ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Paštas: The Distributed Messaging System

Advanced Systems Lab, Milestone 2 Report

Martynas Pumputis

December 18, 2014

Advisor: Zsolt István

Department of Computer Science, ETH Zürich

# Contents

Chapter 1

# Introduction

This is a report for Milestone 2 of Advanced System Lab course project work. The report presents series of models and an extensive analysis of the system presented in Milestone 1. Most of the analysis in this report relies on the experimental results conducted in Milestone 1.

## 1.1 Overview

Firstly, in Chapter 2 and Chapter 3 we present `M/M/1` models of the system. Next, Chapter 4 introduces `M/M/m` models of middleware and database. Further on, in Chapter 5 we present fine-grained model of the entire system on which we apply MVA analysis. Later, Chapter 6 presents validation of Milestone 1 test experiments. Next, in Chapter 7 we analyze the impact of system configuration parameters by conducting $2^k$ analysis. Finally, Appendix C contains an experiment in which we exploit the system's behaviour with clients number higher than 30.

Chapter 2

# System as One Unit

In this section we present a coarse-grained model of the system. The evaluation of the model is based on trace of the stability test.

## 2.1 Model

As a first step, we decided to model the entire system as a single-processor system. Formally speaking, it can be described as a `M/M/1` queue where the first two `M's` means that the time between successive arrivals and the service times are exponentially distributed and `1` denotes that there is only one processing server.

Although it is not explicitly stated in the formal definition, the modeled system consists of an unbounded queue and there are no limitations for population size. The service discipline is `FCFS` which means that requests are served in *First Come, First Served* manner.

## 2.2 Evaluation

To evaluate the described model, we choose a data from the first stability test and the scalability test conducted in Milestone 1 (Section 6.2 and Section 6.3.1 in Milestone 1 respectively).

To recap, the stability test was performed with the configuration described in Table 2.1. The measured average throughput in the test is used as an arrival rate parameter in the evaluation which is a case for the closed systems because all jobs complete correctly. Thus, $\lambda = 2011.28 \frac{jobs}{sec}$.

| Parameter | Value |
|---|---|
| Number of client nodes | 1 |
| Number of client nodes | 1 |
| Number of clients | 10 |
| Number of middleware workers | 10 |
| Number of database connections | 10 |
| Number of queues | 2 |
| **insert_message** | |
| Probability for request | 0.2 |
| Probability for 200 and 2000 byte message | 0.5, 0.5 |
| Probability for broadcast message | 0.1 |
| **read_message** | |
| Probability for request | 0.6 |
| Probability for removing message | 0.9 |
| **read_message_broadcast** | |
| Probability for request | 0.2 |
| Probability for removing message | 0.9 |

Figure 2.1: Configuration parameters for stability test

For the service rate parameter ($\mu$), we chose a maximum achieved throughput from the scalability tests for the same configuration which is equal to $2133.84\frac{jobs}{sec}$. We base our decision on the following facts:

- All jobs finish correctly.
- Throughput is stable, i.e. does not change during the run.
- There are always jobs ready after any job has been processed.
- Throughput does not depend on the number of jobs in any queue.

However, we could measure the service rate in a different way. For example, one could run the test with a single client (at most one job in the system) and multiple the obtained service rate by some constant because of the parallelism we have in the system. But in that case, the constant should have been derived by analyzing the system. Thus, the assumption of the system as a blackbox would be broken.

Having defined the parameters, we can calculate the following values:

- **Traffic intensity:** $\rho = \frac{\lambda}{\mu} = 0.94$. The intensity is less than 1, therefore the system fulfils the stability condition.
- **Probability of zero jobs in the system:** $p_0 = 1 - \rho = 0.06$.

- **Mean number of jobs in the system:** $E[n] = \frac{\rho}{1-\rho} = 16.40$. In our case, we had 10 clients which means that there cannot be more than 10 jobs in the system. Therefore, the estimation has an error which is $\geq 6.40$ jobs.
- **Variance of number of jobs in the system:** $Var[n] = \frac{\rho}{(1-\rho)^2} = 285.67$. The variance cover the error we defined in estimation above.
- **Mean response time:** $E[r] = \frac{(1/\mu)}{1-\rho} = 8.16ms$. The measured response time is equal to $4.92ms$ with standard deviation of $4.49ms$.
- **Variance of the response time:** $Var[r] = \frac{1/\mu^2}{(1-\rho)^2} = 66.57\mu s$.
- **95-Percentile of the response time:** $E[r]ln[100/(100-95)] = 24.44ms$. The measured 95-Percentile is equal to $13.69ms$.
- **Mean waiting time:** $E[w] = \rho\frac{1/\mu}{1-\rho} = 7.69ms$.
- **Mean number of jobs in the queue:** $E[n_q] = \frac{\rho^2}{(1-\rho)} = 15.46$.

## 2.3 Analysis

As it can be seen from the estimations presented above, the model is quite accurate considering the fact, that the real system consists of many queues and also it inherits some degree of parallelism. The estimated response time is `two` times higher compared to the measured one ($8.16ms$ and $4.49ms$ respectively). The estimated average number of jobs in the system is higher than the maximum number of clients, but the difference is not that big (16 and 10 respectively).

If the system utilization in stability test was close to 100%, i.e. arrival rate equals to maximum throughput, then `M/M/1` model would not have had the same level of accuracy because most of the provided formulas contains a division by $1-\rho$ which would be close to 0.

On the other hand, removing parallelism from the system would increase the accuracy of the model, because the model implies that there is a single processing device. However, such modification was out of the scope of this project.

Chapter 3

# Analysis of System Based on Scalability Data

In this chapter we present Mean Value Analysis for `M/M/1` models of the system. We evaluate the models by using the scalability test results.

## 3.1 Model

To begin with, we model the entire system as a `M/M/1` queue. To apply MVA analysis for the queue, we model the queue as a `fixed capacity device`. The service time of the device corresponds to the service time of either database or middleware's workers depending on the experiment.

## 3.2 Evaluation

To evaluate both models, we chose scalability test which is presented in Chapter 5. To recap, in the experiments we have fixed number of clients which is equal to `10` and either fixed number of database connections or worker threads which is equal to `10`.

## 3.3 Analysis

The first case in which we change the number of worker threads is depicted in Figure 3.1. As it can be seen, the analytical response time and throughput is at the same order of magnitude as the observed one. However, the analytical model does not properly detect the behaviour when there are $\leq 10$ threads in the system.

The second case in which the database connection number increases is shown in Figure 3.2. In this case, the analytical response time and throughput is

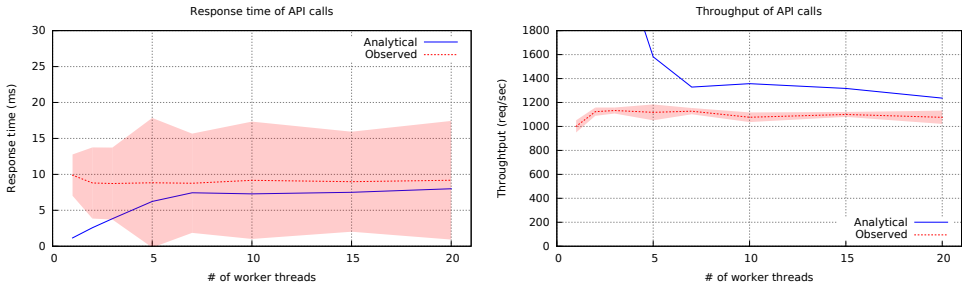


Figure 3.1: Response time and throughput of API requests when the number of worker threads increases.

completely mispredicted. The main reason is that database should be modeled as `load dependent device`, because its service time depends on number of active connections in our implementation.

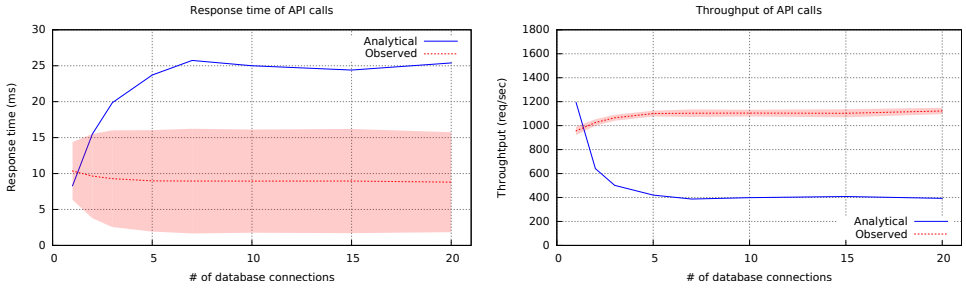


Figure 3.2: Response time and throughput of API requests when the number of database connections increases.

To conclude, only the second model shows that it is not possible to model the entire system as `M/M/1` queue which assumes a system consisting of a single processor which is not true for our design.

Chapter 4

# Model of the Components as Independent Units

In this chapter we present two M/M/m models built for database and middleware.

## 4.1 Models

M/M/1 models presented in previous chapters cannot take parallelism into consideration. The parallelism is inherit in two components of the system - middleware and database. The former can process multiple requests at once due to the workers thread and the latter is due to connection pool and PostgreSQL database (for further details of the pool and database analysis please refer to Chapter 5).

To analyze the components we chose M/M/m model where m stands for either number of worker threads or number of database connections.

## 4.2 Evaluation

To evaluate the models, we chose the scalability test from Chapter 5 where number of database connections and worker threads is fixed and equal to 10 while number of clients increases.

To estimate a mean response time of the each model we used the following formula:

$$E[r] = \frac{1}{\mu}(1 + \frac{\sigma}{m(1-\rho)}) \tag{4.1}$$

Where $\mu$ is mean service rate of component, $\sigma$ - probability of queueing, $\rho$ traffic intensity.

The arrival rate ($\lambda$) is equal to the throughput of the system and the mean service rate ($\mu$) is equal to inverse of response time of each component when there is one single client.

## 4.3 Analysis

Figure 4.1 and Figure 4.2 shows the measured and estimated throughput of the system where system is modeled either as the database or the middleware.

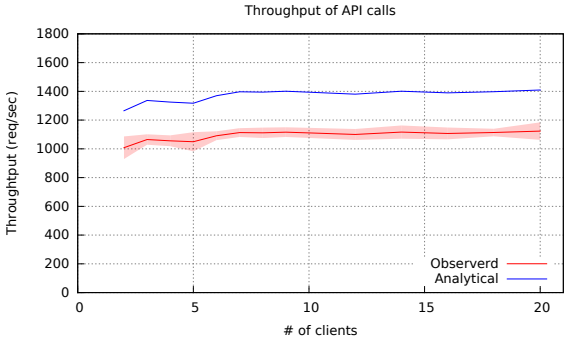


Figure 4.1: Throughput of the system including throughput of the model which consists of the database modeled as `M/M/m` queue.

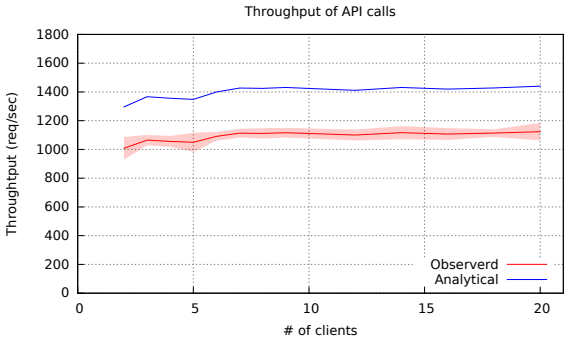


Figure 4.2: Throughput of the system including throughput of the model which consists of the middleware modeled as `M/M/m` queue.

Although the given models are still coarse-grained and do not cover the whole complexity of the system, the predicted throughputs of the system are more precise than in `M/M/1` models presented in previous chapter.

Chapter 5

# Model of System as a Network of Queues

This section presents a fine-grained model of the system which is accompanied with Mean Value Analysis.

## 5.1 Model

In order to achieve more precise estimations, we modeled the system as a closed queueing network. The model is presented in Figure 5.1 and consists of the following components:

- `Clients`: clients are generating load by sending requests. A request is sent only after a response to a previous request has been received. Such behaviour is a requirement for closed queueing networks.
- `Network`: the network covers not only packets sent across the wire, but also it includes OS network handling stack and Java NIO part. Because the network can be parallelized up to some certain degree, we modeled it as a load dependent device. Although, the device represents communication which flows from clients to dispatcher and from workers to clients, we modeled it as a single device, because both paths inherits the same properties.
- `Dispatcher`: the dispatcher is modeled as a fixed capacity device, because we observed that its service time does not depend on a load and other parameters. Also, there is a single instance of the dispatcher per middleware node.
- `Workers`: the workers are represented as a load dependent device, because usually there are a few worker threads per middleware, therefore it can be treated as a parallelized entity. Also, a response to the client is sent by worker, although it is not shown in the model, but due

to simplicity reasons we decided not to split the worker up into two components.

- `Database`: the database is modeled as a load dependent service, because we have multiple connections to it. Also, the device includes networking part which happens between worker and database listener.

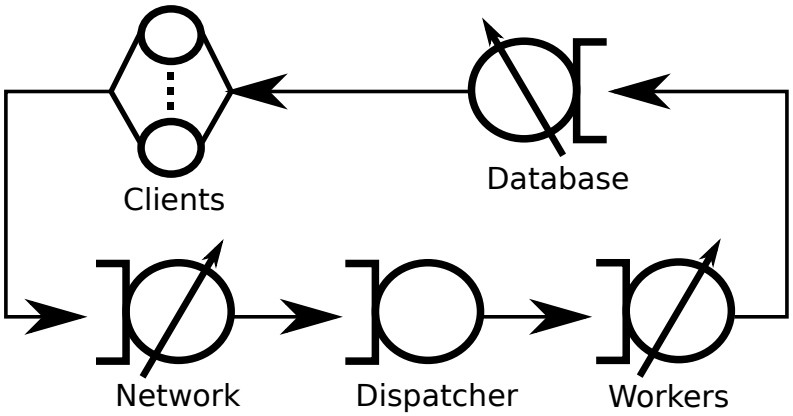


Figure 5.1: A network model of the messaging system

The model closely resembles the architecture of the system presented in Chapter 3 of Milestone 1.

It is worth to mention that PostgreSQL for each connection spawns a new thread and does not do any queueing of the requests. The load regulation such as pooling is up to the client side and in our case is done by `PGPoolingDataSource`. The library does not provide any fairness because it uses `lock` and `wait` Java synchronization primitives which implies that a connection is given to a waiting threads not necessary in `FCFS` order. Although this might increase a variance of client waiting time, we did not consider such effect when modeling the system.

## 5.2 Evaluation

First of all, to evaluate the model we had to know service time or service rate of each device. Unfortunately, the data gathered in Milestone 1 did not contain all required information, therefore we had to re-run some benchmarks after enabling more verbose logging. The re-run benchmarks include:

- Scalability test when increasing number of worker threads on a single node (Section 6.3.1 in Milestone 1).
- Scalability test when increasing number of database connections on a single node (Section 6.3.2 in Milestone 1)

- Scalability test when increasing number of clients on a single node (Section 6.3.3 in Milestone 1).

In all experiments we did not achieve the response time of Milestone 1. It might be due to significant increase in logging frequency which results in higher IO utilization. Amazon AWS has policies for IO which ensure that user gets limited amount of IO operations.

The common configuration parameters of the benchmarks are presented in Table 5.2.

| **Parameter** | **Value** |
|---|---|
| Number of client nodes | 1 |
| Number of client nodes | 1 |
| Number of clients (if not increasing) | 10 |
| Number of middleware workers (if not increasing) | 10 |
| Number of database connections (if not increasing) | 10 |
| Number of queues | 2 |
| **insert_message** | |
| Probability for request | 0.2 |
| Probability for 200 and 2000 byte message | 0.5, 0.5 |
| Probability for broadcast message | 0.1 |
| **read_message** | |
| Probability for request | 0.6 |
| Probability for removing message | 0.9 |
| **read_message_broadcast** | |
| Probability for request | 0.2 |
| Probability for removing message | 0.9 |

Figure 5.2: Common configuration parameters for re-run benchmarks

Next, the service rates of each device has to be set. We used the data from the scalability test in which we increase number of clients. Because we observed non-neglectable differences in service time of each load dependent device with different number of clients, therefore we did not use a mean service time and we used particular service time for a given number of clients in each MVA iteration.

The service rate of the network device can be expressed as:

$$\mu = \frac{min(C_{clients}, C_{workers}, C_{db})}{S_{network}(C_{clients})} \tag{5.1}$$

Where $C_{clients}$ stands for number of clients, $C_{workers}$ number of worker threads, $C_{db}$ number of database connections and $S_{network}(C_{clients})$ is service time of the network device when there is $C_{clients}$ numbers of clients. The motivation for limiting the service rate is that it is bounded by number of open connections.

The service rate of the workers device corresponds to:

$$\mu = \frac{min(C_{clients}, C_{workers})}{S_{workers}(C_{clients})} \tag{5.2}$$

Where $S_{workers}$ is a service time of the device when there is a given number of clients. The service rate is limited by needed amount of workers.

The database device has the following service rate:

$$\mu = Const * \frac{min(C_{clients}, C_{db})}{S_{db}(C_{clients})} \tag{5.3}$$

Because the database does not scale vertically with the increasing number of connections, therefore we need to limit the service rate by some constant which is $\leq 1$. We found out empirically that $Const = 0.6$ gives the most precise results.

Table A.1 in Appendix A contains a list of service times for each device including client think time.

## 5.3 Analysis

Figure 5.3 shows estimated and measured system throughput and response time when number of clients is increasing. The estimation was done by applying MVA algorithm. The number of clients and database connections is equal to `10`, while the number of middlewares is equal to `1`.

As it can be seen, the model roughly predicts the exact behaviour of the system, although the analytically derived response time is lower than the observed one which makes the predicted throughput higher. Also, the breaking point in analytical model at which the system's throughput becomes stable is when there are more clients in the system compared to the real behaviour.

Afterwards, we reuse the service times from the experiment above and by using MVA predict the system behaviour when there the number of workers

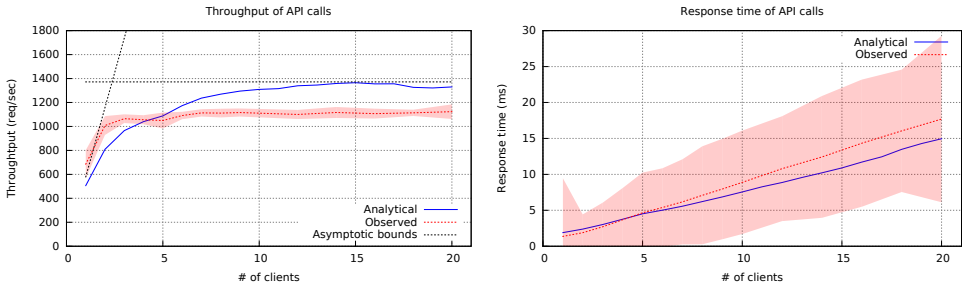


Figure 5.3: The estimated and measured throughput and response time of the system when increasing number of clients

and database connections increases. The results can be seen in Figure 5.4 and Figure 5.5 respectively.

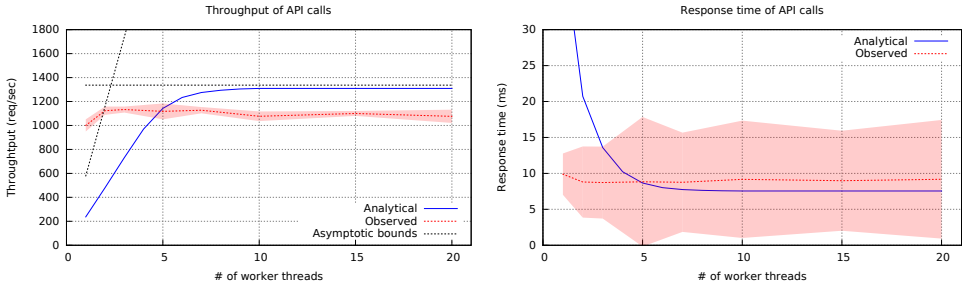


Figure 5.4: The estimated and measured throughput and response time of the system when increasing number of worker threads

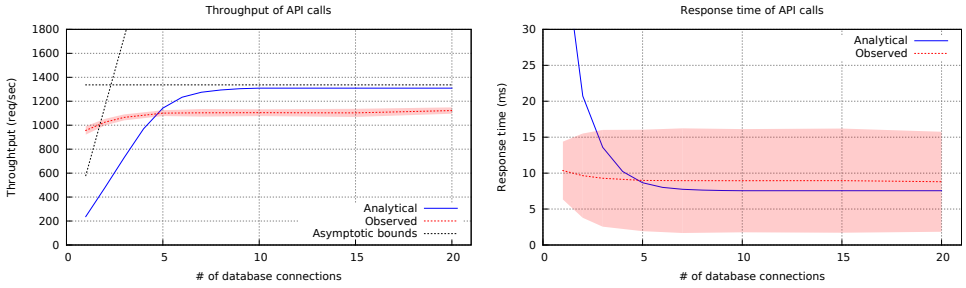


Figure 5.5: The estimated and measured throughput and response time of the system when increasing number of database connections

As it can be expected, the predicted behaviours are basically the same, because the number of workers and the number of databases have the same limiting effect, i.e. increasing number of workers does not improve system's

behaviour if the number of database connections is constant which is less than the count of workers and vice versa.

Next, the predicted behaviours do not map so precisely to the real one as it was in the previous test. The explanation for it is that the number of jobs in any device is not equal to the total number of jobs in system (in our case it is a number of clients) and there is no easy way to estimate that. Also, knowing the parallelization degree of network and database could improve the precision of the model.

## 5.4 Bottlenecks

Utilization of each device obtained by doing MVA when there were 20 clients is listed in Table 5.6.

| **Device** | **Utilization** |
|---|---|
| Network | 15.16% |
| Dispatcher | 99.84% |
| Worker | 16.54% |
| Database | 96.09% |

Figure 5.6: Utilization of each device obtained by MVA analysis

As it can be seen, the dispatcher has the highest utilization which implies that it is the bottleneck for the given configuration of the system. This observation corresponds to the findings in Milestone 1 where we empirically discovered that the dispatcher is bottleneck when middleware is running on a single node. This fact is due to our design decision in which we decided to offload reading and parsing of incoming requests to the dispatcher and there is only one instance of dispatcher per entire middleware.

Chapter 6

# Interactive Response Time Law

The interactive response time law states that if user generates requests which are served by a central system and a new request is submitted after `Z` think time after a response to the previous request has been received, the response time can be expressed as:

$$R = \frac{N}{X} - Z \tag{6.1}$$

Where `N` is number of clients and `X` is a throughput of the system.

By using the interactive law we validated results of the experiments conducted in Milestone 1. No discrepancies between measured and calculated response times were found in any experiment. Plots from Milestone 1 extended with response time from the law can be found in Appendix B.

Chapter 7

# $2^k$ Analysis

This chapter presents $2^k$ analysis which was conducted on our system in order to determine which configuration parameters have the highest impact factor on a performance of the system.

## 7.1 Factors and Levels

First of all, $2^k$ analysis starts by choosing factors. We decided to use the same factors as we used in scalability test in Milestone 1 (Section 6.2) to be able to compare the results from both experiments.

Thus, the factors include:

- Number of middlewares.
- Number of clients.
- Number of worker threads.
- Number of database connections.

Next, the levels of $2^k$ have to be decided. It is important to choose proper levels, because results might be biased by them. Knowing the results of scalability test in advance helped to select such levels that behaviour of all factors is monotonic in a range defined by the levels. Thus, we chose the following levels for each factor:

- `Middleware`: 1 and 2.
- `Clients`: 2 and 8.
- `Worker threads`: 2 and 8 (in total).
- `Database connections`: 2 and 8 (in total).

## 7.2 Results

Table 7.1 presents the results of $2^k$ analysis.

| Levels | Percentage of variation (STT) |
|---|---|
| c | 47.55 |
| m | 28.75 |
| mc | 20.81 |
| wd | 1.41 |
| mwd | 0.42 |
| md | 0.30 |
| cwd | 0.20 |
| mcw | 0.17 |
| mcwd | 0.12 |
| mw | 0.10 |
| cd | 0.05 |
| w | 0.04 |
| d | 0.01 |
| cw | 0.004 |
| mcd | 0.000002 |

Figure 7.1: $2^k$ analysis results. `m` stands for middlewares, `c` for clients, `w` for worker threads and `d` for database connections.

As can be seen, 47.55% of throughput is affected by number of clients. Second the most important factor is number of middlewares. It can be explained, that increasing number of middlewares eliminates the bottleneck of dispatcher. As expected, increasing number of database connections while not changing number of worker threads and vice versa does not help to improve scalability. On the other hand, increasing both numbers has an impact on the performance, but not so significant as two other factor because of the second bottleneck of our system - database.

Comparing the results with scalability test conducted in Milestone 1, we can see similar behaviours - number of middlewares and clients were two most important factors in the scalability test.

Chapter 8

# Conclusions

First of all, we presented coarse-grained models of the system based on `M/M/1` and `M/M/m`. The both types of models are not accurate enough to predict a behaviour of the system. The main problem with `M/M/1` model is that it is not able to take advantages of the system's parallelism in advance. Although the latter model assumes the parallelism, it cannot consider multiple interacting components within the system. Therefore, its accuracy suffers from it.

Next, we modeled the entire system as a network of the queues. The model was analysed by using Mean Value Analysis method. The results were really close to the real measurements. However, not having information how many jobs were at each device has impact on the precision of the model.

Further on, we identified the bottlenecks of the system. The first bottleneck is the dispatcher and the second - database. The former can be eliminated by increasing number of middlewares. This discovery maps to our system design in which we assume that there is only one instance of dispatcher per middleware.

Later, we applied $2^k$ analysis to find out the best configuration parameters of the system. The results come along with findings in the bottleneck analysis.

Finally, we validated all conducted experiments by applying interactive response time law. The results of validation do not contain any discrepancies.

To conclude, all presented models and methods have some limitations which do not allow to achieve high level of precision. To achieve close accuracy one should model the entire system in really detailed way.

Appendix A

# MVA Service Times

| Clients | Network | Dispatcher | Worker | Database | Think time |
|---|---|---|---|---|---|
| 1 | 0.115963 | 0.260159 | 0.132599 | 0.832658 | 0.079355 |
| 2 | 0.191814 | 0.316384 | 0.163952 | 1.165958 | 0.079160 |
| 3 | 0.415860 | 0.474774 | 0.260938 | 1.460342 | 0.082534 |
| 4 | 0.671776 | 0.565514 | 0.337604 | 1.934601 | 0.084853 |
| 5 | 1.027851 | 0.627563 | 0.413276 | 2.336914 | 0.086280 |
| 6 | 1.580227 | 0.688099 | 0.531484 | 2.231150 | 0.088461 |
| 7 | 2.083839 | 0.712063 | 0.567892 | 2.432236 | 0.087289 |
| 8 | 2.891392 | 0.748126 | 0.665130 | 2.355438 | 0.087162 |
| 9 | 3.576806 | 0.757568 | 0.730713 | 2.420279 | 0.085172 |
| 10 | 4.237373 | 0.750822 | 0.790719 | 2.555941 | 0.087866 |
| 12 | 5.711849 | 0.732373 | 0.856967 | 2.711277 | 0.088813 |
| 14 | 7.050768 | 0.728925 | 0.914200 | 2.760395 | 0.090842 |
| 16 | 8.335145 | 0.740282 | 0.904570 | 3.082387 | 0.090613 |
| 18 | 10.199153 | 0.760969 | 0.930603 | 2.760379 | 0.088488 |
| 20 | 11.558659 | 0.750714 | 0.927854 | 2.827028 | 0.089113 |

Figure A.1: Service time (ms) for devices used by the model described in Chapter 5

Appendix B

# Interactive Response Time Law for Milestone 1 Experiments

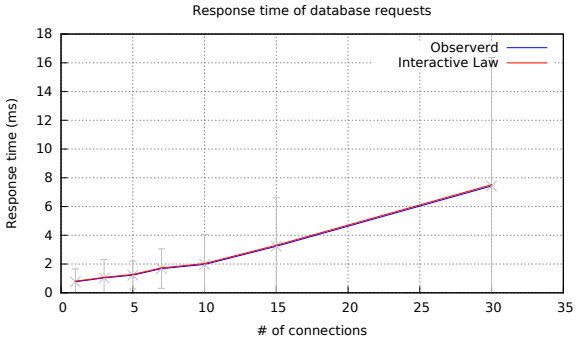


Figure B.1: Response time of the database when increasing number clients. Averaged over one minute interval. Figure 2.8 in Milestone 1.


Figure B.2: Response time of the middleware running on one node when increasing number of clients. Averaged over one minute interval. Figure 3.6 in Milestone 1.

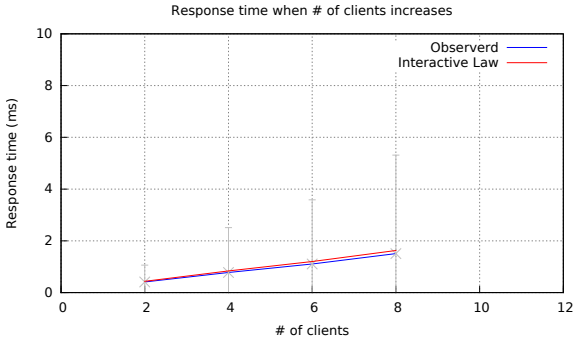


Figure B.3: Response time of the middleware running on two nodes when increasing number clients. Averaged over one minute interval. Figure 3.8 in Milestone 1.

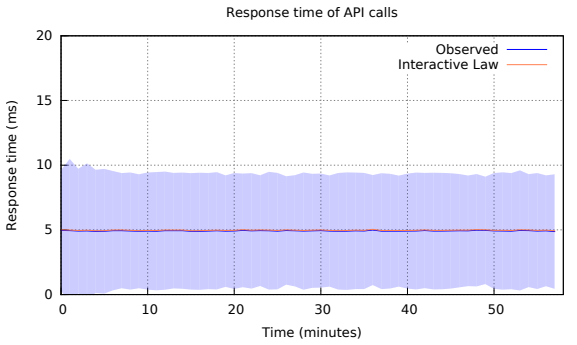


Figure B.4: Response time when running the middleware on a single node. Averaged over one minute interval. Figure 6.4 in Milestone 1.

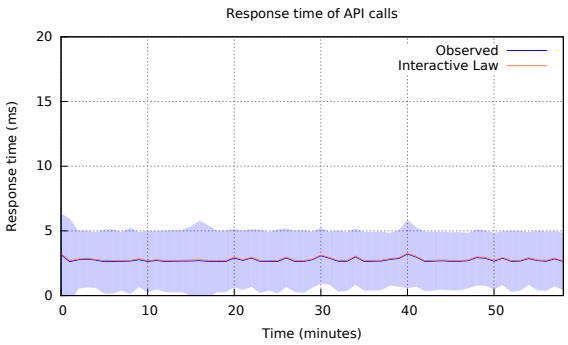


Figure B.5: Response time when running the middleware on two separate nodes. Averaged over one minute interval. Figure 6.7 in Milestone 1.

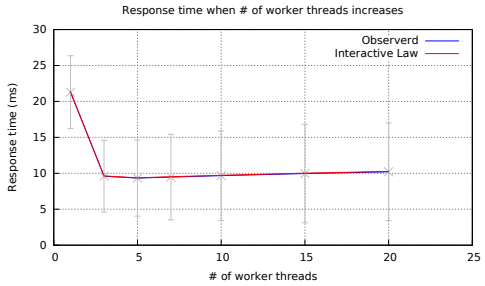


Figure B.6: The middleware is running on a single node

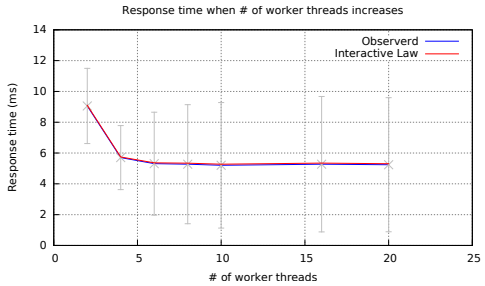


Figure B.7: The middleware is running on two nodes

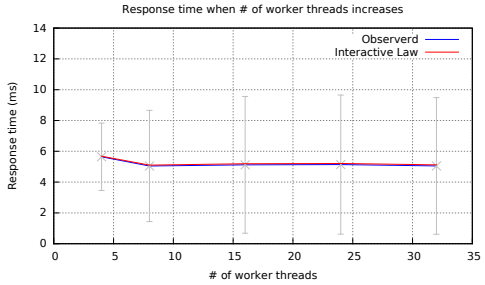


Figure B.8: The middleware is running on four nodes

Figure B.9: Response time and throughput of API requests when the number of worker threads increases. Figure 6.13 in Milestone 1.

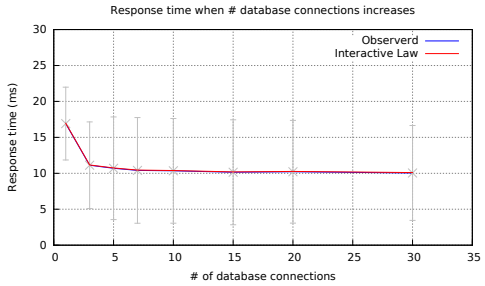


Figure B.10: The middleware is running on a single node

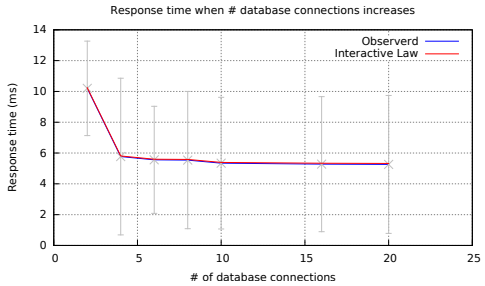


Figure B.11: The middleware is running on two nodes

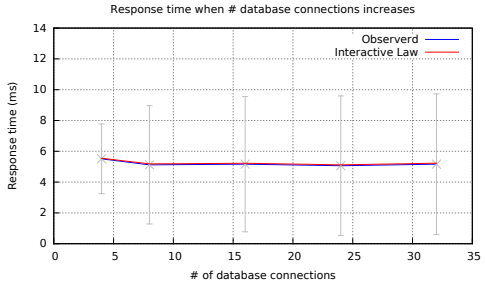


Figure B.12: The middleware is running on four nodes

Figure B.13: Response time and throughput of API requests when the number of the database connection pool increases. Figure 6.17 in Milestone 1.


Figure B.14: The middleware is running on a single node

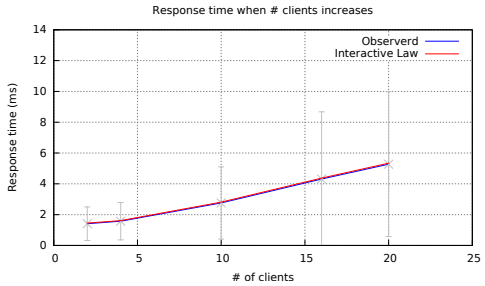


Figure B.15: The middleware is running on two nodes

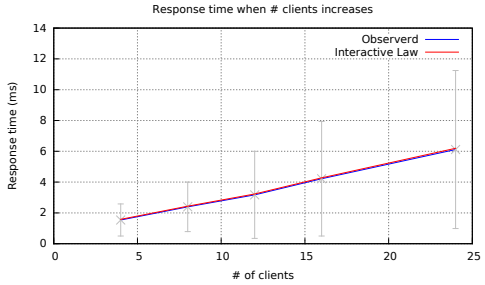


Figure B.16: The middleware is running on four nodes

Figure B.17: Response time and throughput of API requests when the number of the clients increases. Figure 6.21 in Milestone 1.

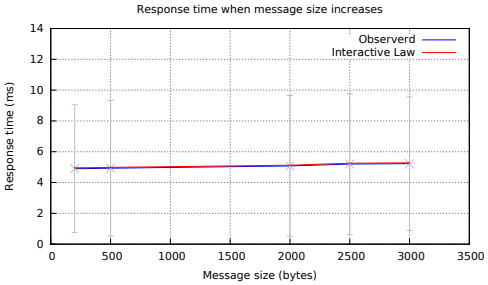


Figure B.18: Response time and throughput of API requests when the size of message increases. Figure 6.23 in Milestone 1.

Appendix C

# Additional Experiments for Milestone 1

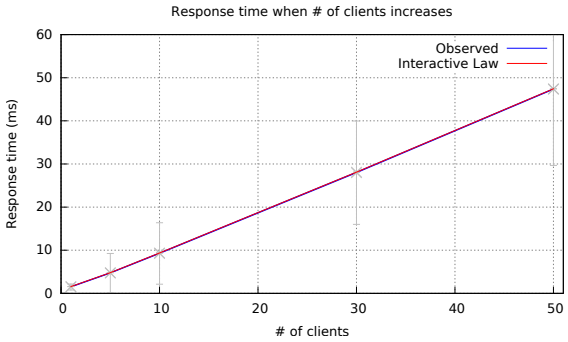


Figure C.1: System response time when increasing number of clients. Number of worker threads and database connections is equal to 30.

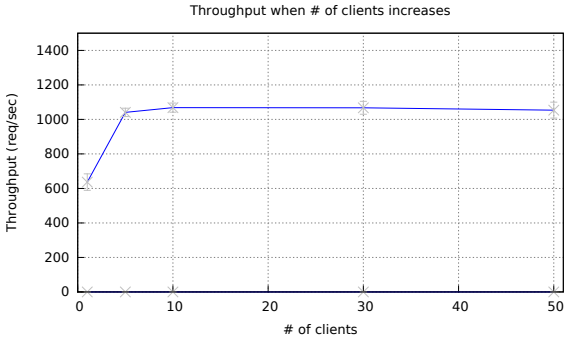


Figure C.2: System throughput when increasing number of clients. Number of worker threads and database connections is equal to 30.

As it can be seen from Figure C.1 and Figure C.2, the system troughput was

not impacted in a case when number of clients is higher than 30.