Ca' Foscari University - Venice

Department of Enviromental Sciences, Informatics and Statistics

Software Performance and Scalability [CM0481] (CM90)

IMDb Web Application Performance Evaluation

Students
Davide Tonetto 884585
Carola Pessotto 885384
Thomas Vego Scocco 884984

Academic Year 2023/2024

Contents

1 Introduction **2**

2 Experimental Setup **3**

3 System Design **4**

4 Test Implementation **7**

5 Results **9**

6 System Theoretical Analysis **12**

7 System Design Improvement **17**

Introduction

This report outlines the implementation and comprehensive performance analysis of a web application designed to emulate the IMDb site from Amazon. The developed system allows users to search for a specific movie ID and display key details such as writers, directors, publication year, rating, and actors of the obtained result. The data used is a replica of the original IMDb database available on the official website. We aim to replicate the functional behavior of the original website rather than its appearance. Therefore, we have created a simple interface that allows users to query the aforementioned database.

Experimental Setup

To carry out the tests for the performance evaluation of our web application we need two machines: the first one, called the system under test (also written as SUT), is the one hosting the database to be queried and the web server, while the second one is used to perform the tests on the web application by simulating many user's requests. In particular, we have the following:

- The SUT used for our experiments is a PC with 4 core CPU Intel i-5 5200U @ 2.5GHz, 8GB RAM and 1TB hard disk. The application is written in Python, using FastAPI, a web framework that manages HTTP requests and responses, and Psycopg, a PostgreSQL database adapter, to interact with the database.
- The testing machine used for our experiments is a PC with 8 core and 8 thread CPU AMD Ryzen 9 5900HS @ 3GHz, 24GB RAM and 515GB SSD. On this device, we used the Apache JMeter tool to analyze the performance of our web server.

System Design

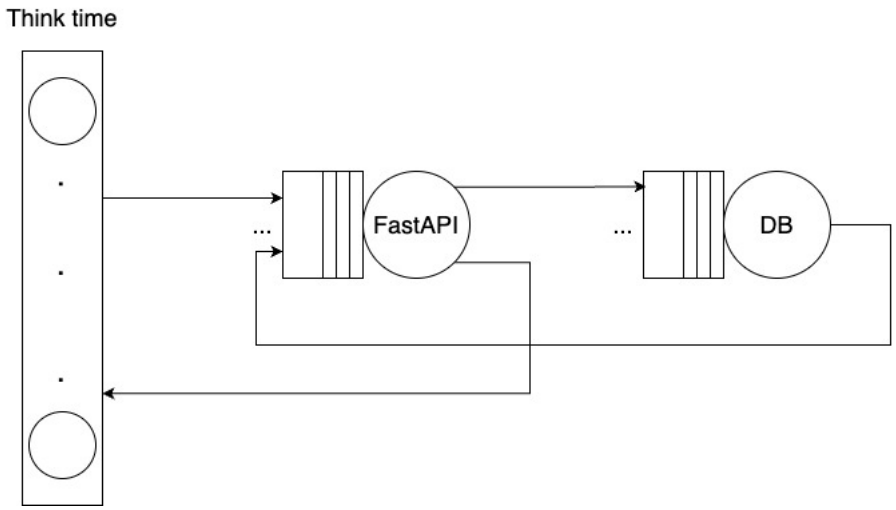

Figure 1

The system, which we show in Figure 1, is composed of an infinite server queue (Think time) and two M/G/1/PS queues (FastAPI and Database). This queuing system represents the architecture of our web application where user interactions are managed through a web server (FastAPI), which in turn interacts with a backend database to fulfil user requests. The workflow is described as follows:

1. Users complete their think time and send a request to the FastAPI server.
2. The FastAPI server receives and processes the request.
3. If data is needed from the database, the FastAPI server sends a query to the database.

4. The database processes the query and returns the data to the FastAPI server.

5. The FastAPI server processes the data (if needed) and sends the response back to the user.

6. The user receives the response and enters the thinking phase before making another request.

From the theoretical point of view, our system architecture can be seen as an interactive system which enjoys the following properties:

- the system is always stable;
- the system's workload is characterized by a fixed number of jobs;
- the system has a constant number of customers;
- the thinking phase is executed independently.

The Processor Sharing (PS) discipline, used to model the last two stations (both physically residing on the SUT machine), is considered a good approximation of standard operating systems' scheduling policies. This is because it assumes that the context switch time is negligible compared to the processing time. PS also provides theoretical benefits, such as insensitivity to the service time distribution, which is crucial for this analysis. Thanks to this property, it can be shown that the results obtained with M/M/1/FIFO are equivalent to those with M/G/1/PS, even without the additional assumption of exponential service times.

We show the M/G/1/PS - M/M/1/FIFO queue formulas:

$$\bar{R} = \frac{1}{\mu - \lambda}$$
$$\bar{N} = \frac{\rho}{1 - \rho}$$
$$\rho = U = \frac{\lambda}{\mu}$$
$$\bar{X} = \frac{\bar{N}}{\bar{R}}$$

where $\bar{R}$ the expected response time, $\bar{N}$ is the expected number of users, $\bar{X}$ is the system throughput,

We model the system as a single class, ensuring that every job is statistically identical to the others. This approach significantly simplifies the

theoretical evaluation of the system. Specifically, after each user request, there is a designated think time, reflecting the real-world scenario where users do not continuously make requests but instead pause between interactions. For the final performance evaluation, only the average statistics of the requests are considered. This averaging allows for a more straightforward analysis, particularly when computing asymptotic bounds and applying Mean Value Analysis (MVA). The single-class assumption reduces the complexity involved in dealing with multiple job classes, enabling a more efficient and manageable assessment of system performance and allowing for deriving accurate performance metrics, such as average response time and throughput, and understanding the system's behaviour under various load conditions.

Test Implementation

The closed-loop workload test was designed to simulate real-world usage scenarios, ensuring efficient system management, even under heavy loads.
It simulates client requests for movie titles, sampling from a pool of 10'000 queries that reflect the popularity of the films based on their actual ratings.

In this experiment, we conducted 10 tests for each progressively increasing number of users, to analyze the average response time compared to the number of users involved.
Those experiments were conducted with the following sequence of number of users [1, 10, 20, 40, 60, 80, 100, 120, 140, 160, 180].
Moreover, each experiment had a loop_count of 20, meaning each user (therefore JMeter's threads) will perform the specified number of requests before stopping.

Each user assumed the following behaviour:

- Accesses the site's homepage;
- Think for a random number of seconds using a Uniform Random Timer with Random Delay Maximum of 3000ms and Constant Delay Offset of 1000ms;
- Sends a request to view a movie's details;
- Think again for another random amount of seconds, always using a Uniform Random Timer but with slightly different parameters: Random Delay Maximum of 4000ms and Constant Delay Offset of 1000ms.

To understand better how the thinking time is calculated, it is important to know how JMeter calculates this time.
The factors used by the Uniform Random Timer are:

- The uniformly-distributed generated value in the range [0.0, 1.0);
- The Random Delay Maximum;
- Constant Delay offset;

So, this timer will put the thread in a thinking state using the simple formula:

$$|X \cdot \mathrm{RDM} + \mathrm{CDO}|$$

where RDM and CDO are respectively the Random Delay Maximum and the Constant Delay Offset, and X is the uniformly-distributed random value.

It is important to underline that the experiments were conducted with a warmed-up system, i.e. with the web server cache full of the information necessary to speed up the request time.
This means that the measured performance was influenced by information already stored in cache, which provides a more realistic representation of system performance under normal operating conditions.

This procedure allowed us to collect empirical data on the performance of the system under different levels of load and to study how the number of users influenced the response time.
The resulting graph represents the average response time of requests of all performed experiments.

Results

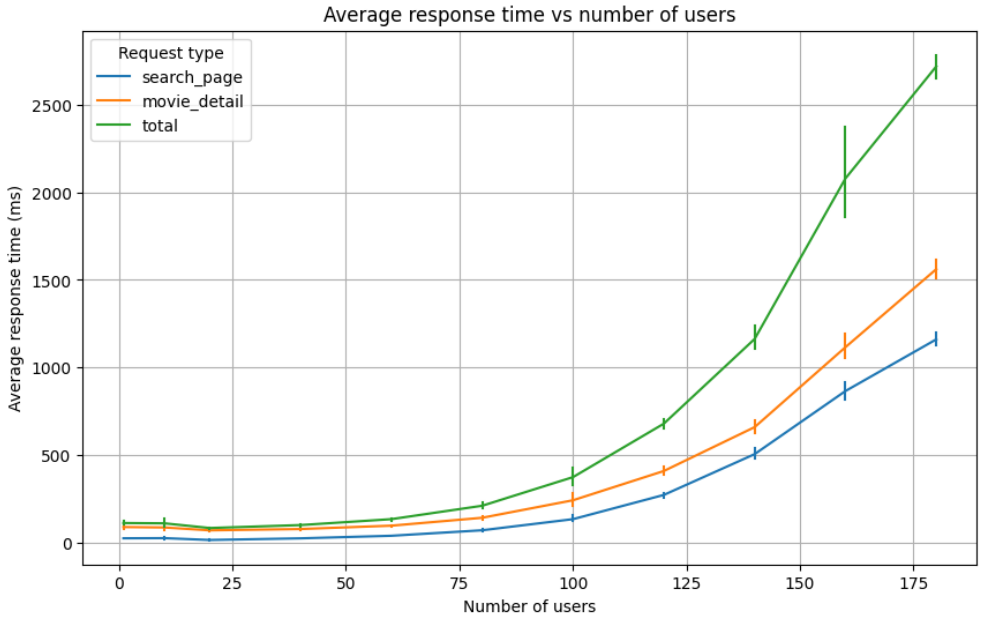

Figure 2: Average response time vs Number of users resulting from the tests described in the previous section (using a 95% confidence interval).

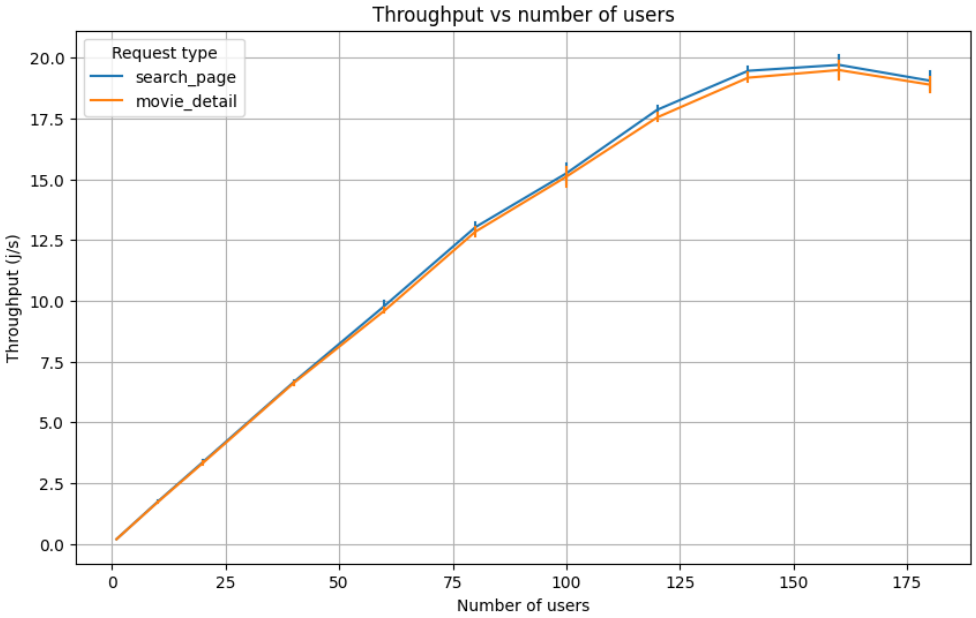

Figure 3: Throughput [$job/sec.$] vs Number of users resulting from the tests described in the previous section (using a 95% confidence interval).

These two graphs represent the empirical results obtained through tests carried out on the web app, in particular the throughput and the average response time as functions of the number of users, divided by the type of request made.

The former shows that the average response time increases as the number of users increases. The increase in response time is quicker for the movie details page than for the search page. This is probably due to the higher complexity of the movie detail page and thus the system requires more time to load it. As we can see, up to around 75 users the response time trend is almost linear, remaining under 500ms, and then growing more and more rapidly until reaching 2000ms with a higher load (around 160 users).

As for the first graph, the latter shows that the throughput of the web app increases as the number of users increases in both request types. However, the throughput trend is not linear: at low loads (up to around 75 users) the throughput increases rapidly, until it slows down at higher loads, around 120 users.

Service Time

During low-load testing with a single user in the system, we've managed to determine the service time of both the server and the database, which are:

$$\mu_{\text{server}} = 0.022945s$$
$$\mu_{\text{db}} = 0.087268s$$

Thinking Time

In our system, threads enter the thinking phase at two distinct moments. These two phases have two different time intervals and we must consider both phases in order to obtain the average thinking time in the system.

We denote with X and Y the two continuous and uniformly distributed random variables that will model these two-time intervals:

$$X \sim U(1, 4)$$
$$Y \sim U(1, 5)$$

and with Z the continuous random variable for the overall thinking time, i.e the sum of the two previous variables:

$$Z = X + Y$$

It is important to note that these two random variables are independent of each other, therefore the following applies:

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

and thus:

$$\mathbb{E}[X] = \frac{1+4}{2} = 2.5s$$
$$\mathbb{E}[Y] = \frac{1+5}{2} = 3s$$

$$\bar{Z} = \mathbb{E}[X] + \mathbb{E}[Y] = 5.5s$$

System Theoretical Analysis

We recall that the web application can be represented as an interactive system, as discussed in the System Design chapter, and its performance metrics can be defined with the following equations:

$$\overline{T}(N) = \overline{Z} + \overline{R}(N)$$
$$\overline{X}(N) = \frac{N}{\overline{T}(N)}$$
$$\overline{R}(N) = \frac{N}{\overline{X}(N)} - \overline{Z}$$

where $\overline{T}(N)$ is the system time, which depends on the number of users, $\overline{Z}$ is the thinking time, and $\overline{R}(N)$ is the expected response time; $\overline{X}(N)$ is the throughput of the system.

Now, we need to identify the bottleneck of the system which can be identified as the component with the highest service demand $\overline{D}_b$ and depends both on the relative visit ratio $\overline{V}_i$ and service time $1/\mu_i$ as follows

$$\overline{D}_i = \overline{V}_i \frac{1}{\mu_i}$$
$$\rho_i = X_1 \overline{D}_i \qquad \text{and } X_i < \frac{1}{\overline{D}_b}$$

The required values of the relative visit ratio in interactive systems are determined by solving the system's traffic equation. For simplicity, the thinking station is chosen as the reference station, with its outflow e_i set to 1.

$$\begin{cases} e_1 = 1 \\ e_2 = e_1 + e_3 \\ e_3 = e_2 \cdot 0.5 \end{cases} \qquad \begin{cases} e_1 = 1 \\ e_2 = 2 \\ e_3 = 1 \end{cases}$$

After some computations, we obtain the service demands for the different components

$$\overline{D}_{server} = \frac{\overline{V}_{server}}{\mu_{server}} = \frac{2}{44} = 0.045$$
$$\overline{D}_{db} = \frac{\overline{V}_{db}}{\mu_{db}} = \frac{1}{11.5} = 0.087$$

which leads us to conclude that the bottleneck is the database.

To study what happens in the limit case, when the system saturation is very low or very high, we compute the asymptotic bounds using the following assumptions, all fulfilled by the theoretical model previously provided in System Design:

- We work with stations with single server assumptions (with the exception of the thinking station that is infinite serves). Please note that when dealing with multiple servers, we can simplify the model by assuming a single server with a speed equal to the speed of one server multiplied by the number of servers. While this assumption may affect predictions under low-load conditions, it remains accurate for high-load scenarios.

- The number of visits at a station does not affect the service time

- There is a single class of users

The asymptotic bounds for an interactive system are as follows:

$$\bar{R} \geq \max(\bar{D}, N\bar{D}_b - \bar{Z})$$
$$X \leq \min\left(\frac{N}{\bar{D} + \bar{Z}}, \frac{1}{\bar{D}_b}\right)$$
$$\text{where} \quad \bar{D} = \sum_{i=2}^{K} \bar{D}_i$$

So, in our case, $\bar{D} = \bar{D}_{server} + \bar{D}_{db} = 0.045 + 0.087 = 0.132$ and

$$\bar{R} \geq \max(0.132, N \cdot 0.087 - 5.5)$$
$$X \leq \min\left(\frac{N}{0.132 + 5.5}, \frac{1}{0.087}\right)$$

Thanks to these results, we also compute the optimal number of users, optimal in the sense that the system is neither under-utilized nor saturated.

$$N_{opt} = \frac{\bar{D} + \bar{Z}}{\bar{D}_b} = \frac{0.132 + 5.5}{0.087} \approx 65$$

N_{opt} corresponds to a situation where the response time is not excessively big, while the throughput remains reasonably high and, from the graphical point of view, it represents the intersection of the bounds referring to the response time and the throughput. Given that the system's theoretical architecture is a closed queuing network with independent and exponentially distributed service times and irreducible probabilistic routing, it is possible to apply the Gordon-Newell theorem and Mean Value Analysis (MVA). The MVA, in particular, is a powerful statistical tool because it enables the computation of mean performance indices for closed product-form queuing networks without the need to directly calculate the distribution's normalizing constant.

We report the plots obtained from the Mean Value Analysis, carried out using Java Modelling Tools.

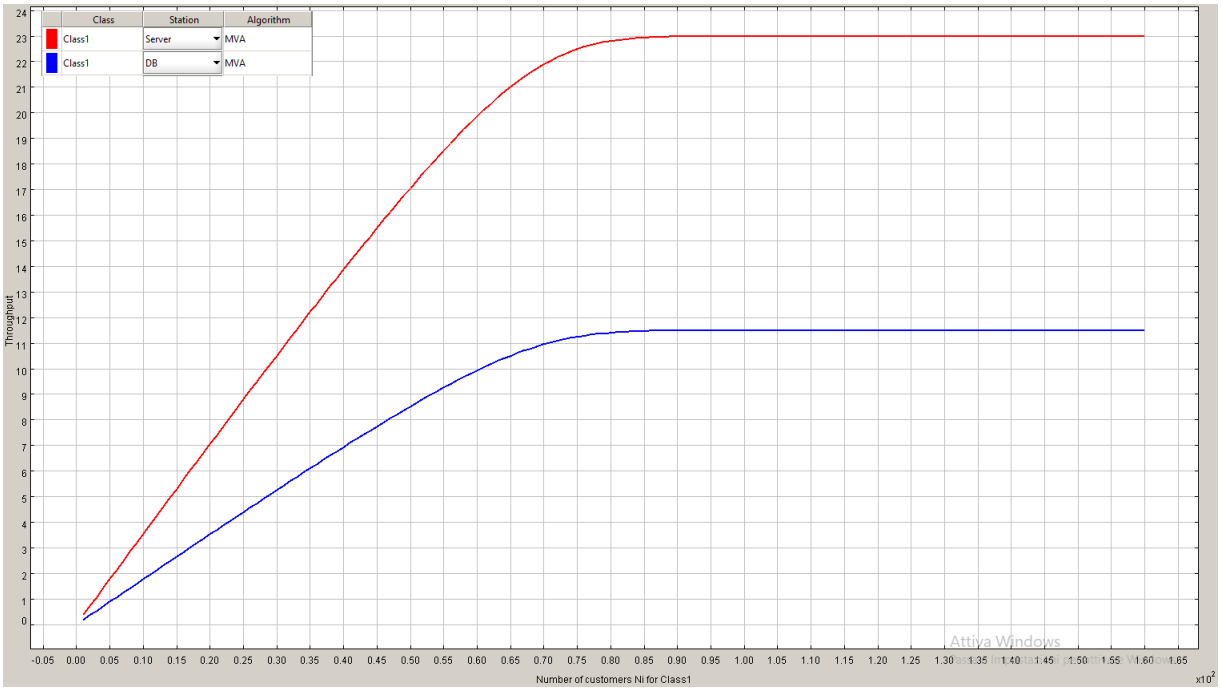

Figure 4: Throughput vs Number of users.

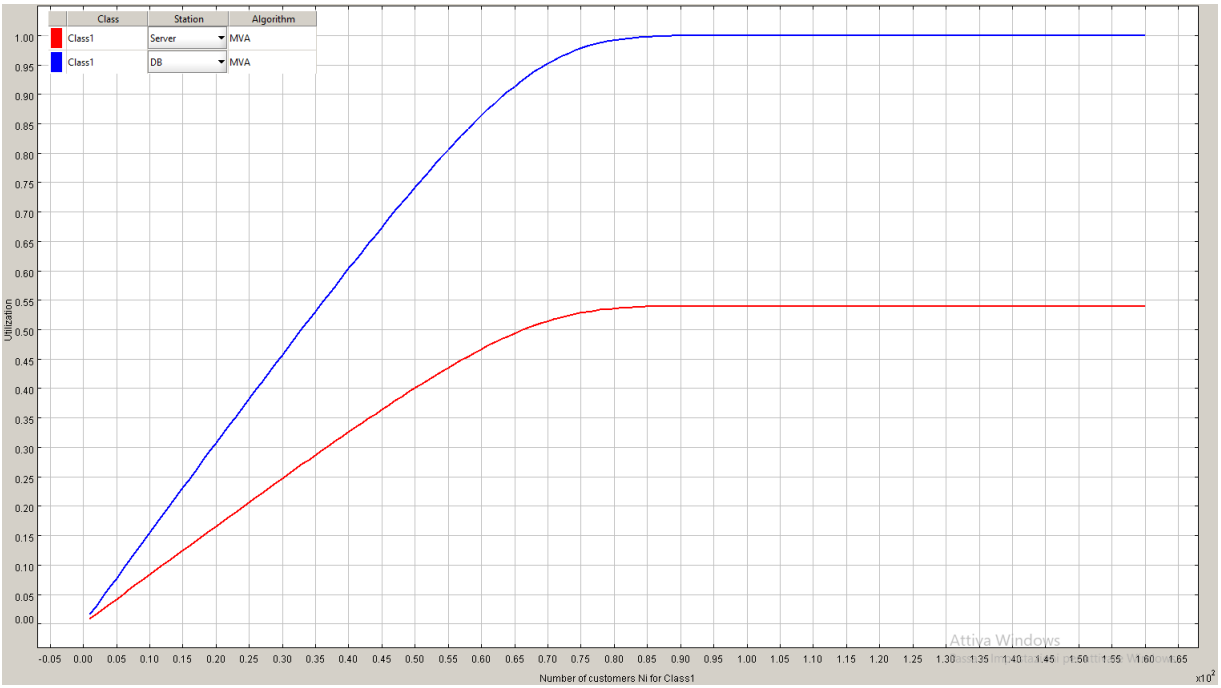

Figure 5: Utilization vs Number of users.

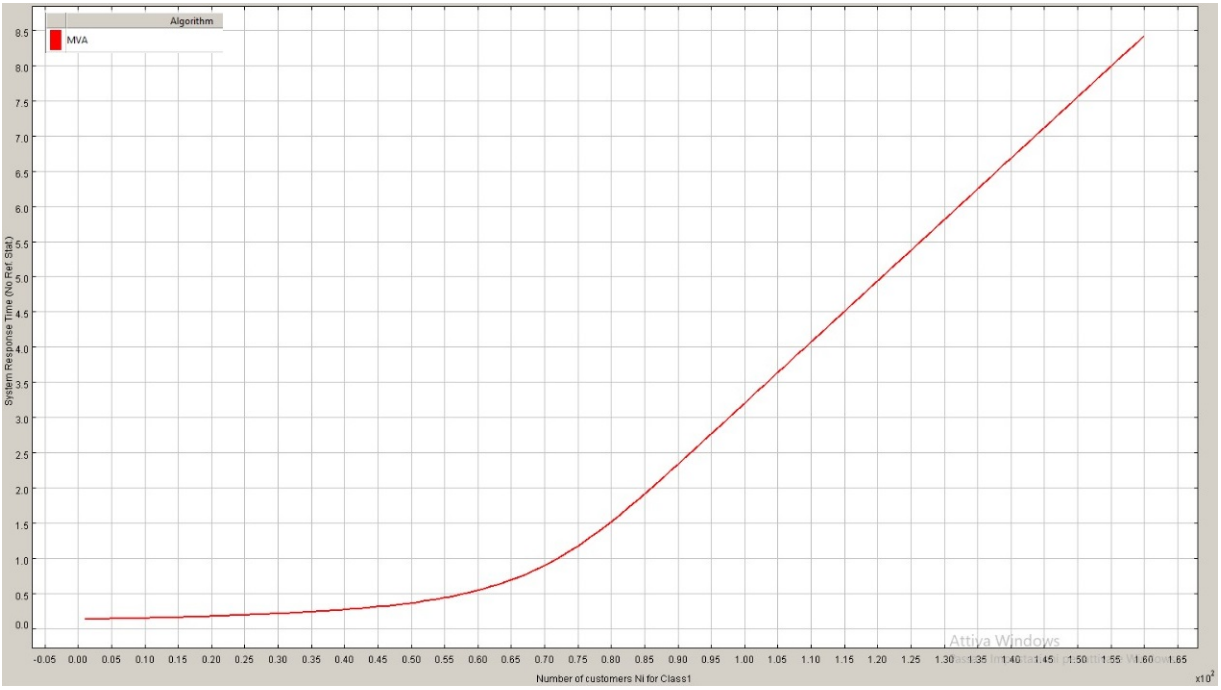

Figure 6: System Response Time (w/o reference station) vs Number of users.

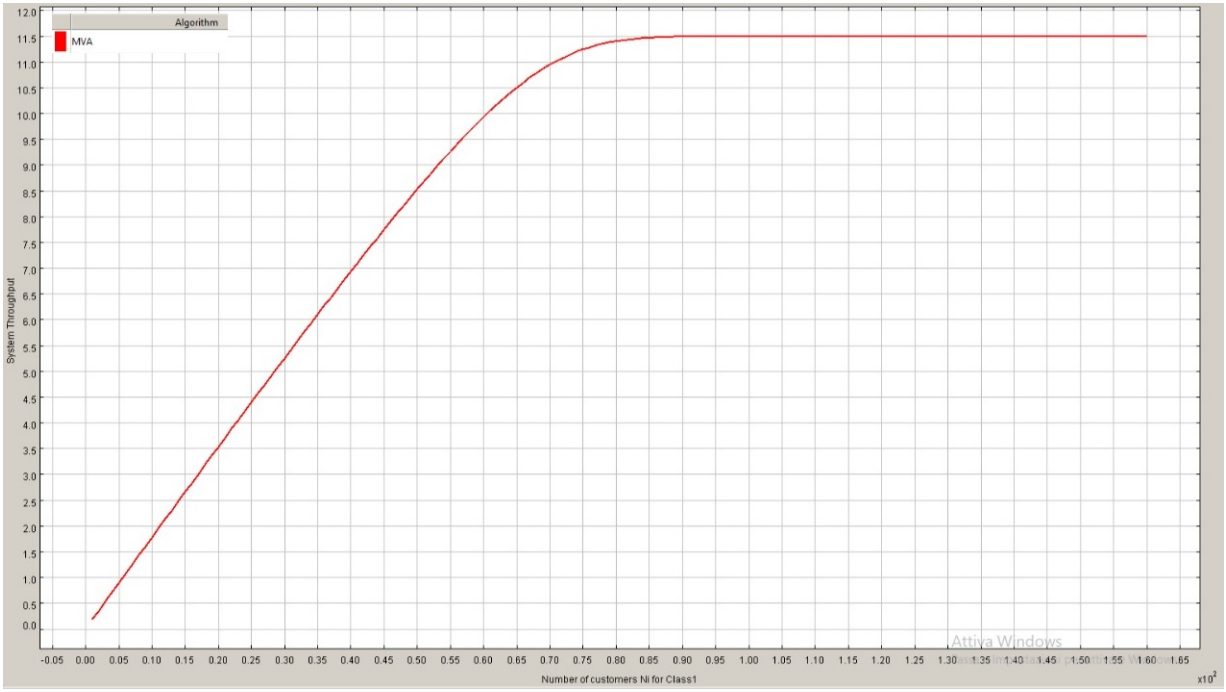

Figure 7: System Throughput vs Number of users.

The plots derived from MVA confirm the results derived from the theoretical formulas and illustrate the trends of the indices as the number of customers varies:

- The bottleneck is identified by observing that the database is the first component to reach the highest utilization.
- The expected response time and throughput adhere to the theoretical asymptotic bounds
- Examining the plots and the indices values for the optimal number of users, the system is observed to be in a good state.
- The expected response time of the system shows an increasing trend as the number of users rises, consistent with the standard closed-loop test plot.

System Design Improvement

To enhance system performance and scalability, it is necessary to address the bottleneck by making it scalable and achieving better resource balancing. The proposed solution involves replicating the database component, consequently distributing the workload more evenly among multiple elements. Another critical consideration is how requests are routed from the server to the new database replicas. A straightforward yet highly effective approach is to use a random dispatcher. In this setup, where all database replicas share the same service time distribution, a random dispatcher is found to be effective, largely due to its stateless nature.

This new system model is illustrated in Figure 8.

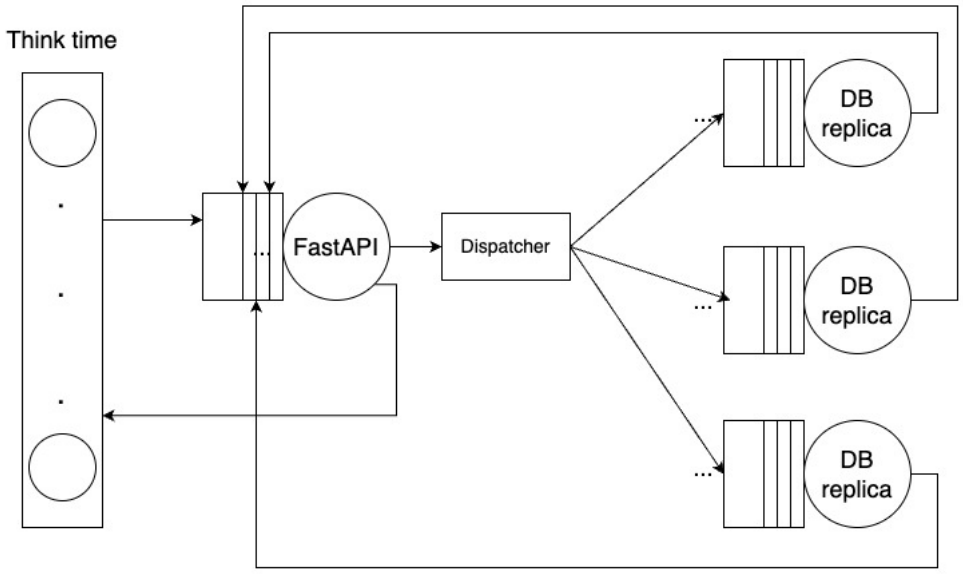

Figure 8

As before, the first step of this new theoretical system analysis is to solve

the traffic equations and determine the relative visit ratio for each station as follows:

$$\begin{cases} e_1 = 1 \\ e_2 = e_1 + e_3 + e_4 + e_5 \\ e_3 = 0.5 \cdot e_2 \cdot 0.33 = 0.165 \cdot e_2 \\ e_4 = 0.5 \cdot e_2 \cdot 0.33 = 0.165 \cdot e_2 \\ e_5 = 0.5 \cdot e_2 \cdot 0.33 = 0.165 \cdot e_2 \end{cases} \quad \begin{cases} e_1 = 1 \\ e_2 = 2 \\ e_3 = 0.33 \\ e_4 = 0.33 \\ e_5 = 0.33 \end{cases}$$

The next step is to calculate the service demand for each station:

$$\overline{D}_{server} = \frac{2}{44} = 0.045$$
$$\overline{D}_{db1} = \overline{D}_{db2} = \overline{D}_{db3} = \frac{0.33}{11.5} = 0.029$$

The bottleneck has shifted and the highest service demand is now related to the FastAPI component $\bar{D}_{server}$. With this information, the asymptotic bounds can also be computed as follows

$$\bar{D} = 0.045 + 0.029 * 3 = 0.074$$
$$\bar{R} \geq \max(0.074, N \cdot 0.045 - 5.5)$$
$$X \leq \min\left(\frac{N}{0.074 + 5.5}, \frac{1}{0.045}\right)$$

Furthermore, the new optimal number of users is

$$N_{opt} = \frac{0.074 + 5.5}{0.045} \approx 124$$

Our modification leads to an increment of the optimal number of users from 65 to 124. These results are confirmed also by the Mean Value Analysis (MVA) performed using JMT. The MVA plots of the improved system architecture are shown below.

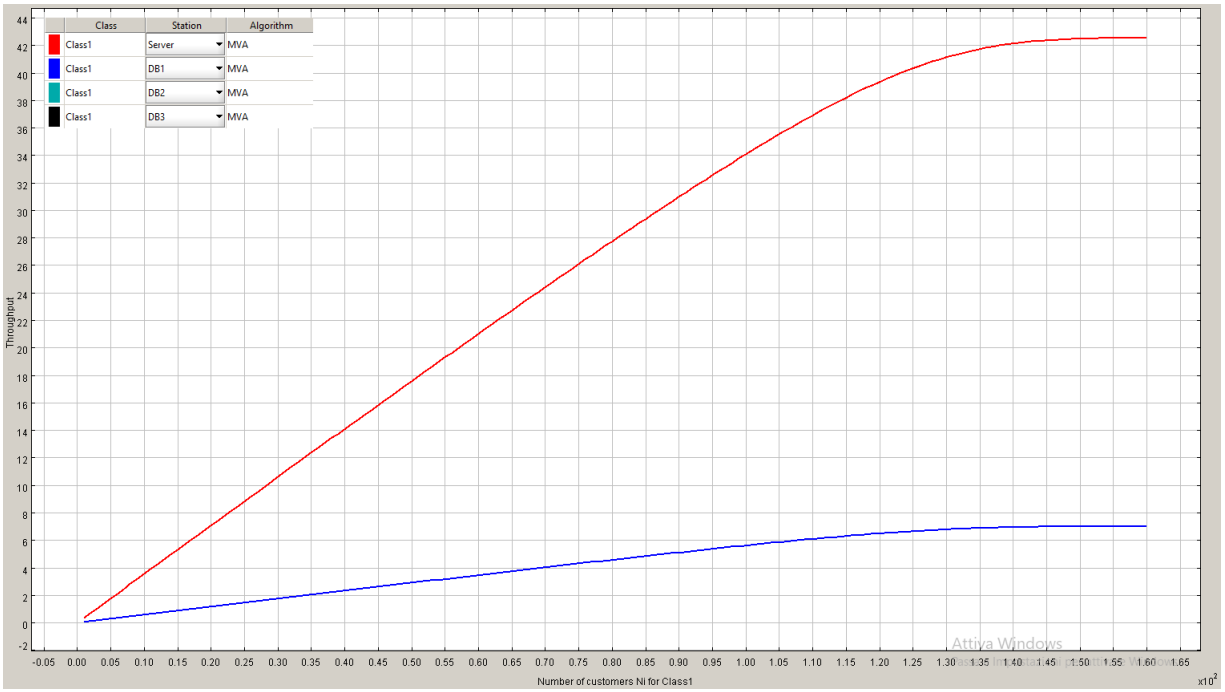

Figure 9: Throughput vs Number of users for the improved system.

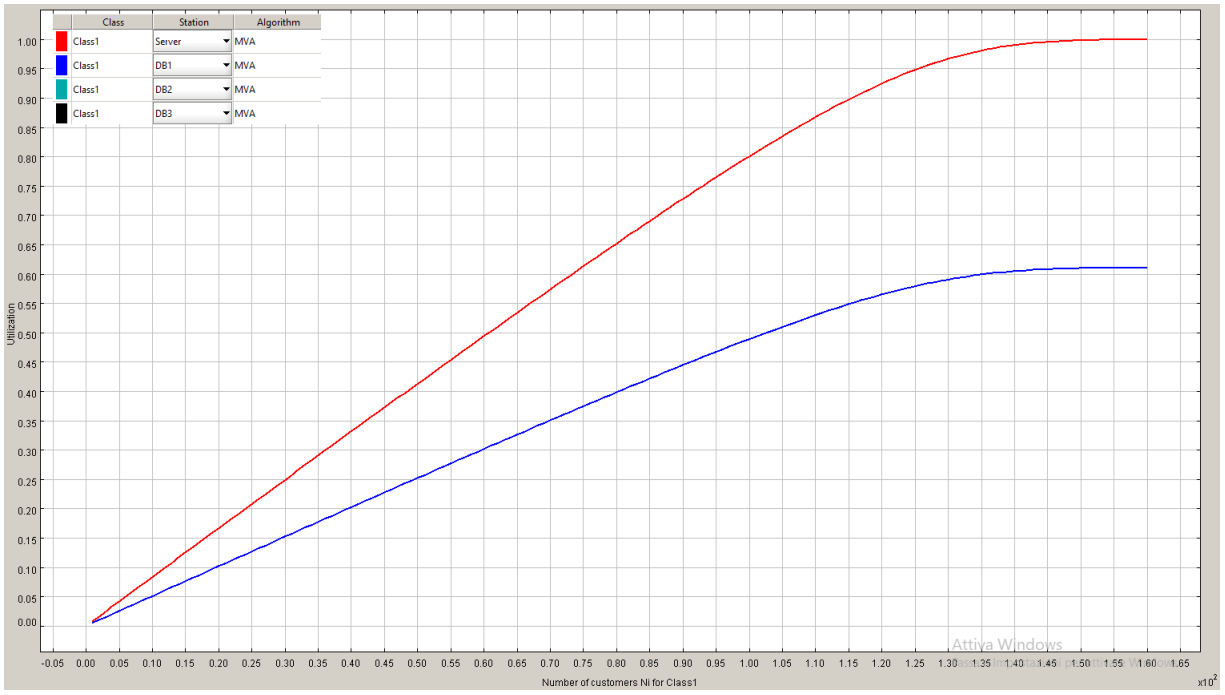

Figure 10: Utilization vs Number of users for the improved system.

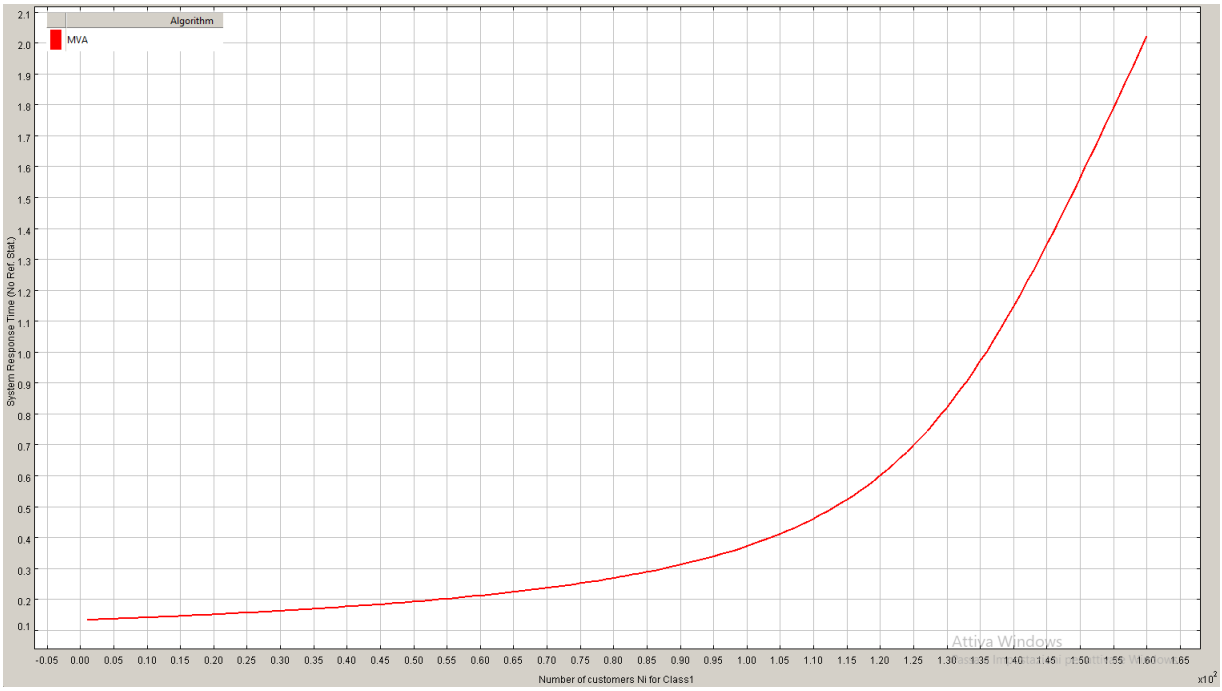

Figure 11: System Response Time (w/o reference station) vs Number of users for the improved system.

Figure 12: System Throughput vs Number of users for the improved system.

Conclusion

This report provided a detailed implementation and performance evaluation of a web application designed to replicate the functionality of the IMDb website. Several key insights into the system's performance were uncovered through a series of methodical tests and theoretical analyses.

Firstly, the performance evaluation identified the database as the primary bottleneck, being the component with the highest service demand. This was confirmed by both theoretical formulas and empirical data obtained from the Mean Value Analysis (MVA), which indicated that users tend to accumulate at the slowest station in the system.

Secondly, the expected response time and throughput closely adhered to the theoretical asymptotic bounds. The plots obtained from the MVA confirmed the trends predicted by the theoretical models, showing a clear relationship between the number of users and the system's performance metrics.

Moreover, it was observed that the system maintains good performance up to the optimal number of users. Beyond this point, the expected response time shows an increasing trend, as typically seen in closed-loop system tests. This finding underscores the importance of managing the number of concurrent users to ensure optimal performance.

Additionally, the proposed system improvement involved replicating the database component to distribute the workload more evenly, thereby enhancing scalability and resource balancing. This modification shifted the bottleneck to the FastAPI component and significantly increased the optimal number of users, confirming the efficacy of this improvement through MVA plots.

In conclusion, the study demonstrates that a careful balance of system resources and thoughtful architecture design can significantly enhance the performance and scalability of web applications. The insights gained from this analysis provide a solid foundation for further optimization and development of similar systems.