International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 11 Nov. 2022, pp: 1025-1030 www.ijaem.net ISSN: 2395-5252

Evaluation of Load Balancing Techniques on Web Servers

[1]O. L Abraham,[2] A, AWaheed &[3]M. O Ayemowa
[1]*ITS Unit, Gateway Polytechnic, Saapade, Ogun State, Nigeria*
[2] *Department of Computer Science, Lead City University, Oyo State, Nigeria*
[3]*Department of Computer Science, Gateway Polytechnic, Saapade, Ogun State, Nigeria*

Date of Submission: 21-11-2022 Date of Acceptance: 30-11-2022

ABSTRACT
Traditional networking architectures have many significant limitations that must be overcome to meet modern IT requirements. To overcome these limitations; Software Defined Networking (SDN) is taking place as the new networking approach. One of the major issues of traditional networks is that they use static switches that cause poor utilization of the network resources. Another issue is the packet loss and delay in case of switch breakdown. This project proposes an implementation of a dynamic load balancing algorithm for SDN based data center network to overcome these issues. In a data center environment, the load balancer is an integral part of the networking ecosystem. The primary function of a load balancer is to distribute traffic among a cluster of servers such that a single server does not become over-utilized and ensure that critical services keep running. Software Defined Networking (SDN) offers a cost-effective and flexible approach in implementing a load balancer. Moving away from traditional hardware-based networking approach, this project implements load balancing with the help of software. The SDN approach reduces the cost, offers flexibility in configuration, reduces time to deploy, provides automation and facilitates building a network without requiring the knowledge of any vendor-specific software/hardware. A test-bed has been implemented using Mininet software to emulate the network, and OpenDaylight platform (ODL) as SDN controller. Python programming language is used to define a fat-tree network topology and to write the load balancing algorithm program. Finally, iPerf is used to test network performance. The network was tested before and after running the load balancing algorithm. The testing focused on some of Quality of Service (QoS) parameters such as throughput, latency, bandwidth, delay, jitter, and packet loss between two servers in the fat-tree network. The algorithm increased throughput with at least 35.8%, and also increased the network utilization within the system.

Keywords: Application Programmable Interface (API), Load Balancer, Mininet, OpenFlow, Python, Round Robin, Software Defined Networking (SDN), Data Plane, Control Plane.

I. INTRODUCTION

Due to the ongoing growth of web applications, there has been an increase in demand for online services and information. Given that all of these services are web-based and run on servers, we may presume that consumers can accurately and quickly find the majority of the information they need for their everyday lives online. Web servers therefore take in client requests, process them, and deliver the answers. Additionally, the effectiveness and efficiency of web servers impact their ability to attract businesses and web developers with features like precise and quick client responses. On the other hand, the demand placed on web servers has an immediate impact on their performance. Controlling the strain on the web servers is therefore essential. Therefore, it is crucial for researchers to design and propose an effective system that could handle a large amount of demand. The massive and ongoing increase in client requests for the services offered by the servers is another major contributor to the overload. Accordingly, server overcrowding impairs a company's ability to attract customers, lowers revenue, and damages its reputation[1]. Establishing a cluster server is yet another similar approach that is frequently utilized in businesses to improve performance. However, using a number of servers to increase resource availability and persistency necessitates a system to evenly balance and disperse the incoming demand among servers. This approach also aids in reducing the amount of traffic that is routed toward a single server inside

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 11 Nov. 2022, pp: 1025-1030 www.ijaem.net ISSN: 2395-5252

the server cluster. In modern web technology, load balancing is a crucial component in setting up and offering a reliable service. In the meantime, load balancing approaches boost the functionality of cluster servers by enhancing traffic management, reducing response times, increasing throughput, and minimizing the burdensome strain on cluster servers[2].

A load balancer, in general, functions similarly to a "traffic controller" for all server and routes traffics to an accessible one that can do so effectively. This guarantees that requests are met quickly and that no server is overworked to the point where performance is compromised. The load balancer helps an organization choose which server can effectively handle the requests in an effort to satisfy the application demands. Better user experience is produced as a result. The load balancer manages the flow of request between the servers and the clients by assisting servers in moving data effectively. Additionally, it evaluates the server's ability to handle requests, and if needed, the balancer eliminates the unfit servers or machine till their operations are fully restored. Traffic are sent to the active servers or machines when a failure is experienced on some servers and as such tagged as been offline, and when a new server is launched, requests are immediately forwarded to it[3]. To determine which load balancing strategies work best in each situation, a number of them will be tested and their impact on the web server analyzed.

Statement of the Problem

Web server application performance, web traffic, and congestion management become issues due to the rise in web users as demands on internet services and information grow steadily as a result of continual expansion of web applications. For a network to transmit data with high throughput and minimal delay, network resources must be managed dynamically. Static switches are used in conventional networks. These networks have the drawback that every flow/request follows a single, preset route through the network[5]. Packets typically drop when a switch fails until a replacement routing is chosen. Poor network resource use, where alternate links to the destination are inactive, is another problem.

Aim and Objectives

This study is aimed at examining different load balancing techniques that may be used in SDN-based data center networks in order to investigate the many ways in which a higher level of performance can be achieved.

The aim is to be achieved by the following objectives: to

i. identify Network Request Distribution load-balancing approaches.
ii. implement the various strategies for load balancing across the network.
iii. assess the network impact of each load-balancing approach.

Significance of the Study

The primary contribution of this study is it to improve the effectiveness, performance, and dependability of web servers and so contributes to the maximization of customer satisfaction.

Scope of the Project

In order to deliver and maintain high throughput and low latency on the network, this study is meant to distinguish, experiment, and assess the effect of load balancing algorithms utilizing key performance indicators based on multiple parameters.

II. METHODOLOGY

Research Approach

This section defines the procedure implemented to realize a dynamic load balancing method, and as well offers the constituents and software implements utilized in this study to establish the testbed.

System Design

In order to exploit the resource competence of each connection in a network, the load balancing algorithm's task is to balance traffic from incoming and outgoing network flows. Maintaining awareness of the network's current condition is important to accomplish this goal. The phases of the load balancing algorithm are illustrated in Figure 3.1 below.

The algorithm's initial phase is to gather operational data on the network structure and the gadgets therein which includes IP addresses, MAC addresses, ports, connections, etc. The succeeding phase is to locate path data based on the load-balancing algorithm. Here, the search needs to be limited to a lesser area of the Fat-Tree network interlinks in order to identify the shortest routes between the basis and target hosts. Next, calculate the overall connection cost for each of these routes between the root and target hosts. After calculating the diffusion overheads of the links, the flows are formed based on the least diffusion rate of the links at the time. The best route is chosen based on the cost, and inert movements are then advanced into respective switch in the chosen fitting route. As a

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 11 Nov. 2022, pp: 1025-1030 www.ijaem.net ISSN: 2395-5252

result, each switch along the chosen path will have the required flow entries to execute the exchange amidst the two termination spots. To end with, the application keeps updating this data every minute, causing it to be dynamic.

Implementation Overview

A test-bed has been set up in this study in Linux, utilizing Mininet software to simulate the network, the open-source OpenDaylight platform (ODL) as the SDN controller, Python language to design the fat-tree topology and create the load balancing set of rules program, and iPerf to measure network functioning. The steps in design are shown in the diagram below.

Requirement Specification

Mininet

Mininet is a network emulator that enables rapid prototyping of huge networks on a single host computer. On a single Linux kernel, it manages a variety of end hosts, switches, routers, and links. By executing the same kernel, system, and user code on a single system, it leverages lightweight virtualization to make the system appear to be a full network.

Mininet main advantages:

1. Mininet is an open source project.
2. Custom topologies can be created.
3. Mininet runs real programs.
4. Packet forwarding can be customized.

In contrast to simulations, Mininet executes actual, unmodified code, including application code, OS kernel code, and control plane code (both OpenFlow controller code and Open vSwitch code), and it connects to real networks with ease.

It is necessary to have a controller machine running on a system-working framework, such as ODL or POX, in order to create the SDN load balancer. With the programming interface in OpenFlow, the crucial components of the forwarder have a separate control arrangement. For SDN, a comprehensive physical foundation that connects every component via a secure channel is required, and this is costly.

This project makes use of a specific test system to handle this problem. Mininet is a tool that simulates Software Defined Networks, which allow for quick prototyping of a sizable virtual foundation system using a desktop PC. In light of programming, it facilitates the use of virtual models of flexible systems. For instance, OpenFlow quickly associates and updates models for Software Defined Networks by using a rudimentary virtualization operating system using these primitives[1]. A few features of the Mininet are the following.

1. It permits that various scientists autonomously test the same system topology.
2. It permits the testing of a complex topology without the need of a physical system.
3. It incorporates apparatuses to troubleshoot and run tests over the system.
4. It underpins various topologies and incorporates an essential arrangement of them.
5. It gives straightforward Python APIs for making and testing systems.
6. It gives a straightforward and modest path for testing systems for the improvement of OpenFlow.

Mininet is more compatible than simulators since it uses the application's original code, as opposed to simulators, which use test code. This facilitates testing and running prior to implementation in production. The OpenFlow procedure is used in the programming of Mininet switches, which makes it effortless to examine and amend the code as needed. By issuing a single command, it constructs a realistic virtual network that functions on an actual kernel, switch, and application code on any virtual machine (VM), cloud, or native platform[2]. According to the OpenFlow website, Mininet is a useful tool for developing, sharing, and experimenting with OpenFlow and Software-Defined Networking systems.

OpenDay Light Controller

The OpenDaylight controller (ODL) is a multi-protocol controller architecture designed for SDN dispositions on contemporary varied multi-vendor links. It is highly accessible, flexible, extendable, scalable, and available in several languages. ODL is a platform for model-driven facility construct that enables users to quickly create applications that run on a variety of south-bound protocols and hardware.

Additionally, it has internal plugins that expand the network's capabilities. For instance, it features dynamic plugins that make it possible to acquire both network topology and statistics.

iPerf

iPerf is a well-known network assessing device for evaluating the effectiveness of a network link and the bandwidth operation of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The user is capable of completing a quantity of assessments that give perception into the network's bandwidth disposal, latency, jitter, and data loss by changing various timing, buffer,

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 11 Nov. 2022, pp: 1025-1030 www.ijaem.net ISSN: 2395-5252

and protocol parameters (TCP, UDP, SCTP with IPv4 and IPv6)[3]. iPerf is open source software that works on a variety of operating systems, including Linux, UNIX, and Windows.

Programming Language used: Python

Python was used in this study to express the Fat-tree topology in Mininet and to create the application for the load balancing technique. Python is an object-oriented, interpreted language that may be used for a variety of tasks. It has a flexible dynamic type system, strong high-level data structures, and a clear, understandable syntax[4]. Python is a flexible language that can be applied interactively, in standalone scripts, for complex systems, or as an add-on to already existing applications. Windows, Macintosh, and Linux computers can all run the language.

Python can be easily extended through C or C++ modules, and it is also capable of been used as a library in other programs. Additional system-specific extensions are available. There is also a sizable library of common Python modules. Python instructs are substantially littler and therefore are quicker to devise than C applications. Python code is simpler to deliver, script, and support than Perl code. Python is more appropriate for copious or more complex projects than TCL is.

Research Methods

Round-Robin and weighted round-robin are the two most popular load balancing algorithms. These algorithms, however, are static, which means that they are not dynamically modified in response to incoming requests. Regardless of the scope of the work or the administering influence of the resources, the RR algorithm distributes the incoming requests in a circular pattern to the available resources. The WRR algorithm takes the processing capability of resources into consideration, and the resource with better processing power is given a larger quantity of incoming jobs. The dynamic WRR algorithm is also used for comparison with these algorithms. This method uses the waiting time to determine the best server for each new request that is made. Using data on the server's processing capacity, task length, task priority, and processing time needed for servers to complete jobs with equal or higher priorities, this algorithm aids in the selection of the most appropriate server. Information regarding a task, including its duration and priority, is recorded once it enters the system. The weight of each server is then determined, as was previously described. The incoming job is then sent with additional weight to the server. The weight of a server essentially relates to the length of time needed for that particular server to finish the tasks with equal or higher priorities in its queue. If the completion time is quicker, the weight of the server increases, and vice versa. Task migration is carried out when a task's runtime varies significantly from expected values. By determining which server has the least number of jobs with the same or higher priority, the high priority task from the overloaded server's queue is sent to the under-loaded or ideal server[5]. This speeds up the completion of high-priority tasks, and the migration helps to improve the efficient use of resources.

Waiting Time Calculation

According to the waiting time to complete the subsequent incoming job, each VM is given a weight in the proposed method. A VM's weight and waiting time have an inverse relationship; a VM has a high weight when the waiting time is short and vice versa. The proposed method bases the calculation of waiting time on the importance of an incoming job. Each task has a priority value between 1 and 10, and one with a lower integer value is regarded as having a high priority. The following formula is used to determine each VM's waiting time, WT_{vm}:

$$WT_{vm} = \sum_{p=1}^{i} T_p$$

Equation (3.1)

where T is a task's execution time and p is the task priority. When a VM gets an incoming task with priority I the waiting time is determined using the equation above. In this experiment, the cloudlet with the lowest priority is regarded as having a high priority, while the cloudlet with the highest priority is regarded as having a low priority[6]. For instance, if a job with priority 5 is received, the waiting times for tasks with priorities 1, 2, 3, 4, and 5 are determined. It determines the sum of all tasks' execution times whose priority is equal to or higher than the entering tasks.

The execution time of a task is calculated as follows:

$$T = \frac{S}{V_m}$$

Equation (3.2)

where S represents the cloudlet size in MI (Million Instructions) and V_m represents the processing capacity of a VM in MIPS (Million Instructions Per Second).

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 11 Nov. 2022, pp: 1025-1030 www.ijaem.net ISSN: 2395-5252

Resource Load Calculation

To conduct task migration between virtual machines, the load of all VMs must be calculated, and it must be split into three categories: overloaded, under-loaded, or balanced. Calculating the overall processing capacity of all VMs (C), threshold variance for each VM, and threshold for each VM are all necessary steps in doing this categorization[7]. The threshold range is computed using two variances, V_{min} and V_{max}. These numbers represent a machine's percentage usage; in this experiment, we utilized a minimum value of 0.7 and a maximum value of 0.9.Total capacity of all virtual machines is calculated as follows:

$$C = \sum_{i=1}^{k} c_i$$

Equation (3.3)

where k represents the number of available virtual machines and c represents the processing capacity of a VM. Threshold for each VM is calculated as follows:

$$T_{min} = \frac{c}{C} * V_{min} * L * n$$

Equation (3.4)

whereV_{min} represents the minimum variance, L represents the total capacity of a node, and n represents the total number of virtual machines. T_{min} represents the minimum threshold value of a VM.

$$T_{max} = \frac{c}{C} * V_{max} * L * n$$

Equation (3.5)

where V_{max} represents the minimum variance and T_{max} represents the maximum threshold value of a VM.

The VMs are classified by using the following equations:

$$WT_{vm} < T_{min},\ Underloaded$$

$$WT_{vm} > T_{max},\ Overloaded$$

$$WT_{vm} = [T_{min}, T_{max}],\ Balanced$$

Any virtual machine (VM) that exceeds the threshold level is regarded as being overloaded. A VM with a load that is below the threshold level receives the task migration from an overloaded VM. The under-loaded VM is prepared to accept the task until it hits the threshold level. No migration is conducted if there are no under-loaded VMs. According to the proposed method, the VM that has the least number of tasks with equal or higher priorities is chosen as the under-loaded VM.

Implementation of Round-Robin Algorithm

The round-robin algorithm's step-by-step procedure is given in Algorithm 1. Among static algorithms, this is one of the most frequently applied. The incoming requests are cycled through the servers according to this algorithm. The first request is assigned to any random server, and the following requests are processed in cyclic order. For perfect static conditions, cloud service providers frequently utilize this algorithm.

III. CONCLUSIONS

In laboratory conditions, the test scenario are performed in which the testbed environment was composed of a minimum number of devices needed to realize the project objectives. This approach produces a particular limit in terms of results - they could be different if it the testing were performed in a realistic or cloud environment. As software-defined network is developed to manage large networks like WAN, cloud computing technologies like data center, big data etc. Growth in today's network leads to large amount of traffic on the link due to which performance and efficiency of the network degrades. The algorithm is not analyzed over such a big network. One can check the performance on these networks and update the algorithm according to the experimental results.

Contribution to Knowledge

Software-defined networking (SDN) load balancing removes the protocols at the hardware level to allow for improved network management and diagnosis. SDN controller load balancing makes data path control decisions without having to rely on algorithms defined by traditional network equipment. An SDN-based load balancer saves running time by having control over an entire network of application and web servers. Load balancing in SDN leads to discovery of the best pathway and server for the fastest delivery of requests.

REFRENCES

[1]. Afolabi, I.; Prados-Garzon, J.; Bagaa, M.; Taleb, T.; Ameigeiras, P. Dynamic

ResourceProvisioning of a Scalable E2E Network Slicing Orchestration System. IEEE Trans. Mob. Comput. 2020, 19, 2594–2608.

[2]. Boutaba, R.; Salahuddin, M.; Limam, N.; Ayoubi, S.; Shahriar, N.; Estrada-Solano, F.; Caicedo

[3]. Rendon, O. A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities. J. Internet Serv. Appl. 2018, 9, 1–99.

[4]. Casas-Velasco, D.M.; Rendon, O.M.C.; da Fonseca, N.L.S. DRSIR: A Deep ReinforcementLearning Approach for Routing in Software-Defined Networking. IEEE Trans. Netw. Serv. Manag. 2021, 1–14.

[5]. Dixit A., Fang H., Mukherjee S., Lakshman T.V., Kompella R.R., "ElastiCon: an elasticDistributed sdn controller," Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, ACM, 2014, pp. 17-28.

[6]. Fu X., Yu F.R., Wang J, Qi Q. and Liao J., "Service Function Chain Embedding for NFV-Enabled

[7]. IoT Based on Deep Reinforcement Learning," IEEE Communications Magazine, 2019, vol. 57, no. 1, pp. 102-108.

[8]. Jamali S., Badirzadeh A., Siapoush M.S., "On the use of the genetic programming for balanced

[9]. Load distribution in software-defined networks," Digital Communications and Networks, 2019, vol. 5, no. 4, pp. 288–296.

[10]. Khan, A.; Abolhasan, M.; Ni, W.; Lipman, J.; Jamalipour, A. An End-to-End (E2E) Network

[11]. Slicing Framework for 5G Vehicular Ad-Hoc Networks. IEEE Trans. Veh. Technol. 2021, 70, 7103–7112.

[12]. Wireless networks," International Journal of Distributed Sensor Networks, 2015, pp. 1-8.

[13]. Yan, M.; Feng, G.; Zhou, J.; Sun, Y.; Liang, Y.C. Intelligent Resource Scheduling for 5G

[14]. RadioAccess Network Slicing. IEEE Trans. Veh. Technol. 2019, 68, 7691–7703.

[15]. Yang C.-T., Chen S.-T., Liu J.-C., Su Y.-W., Puthal D., and Ranjan R., "A predictive loadBalancing technique for software defined networked cloud services," Computing, 2019, vol. 101, no. 3, pp. 211–235.

[16]. Zhang P., Wang C., Qin Z. and Cao H., "A multi-domain virtual network embedding algorithm

[17]. Based on multi-objective optimization for Internet of Drones architecture in Industry 4.0," SoftwPractExper, 2020, pp. 1-19.