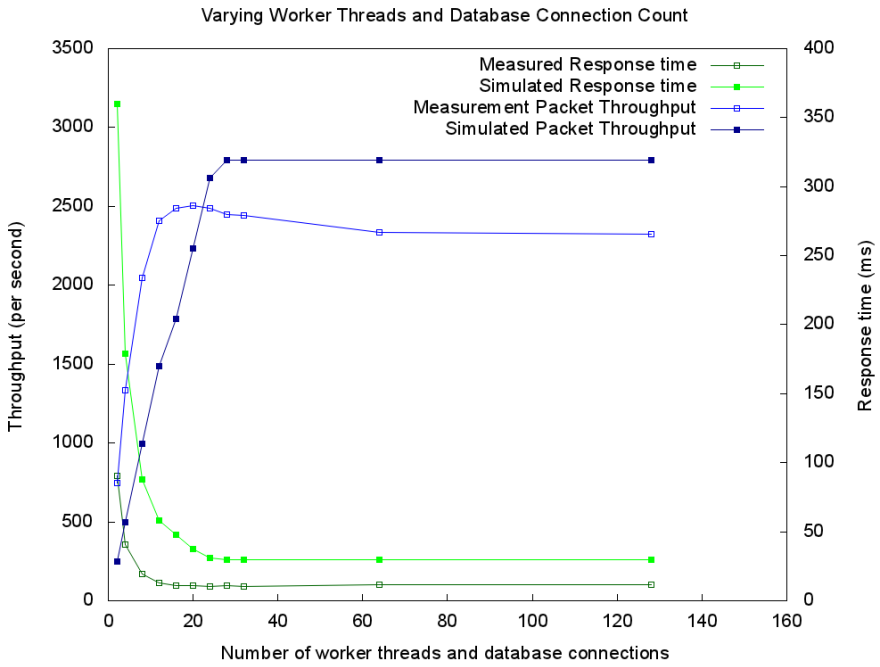

Figure 2.1: Varying number of worker threads and database connections

Figure 2.1 shows the comparison of both the response time and the throughput of the simulation and experiments varying the number of worker threads and database connections simultaneously.

We see that the simulation roughly fits the measurements of the real *Telesto* system. The main differences are the extrema that are about 8% off for the throughput and nearly trippled for the response time. Also they are reached at a later point than with the real world measurements. Over all, all simulated values start at higher values for the response time and lower values for the throughput but then follow approximately a similar slope like the comparison data.

An other minor difference is, that the throughput does not show a clear and single maximum like in the measured values but stays constant after reaching the maximum value of 2792 packets per second.

2.2.2 Client Count

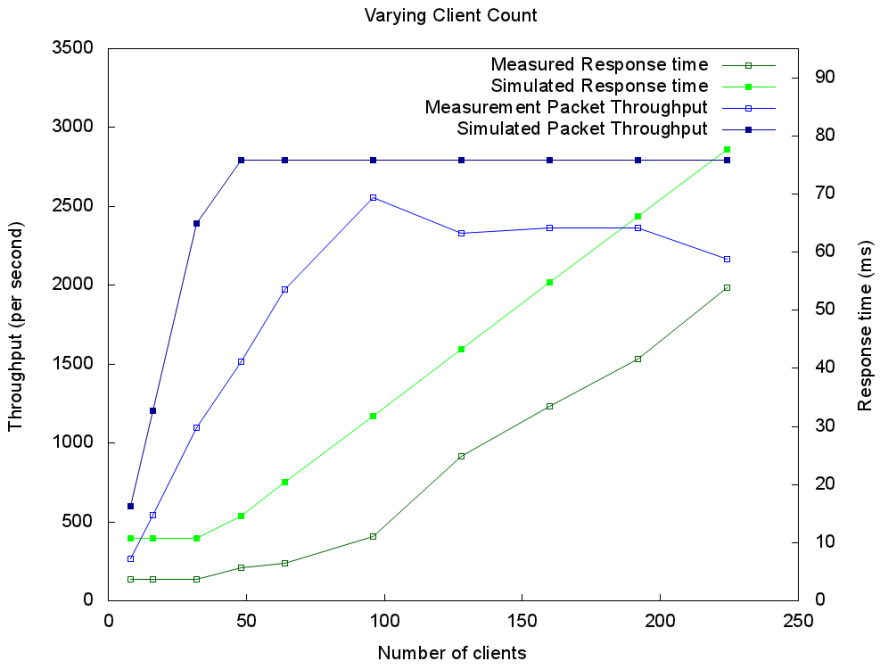

Figure 2.2: Varying client count

In figure 2.2 we plotted the response time and throughput for *Telesto* when varying the number of clients and therefore jobs in the network.

We can again say, that the curves take approximately the same development. The maximum of the throughput is about 9% off for the throughput while the response time has a very similar slope but is constantly off by about $20 - 25$ ms. Also we see, that the MVA model is limited in a way that it cannot simulate the descending throughput after a certain amount of clients are surpassed. (i.e. 100 clients)

Again the starting points are already not fitting the real values which itself produces a high offset.

2.2.3 Client Retry Delay

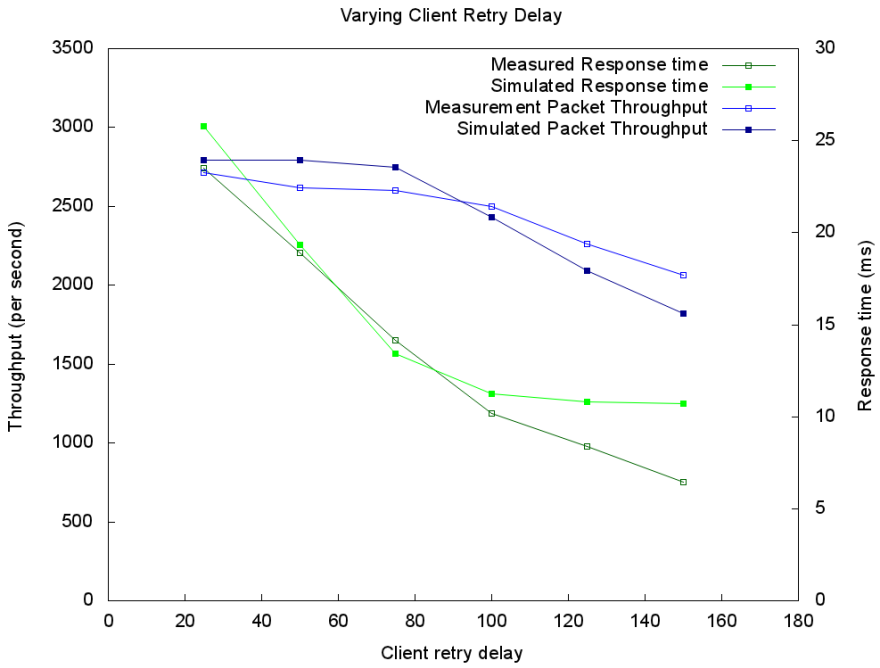

Figure 2.3: Varying client retry delay (i.e. think time)

Since the network think time is directly dependent on the client retry delay in *Telesto*, we also simulated this experiment with our MVA model. The results are shown in figure 2.3.

As presented in section 1.3, the client retry delay directly affects the think time of the modeled network. This can very nicely be seen in this comparison because we actually get very similar data for our simulation and the measurements from milestone 1.

Especially the retry delay in the interval of $75 - 100$ ms which leads to the best performance, produces very similar results with an offset of less than 7% and 3% for 75 and 100 ms respectively in throughput and less than 6% and 11% in response time.

An other observation is that the measured response time continues to decrease with higher retry delay while the simulated one evens out at around 10.7 ms. This is most likely because the probability of having to wait that long for a new message to be accessible is very low and doesn't behave in such a linear manner as it was modeled in the network model.

2.2.4 Bottleneck

For bottleneck analysis we use the calculated utilization values from our MVA simulation. We used the standard configuration with 20 worker threads and database connections, and 90 clients which is the best performing configuration in our *Telesto* benchmarks in milestone 1. Table 2.1 shows the calculated data where we can clearly recognize that the database is as expected the bottleneck of the system.

Center	Utilitzation
Dispatcher	2.7 %
Middleware In	3.1 %
Database	100 %
Middleware Out	4.8 %

Table 2.1: Utilization of the load centers in the MVA simulation

2.3 Database

To get further information about how the database behaves during execution, we use a M/M/m queue model as described in [2] and feed it with our measured values for the reference test of *Telesto* with 90 clients and 25 database connections.

We will do such an analysis only for the database part of *Telesto* since the middleware has such low utilization that it wouldn't really produce very interesting results. Our assumptions are that the middleware would have a very high probability for very low number of jobs in the queue, very low traffic intensity and practically no waiting time.

In the following we state the formulas and calculations results directly as listed in Box 31.2 in [2]. The calculations were done using a small python script that makes use of the `decimal`[2] package in order to achieve very high precision with floating point numbers. A first try with a Java program using normal `double`s yielded highly unstable results due to rounding errors and bad accuracy.

$$\begin{aligned} \text{Arrival rate} &\quad \lambda = 2498 \text{ jobs} \cdot s^{-1} \\ \text{Service rate} &\quad \mu = 124.1 \text{ jobs} \cdot s^{-1} \\ \text{Number of Servers} &\quad m = 25 \\ \text{Number of Jobs / Clients} &\quad n = 90 \end{aligned}$$

[2]Python Documentation: decimal Package
Available at: http://docs.python.org/2/library/decimal.html [Accessed December 20, 2013]

While the value for λ was directly extracted from the throughput measurements, μ was calculated using the mean service time for the database as already used for the MVA network simulation. The necessary calculations are shown below:

$$\mu = (0.008059s)^{-1} = 124.1 \text{ jobs} \cdot s^{-1}$$

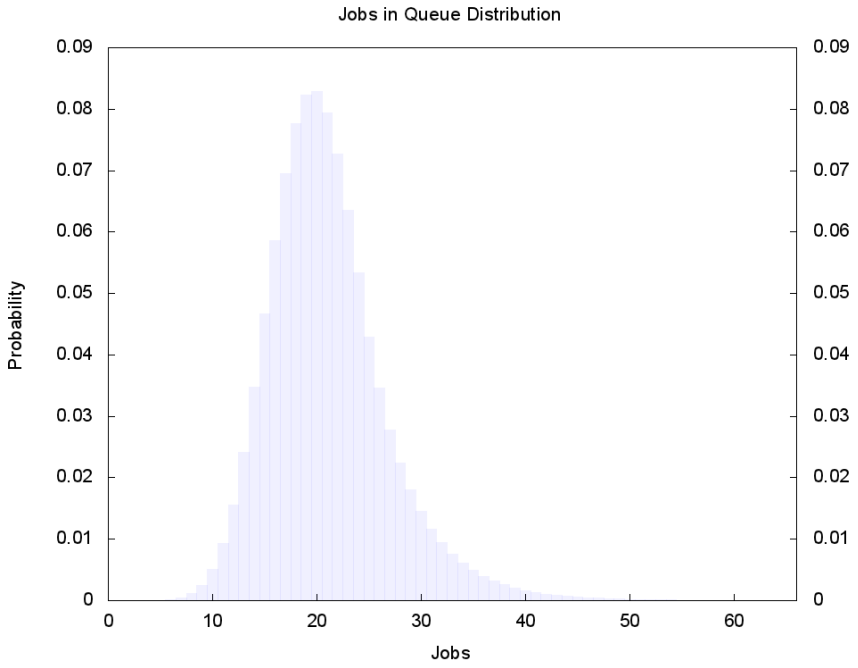

Figure 2.4: Distribution of probabilities for a certian number of jobs in the system

Traffic Intensity:

$$\rho = \frac{\lambda}{m \cdot \lambda} = 0.805$$

Probability of 0 jobs in Queue:

$$p_0 = \left[1 + \frac{(m*p)^m}{m!*(1-p)} + \sum_{n=1}^{m-1} \frac{(m*p)^n}{n!}\right]^{-1} = 1.690 * 10^{-9}$$

Probability of n jobs in Queue:

$$p_n = \begin{cases} p_0 * \frac{(m*p)^n}{n!} & n < m \\ p_0 * \frac{p^n * m^m}{m!} & n \geq m \end{cases}$$

This formula yields figure 2.4 which roughly fits a normal distribution with highest probability for 21 jobs with a value of approximately $0.083 = 8.3\%$.

Probability of queuing:

$$\varrho = P(\geq m \text{ jobs}) = \frac{(m * \rho)^m}{m! * (1 - \rho)} * p_0 = 0.220 = 22.0\%$$

Mean number of jobs in the system:

$$E[n] = m * \rho + \rho * \varrho/(1 - \rho) = 21.039$$

Variance of number of jobs in the system:

$$Var[n] = m * \rho + \rho * \varrho * \left[\frac{1 + \rho - \rho * \varrho}{(1 - \rho)^2} + m\right] = 242.363$$

Mean number of jobs in the queue:

$$E[n_q] = \rho * \varrho/(1 - \rho) = 0.910$$

Variance of number of jobs in the queue:

$$Var[n_q] = \rho * \varrho * \left[\frac{1 + \rho - \rho * \varrho}{(1 - \rho)^2}\right] = 7.605$$

Average utilization of each core (i.e. database connection):

$$U = \lambda/(m * \mu) = \rho = 0.805 = 80.5\%$$

Mean response time:

$$E[r] = \frac{1}{\mu} * (1 + \frac{\varrho}{m * (1 - \rho)}) = 8.4ms$$

Variance of the response time:

$$Var[r] = \frac{1}{\mu^2} * \left[1 + \frac{\varrho * (2 - \varrho)}{m^2 * (1 - \rho)^2}\right] = 0.066ms = 66\mu s$$

Mean waiting time:

$$E[w] = E[n_q]/\lambda = \varrho/[m * \mu * (1 - \rho)] = 0.364ms$$

Variance of waiting time:

$$Var[w] = \varrho * (2 - \varrho)/[m^2 * \mu^2 * (1 - \rho)^2] = 1.072\mu s$$

This implies, that most jobs have to wait no more than about $1.3ms$. Details can be calculated using the $90-$ and $99-$percentile as below:

90-Percentile of the waiting time:

$$\frac{E[w]}{\varrho} * ln(10 * \varrho) = 0.0013s = 1.3ms$$

99-Percentile of the waiting time:

$$\frac{E[w]}{\varrho} * ln(100 * \varrho) = 0.0051s = 5.1ms$$

This means only 1% of all jobs have to wait more than 5.1 ms and 10% more than 1.3 ms.

As expected the utilization of the database with the selected input parameters is quite high while the waiting time is still rather low. We expect this to raise with increasing load. We also see a rather good correlation of the calculated values with the real ones, given that we had very limited modeling parameters.

Chapter 3

Conclusion

Given that we only used the measured times of one execution of *Telesto* as input for the whole network model and for simulation of completely different configuration than those used during the reference measurement, the results were pretty good.

The following sections will show some limits of the simulations and list some possible improvements.

3.1 Limitations of MVA

First of all, it is very hard to model such a complex system with such a simple model that basically only takes runtimes of separate components as input.

Some factors like blocking, contention, mutual exclusion or non-exponential service times can simply not be modeled with the limited MVA model.

Then there is the problem of accurately selecting a μ function to model the load dependent behaviour of the database and the workers. In case of the workers, it is nearly irrelevant what is selected because they only make a very small part of the entire response time. But with the database we have a enormously complex software component that contains various optimizations, does many unpredictable memory reads and is just to complex as that if we could simply say it scales up to 25 connections in parallel but with 10% overhead.

We at this point assume, that most of the difference between our simulation data and the measured data is due to a too simplistic model of the database behaviour. However it is not the scope of this work to improve that model.

In the usual case one would first do some very limited measurement with the system in order to have ground truth data for this model. There wouldn't be the whole set of real data to compare to and therefore we wouldn't be able to improve the model and change parameters in order to achieve a better fit. This is exactly why we didn't do this because this is not the goal of this simulation process.

A second factor that impacts the correlation of our simulated data, is of course the think time of the network. Our estimation is very vague and holds only in a very small subset of the possible configurations. In order to improve the predictability of this parameter, it would have been better to only use *Telesto* tests with exactly one type of client and a guarantee of always getting a message back together with a constant waiting time after every response. This would lead to a predictable and balanced workload which is free from fluctuations.

In the end we think the queuing network models as seen in [2], are quite useful for approximate simulation of systems that cannot be tested in real very easily. There is a big information source in modeling a system like that if it isn't already built completely or if further optimization options want to be explored without doing extensive real world testing.

3.2 Multiple Middlewares

This report does not include any simulation of *Telesto* with multiple middlewares because it was relatively clear, that

1. there wouldn't be any significant performance difference because the database is the bottleneck
2. the model would get even more complicated with multiple middlewares.

We therefore think that there is no real value in simulating this case and omitted it for this report.

Bibliography

[1] Langenegger, D., Marti, S.: Telesto - A Distributed Message Passing System. Advanced Systems Lab, ETH Zurich (November 2013)

[2] Jain, R.: The Art Of Computer Systems Performance Analysis:. Wiley India Pvt. Limited (2008)