Optimizing Performance of Distributed Web Systems

Marian ILEANA
National University of Science and Technology Politehnica Bucharest,
Pitești University Center, Romania
marianileana95@gmail.com

Distributed systems have been a hot topic of study in recent years. In general, a distributed system is defined as a computer system implemented within a computer network in which both software and hardware components located on computers within the network communicate and coordinate their actions via messages. Optimizing the performance of distributed web systems has become a top priority in the current context of technological evolution and the continuous growth in the number of users and online traffic. These systems are fundamental to delivering services and content to users worldwide, but the ever-increasing demands for availability and scalability have brought new challenges. Finite state machines are a powerful and flexible technique for modelling distributed systems using the Python language. Finite state machines are popular for modelling distributed systems due to the fact that they represent an easy and flexible way to represent these systems, and together with the Python language, they manage to model complex systems. The two are used to illustrate complex system behaviour, communication between system nodes, node synchronization, error handling, and performance optimization. In this study, various strategies and techniques for optimizing their performance have been analysed.

Keywords: *Distributed Systems, Performance Optimization, Web Architecture, Scalability, Load Balancing, Finite State Machine, Python*
DOI: 10.24818/issn14531305/27.4.2023.06

1 Introduction

Nowadays, we live in an increasingly interconnected digital age, and distributed web systems play a major role in ensuring that online services and applications operate at optimal capacity [1]. Architects, together with software developers, are looking to optimize these systems as performance and scalability requirements continue to increase [2].

This article focuses on optimizing the performance of these systems, offering innovative strategies and solutions to achieve maximum efficiency and the best possible user experience. It will examine the main issues, such as handling heavy traffic and managing real-time data, and examine ways in which these issues can be avoided.

Concepts such as scalable architecture, load balancing, optimizing distributed databases, and using the latest caching technologies to speed up responses from the server to the users will be discussed. In addition, methods for improving network latency and reducing downtime will be examined.

By using the interpreted programming language Python and its libraries, among the most important are PyTransition and Matplotlib. Using the free and open-source PyTransition framework, one can manage transitions or changes within a system, used to define and counter how various states or stages evolve and change [6]. Transitions change as the network evolves. The Matplotlib library is one of the most important visualization libraries of the Python programming language, created to render the highest quality graphs and charts [7].

Whether you are a student, a professor, an architect about to design a major distributed system for your company, or even a developer with a web application in development. This article aims to provide insight into performance optimization approaches so that you can achieve your efficiency goals and keep your systems running smoothly in the dynamic environment of modern technologies.

2 Literature review

In order to see the dynamics of the subject of distributed web systems within the specialized literature, a brief analysis of the literature was carried out that focuses on the subject of distributed web systems, a topic debated by previous authors.

In carrying out this empirical investigation, the scientific database Dimensions.ai was used as a search space, and the results returned by it were processed using the VOS Viewer application in order to be able to identify the key authors found in the research.

Using the application methodology, a study was conducted. Authors who published on the topic "distributed web systems" were searched, and the results were gathered by a network of authors, as illustrated in Figure 1.. The obtained results included a total of 9006 authors, these are the authors with publications on this subject; for an author to be revealed, the condition was added that his name appear in at least five publications. Thus, a number of 102 authors who meet this threshold were filtered, based on which the graph in Figure 1 was created.

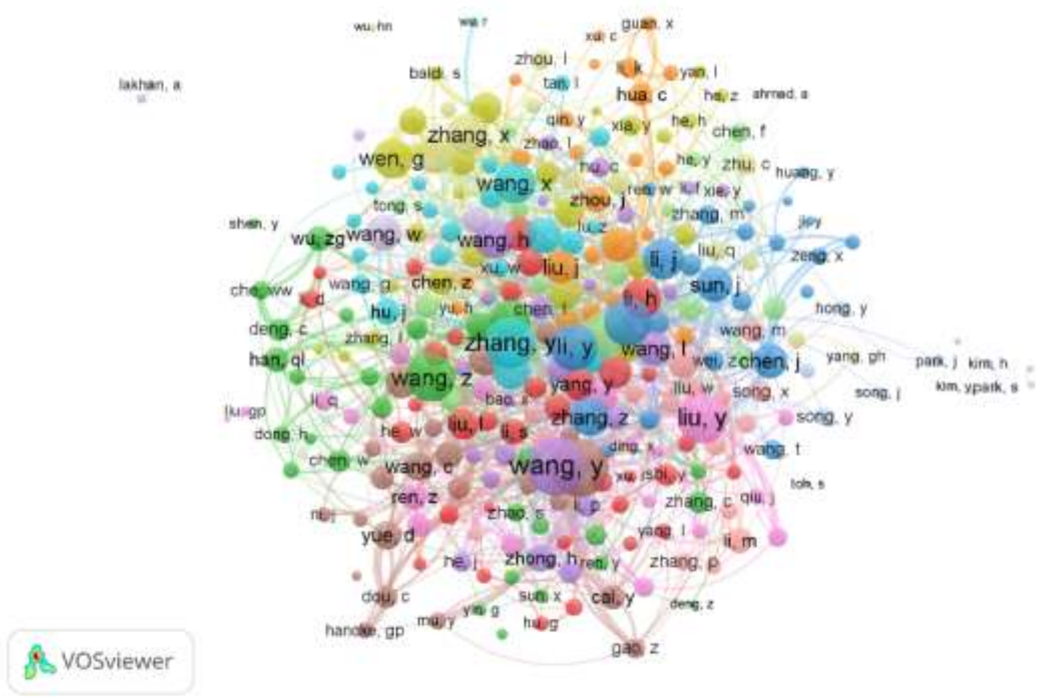

Fig 1. Result of the bibliographical study

The authors expose some of the most common problems of distributed web systems, including [4]:

1. Communication between components
2. Data consistency
3. Failure tolerance
4. Management of resources
5. Security and privacy
6. Synchronization
7. Management of the transaction
8. Performance
9. Scalability
10. Administration and monitoring

The literature on distributed web systems provides a deep understanding of the technological advances and the problems that may arise during the study, research, and management of such systems with a high degree of complexity.

This branch of literature brings to the fore how distributed architectures have become essential to the smooth functioning of the online platforms we are all used to, including e-commerce, social networks, and web services.

The literature in this field examines concepts such as horizontal scalability, traffic management and processing methods, load distribution, and ensuring redundancy to manage availability and performance in the face of fluctuating user demands. In addition, it focuses on concrete issues such as data synchronization, data security, and error resolution in legacy infrastructures. In general, the literature brings to the attention of the reading public how distributed web systems have become essential in modern society.

The contemporary online environment is the basis of modern society. It highlights ongoing efforts to address and resolve the complex issues and challenges posed by this essential technology.

A mechanism for assessing the scalability of a distributed system was proposed in a study by Jogalekar and Woodside in 2000. They concluded that the system must be scalable, which means that it must be able to be modified in different configurations and dimensions, which is the most important element of a distributed system [1]. In their paper, it is pointed out that a system is infinitely scalable if the degradation of the response time is allowed, is directly proportional to the number of users, and is included in the scalability function.

In the book "Designing Data-Intensive Applications", M. Kleppmann brings to the reader's attention the design and construction of distributed web applications that can manage large volumes of data. The author explores concepts such as data storage, consistency, scalability, and fault torrenting [3]. The book provides a detailed analysis of the technologies needed to develop powerful and efficient web platforms, such as distributed databases, real-time messaging systems, and more. A bible of this field written by A. S. Tanenbaum and M. van Steen, "Distributed Systems: Principles and Paradigms". This draws attention to the principles of distributed systems. The two authors address areas such as inter-process communication, resource distribution, defect management, and, last but not least, data consistency and coherence [4]. In addition, the book covers important concepts such as service-oriented architectures (SOA) and cloud architectures.

"Distributed Systems: Concepts and Design" by G. Coulouris, J. Dollimore, and T. Kindberg brings to the attention of the reading public the essential ideas, technologies, and principles of distributed systems. The authors scrutinize distributed systems in terms of inter-process communication, resource distribution, data security, and fault tolerance [5]. To be able to illustrate how these concepts are applied to distributed web platforms, the book provides appropriate examples and also brings a number of case studies.

3 Model description

3.1. Initial Model: Web Distributed Systems Architecture

To provide a wide range of services and engaging content to users who want memorable online experiences, they need distributed web systems. These are complex entities that are made up of a network of interconnected servers. The performance that must be high is essential to satisfying the diverse needs of users who have a high degree of expectation regarding the speed of the platform and the connection [1]. The high quality of their infrastructure determines the popularity of platforms and stores that gravitate toward the virtual sphere. By increasing quality, it is desired to increase the level of consumer satisfaction and extend the duration of interaction with the platform, which directly leads to an increase in the financial income of the respective virtual business.

3.1.1 Performance Optimization Challenges

To provide users with a fast and secure experience, optimizing performance in distributed web systems is a complex and essential process. This means overcoming several key challenges, such as [1]:

- Latency - Ensuring minimal delay in delivering content to users;
- Load Balancing - Distributing workload evenly across servers;
- Caching - Keeping frequently accessed data near users;
- Fault Tolerance - Maintaining system availability in case of failures.

Latency is the time it takes for data to travel between the two actors, client and server, resulting in increased web page load times. Ways to reduce latency [1]:

- Networks for Content Delivery (CDN): Content delivery networks allow content to be stored on servers located in different geographic locations so that users can connect to the closest server.
- Optimization of resources and images: Resizing and compressing images can lead to shorter loading times and page sizes; these methods can also be applied to static resources.
- Reducing the number of HTTP requests: Concatenation and code minification methods are used to reduce the number of HTTP requests by combining these resources.

3.1.2 Load Balancing

Load balancing optimizes performance in distributed web systems. Load balancing evenly distributes network traffic across multiple servers, ensuring scalability, reliability, and efficient resource utilization [1]. Techniques like round-robin and weighted distribution minimize response time and enhance system performance. As can be seen in Figure 2, the devices connect to the Internet, and between the Internet and the servers, there is software and hardware that correctly distributes the traffic so as not to overload a certain server.

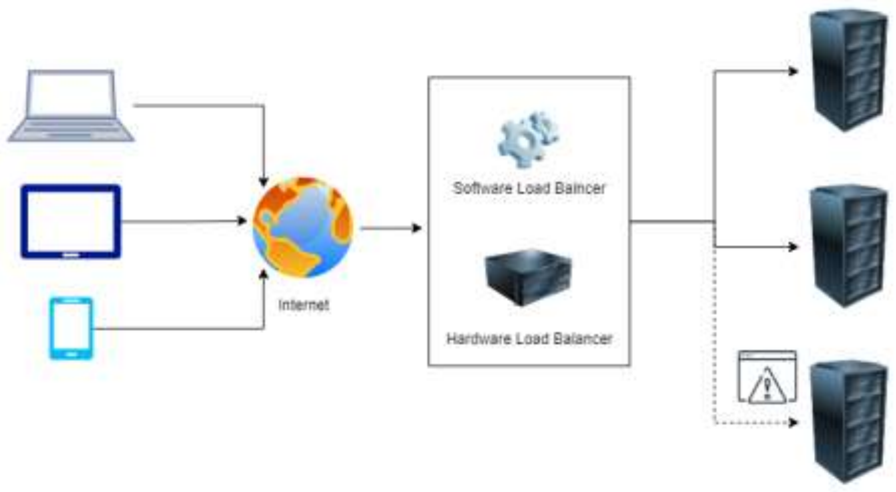

Fig 2. Load Balancing – Example

3.1.3 Caching

Caching is an effective technique for improving the performance of distributed web systems. By temporarily storing data or pre-computed results in a fast-access area such as cache memory, response times can be significantly reduced [1]. Caching can be applied not only at the level of individual servers but also at the network or client level.

Caching can be applied to multiple levels of distributed web systems. At the individual server level, caching can represent the storage of precomputed results from frequent queries. This helps reduce the processing time of requests from customers.

When talking about the network layer, cache memory is used to store copies of data from distributed servers [1]. This helps reduce network traffic, as clients no longer have to connect to each server to get the data they need. Copies of pre-calculated data or results can be stored at the client level. This helps to enhance the user experience.

Caching is an effective technique that can be successfully used to improve the performance of distributed web systems [1]. By temporarily storing data in a fast-access area, it significantly reduces the user's response time (Figure 3).

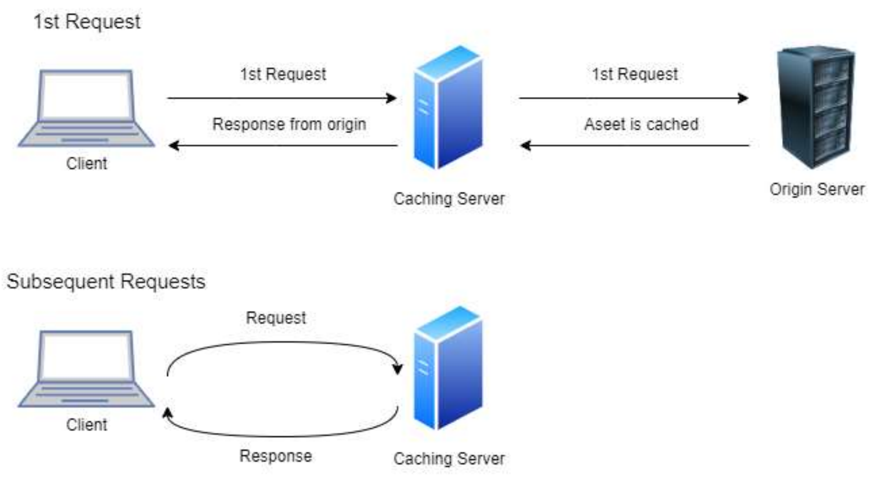

Fig 3. Cache Definition

3.1.4 Content Delivery Networks

CDNs are geographically distributed networks of servers that optimize the delivery of web content to users (Figure 4). By caching and distributing data efficiently, CDNs reduce latency and provide improved performance and security [1].

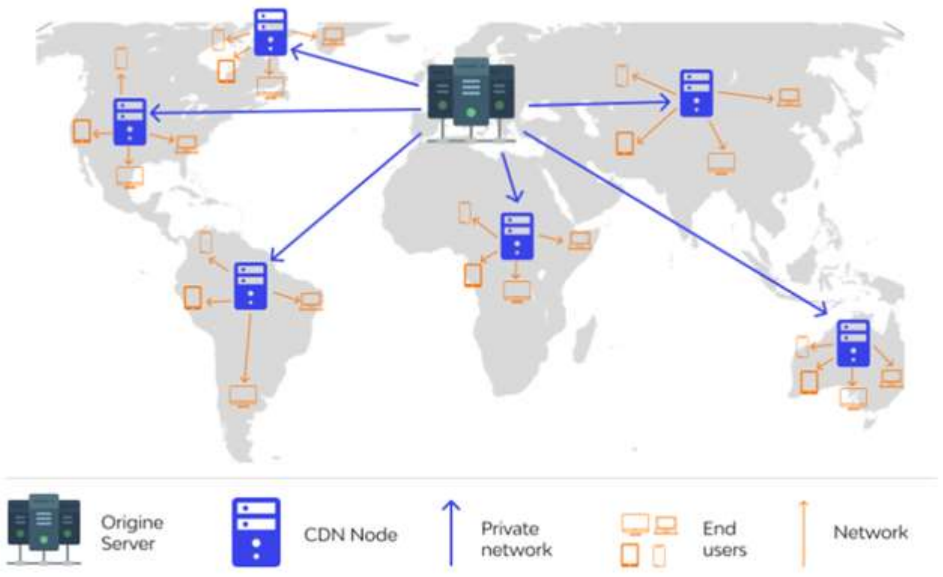

Fig 4. Content Delivery Networks [11]

Geographic load balancing is a technique used to distribute web traffic between multiple servers located in different geographic regions. This helps improve the performance of web applications by reducing the distance data travels and ensuring that users are served by the server closest to them.

There are two main approaches to geographical load balancing in the specialized literature. DNS-based load balancing: This method uses the Domain Name System (DNS) to route requests to the nearest server. When a user enters a URL into a browser, the DNS server returns the IP address of the server closest to the client.

HTTP-based load balancing: This method uses HTTP headers to direct requests to the closest available server. When the server receives a request, it examines the HTTP headers to determine the user's location. The server

will then forward the request to the server closest to the user.
Geographical load balancing is the most effective way to improve the performance of web platforms. The user experience is also improved by reducing request latency.
The most important advantages of using geographic load balancing for scalable distributed web systems are [2]:

- Improved performance: By distributing web traffic across multiple servers, local load balancing can help reduce the load on each server. This can lead to better performance, especially for web applications that experience traffic spikes (times when traffic increases suddenly).
- Reduced latency: By forwarding requests to the server closest to the user, geographic load balancing can help reduce latency. This can improve the user experience, especially for applications that require real-time data or interactions.
- Increased Availability: By distributing web traffic across multiple servers, geographic load balancing helps increase the availability of web applications. If one web server goes down, other servers can continue to serve requests.
- Improved scalability: By adjusting the geographic load, the scalability of web platforms is improved. As the number of users increases, additional servers can be added to the load-balancing group.

The most common challenges in using load optimization for scalable distributed web systems are [1]:

- Complexity: Geographical load balancing can be a highly complex and difficult operation to implement. It requires careful planning and coordination; developers must be careful that the load is distributed evenly across all servers.
- Cost: Geographical load balancing requires additional servers and network infrastructure, which can increase overall costs.
- Management: Geographic load balancing can be difficult to implement and manage because it requires monitoring and calibration to ensure that the load is evenly distributed and that the system is operating within parameters.

In most cases, geographic load balancing is a valuable tool for improving the performance, scalability, and availability of web applications. However, it is important to consider the challenges of such an implementation.

3.2 Revised Model: Distributed Web Systems Using FSM

Using a finite state machine, we can model distributed web systems. With the help of FSMs, we can track the states that the system goes through. Using the Python language together with PyTransition and Matplotlib, it is possible to visualize the states that the distributed system goes through.
Starting from the following finite state machine. Transitions "Request1", "Request2", and "Request3" that go from the "Idle" state to the "Processing" state, and the "ErrorHandling" transaction goes from the "Processing" state to the "Error" state (Figure 5).

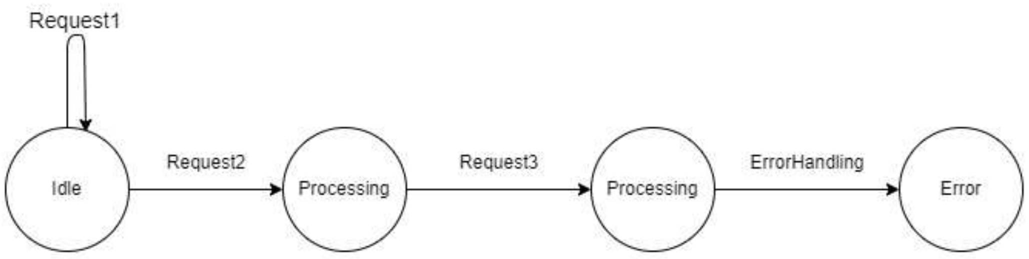

Fig 5. FSM modelling

A finite state machine can be modelled in Python using the following code:

```python
from transitions import Machine
import matplotlib.pyplot as plt

class DistributedSystem:
    def __init__(self, name):
        self.name = name
        self.states = ['Idle', 'Processing', 'Error']
        self.transitions = [
            {'trigger': 'receive_request1', 'source': 'Idle', 'dest': 'Processing'},
            {'trigger': 'receive_request2', 'source': 'Idle', 'dest': 'Processing'},
            {'trigger': 'receive_request3', 'source': 'Idle', 'dest': 'Processing'},
            {'trigger': 'process_request1', 'source': 'Processing', 'dest': 'Idle'},
            {'trigger': 'process_request2', 'source': 'Processing', 'dest': 'Idle'},
            {'trigger': 'process_request3', 'source': 'Processing', 'dest': 'Idle'},
            {'trigger': 'error_handling', 'source': 'Processing', 'dest': 'Error'}
        ]
        self.machine = Machine(model=self, states=self.states, transitions=self.transitions,
initial='Idle')
```

The function of drawing the obtained result is:

```python
def draw_fsm(self):
    G = nx.DiGraph()
    G.add_nodes_from(self.states)
    for transition in self.transitions:
        G.add_edge(transition['source'], transition['dest'], label=transition['trigger'])

    pos = nx.spring_layout(G)

    plt.figure(figsize=(8, 6))
    nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=2000,
edge_color='gray', arrows=True)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=nx.get_edge_attributes(G, 'label'))
    plt.title(f'FSM Graph - {self.name}', fontsize=12)
    plt.tight_layout()
    plt.savefig(f'fsm_graph_{self.name}.png', format='png')
    plt.show()
```

The result obtained by modelling the system and calling the previously presented drawing function is presented in Figure 6.

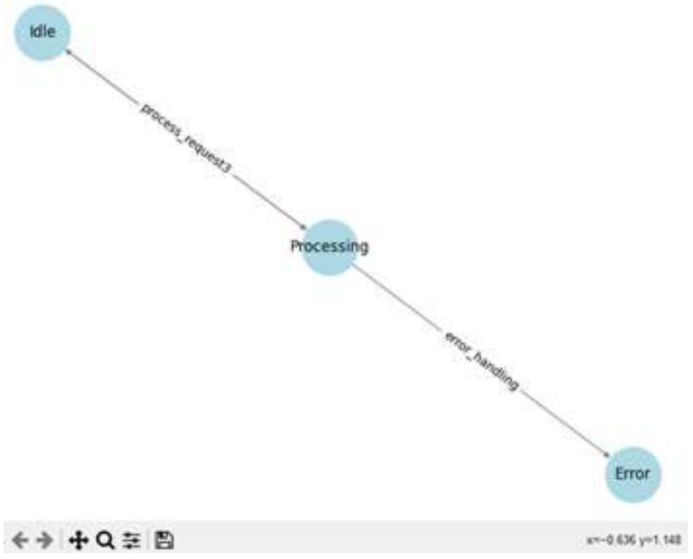

Fig 6. The resulting diagram

4 Reviewing process

The development and optimization of distributed web network systems have behind them an exhaustive review process to be able to ensure the reliability, efficiency, and scalability of the implemented solutions. Modelling the behaviour of the system using finite state machines (FSM) using the libraries provided by Python, PyTransition, and Matplotlib brings a new approach to the design of such a system. However, the effectiveness of such a system must be carefully evaluated and validated.

By modelling from the previous point, we want to bring into discussion another perspective that wants to improve the performance of distributed web systems. With technological change and the increased complexity of requirements in the field of distributed web systems, the design and development processes have become more important than ever [8]. In search of new, more efficient methods with higher scalability, developers eagerly embraced new methods and tools to be able to turn their visions into reality.

In this case, a type of distributed web modelling is developed that is supported by modern and innovative tools. Using these tools, developers are able to create a distributed web system that successfully meets the needs of today but also prepares the system for the demands of the future. One of the defining aspects of this approach is the adoption of finite state machines (FSMs) as a way to represent the complex behaviour of systems. In what was previously described, these new types of design were examined, and the main tools that can be used to support and define them were presented, demonstrating their impact and value in the design and development of new distributed web systems.

Finite State Machines (FSMs) represent a modern paradigm of system modelling. Finite state machines have constantly evolved in various disciplines to become important tools in the development and testing of complex systems. In traditional methods, distributed web systems are built using complex graphs that are often difficult to manipulate and scale. By adopting FSM, developers can define system behaviour in a structured and understandable way. Each state of the FSM represents a different level of system operation, and transitions between states represent changes from one level to another.

Modern tools: Python, PyTransition, and Matplotlib [6], [7]. Modern tools have played an important role in the development of this modelling system. The Python programming language provides a powerful and versatile framework for implementing finite state machines and operations associated with distributed web systems. Pytransition, a Python library specializing in managing finite state machines, provides developers with a set of tools needed to define and manage transitions between states as well as control the behaviour associated with each state. Matplotlib, another Python library, completes the triangle of modern tools by providing advanced visualization capabilities. With Matplotlib, developers can create graphical representations of finite state machines, making it easier to understand and optimize the behaviour of distributed web systems [1].

The benefits of this approach. This modern way of presenting a distributed web network comes with a lot of advantages. The transparent nature of the finite state machine simplifies the development and debugging processes, while the graphical visualization provides a quick overview of the system's behaviour [9]. In addition, the system is highly modular, allowing developers to add and change states and transitions easily as system requirements evolve as the number of users increases.

Modelling distributed web networks using this new model, based on FSM and modern tools such as Python, PyTransition, and Matplotlib, represents a significant paradigm shift in the field of complex systems development. This approach emphasizes efficiency, scalability, and creating sustainable systems that meet future challenges through high adaptability.

5 Conclusions

In this paper, strategies and approaches have been explored to optimize the performance of distributed web systems using finite state machine models. It has been demonstrated how the dynamic programming language Python and the PyTransition library can be used to create and manage such models, and the results obtained highlight the significant benefits of these techniques in improving the performance monitoring of distributed web systems.

By using finite state machines, it was possible to capture the essential behaviour of our web system, allowing critical performance aspects to be identified and optimized. This enables informed decisions about scalability, efficiency, and resource utilization.

An FSM model was implemented in Python using the PyTransition library, and the graphics determining the behaviour of the FSM were made with Matplotlib. It has been demonstrated how, using this model, one can create test cases that simulate different types of web traffic. By implementing the two libraries, they allow the definition, simulation, and analysis of the transition states of the system with ease. The easy integration of Matplotlib allows efficient visualization of the system's performance in the various proposed scenarios, helping to identify potential weak points and bottlenecks [10].

The obtained results show that FSM is an effective technique to optimize the performance of a distributed web system. FSM can be successfully used to model user and system behaviour as well as create test cases to identify and fix performance issues.

Based on the obtained results, a significant improvement in system performance was observed when the optimization measures suggested by the model analysis were applied. This highlights the importance of using finite state machines and modelling approaches using the Python language to achieve optimization of distributed web systems.

Starting from the expertise accumulated for the realization of this material, other research directions that will improve distributed web systems and from which future research can start and expand in the following directions are:

- Using FSM to Model User Behaviour and Complex Web Systems;
- Development of test scenario automation techniques using FSM;
- Using FSM to optimize system performance under real traffic conditions.

In conclusion, this paper highlights the benefits of using machines to optimize the performance of distributed web systems. The integration of Python and the PyTransition and Matplotlib libraries plays a significant role in addressing and solving complex performance and scalability challenges in such environments. Although this research has been successfully accomplished, there is potential for future research and improvements in the approaches and techniques used to meet the changing needs of the increasingly complex and demanding distributed web systems in the modern world.

References

[1] P. Jogalekar and M. Woodside, "Evaluating the scalability of distributed systems," IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 6, pp. 589–603, Jun. 2000, doi: 10.1109/71.862209.

[2] M. Mansouri-Samani and M. Sloman, "Monitoring distributed systems," IEEE Network, vol. 7, no. 6, pp. 20–30, Nov. 1993, doi: 10.1109/65.244791.

[3] M. Kleppmann, Designing Data-Intensive Applications: The big ideas behind reliable, scalable, and Maintainable systems. 2017. [Online]. Available: http://repo.darmajaya.ac.id/4191/

[4] A. S. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, 2nd edition. 2007. [Online]. Available: https://dl.acm.org/citation.cfm?id=1202502

[5] G. Coulouris and J. Dollimore, Distributed Systems: Concepts and design. 1988. [Online]. Available: http://cdk5.net/errata/Errata.pdf

[6] Pytransitions, "GitHub - pytransitions/transitions: A lightweight, object-oriented finite state machine implementation in Python with many extensions," *GitHub*. https://github.com/pytransitions/transitions

[7] G. Moruzzi, "Plotting with Matplotlib," in *Springer eBooks*, 2020, pp. 53–69. doi: 10.1007/978-3-030-45027-4_3.

[8] V. Cardellini, M. Colajanni, and P. S. Yu, "Geographic load balancing for scalable distributed Web systems," Proceedings 8th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Sys-tems, Nov. 2002, doi: 10.1109/mascot.2000.876425.

[9] S. Mohapatra et al., "A Cross-Layer Approach for Power-Performance Optimization in Distributed Mobile Systems," 19th IEEE International Parallel and Distributed Processing Symposium, Apr. 2005, doi: 10.1109/ipdps.2005.13.

[10] A. Ledmi, H. Bendjenna, and S. M. Hemam, "Fault Tolerance in Distributed Systems: A Survey," 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS), Oct. 2018, doi: 10.1109/pais.2018.8598484.

[11] Wallarm, "What is Content Delivery Network (CDN) How does it Work?," Wallarm, Feb. 20, 2023. https://www.wallarm.com/what/what-is-content-delivery-network.

Marian ILEANA graduated from the Faculty of Mathematics and Computer Science of the University of Bucharest with a bachelor's degree in Computer Science, in 2017. He obtained a research master's degree in Economic Informatics, within the Bucharest University of Economic Studies in 2020, and a professional master's degree in Databases and Web Technologies, within the University of Bucharest. He is a PhD student in Computer Science at the National University of Science and Technology Politehnica Bucharest, Pitești University Center, since 2020. His current interests are in the development of finite state machines, timed automata, distributed computing, and distributed web systems.