Group-5(CC Report)

Group Members
Tushar - (S20210010231)
Vivek Odedra - (S20210010245)
Abhishek Manithia - (S20210010004)
Lakshay Yadav - (S20210010129)

Implementation and Comparative Analysis of Static and Dynamic Load Balancing Algorithms

Introduction

The efficiency of load balancing algorithms is important in the rapid changing world of cloud computing, where the need for flawless connectivity and efficient resource use is consistent. We have explored the unique characteristics of these algorithms and the domains of static and dynamic approaches. The more we learn about networking and the increasing demands on bandwidth, throughput, and latency reduction, the more we realize that load balancing acts as a guardian, distributing incoming network traffic among several servers in an equal and fair manner. We examine the complexities of load balancing, highlighting the subtle differences between static and dynamic approaches.

I. Load Balancing in Cloud Computing

The modern trend of increased internet usage has increased server strain and called for a careful balancing act when it

comes to resource allocation. As a result, load balancing becomes an essential method for making sure that no server is overworked. It is the key to improve availability, responsiveness, and eventually the overall performance of the programme on the server. With cloud computing at the forefront of today's technological architecture, load balancing performance has enormous consequences. As we will see, an optimized load balancer improves throughput, minimizes response time, lowers transmission latency, and prevents bottlenecks in addition to improving end-to-end Quality of Service.

II. Static vs Dynamic Load Balancing

Static Load Balancing uses a fixed setup that spreads requests evenly to servers. It doesn't consider how busy servers are or how well they perform. This method is fair and straightforward to set up. It works well in places with stable server loads. But it's not great for handling changing traffic or server loads because it can't adapt easily. Static Load balancing algorithms, we have used in this implementation are Random Selection, Round Robin algorithm and weighted Round Robin algorithm.

In the Dynamic load balancing mechanism, the server with minimized load is targeted and the load from the over loaded server is migrated to the targeted server. It therefore requires the information about the current status of the net work servers and accordingly the load is distributed from maximum utilized server to underutilized servers and the load is balanced. DLB is more efficient than the SLB due to the dynamic distribution of load with respect to dynamic changes of load pattern. It is suitable for the applications in which the load keeps changing in the execution period and it provides better performance compared to static LB. The Dynamic LB can be performed in three methods:
Non Distributed, Distributed or semi-Distributed methods. In the Non-distributed method, only one centralized node exists which is responsible for receiving all incoming requests and allocates them to the connected servers. In the distributed

method all nodes share the incoming requests by mutually distributing among them. In semi-distributed method, the nodes are convened into clusters and each cluster is managed by one central node which is responsible for distributing the requests within the cluster nodes.

III. Static Load Balancing Algorithms

i. Random Selection Algorithm : Requests are assigned to servers randomly with equal probability. This means that each server in the pool has an equal chance of receiving a request, regardless of its current load or status. The Random Selection algorithm is easy to implement and requires minimal configuration. It's a straightforward approach that can work well for certain scenarios, especially when server resources are relatively evenly balanced. Random Selection doesn't take into account the current load or performance of the servers. It doesn't prioritize servers with lower traffic or better health.

ii. Round Robin Algorithm : Requests are distributed in a cyclic order to each server, ensuring an even distribution of traffic. Each server gets an equal chance to process requests, regardless of its current load or performance. It promotes fairness by preventing any single server from being overloaded, making it suitable for scenarios with similar server capabilities. It doesn't consider server load or health, so it might not be the most efficient choice for dynamic or unbalanced workloads.

iii. Weighted Round Robin Algorithm : Requests are distributed to servers based on predefined weights, allowing you to prioritize certain servers over others. It provides more control and flexibility by assigning higher weights to servers with greater capacity or performance. Even with different server capacities, the algorithm ensures a proportional distribution of traffic based on the assigned weights. Suitable for scenarios where some servers are more powerful or capable than others, making it an efficient load balancing choice. While more versatile, it requires configuring and

maintaining the weights for each server, which adds complexity compared to basic algorithms like Round Robin.

IV. Dynamic Load Balancing Algorithms

i. Least Response Time Load Balancing : Load balancers continuously monitor the response times of all available servers. When a new request arrives, the load balancer calculates the current response times of the servers. The server with the fastest response time at that moment is selected to handle the request. This technique minimizes response times, providing the best possible user experience. As server loads and response times change, the load balancer adapts its server selection decisions dynamically. Ideal for applications where low latency and fast response times are crucial, such as real-time communication or gaming platforms.

ii. Dynamic Weighted Random Selection : In DWRS, server weights are assigned dynamically based on real-time server loads. The least-loaded server receives more requests. The controller periodically monitors server loads and converts load values into weights using the formula: Weight of a server = 10 - (server load / 10). The formula ensures that server weights range from 1 to 10, preventing excessively high or low weights. Server weight represents the probability of selection. Servers with higher loads have a lower chance of being selected.

Objective

The goal of putting static and dynamic load balancing algorithms into practise and comparing them is to fully assess and comprehend how well they work in cloud computing environments. This entails evaluating metrics like latency, throughput, and response time; it also entails optimising resource usage and testing workload adaptability. Finding out whether these algorithms are scalable, performing a thorough

comparative analysis of particular static (Random Selection, Round Robin, Weighted Round Robin) and dynamic (Least Response Time, Dynamic Weighted Random Selection) algorithms, and assessing their effects on end-to-end Quality of Service are the main objectives of this research. It also evaluates the complexity of implementation and maintenance and looks for use cases that best suit each type of algorithm. The ultimate objective is to provide insightful information to help choose the best load-balancing approach.

Contributions

- Implementation of the Random Selection (RS) Algorithm by Lakshay Yadav: Lakshay led the charge by putting the Random Selection (RS) algorithm into practise, which is a quick and flexible load balancing method. The method was created in the.py file to choose a server at random for every request that comes in, offering simplicity and flexibility. The RS algorithm was contained by Docker and docker-compose, which made it easier for it to be integrated into the research environment. The RS algorithm is simple, but it introduces a certain unpredictable element.

- Implementation of the Round Robin (RR) Algorithm by Lakshay Yadav: Lakshay provided the fundamental approach to static load balancing using Round Robin Algorithm. The algorithm was carefully written in the.py file to guarantee a cyclic distribution of incoming requests among servers, preserving efficiency without sacrificing simplicity. The algorithm was encapsulated using Docker and docker-compose, resulting in a modularized and easily deployable environment for simulation and testing. When assessing static load balancing's baseline performance, the RR algorithm is used as a standard. He made sure the algorithm

would work flawlessly within the larger study framework by putting it through a rigorous testing and tuning process.

- Implementation of the Weighted Round Robin (WRR) Algorithm (Vivek Odedra): A more advanced static load balancing technique called Weighted Round Robin (WRR) was put into practise by Vivek. Static weights that are assigned to each server according to predetermined criteria like traffic load, response time, and server capacity were carefully considered and incorporated into the.py file during its meticulous creation. This algorithm was encapsulated using Docker and docker-compose, which allowed for its effective deployment in a variety of environments. The application of the WRR algorithm sheds light on the benefits of load distribution that takes server heterogeneity into account—a critical component in contemporary cloud environments.

- Implementation of Least Response Time (LRT) by Tushar: Tushar implemented the LRT dynamic algorithm, exploring the dynamic terrain. This state-of-the-art load balancing method is carefully integrated into a.py file and dynamically allocates requests to servers according to response times in real time. The LRT algorithm was encapsulated to ensure seamless flexibility and adaptability in dynamic cloud environments by utilising the power of Docker and docker-compose. The complex algorithm was fine-tuned to leverage real-time response time data, enabling the system to allocate incoming requests intelligently and maximise server utilisation. Extensive testing carried out by him demonstrated the LRT dynamic algorithm's ability to handle varying workloads with ease, emphasising its usefulness in practical situations.

- Implementing the Dynamic Weight Random Selection (DWRS) Algorithm (Abhishek Manithia): By applying the Dynamic Weight Random Selection (DWRS) technique,

Abhishek entered the dynamic domain. Weights are dynamically assigned to servers by this dynamic load balancing approach, which is programmed into a.py file, based on actual loads. The DWRS algorithm was wrapped through the use of Docker and docker-compose, guaranteeing flexibility and adaptation in dynamic cloud environments. The complex algorithm was fine-tuned to take advantage of real-time load data, which allowed the system to optimise server utilisation and intelligently distribute requests. Abhishek helped demonstrate the versatility of dynamic load balancing techniques in managing varying workloads through comprehensive testing.

- Overall Contributions and Synergy: Each team member worked together to create a comprehensive suite of load balancing algorithms that were executed in Python (.py) and packaged in Docker for easy deployment. This modular approach allowed for a comprehensive assessment of both static and dynamic load balancing strategies in addition to facilitating individual algorithmic contributions.

- Experimental Details : Docker and docker-compose, the core components of the experimental framework that provided consistency and portability across various systems, were used.Docker facilitates containerization by encapsulating dependencies and applications into small, lightweight containers. Multi-container Docker applications were defined using Docker-Compose, with a YAML file used to configure volumes, networks, and services. This made the experimental setup efficient and repeatable.Python was used to implement each load balancing algorithm (.py files). Python was a great option because of its readability and adaptability, which allowed for simple modification and clear expression of algorithms.An essential part of coordinating the communication between containers was Docker-Compose. It made the load balancing algorithms that were implemented within Docker containers easier to deploy and interact with.

Performance Metrics and Analysis

Latency:

Latency, a fundamental performance metric, measures the round-trip time for a request to travel from the client to the server and back. A lower latency indicates quicker response times and improved system responsiveness. The analysis focused on understanding how each load balancing algorithm influenced latency under different workloads. For instance, Round Robin's cyclic distribution may lead to consistent but potentially higher latencies, while Weighted Round Robin's consideration of server weights could optimize response times.

Throughput:

Evaluated as the speed at which requests were handled and revealed how effective the load balancing techniques were. A higher throughput indicates that more requests are handled and processed by the system efficiently. The analysis examined the effects of the load distribution strategy used by each algorithm on the total throughput. While Dynamic Weight Random Selection may dynamically adjust to changing workloads and potentially optimise throughput in dynamic scenarios, Random Selection's simplicity may help it process information more quickly under lighter workloads.

Packet Delivery Ratio (PDR) and Packet Loss Percentage:

These two metrics were essential for determining how well the algorithms handled network traffic in terms of dependability and efficiency. Robustness in packet delivery across the network was demonstrated by a high PDR and a low packet loss percentage. The examination examined the ways in which every load balancing algorithm reduced packet loss and preserved PDR in various scenarios. For example, the deterministic nature of static algorithms such as Round Robin

may help maintain consistency in PDR, whereas the flexibility of dynamic algorithms may have a positive effect on Packet Loss Percentage.

Experimental Setup:

The chosen tech stack, including Docker, Docker-Compose, and Mininet, provided a versatile and scalable environment for this performance analysis. Docker's containerization ensured consistent deployments, while Docker-Compose orchestrated interactions between containers.

Results/Outcome

The thorough analysis demonstrates that the Dynamic Weight Random Selection (DWR) algorithm outperforms static techniques, exhibiting superior outcomes in terms of packet delivery, speed, latency, and resource consumption, among other network performance metrics. Nevertheless, there is a discernible rise in packet loss, primarily as a result of a single central controller handling numerous client requests. Adding more controllers has the potential to enhance DWR's performance, which is important for effectively controlling network scalability. This observation suggests that although DWR is a great tool for dynamic load balancing, it can be improved in the current environment by implementing strategic changes, such as moving to a multi-controller network.

Limitations of the work and future scope

Although our study has provided insightful understanding of cloud computing load balancing algorithms, it is important to recognize certain limitations. The use of a single centralized

controller architecture is a significant drawback, particularly for dynamic algorithms such as Dynamic Weight Random Selection (DWR). While this method works to a certain degree, it creates a bottleneck problem, particularly when there is an increase in client requests. As a result, packet loss increased, highlighting a scalability issue and emphasizing the need for a more resilient network architecture. In addition, the majority of our study was conducted in a simulated setting, which means that real-world nuances and factors that were not fully included in our analysis may have been overlooked.

The future scope of our research entails a more thorough investigation of the multi controller challenge in load balancing in order to address these limitations. Performance bottlenecks caused by a single controller's inability to effectively handle an increasing number of requests give rise to the multi controller problem. The main focus of future work should be on developing and putting into place a multi-controller network infrastructure. The goal of this evolution is to make load balancing algorithms more efficient overall, more scalable, and less prone to bottlenecks.

Observation of the study

The study is well-planned, using different ways to balance work in a computer setup to answer research questions. It looks at both changing (Dynamic Weight Random Selection) and unchanging (Round Robin, Weighted Round Robin, Random Selection) methods. These methods are neatly packed using Docker and docker-compose, making them easy to study and use. The study's organized approach, covering many methods and ways to measure performance, ensures a detailed look at balancing work in cloud computing.

The results clearly show that the Dynamic Weight Random Selection algorithm works better than static methods. As expected, there are trade-offs, especially in dynamic situations where more data packets can be lost. While not surprising, this emphasizes the need to fix scalability issues linked to centralized controllers in dynamic environments.

This study is a big help in understanding how to balance tasks in cloud computing. It gives useful information about different methods, whether they stay the same or change. This helps people who make decisions about computer systems. Overall, the study teaches us more about how these methods work in different cloud situations.

The study tells us important things, but there are still questions about how these methods work in real cloud situations. The study used a pretend setup, and there might be things happening in real situations that the study didn't look at. This means we need more research to be sure. The study also talks about problems with the main systems it used, saying we need to think about the good things and what to be careful about when using lots of controllers in networks.