UNIVERSITÀ DI PISA

Master's degree in Computer Engineering

592II Performance evaluation of computer systems and networks

Multiprogrammed server

Designers:

Tommaso Califano

Nicola Ramacciotti

Gabriele Suma

ACADEMIC YEAR 2023/2024

Contents

Chapter 1

Overview

1.1 Objectives

A multi-programmed server provides service to **N concurrent clients** that request the server to perform transactions. **Local CPU** computations, interactions with the **local disk** and remote queries to a **distant web server** are all processes that may be involved in transactions. An interaction between clients and server can be defined as follows:

1. A new transaction always requires some processing time as a first step.

2. Transactions can follow different flows based on probability.

3. A reply is sent to the client that originated the request.

4. A user that receives a reply immediately issues another request.

Utilizing the FIFO policy, each module within the system (Local CPU, local disk and the remote web server) is capable of processing a single request at any given time. Considering the previous assumptions, it becomes crucial to evaluate the performance of the system with a particular emphasis on **throughput and utilization**. With the aid of the Omnet++ simulation software, we can gain insights into the behaviour of the system under various conditions. Furthermore, utilizing MS Excel and Python, we can collect the data obtained from simulations for data analysis and chart representations.

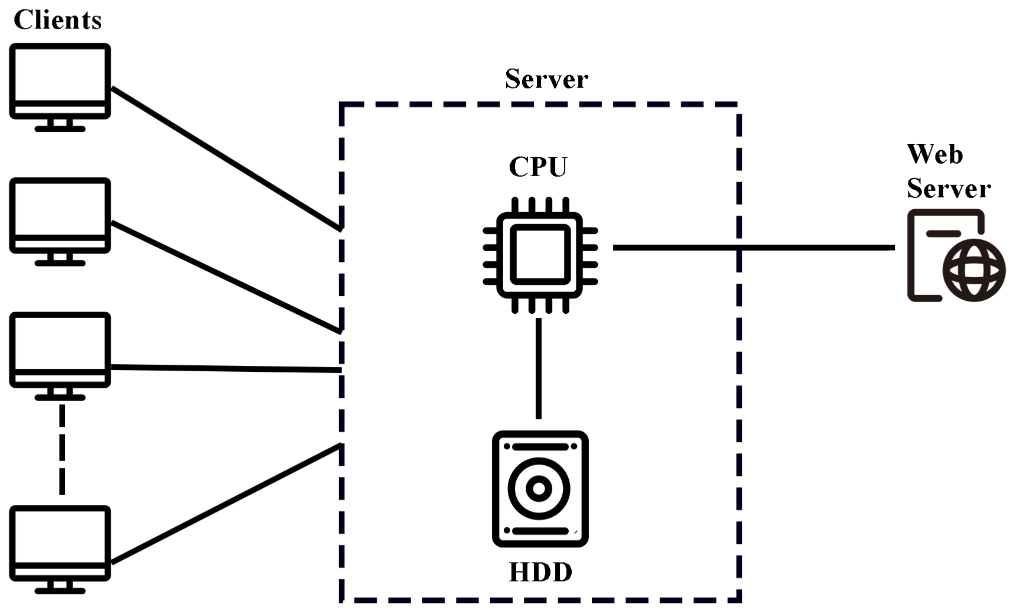


Figure 1.1: Simplified and high-level scheme

1.2 Performance Indexes

To evaluate the performances of the system, the following indexes were defined:

- **Throughput**: the number of completed transactions per unit of time.
- **Utilization**: the time percentage during which each node is busy.

Chapter 2

Model

2.1 General assumption and preliminary validation

- **Clients:** the number of clients is finite and it corresponds with the number of jobs within the system. Given that, a client instantly sends a new request upon receiving a response from the server, implementing a queue or modeling clients as service centers would be unnecessary.

- **Queues:** the queue size is not dimensionally limited for simulation purposes. Essentially, this has been an assumption in the simulation. Indeed, having N clients implies that the queue could reach a size of N jobs in the worst-case scenario.

- **Service centers:** CPU, HDD and Web server query SCs were needed to represents the actual system. Each SC has an exponential distributed service time with a different rate $\boldsymbol{\mu_{CPU}}$, $\boldsymbol{\mu_{HDD}}$, $\boldsymbol{\mu_{Q}}$. Each request is processed individually in a **FIFO** order.

- **Handling Transactions:**

  - A new transaction always requires some processing time as a first step.
  - Then:
    * with probability **p1** the transaction is terminated.
    * with probability **p2** an access to the local disk is required, and then a new CPU processing is required.
    * with probability **1-p1-p2** a remote query is required, and then a new CPU processing is required.
  - A reply is sent to the client that originated the request.

Considering these assumptions, it's clear that the system can be represented as a **Close Jackson's Network**. It follows a potential representation:

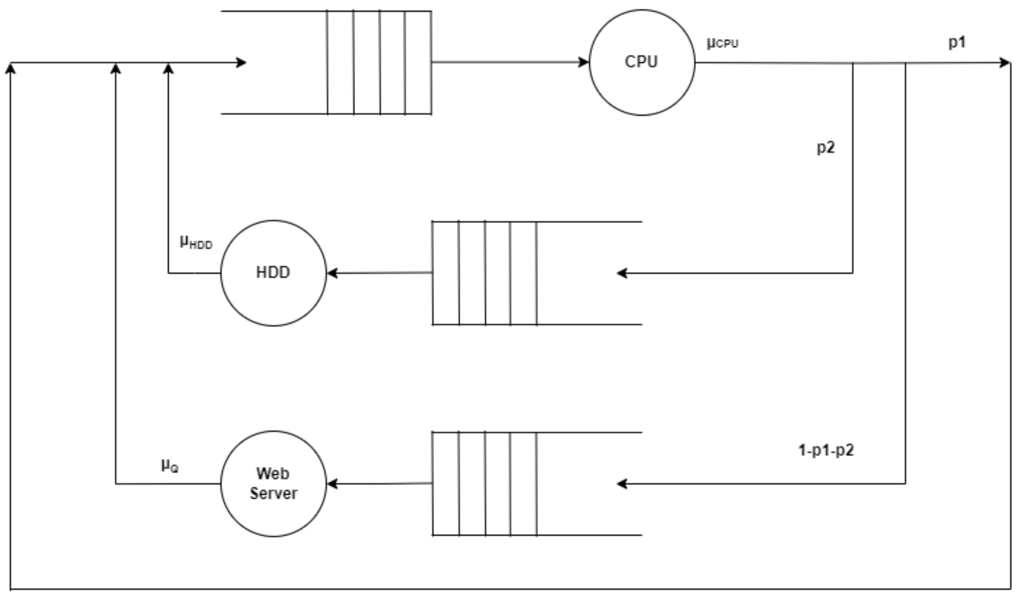


Figure 2.1: Close Jackson's Network modeling representation

2.2 Factors

The following factors may affect the performances of the system:

- **N**: the number of clients.
- **p1**: the probability that a transaction is terminated.
- **p2**: the probability that an access to the local disk is required and then a new CPU processing is required.
- $\boldsymbol{\mu_{CPU}}$: CPU's service rate.
- $\boldsymbol{\mu_{HDD}}$: HDD's service rate.
- $\boldsymbol{\mu_{Q}}$: Web query server's service rate.

Chapter 3

Implementation

3.1 Modules

The following modules have been defined:

- **Client**: a simple module which represents N Clients.
- **Server**: a compound module representing the local server, composed by:
  - **CPU**: a simple module which represents the local processor.
  - **HDD**: a simple module which represents the local hard disk.
- **Web Server**: a simple module which represents a distant web query server.

3.2 Messages

The class **cMessage** has been extended in **Transaction**. It was necessary to monitor the initiation time of each transaction, which is crucial for calculating the **response time** of the system.

3.3 Verification

3.3.1 Degeneracy Test

The degeneracy test is used to analyze the behaviour of the system under conditions where factors take on extreme or degenerate values.

1. When the number of clients is **zero**, the system enters an idle state due to the absence of requests.
2. When $\mathbf{p2} = \mathbf{0}$, the system doesn't rely on HDD computation, but only on CPU and distant web query server computations.
3. If $\mathbf{p1} = \mathbf{1}$, the system relies only on the elaboration of the CPU.

3.3.2 Continuity Test

To verify the accuracy of the model, it is necessary to create two configurations with slightly different values of factors [Table 3.1] and see if the output does not change significantly.

Configuration	**Factors**
First Config	p1 $= 0.35$ **p2 = 0.40** $\mu_{CPU} = 1000$ $\mu_{HDD} = 250$ $\mu_Q = 75$ Number of clients $= 40$
Second Config	p1 $= 0.35$ **p2 = 0.41** $\mu_{CPU} = 1000$ $\mu_{HDD} = 250$ $\mu_Q = 75$ Number of clients $= 40$

Table 3.1: Two slightly different configurations selected for the continuity test

Simulating the previous scenario for 25 repetition, the following chart [Figure 3.1] shows:

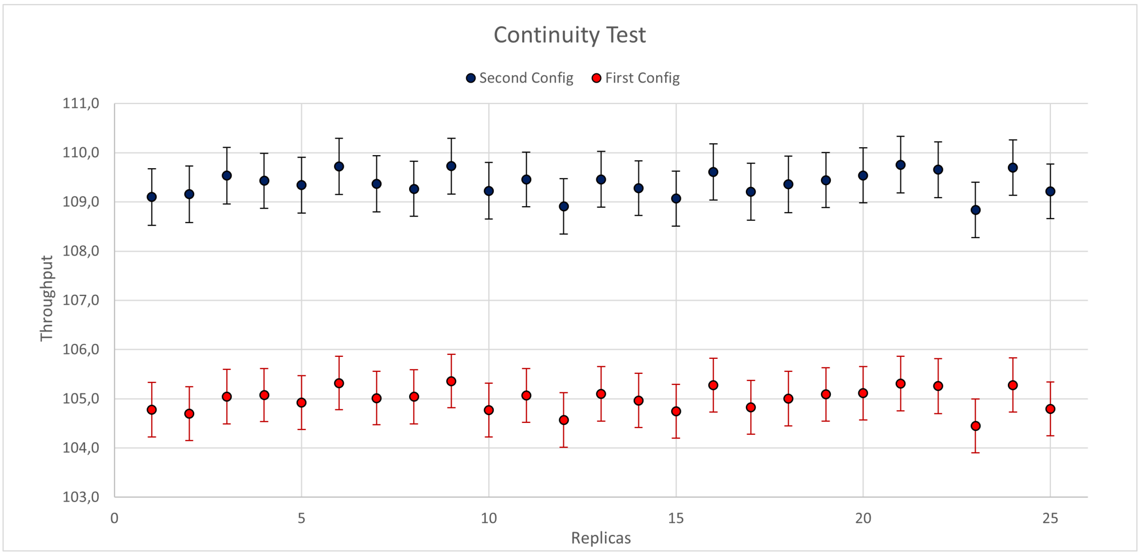


Figure 3.1: Performing a continuity test at a 95% confidence level.

The system operates as anticipated without any unexpected behaviour. There is a minor fluctuation in throughput, attributable to the reduced service rate of the web server [Figure A.2]. Notably, in the second configuration, an increase in **p2** results in a marginal decrease in the utilization of the web server.

3.3.3 Consistency test

In order to validate the consistency test, it was necessary to conduct a study on the behaviour of the system by varying the number of clients [Appendix A] making service requests to the server.

Configuration
p1 = 0.35
p2 = 0.40
$\mu_{CPU} = 1000$
$\mu_{HDD} = 250$
$\mu_Q = 75$

Table 3.2: Configuration adopted for the simulation runs.

Clearly, the system becomes saturated due to the volume of requests and is unable to handle the load when the number of clients exceeds 10. It is expected due to the low service rate provided by the distant web server. The following chart [Figure 3.2] focuses on doubling the number of clients (from 20 to 40) during the bottleneck scenario [Figure A.2]:

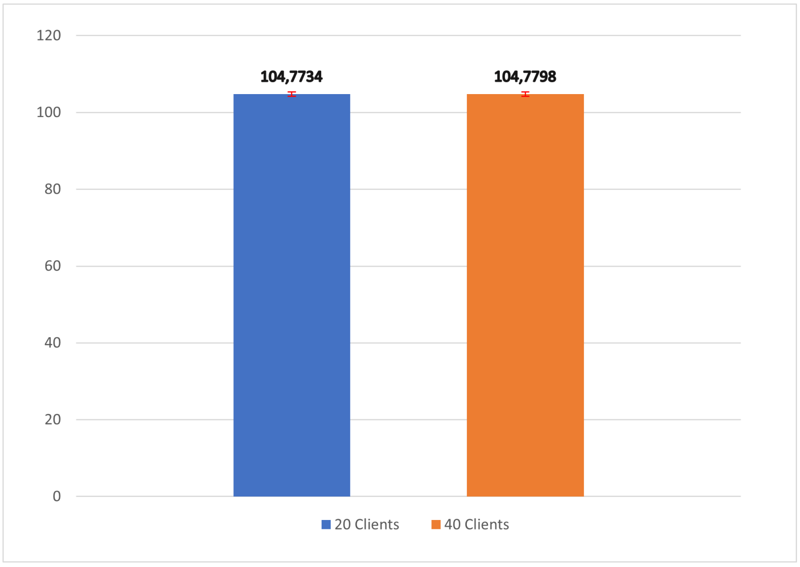


Figure 3.2: Comparison of throughput (with a 95% confidence interval) between 20 and 40 clients.

3.3.4 Theoretical verification

Given that our system can be modeled as a **Closed Jackson Network**, we can theoretically prove its performance indexes. By setting $\mathbf{M} = \mathbf{3}$ and $\mathbf{K} = \mathbf{40}$ and then applying the **Buzen's convolution algorithm**, we were able to compute the matrix $\mathbf{G(M,K)}$ efficiently.

Utilization:

For a single SC, this can easily be computed as:

$$\mathbf{U_i} = \mathbf{P}\{\mathbf{N_i} >= \mathbf{1}\} = \rho_\mathbf{i} \times \frac{\mathbf{G(M,K-1)}}{\mathbf{G(M,K)}}$$

Theoretical utilization values computed:

- $\mathbf{U_{CPU}} = 0{,}4$
- $\mathbf{U_{HDD}} = 0{,}64$
- $\mathbf{U_{WS}} = 1$

Collecting the utilization data from our simulations, we were able to verify our model with queueing theory. In fact, in the following Figure 3.3, the empirical and theoretical values closely align.

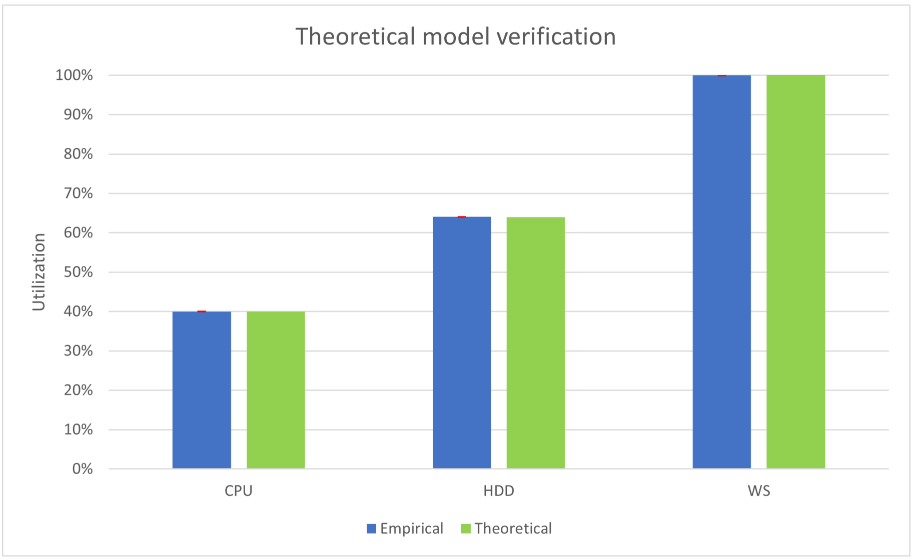


Figure 3.3: Comparison between empirical and theoretical values (with confidence interval of 99%.

Chapter 4

Data Analysis

4.1 Calibration

In order to accurately simulate each scenario, we must compute the warm-up period and the simulation time. The former is crucial to avoid taking ineffective samples before the system is in a steady-state. On the other hand, the latter is required to gather data that is statistically meaningful during the steady-state.

4.1.1 Warm-Up period

To estimate the warm-up period, we relied on the study of the time average. Once the time average function begins to **stabilize**, we can confidently conclude that the system has completed its warm-up period.

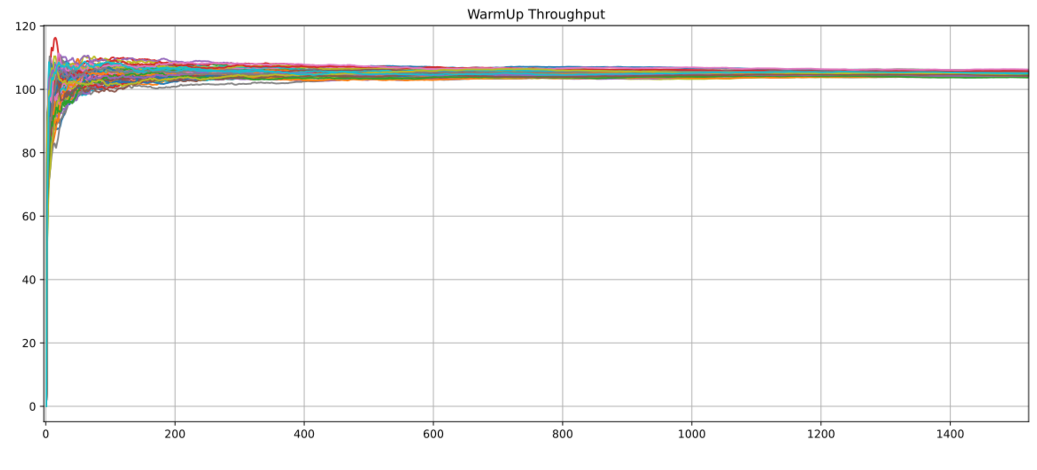


Figure 4.1: Time-average function calculated from 100 repetitions using [Table 3.2] configuration.

In the [Figure 4.1], it is clear that starting from **500s** the time-average function becomes **stable**. In the [Appendix B] it is possible to go into further details of warm-up period evaluation.

4.1.2 Simulation Time

In order to estimate the simulation time, our focus was primarily on the utilization of the system. By adopting the [Table 3.2] configuration and conducting an analysis of **500 repetitions**, we were able to compute the **utilization** of the system from the single SC utilizations.

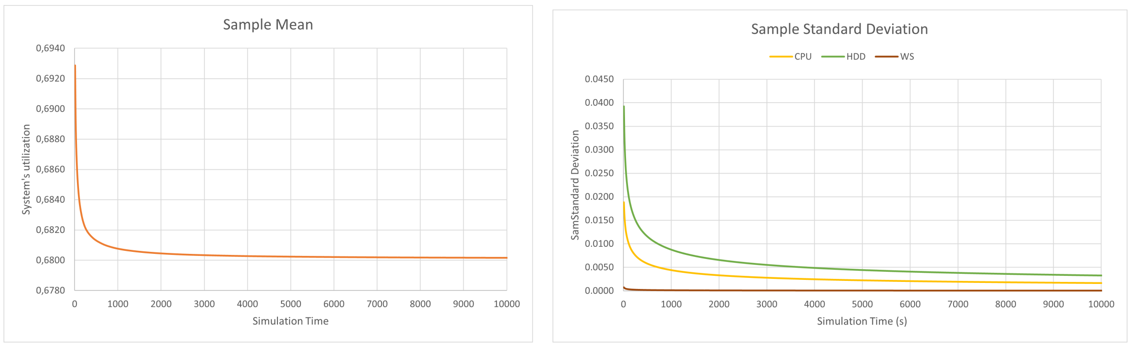


Figure 4.2: Charts showing the sample mean and the sample standard deviation of the utilization of the system as a function of the simulation time.

The validity of the result is confirmed as the sample mean **progressively approximates** to:

$$\frac{E[Activity]}{E[Activity] * E[Idle]} = 0,680078144$$

After **4000 seconds** of simulation time, the sample standard deviation tends to diminish significantly (by two orders of magnitude).

4.2 Simulation Scenarios

In order to evaluate the performances of the system in a meaningful way, we grounded our assumptions on the data collected by using the configurations in the [Table 4.1] below. Analyzing the low **CPU utilization**, we decided to keep the $\boldsymbol{\mu_{CPU}}$ constant and equal to **1000**. In the following charts, we **tweaked** two factors and established **three subcases**. In these subcases, the remaining **two factors were assigned low**, **medium** and **high values** respectively as the [Table 4.1] shows.

	Low	**Medium**	**High**
μ_{HDD}	175	225	300
μ_{WS}	50	100	150
p1	0,1	0,3	0,4
p2	0,2	0,3	0,5

Table 4.1: Configurations defined for simulating scenarios. μ_{CPU} is presumed to be constant and equal to 1000.

4.2.1 Adjusting p1 and p2

In the first scenarios, we focused on the impact of **p1** and **p2** on the throughput of the system. In the following charts [Figure 4.3], it is possible to observe that **increasing** p1 and p2 (this action results in a **decrease** in web-server utilization and **increases** the HDD utilization [Figure 4.4]), brings the throughput of the system to **increase** quite rapidly. This is completely logical, given that we are reducing the load on the slowest SC.

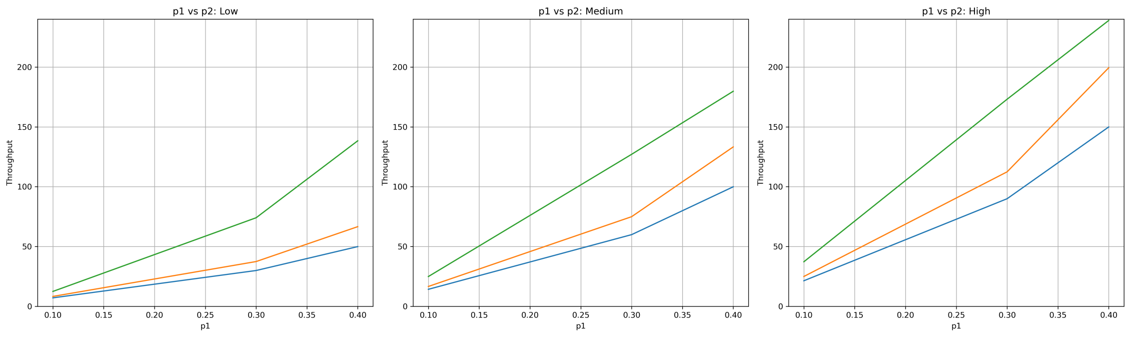


Figure 4.3: Charts representing the impact of p1 and p2 on throughput.

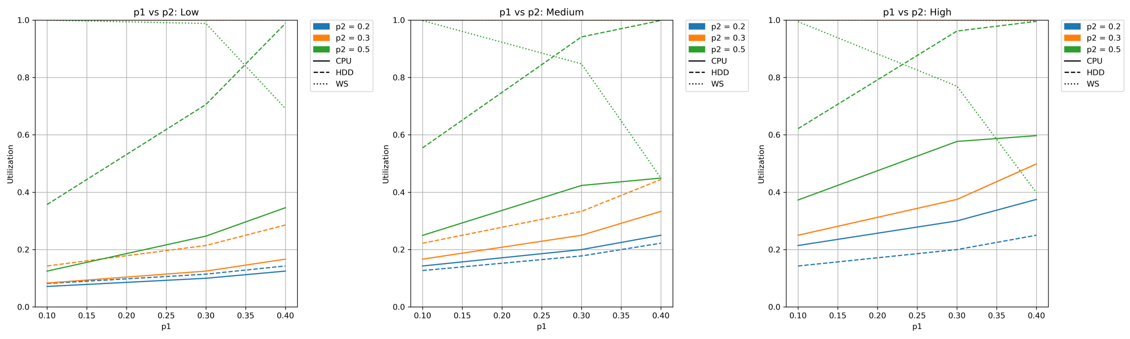


Figure 4.4: Charts representing the impact of p1 and p2 on utilization.

4.2.2 Adjusting μ_{HDD} and p1

In the next scenarios, we focused on the impact of $\boldsymbol{\mu}_{\mathbf{HDD}}$ and **p1** on the throughput of the system. In the following charts [Figure 4.5], it is quite clear that **increasing** the service rate of HDD and p1 doesn't improve the throughput of the system. The performances **increase only** if the **web-server service rate and p2 increase**, thus **decreasing** the web-server load [Figure 4.6].

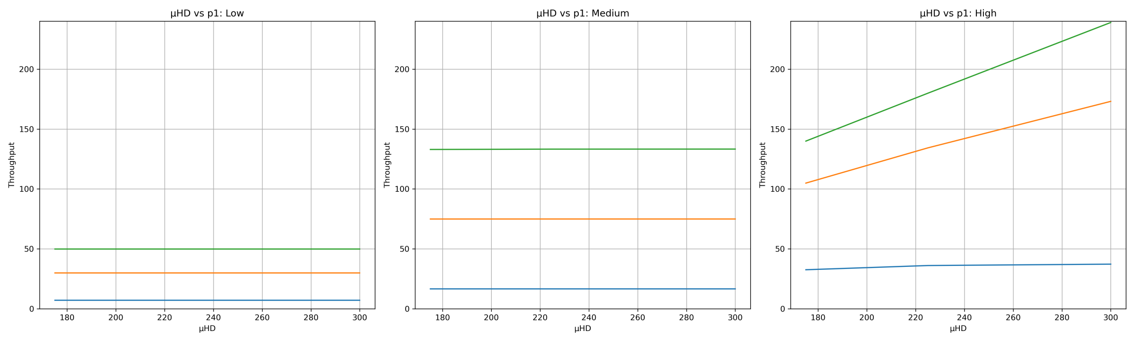


Figure 4.5: Charts representing the impact of μ_{HDD} and p1 on throughput.

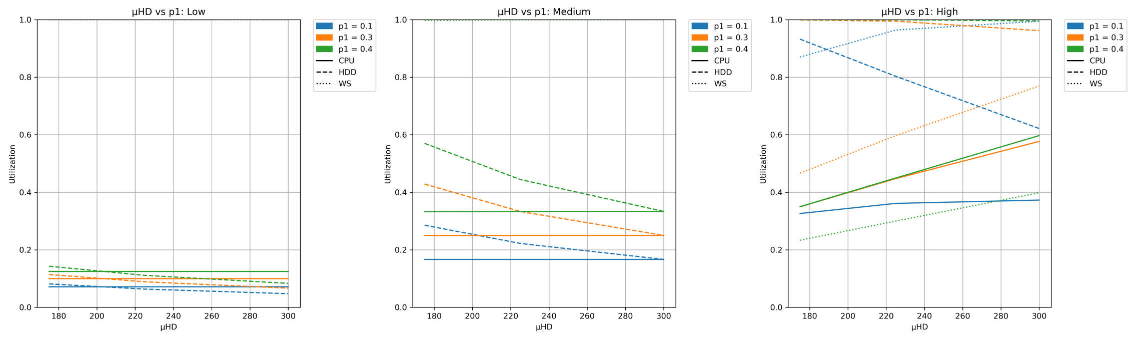


Figure 4.6: Charts representing the impact of μ_{HDD} and p1 on utilization.

4.2.3 Adjusting μ_{WS} and p1

In the last scenarios, we focused on the impact of $\boldsymbol{\mu}_{\mathbf{WS}}$ and **p1** on the system. In the following charts [Figure 4.7], it is quite obvious that **increasing** the **web-server service rate** and **p1** improves the performance of the system. The final chart, which corresponds to the subcase where the remaining two factors (**p2 and** $\boldsymbol{\mu}_{\mathbf{HDD}}$) assume their maximum values, suggests that the throughput tends to stabilize when the web-server service rate hits **100**. In fact, in the [Figure 4.8] there is a **reduction** in web-server utilization, contrasted by an **increment** in HDD utilization (which becomes the new bottleneck of the system).

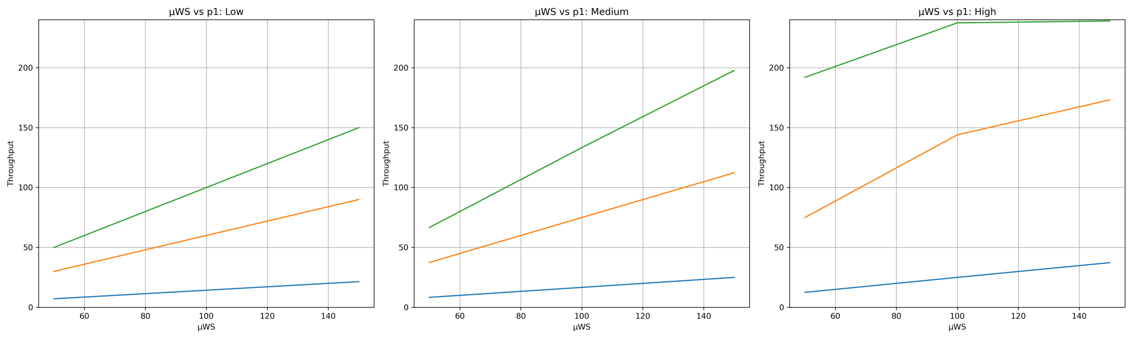


Figure 4.7: Charts representing the impact of μ_{WS} and p1 on throughput.

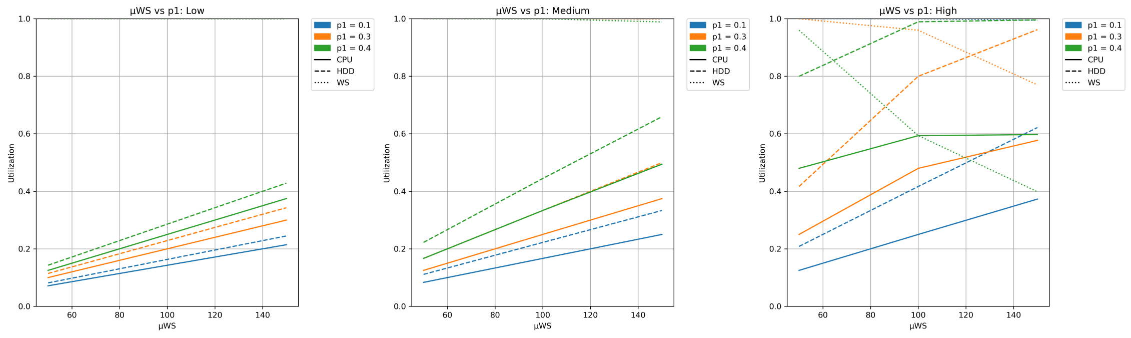


Figure 4.8: Charts representing the impact of μ_{WS} and p1 on utilization.

We have **omitted** the representation of confidence intervals in each chart due to their negligible size. To prove this, we calculated the **ratio** of utilization [Figure 4.10] and throughput [Figure 4.9] over their respective confidence intervals at **99%** confidence level:

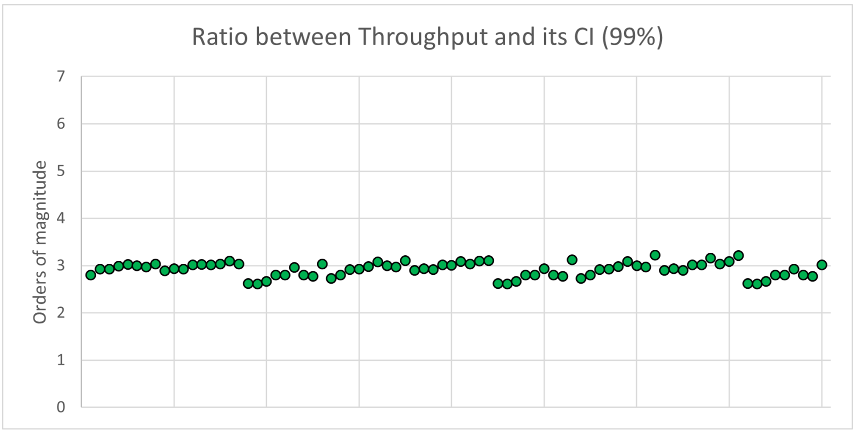


Figure 4.9: Ratio between throughput and confidence intervals (99% confidence level)

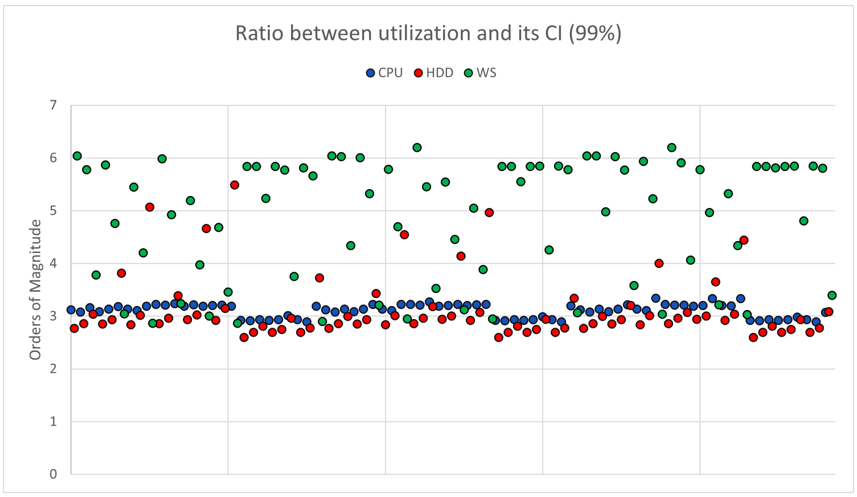


Figure 4.10: Ratio between utilization and confidence intervals (99% confidence level)

Appendix A

Evaluation varying the number of clients

The following chart shows how the system react varying the number of clients:

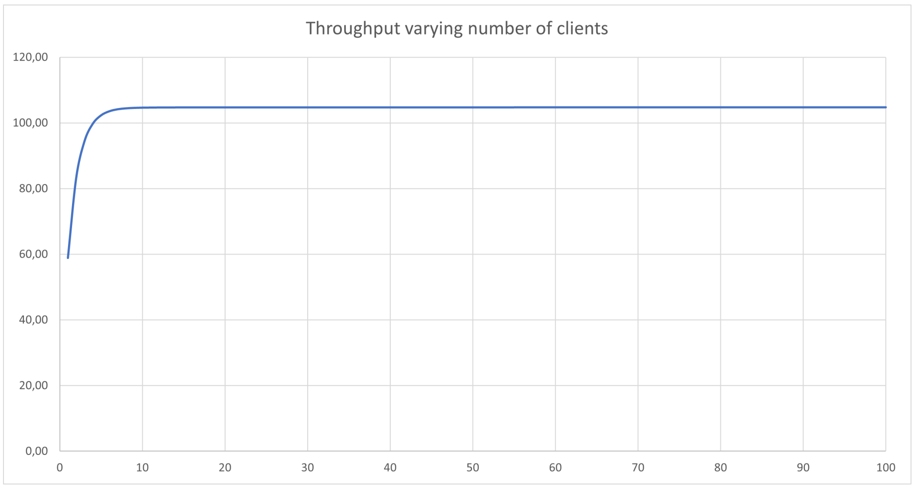
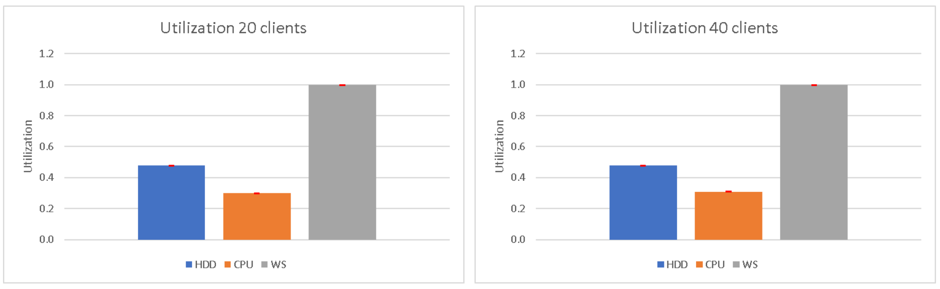


Figure A.1: Throughput (Y-axis) of the system varying the number of clients (X-axis).



(a) 20 Clients

(b) 40 Clients

Figure A.2: Comparison between 20 and 40 clients utilization.

Appendix B

Warm-Up deeper evaluation

In order to evaluate correctly the warm-up period, we computed the time-average function of the **mean response time** and each SC **utilization**:

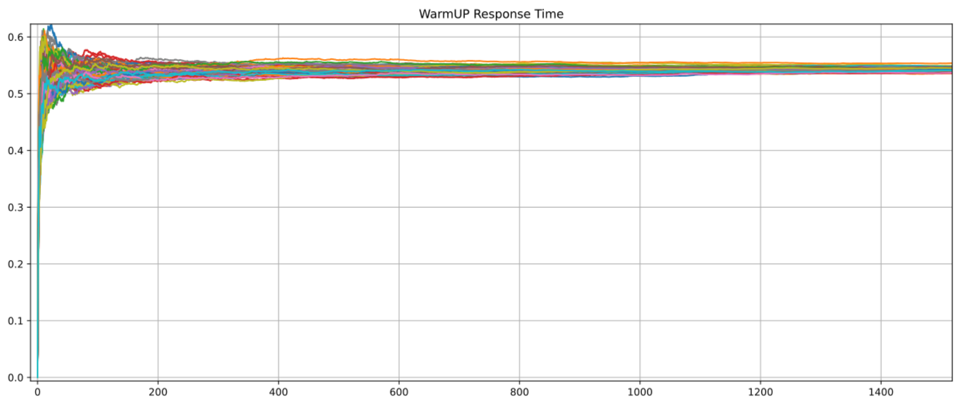


Figure B.1: Time-average function applied on the mean response time on 100 repetitions.

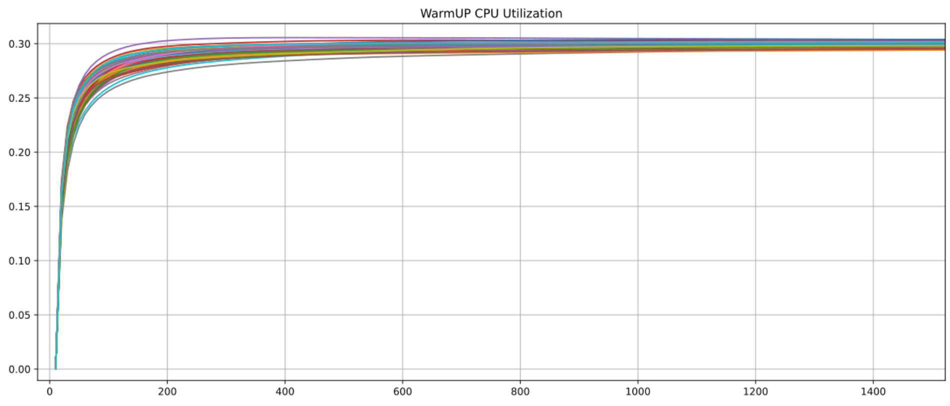


Figure B.2: Time-average function applied on CPU utilization on 100 repetitions.

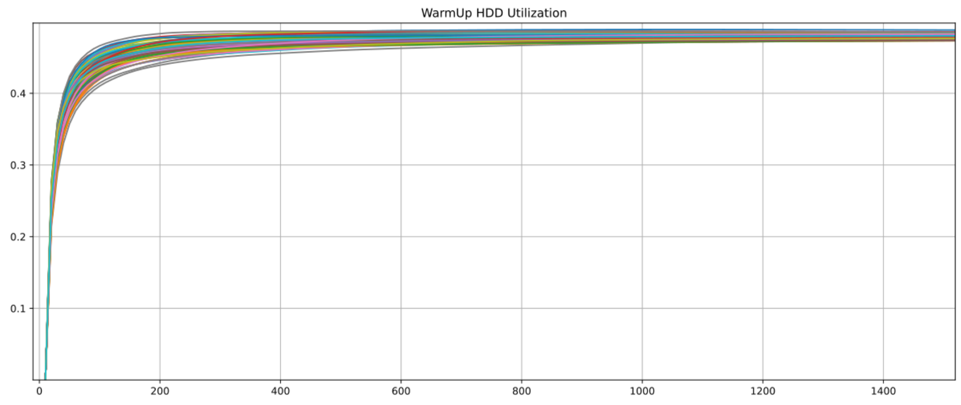


Figure B.3: Time-average function applied on HDD utilization on 100 repetitions.

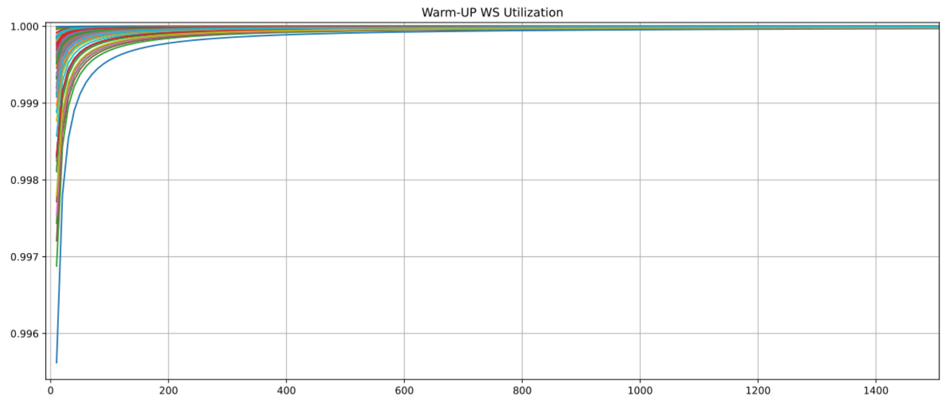


Figure B.4: Time-average function applied on WS utilization on 100 repetitions.

From each chart, we can estimate that the time-average function appears to stabilize around **500 seconds**.

Appendix C

Factorial analysis

We aim to evaluate the impact of each factor on the performances of the system. After we studied how the system behaves when the number of clients increases [Appendix A], we decided to maintain a constant count of **40 clients**.

The following configurations were chosen, consisting in a total of **32 observations** [Table C.1]:

	Low	**High**
μ_{CPU}	500	1500
μ_{HDD}	175	300
μ_{WS}	50	150
p1	0,1	0,4
p1	0,2	0,5

Table C.1: Factors defined for the factorial analysis.

After running the simulation **50 times**, we have come up with the following result: the chart [Figure C.1] clearly indicates that **p1** has the most significant influence at (**67%**), while **p2** (**9%**), μ_{WS} (**7%**) and a combination between **p1** and **p2** (**5%**) have marginal impacts.
Unfortunately, the factorial analysis didn't meet the following hypothesis:

1. **Residuals (errors) are IID Normal RVs with a null mean and a constant standard deviation**

Consequently, our findings are not statistically significant and provide no insights into the system.

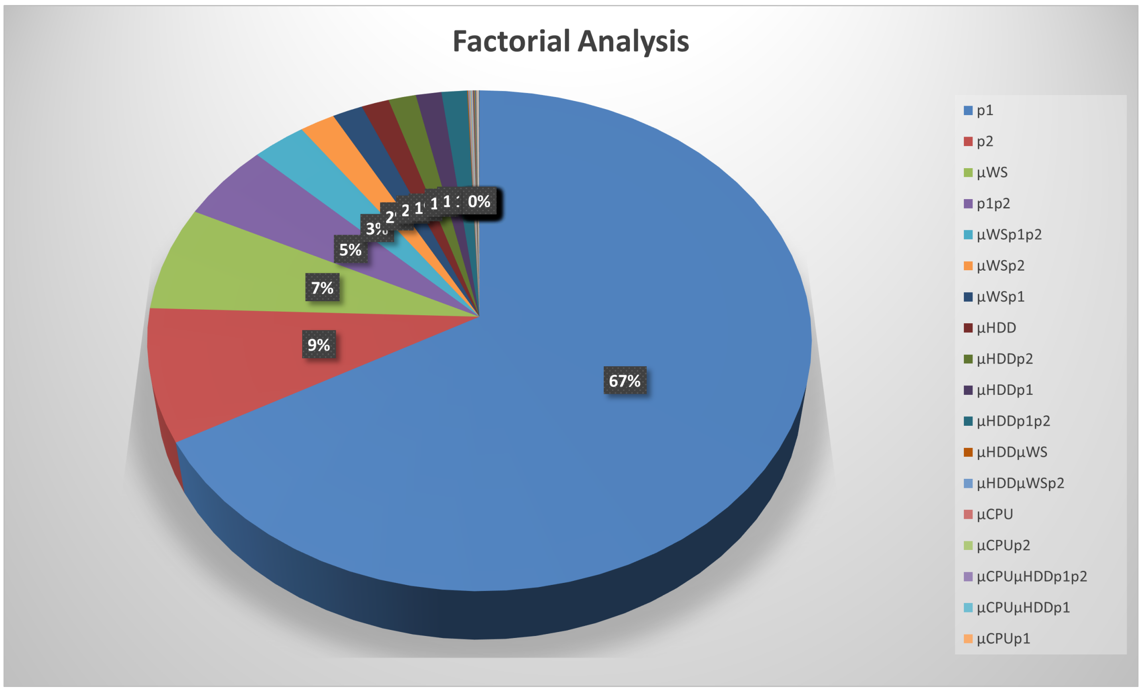


Figure C.1: The pie chart illustrates the impact of each factor on the throughput of the system.

Testing normal hypothesis:

The **QQ-Plot** shows a non-linear behaviour.

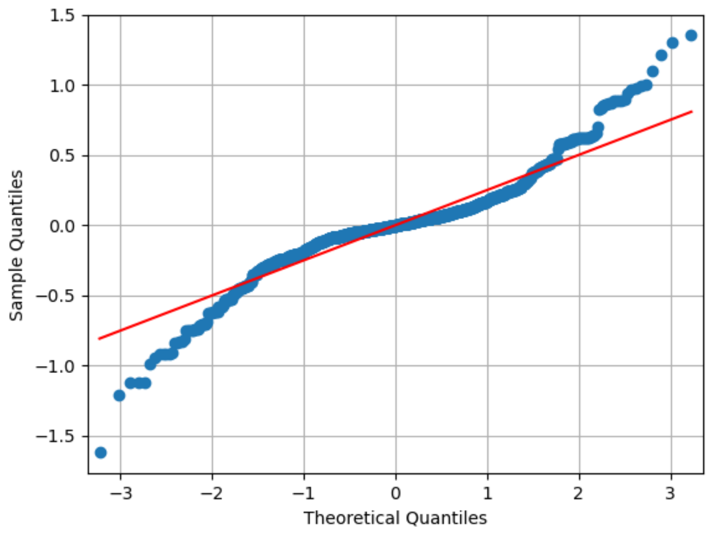


Figure C.2: QQ-Plot: On Y-Axis the samples quantiles and X-Axis the theoretical quantiles of a normal distribution.

Homoskedasticity:

The scatter-plot of residuals vs the predicted response [Figure C.3] shows a visible trend:

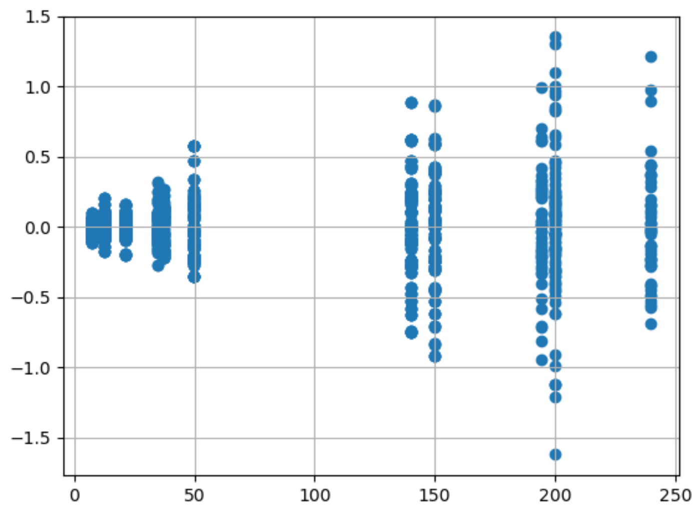


Figure C.3: The residuals are plotted along the X-axis, while the predicted responses are represented on the Y-axis.

Essentially, we can **ignore trends** if the errors are **one or more orders of magnitude** below the predicted response as is the case in our analysis:

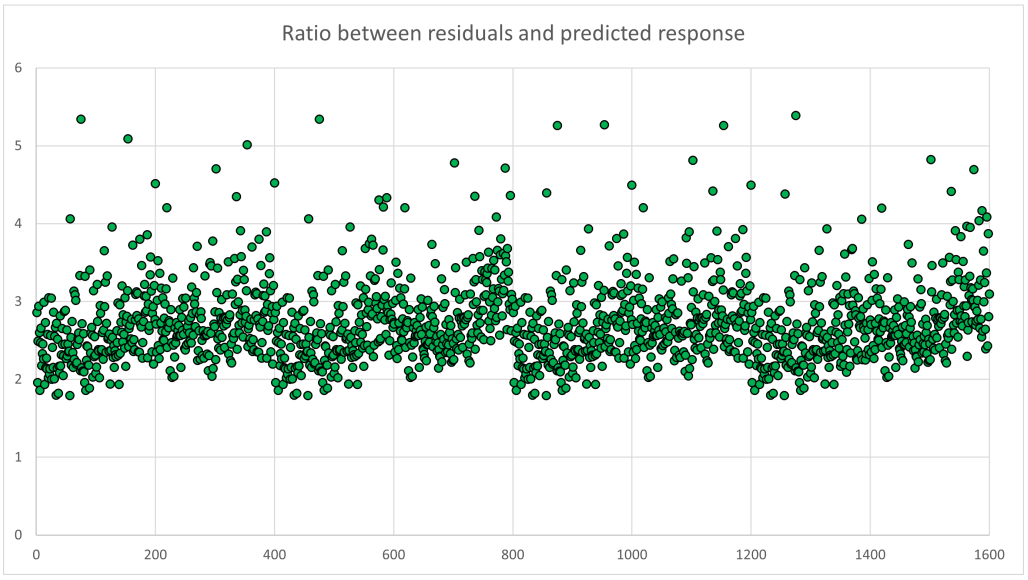


Figure C.4: Each residual varies by an order of magnitude or more below the predicted response.