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 system's performance with a particular emphasis on **throughput, mean response time and utilization**. With the aid of the Omnet++ simulation software, we can gain insights into the system's behavior under various conditions. Furthermore, utilizing MS Excel, we can collect the data obtained from simulations for data analysis and graph representations.

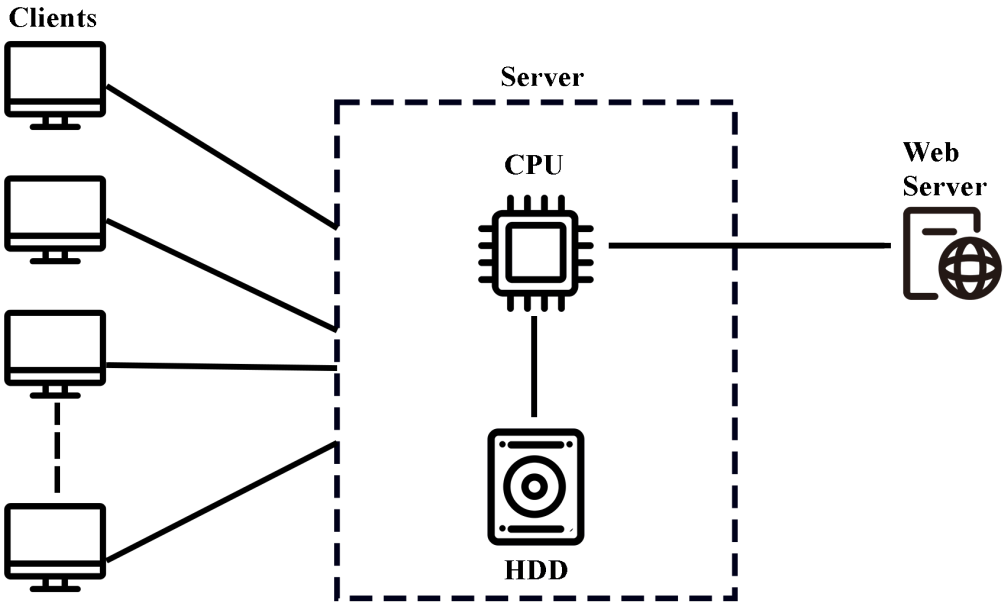

Figure 1.1: Simplified and High-level scheme

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.
- **Mean response time**: the mean duration required for a particular client request to obtain the corresponding response.
- **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. 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 μ_{CPU}, μ_{HDD}, μ_{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:

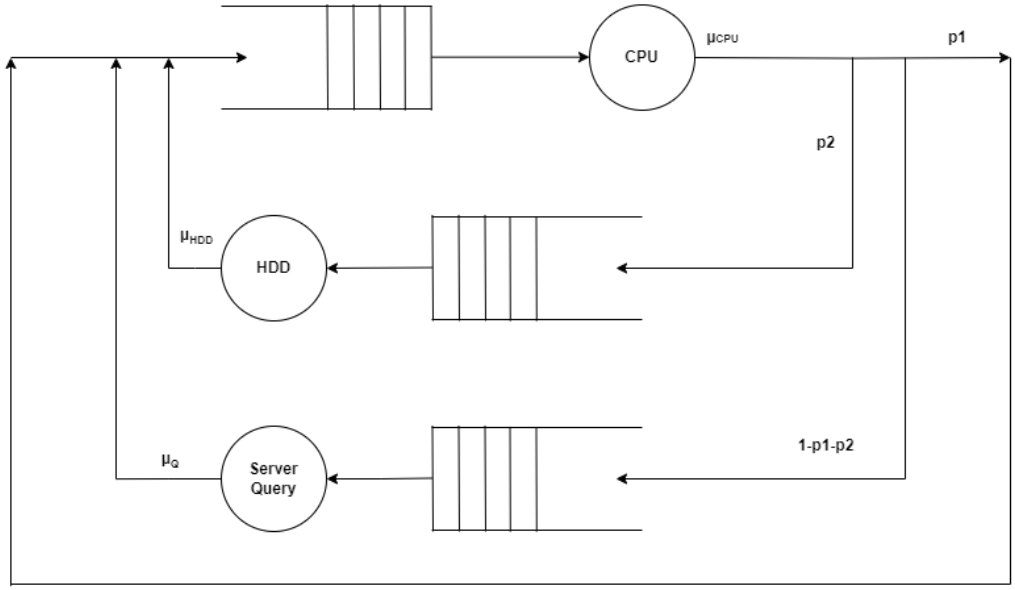

Figure 2.1: Close Jackson's Network modeling representation

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.
- $\mu_{\mathbf{CPU}}$: CPU's service rate.
- $\mu_{\mathbf{HDD}}$: HDD's service rate.
- $\mu_{\mathbf{Q}}$: 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 system's response time.

3.3 Verification

3.3.1 Degeneracy Test

The degeneracy test is used to analyze the system behaviour under conditions where parameters 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 CPU's elaboration.

3.3.2 Continuity Test

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

Configuration	Parameters
First Config	p1 = 0.35 **p2 = 0.40** μ_{CPU} = 1000 μ_{HDD} = 250 μ_{Q} = 75 Number of clients = 40
Second Config	p1 = 0.35 **p2 = 0.41** μ_{CPU} = 1000 μ_{HDD} = 250 μ_{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 the system behaviour:

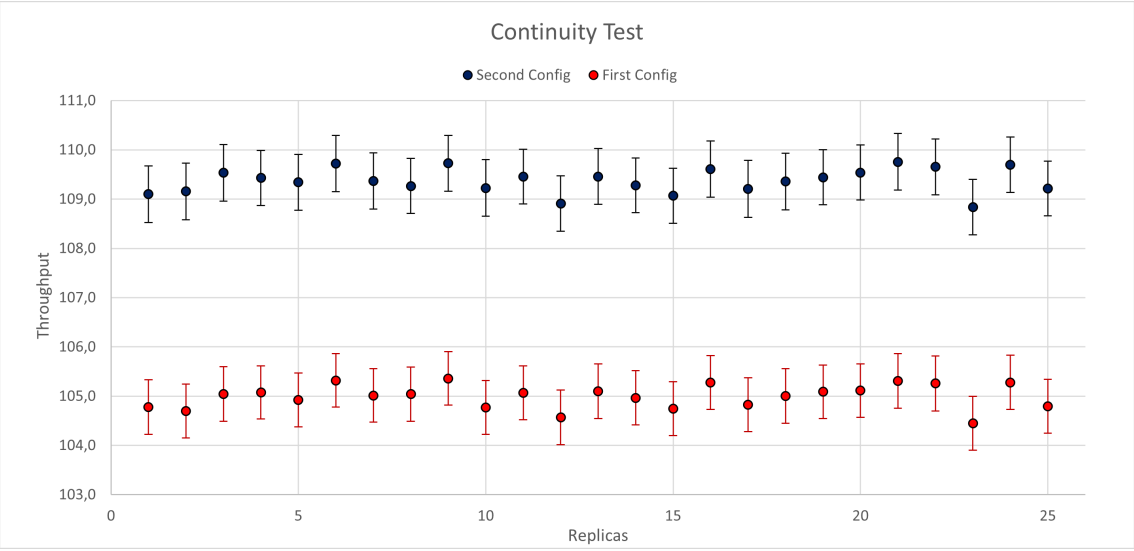

Figure 3.1: Performing a continuity test at a 95% confidence level.

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. 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 system's behavior by varying the number of clients [Appendix A] making service requests to the server.

Parameters
$p1 = 0.35$
$p2 = 0.40$
$\mu_{CPU} = 1000$
$\mu_{HDD} = 250$
$\mu_{Q} = 75$

Table 3.2: Fixed values of parameters 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] focus on doubling the number of client (from 20 to 40) during the bottleneck scenario:

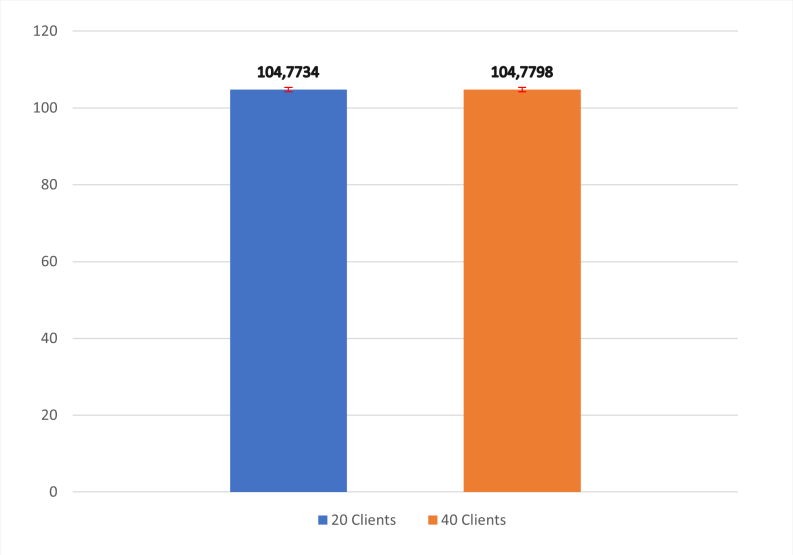

Figure 3.2: Comparison of Throughput (with a 95% Confidence Interval) between 20 and 40 clients.

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 to take 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.

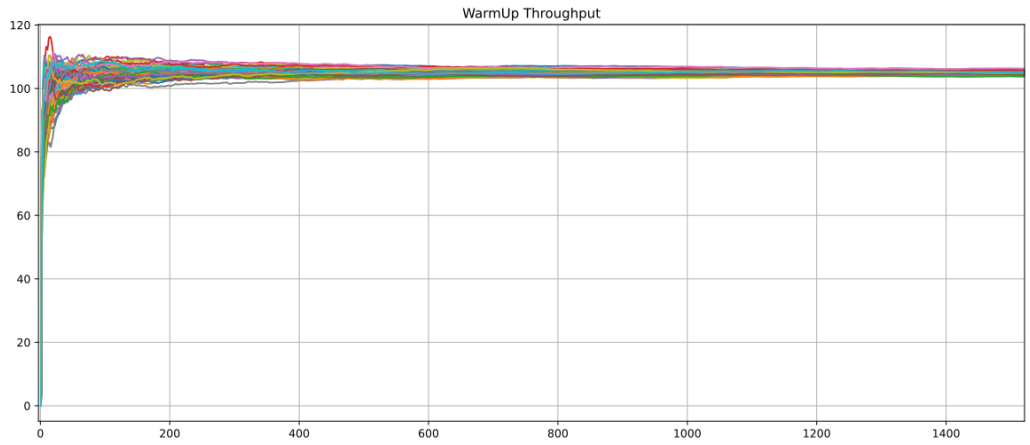

Figure 4.1: Time-average function calculated from 100 repetitions.

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

4.1.2 Simulation Time

Appendix A

Varying number of clients evaluation

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

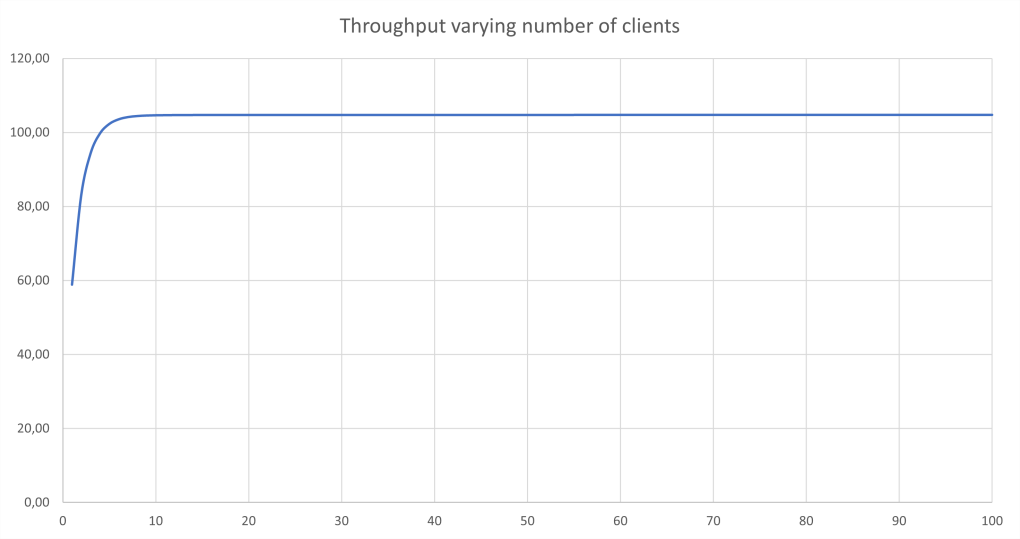

Figure A.1: Throughput (Y-axis) of the system varying the number of clients (X-axis).

Appendix B

Warm-Up deepen 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**:

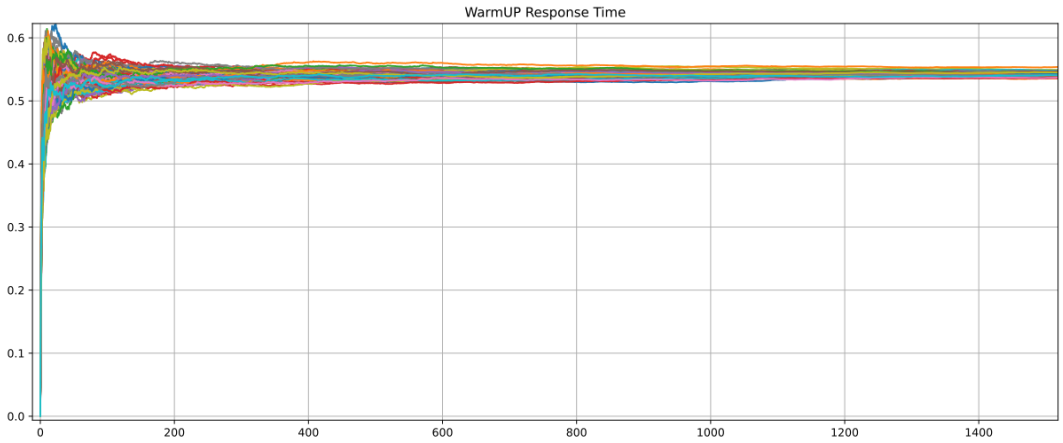

Figure B.1: Time-average function calculated from 100 repetitions.

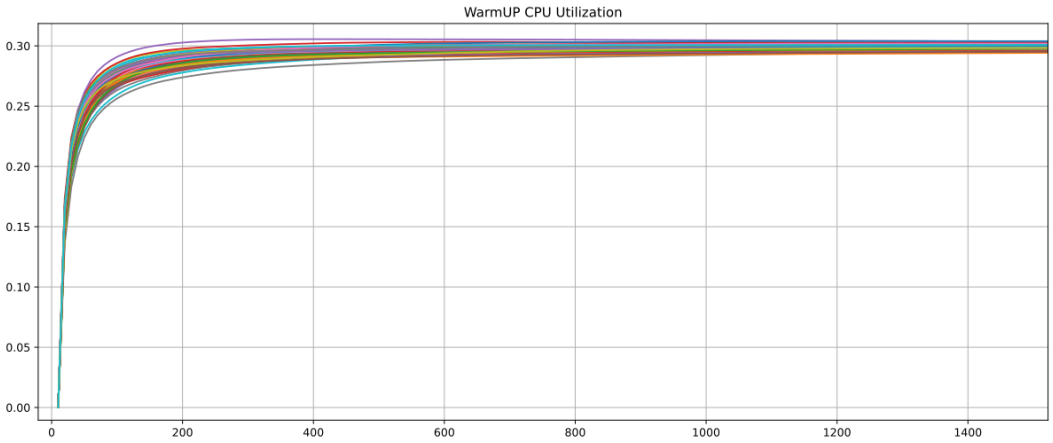

Figure B.2: Time-average function calculated from 100 repetitions.

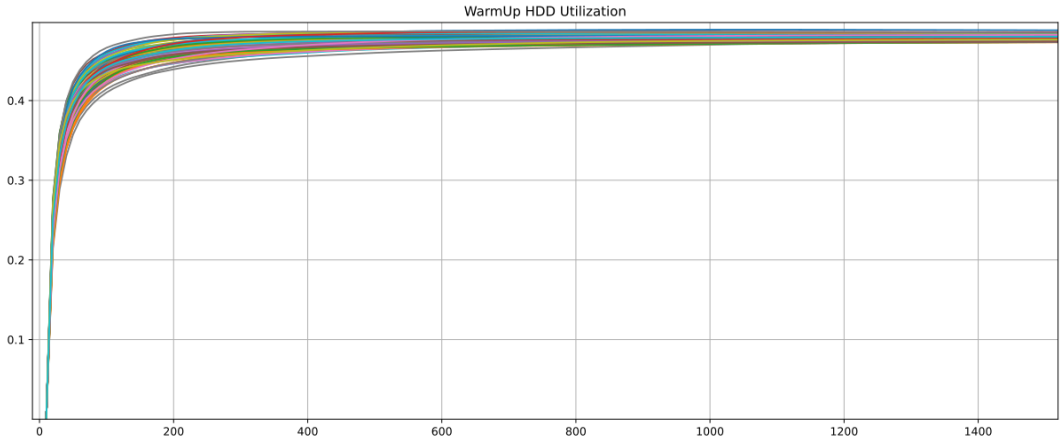

Figure B.3: Time-average function calculated from 100 repetitions.

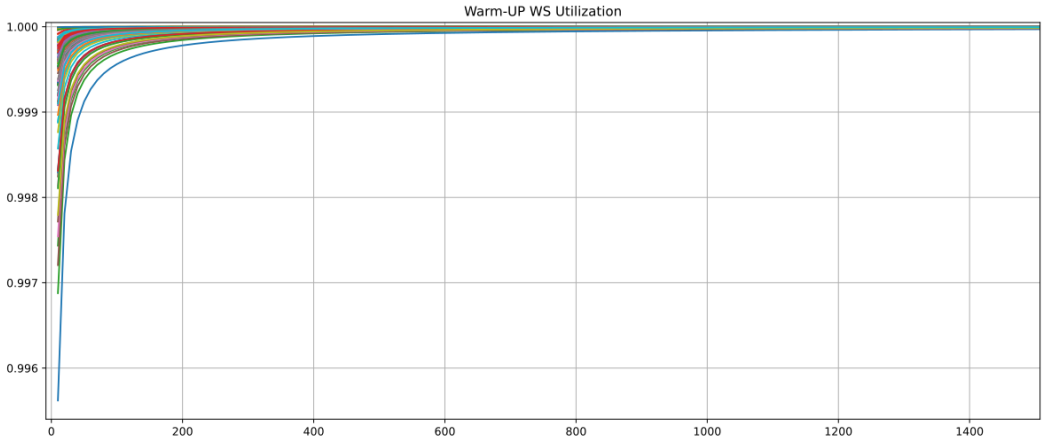

Figure B.4: Time-average function calculated from 100 repetitions.

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