A Proposed Dynamic Hybrid-Based Load Balancing Algorithm to Improve Resources Utilization in SDN Environment

Haeeder Munther Noman[1(✉)] and Mahdi Nsaif Jasim[2]

[1] Software Department, College of Information Technology, University of Babylon, Babylon, Iraq
[2] University of Information Technology and Communications, Baghdad, Iraq

Abstract. Several load balancing schemes are proposed to tackle the webserver overloading problems. Static load balancing is appropriate for systems with low load variations where the traffic fairly distributes among servers and Prior information about system resources is required. Dynamic load balancing monitors the system's current state to perform load controlling actions and respond to the current system state while making load transferring decisions. Consequently, processes may dynamically switch from an overused machine to underuse during real-time. However, to utilize the resources more efficiently this paper proposes a new hybrid-based load balancing algorithm that relies on inheriting the distinctive characteristics, overcoming the existing limitations, and combining the desired features of static as well as dynamic load balancing. The experimental analysis witnessed the utilization of the OpenLoad benchmarking tool that generate a concurrent users from 0 up to 350 to provide a near real-time performance measurement of the application under test and to evaluate the performance of load balancing algorithms. Results reveal that proposed hybrid-based load balancing algorithm interestingly enhances server transactions per second up to 10.41%, average server response time up to 24.61%, and server CPU capacity up to 9.55% when compared with other load balancing algorithms like static weighted round-robin (WRR) and dynamic least connection-based (LCB). Accordingly, this research recommends deploying the proposed load balancing technique in SDN-Based Platform data center networks (DCN's).

Keywords: Weighted Round-Robin (WRR) · Least Connection-Based (LCB) · Software Defined Network (SDN) · Advanced Server Monitoring module (ASM) · Hybrid-based (HB)

1 Introduction

SDN is a modern architecture that separates control and forwarding functions in the network. The separation is a departure from the conventional architecture where complex tasks are abstracted from the repeated forwarding tasks. A complex process is automated and handled separately in a centralized SDN controller or Network Operating System. The controller uses the OpenFlow protocol to connect with the real physical or virtual switch, and the data path uses the flow entries inserted by the

A. M. Al-Bakry et al. (Eds.): NTICT 2021, CCIS 1511, pp. 147–162, 2021.
https://doi.org/10.1007/978-3-030-93417-0_10

controller in the flow table for routing data. Three methods exist to separate the control plane and the data plane: fully distributed, logically centralized, and strictly centralized. In the Fully distributed method, switching devices provide one essential feature for forwarding packets, but unfortunately, without a control power, that may lead to a failure point. The logically centralized method includes a remarkable feature signified in the devices with a partial functionality embedded inside. The strictly centralized method adopts the conventional way of making all machines within all planes route packets through the network. However, the majority of clients, organizations, and businesses rely on the internet for their daily activities to deliver services online [1]. The hardware performance improvements are no longer enough to cope with the rising volume of customer requests while preserving the desired quality of service [2]. Accordingly, standard practice is to utilize servers to process the client requests [3]. Suppose it is difficult to allocate the incoming requests among servers evenly. In that case, some servers could be overloaded while others remain idle, which leads to low server utilization and poor service quality. Moreover, due to the ever-increasing request loads of popular websites, it's challenging to adopt a single robust server or mirrored servers [4]. Therefore, a load balancer (also known as a dispatcher) is employed to distribute client requests to a particular server in the back end and removes any possible single point of failure to maximize system reliability [5]. The load balancing strategies invoked by the load balancer are viable to redirect requests among the server members and become necessary when multiple servers operate simultaneously. Moreover, load balancing techniques reduce the overall response time and optimize the total throughput when all activities are transparent to the user [6]. The contribution is to develop and implement a hybrid-based load balancing algorithm that eliminates the issues and inherits the merits of static and dynamic load balancing schemes.

The organization of this paper is as follows: Sect. 2 introduces a literature review related to the development of load balancing algorithms running in SDN, the main features of static and dynamic load balancing algorithms, and the limitations for both types of algorithms. Section 3 outlines the proposed hybrid-based (HB) load balancing algorithm. Section 4 discusses the experimental set up. Section 5 is committed to discuss the flow-sequence diagram of the proposed algorithm. Section 6 describes the experimental results, evaluation, and finally Sects. 7 and 8 deal with the conclusion and future work.

2 Related Work

Many approaches have been hypothesized to address the load balancing issue. Traditional load balancing algorithms suffered from being non-programmable and vendor locked, which turned them to be rendered inoperable. Moreover, network administrators could not build customized applications [7]. Preliminary work focused on a single load balancing parameter was later insufficient to pick up the best server for processing user requests [8]. A systematic study to investigate a dynamic flow entry saving multipath (DFSM) system for inter-data center WAN transmission was carried out [9]. The DFSM system employed the concept of source-destination multipath forwarding mechanism with latency awareness flow-based traffic splitting to preserve

flow entries and get a high level of achieved performance. However, the DFSM saved from 15% up to 30% of system flow entries and reduced from 10% up to 48% of average latency. The computation of the shortest route among hosts and performing the load calculations associated with each link was proposed by [8]. If congestion in a route occurs, it substitutes the old route with the alternative best route having the lowest traffic flow [10]. The utilization of the dispatcher architecture represented the major advance regarding load balancing to employ the back-end server, later investigated to be inappropriate for distributing client requests evenly to different servers [11]. The hybrid–based load balancing algorithm's proposition relying on each of the Least Load and Round-Robin load balancing solutions running in an SDN environment was suggested by [12]. Accordingly, the time has come to develop different hybrid load balancing algorithm that merges two or more load balancing schemes, static or dynamic. The aim is to produce a new algorithm that incorporates the benefits, inherits the distinctive characteristics, and overcomes both strategies' limitations and inconveniences.

2.1 Static Load Balancing Algorithms

An equal division of traffic is performed inside the servers in a static algorithm, which is appropriate for platforms where the load changes at a low rate. Prior information about system resources is required to ensure that load shifting decisions do not rely on the system's current state. Moreover, initial tasks are assigned to the individual processors for execution by the master processor [13]. Accordingly, the performance of the workload is calculated from the beginning through the master processor. Slave processors calculate the expected work and provide the results for the master processor. Static load balancing is appropriate for systems with low load variations where the traffic fairly distributes among servers. The processor's efficiency is computed at the start of execution, and the decision to transfer loads avoids relying on the system processors' current status. Tasks are assigned to virtual machines and processors after their generation because they could not be transferred to any other device for load balancing during the execution. Although a static scheme is less overhead, suitable for a homogeneous environment, and easy to implement, several drawbacks are reported: like the non-versatile nature, the inability to accept dynamic changes, pure dependency on statically obtained data, and the non-pre-emptive behavior. WRR is a good instance of this type of algorithm regarded as the Round-Robin's improved version. Static weight is assigned to each server in the network pool based on servers' actual capacities and specifications regarding CPU, RAM, etc. [14].

2.2 Dynamic Load Balancing Algorithms

Due to the dynamic distribution of the pre-programmed load balancing patterns, the dynamic load balancing scheme is more efficient than static [15]. Two modes are available to implement the dynamic load balancing scheme: the non-distributed and distributed. In the nondistributed approach, a (centralized) node receives and distributes all requests to the servers, whereas the distributed mode shares the nodes within the distribution of the requests [16]. Features of dynamic load balancing schemes include

monitoring the system's current state to perform load controlling actions and responding to the current system state while making load transferring decisions. Consequently, processes may dynamically switch from an overused machine to an underused device in real-time. Dynamic load balancing schemes come with several advantages, like the absence of a single web server overloading problem. The current system load is kept under consideration to select the following data center. Drawbacks include communication overheads, the growing number of procedures, and the higher run time complexity. However, LCB is one of the traditional dynamic load balancing schemes in circulation. The load distribution is decided based on the current number of connections for every node. A load balancer keeps a log of a node's number of connections. However, the node with the fewest number of connections is firstly selected [17] such that when a new operation or order occurs, load increases, whereas when the operation terminates, load decreases. This technique is a good choice if the request load has a high degree of variance and is ideal for an individual server pool with a similar capacity for each member node.

2.3 Limitations and Drawbacks

Load balancing schemes perform well but unfortunately hold certain limitations. WRR is simple and runs fast but lacks accuracy during load balancing of varying load size or request complexity. Current server load is not considered during request distribution [18]. In the LCB algorithm, the server load is taken into account before load delivery, which may have the load distributed sparsely, leaving some of the servers idle [19].

3 Proposed Hybrid-Based (HB) Algorithm

The proposed load balancing algorithm consists of two functional models, the Advanced Server Monitoring Module (ASMM) and the Hybrid Load Balancing Module (HLBM). The modules usually mount on the top of the POX controller. The ASMM module tracks all of the information related to the server's status, modify some of the corresponding parameters, and forwards it periodically to the load balancing module. The Hybrid load balancing Module (HLBM) is considered the main operating module responsible for handling and controlling the load balancing decisions. Each server connects to the OpenFlow switch with a static IP address, and each server pool acquires a virtual IP and MAC address. Without recognizing the server's physical address, users send their requests to the OpenFlow switch's virtual MAC address, which sends a Packet_in message to the controller running the modules mentioned above. If there is no matching regarding flow entries, the controller inserts the corresponding flow entry into the OpenFlow switch via the southbound OpenFlow protocol.

3.1 Advanced Server Monitoring Module (ASMM)

The ASMM is developed to collect the OpenFlow switch ports, flow entries, and flow table counters statistics. The OpenFlow switch provides three types of counters

statistics: the first type is the statistics per flow table that consist of matched packets, the number of looked-up packets, and the number of active flow entries. The second type is the statistics per flow entry in a flow table that consists of the received bytes, received packets, and duration. The last type is the statistics per port consisting of received packets, transmitted packets, received bytes, transmitted bytes, receive drops, transmit drops, receive errors, transmit errors, and collisions. The module executes a runnable class, which involve the running data collection function according to the time interval set to 5 s by passing out two parameters: the first parameter is the OpenFlow switch port number (OpenFlowPortNo.) used to retrieve the counters statistics related to OpenFlow switch ports such as the received packets, transmitted packets, received bytes, transmitted bytes, received drops, transmitted drops, and the collisions. The second parameter is the Flow entry number (FlowEntryNo.) is used to retrieve the counters statistics related to flow entries such as received bytes, received counters, time and OpenFlow table statistics such as active entries, packet lookups, and packet matches. The ASMM is incorporated into the HLB by using a thread that handles the request and response statistics as explained in Algorithm 1.

Algorithm1: Advanced Server Monitoring Module(ASMM)

Input: *Network topology, O.FSwitch PortNo, O.FSwitch FlowEntryNo, t //adopted to collect OpenFlow Switch traffic information requested by POX*
Output: *Collect OpenFlow switch counters statistics periodically every 5 seconds*

SwitchStatistics: SwitchStatisticsType

While *TRUE do*
Read *SwitchStatisticsM*
Count SwitchStatisticsM.Port for the Server Connected to the OpenFlow Switch by OfPortNo.
Count SwitchStatisticsM.FlowEntry for the Server Connected to the OpenFlow Switch by OfPortNo
Update *SwitchStatisticsM.FlowTable, SwitchStatisticsM.FlowEntry, SwitchStatisticsM.Port*
Sleep (t units of time)
End while

3.2 Hybrid Load Balancing Module (HLBM)

This section aims to explain the approach followed by this module to compute and distribute the incoming traffic to the server. The load is determined based on least connections as well as static weight for each server. The requests usually demand pages of type text with small sizes as the HTTP web server itself could process them and do not require many resources. Thus, consuming less bandwidth and avoiding server-side

scripting (e.g., JSP, ASP, and PHP). In contrast, requests demanding pages of big sizes like images that require additional resources cause extra time and consume high bandwidth. The HLBM implements the load of each server and selects the optimal server to handle the incoming request, as shown in the processes listed below:

1. **Computing the Load of Servers**
   The module adopts a single pool (P) assigned with a single service type (HTTP). The pool includes server members acquiring varying load, L = (L(S l), L (S2)….L (SN)) and R is the set of requests that need to be scheduled: R = (R1, R2….RN). The load of the pool is calculated according to Eq. 1:

$$L(Si)Pool = \sum_{i=0}^{n} L(Si) \tag{1}$$

   The load of the server members is calculated based on the algorithmic rule that consists of two stages. The first stage continuously monitors server members and computes the active number of connections of each server, and selects the server with the least number of real-time active connections according to Eq. 2:

$$Sm = Min[Conn \sum_{i=1}^{n} Si] \tag{2}$$

   Where Sm: refers to the server member having the least number of connections. However, the second stage carries out the mathematical multiplication of least connections server member resulting from the first condition with static weight for each server in the pool. The server producing the highest product value, is selected and the request is forwarded to that server depending on Eq. 3:

$$Soptimal = Max[\sum_{i=1}^{n} (Sm) * W(Si)] \tag{3}$$

   Where Soptimal: refers to the final server member responsible for handling the incoming request among a pool of n servers. n, i = {1, 2, 3,}, where n is the total number of servers in the pool. W (Sw): refer to the assigned static weight for each server member.
2. **Selecting the Optimal Server**
   As the POX controller receives requests from clients, then relying on the calculations of the mathematical rule mentioned in the above step, the server with the least load is selected to process the client request. However, each time a load of server members varies, then related information is updated as explained in Algorithm 2.

Algorithm2: Hybrid–Load Balancing Module (HLBM)

Input: *ServersIDArray = {S1, S2… Sn}, ServersWeightArray = {ServerWeight (1), ServerWeight (2),…..ServerWeight(n)*
Output: *The Best server to Handle the Incoming Request Among a Pool of n servers*
Type1: Type
ServerNo.: integer
ServerNo.of.Connections: integer
ServerWeight: integer
OptimalServer: integer
SelectServNo: integer
No.ofServers: integer
end Type1
Servers: Array[No.of Servers] of Type1
OptimalServers: Array[No.ofServers]
Type2: *SwitchStatisticsType*
Record Per FlowTable
ActiveEntries: Integer
PacketLookups: Integer
PacketMatches: Integer
End Record
Record PerFlowEntry
ReceivedPackets: Integer
ReceivedBytes: Integer
End Record
Record PerPort
ReceivedBytes: Integer
TransmitBytes: Integer
ReceivedPackets: Integer
TransmitPackets: Integer
ReceiveDrops: Integer
TransmitDrops: Integer
Collisions: Integer
End Record
end Type2
SwitchStatistics: SwitchStatisticsType
Call Advanced Server MonitoringModule(SwitchStatistics)
for *i=1 to No.of Servers*
Call FindMinServerConn. (No. of Servers, Servers, and Var SelectedMinServerConn.
MinConnServer = SelectedMinServerConn
End for
for *i=1 to No.of Servers*
OptimalServer[i]=Server[i].weight MinConnServer*
end for
SelectedServNo=findMax(OptimalServers)
forward the request to (SelectedServNo)
Algorithm FindMinServerConn.(N, Servers, Var SelectedMinServConn.)
N= Number of Servers
ServerMinConnections:int=0
for *m = 0 to n-1*
If *Servers[m].Conn. <ServerMinCon*n.
Then *ServerMinConn. = Servers[m].conn.*
end if
end for
Return *ServerMinConn*
End.

4 Experimental Setup

The experimental work is carried out using a POX controller. The POX is written with Python, designed and implemented as a friendlier alternative by several SDN developers and Engineers. POX performs well as compared to NOX. The POX is a more straightforward development environment to deal with that offers a web-based GUI with a fairly well-written API and documentation. Moreover, the POX facilitates the addition and removal of restored units, and enhances the experiments flexibility and performance. The POX includes a list of IP addresses assigned to each server statically. The Mininet emulator carries creates the virtual network topology that consists of three hosts acting as Apache HTTP web servers holding the same configuration to provide the same web services to the clients. Nine hosts acting as clients, which attempt to access the Apache HTTP web servers by using a virtual IP address. The OpenLoad benchmarking tool plays an important role in evaluating the performance of the proposed load balancing scheme. OpenLoad is easy to use and provides a near real-time performance measurements of the application under test. This is particularly useful during performing optimization due to the fact that OpenLoad reflects the impact of modifications almost immediately. This tool examines the impact of the gradual growth of load represented by concurrent users from 0 up to 350 on the server's quality of service (QoS) metrics defined by the server's average response time, server's transactions per second, and server's CPU capacity to handle requests. Port 6633 is the default connection port for establishing communications between the OpenFlow switch and POX controller.

5 Flow Sequence Diagram

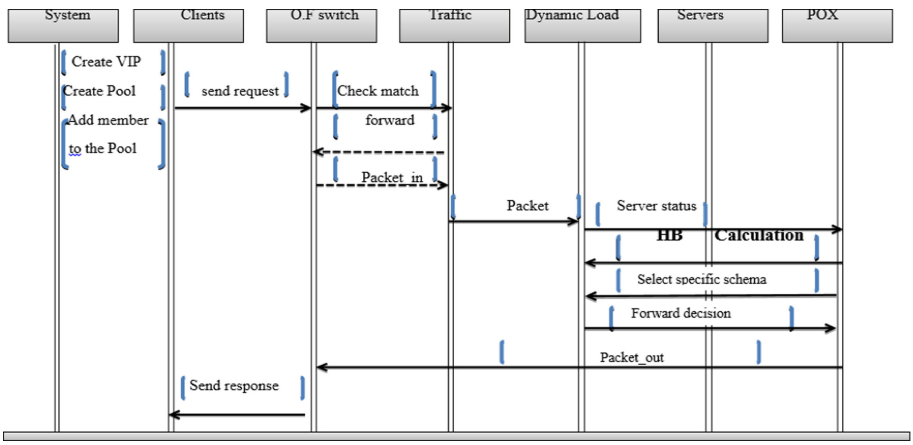

Fig. 1. HB load-balancing algorithm flow-sequence diagram

As shown in Fig. 1, the flow-sequence diagram Steps of HB load balancing algorithm is listed as follows:

- **System Configuration Process**
  The user should initially set up the server pool, the VIP address associated with that pool, and the traffic type. The configuration of the server pool is based on the number of member servers. Since a single service is provided by the server members and single type of transmission that is TCP. Therefore, the proposed system requires a single pool
- **The Addition of Member Servers to Pool**
  Three HTTP servers (HTTP Server1, HTTP Server2, and HTTP Server3) are launched into the pool relying on their layer three static IP address with the same port of VIP.
- **Sending Requests**
  The OpenLoad benchmarking tool plays an essential role in evaluating the performance of server members. OpenLoad is discovered to be flexible for use and provides a near real-time performance measurement of the application under test, which is particularly useful when performing optimization as the impact of changes could be immediately observed. This tool relies on investigating the effect of the gradual growth of load represented by concurrent users on the server's performance metrics. The number of concurrent users' requests varies from 0 up to 350 (req/sec) with a uniform increase of 50 (req/sec)
- **Receive and Process Packet_in Messages**
  When the message reaches the OpenFlow switch and no base entry is found, then it is forwarded to the POX controller, which verifies whether the message is IPv4 or not. If yes, then the packet is parsed to obtain details like the type of service, destination IP, and source IP. The POX controller utilizes the IP address of server members and the VIP to send ARP messages in order to check if there is a server crash (the server that has not responded to this message for some time) and must be deleted from the server list in the load balancing application.
- **Load Balancing Mechanism**
  The ASMM reports the OpenFlow switch statistics, including ports and flows entry counter statistics of each server member in the pool periodically every 5 s. The HLBM executes the tasks represented by the computation of the load for each server member according to Eq. (4) and Eq. (5) to select the optimal server for handling the incoming request from the client. The experimental evaluation compares the proposed HB load balancing algorithm with two dynamic-based load balancing algorithms represented by WRR and LCB under the same conditions. WRR is simple and runs fast but lacks the accuracy during load balancing of varying load size or request complexity, and current server load is not considered during request distribution. The LCB takes into account the server load before load delivery, which ends up when the load is acquired sparsely, leaving some servers idle beside the Addressing the imperfect system performance when the processing capabilities of the servers are different, thereby reducing system performance (Table 1).

Table 1. Experimental parameters values.

Parameter	Value
Host operating system	Linux (Ubuntu) version 14.04
Programming language	Python 2.7.6
Simulation tool	OpenLoad
Max no. concurrent users	350
Min no. concurrent users	0
No. of servers (3)	3
No. of clients	9
SDN controller	(POX)
Virtual SDN switch	OpenFlow switch
POX Controller to OpenFlow switch port	6633
HTTP web servers listening port	80
Virtual IP (service IP)	10.0.1.1
No. of iterations	7
Link Latency	No

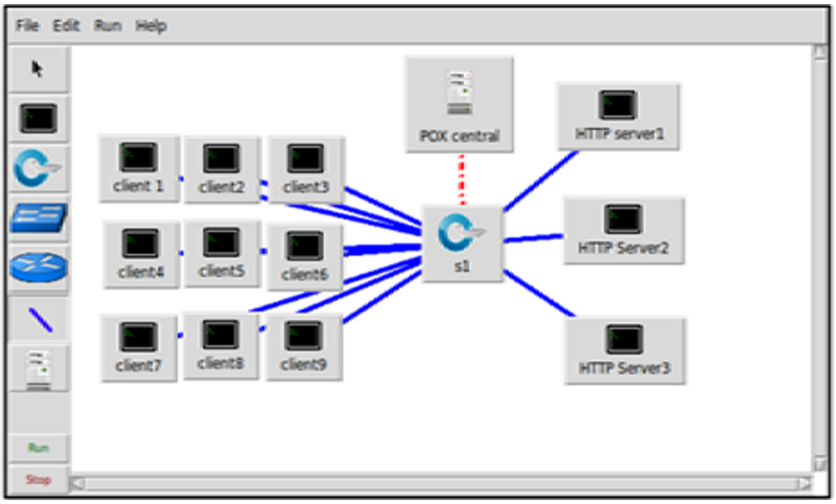

Fig. 2. SDN-based platform network topology

The network topology in Fig. 2 utilizes a single remote POX due to owning free, open-source utilities that facilitate the addition and removal of restored units and allow the experiments to be more flexible and reliable. Moreover, this section adopts an OVS switch along with a companion Linux kernel module for flow-based switching as shown in Fig. 2. The listening port 6633 is the default connection port for the mininet to create connections between the OpenFlow switch and the POX. The Mininet emulator creates a virtual network topology that consists of 3 hosts acting as

Apache HTTP web servers that offer the same web services and 9 hosts' serves as clients who access the web servers using a virtual IP address.

6 Results and Discussions

The outcomes of the research can be discussed as follows:

6.1 Server Transactions per Second

The Server Transactions per Second is the first metric to investigate, which is regarded as one of the essential performance parameters capable of handling routine and keeping records. In general, this metric defines the total number of transactions accomplished by a server in a given period divided by the seconds per that period. The load is represented by the simultaneous number of users attempting to access the local server webpage, which starts from 0 up to 350 to verify the ability of load balancing schemes to distribute several flows in a parallel path between the source and the destination.

$$\text{Transactions per Second (TPS)} = \frac{TPS1 + TPS2 \cdots \ldots TPSn}{\sum T} \tag{4}$$

Related to Eq. (4), Fig. 3 Demonstrates servers' transactions per second when adopting WRR, LCB, and proposed HB load balancing schemes. The experimental results indicate a noticeable improvement to the proposed HB scheme, recording 67.73 compared to 53.5, 60 to WRR and LCB schemes, respectively.

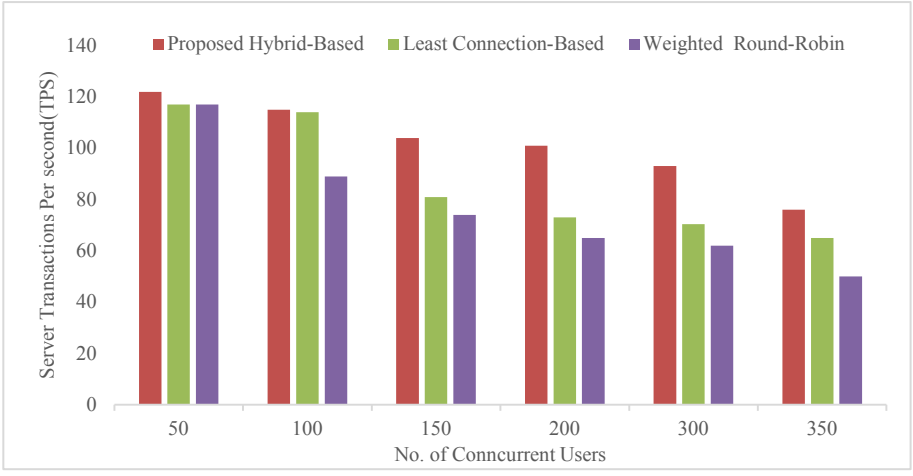

Fig. 3. Server transactions per second vs. the no. of concurrent users

6.2 Server Average Response

This parameter expresses the required time to deliver request results to the clients. Different variables may affect the response time, such as average thinking time, number of requests, number of users who accessed the system, and the bandwidth [20].

$$\text{Server Average Response Time (SART)} = \frac{Un}{Rn} - Tt. \tag{5}$$

Whereas T_T is the thinking time per request, Un is the number of concurrent users, and Rn is the number of requests per second. According to Eq. (5), Fig. 4 proves that polylines fluctuate in a stable and minimal form during the application of the proposed HB scheme, which leads to selecting the server with the minimum average response time of 1.28 ms as compared with 1.49, 1.51 ms associated to WRR and LCB load balancing policies (Fig. 5).

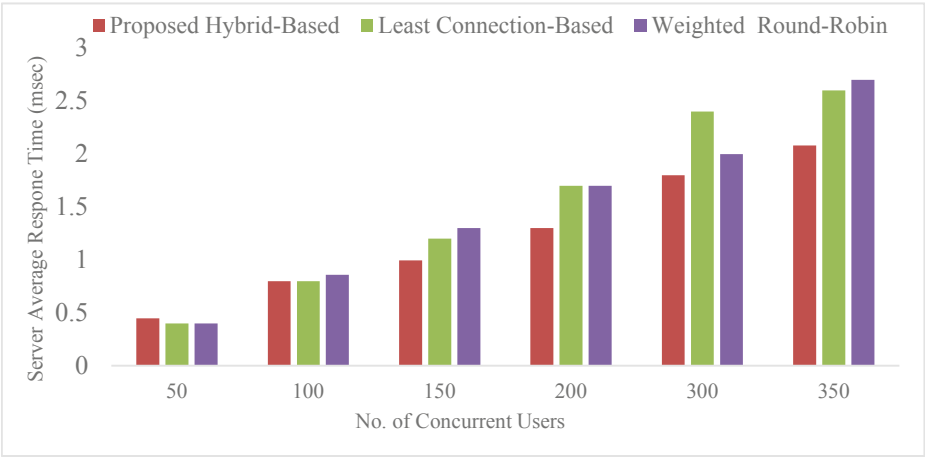

Fig. 4. Server Average Response Time vs. the no. of concurrent users

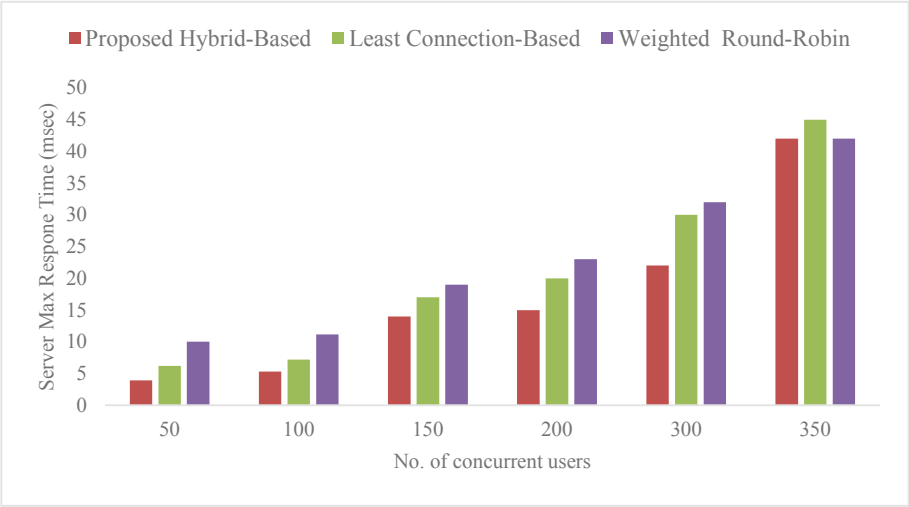

Fig. 5. Server Maximum Response Time vs. the no. of concurrent users

6.3 Server CPU Capacity

The Server CPU capacity is the final metric to investigate, which refers to the process responsible for calculating the number of resources needed to provide the desired level of services for a given workload, as presented in Eq. (6) [20].

$$\text{Server CPU capacity} = \frac{1}{SART} \tag{6}$$

According to Eq. (6), the results from Fig. 6 unmistakably imply that the proposed HB scheme records the highest ratio of Server CPU capacity up to 1.033 compared to 0.988 and 0.981 belonging to the LCB and WRR load balancing schemes.

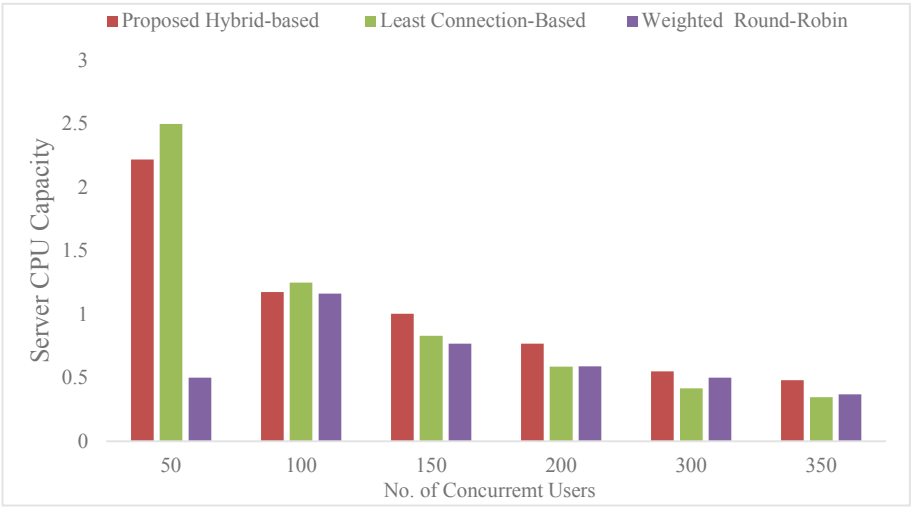

Fig. 6. Server CPU Capacity vs. the no. of concurrent users

7 Conclusion

The development of SDN architecture offered new possible options to solve the conventional load balancing problems. This article suggested a new load balancing scheme using the POX controller under the SDN architecture to pick the server, achieving several vital metrics in SDN like the maximum transactions per second, minimum average response time, and maximum server CPU capacity. Moreover, it seemed to be sufficient to overcome low-performance problems and high load balancing costs. It is possible to conclude that the proposed HB load balancing algorithm achieves a significant progress in terms of server transactions per second up to 10.41%, server average response time up to 24.61%, and server CPU capacity up to 9.55%. It is also essential to state that any attempt to increase the number of concurrent users above 350 lead servers to enter the saturation region, which causes several drawbacks and performance degradation.

8 Future Work

In the following, we present possible future research directions that may be conducted to extend the dissertation innovations:

- Future work will address the scalability of the proposed HB load balancing algorithm achieved by the dynamic addition of the hosts to the existing pool when all pool members are being overloaded.

- Adopting multiple controllers to avoid the single point of failure problem as additional controllers carry out the load balancing task when the master controller goes down.
- The assignment of dynamic weights instead of static ones for the servers participating in the proposed hybrid-based load balancing scheme. Dynamic weights generally reflect actual server capabilities and further improve the performance of the balancing scheme.

References

1. Trestian, R., Katrinis, K., Muntean, G.M.: OFLoad: an OpenFlow-based dynamic load balancing strategy for datacenter networks. IEEE Trans. Netw. Serv. Manag. **14**(4), 792–803 (2017)
2. Semong, T., Maupong, T., Anokye, S., Kehulakae, K., Dimakatso, S., Boipelo, G., Sarefo, S.: Intelligent load balancing techniques in software defined networks: a survey. Electronics **9**, 1091 (2020)
3. Xie, J., Guo, D., Hu, Z., Qu, T., Lv, P.: Control plane of software defined networks: a survey. Comput. Commun. **67**, 1–10 (2015)
4. Mendiola, A., Astorga, J., Jacob, E., Higuero, M.: A survey on the contributions of software-defined networking to traffic engineering. IEEE Commun. Surv. Tutor. **19**(2), 918–953 (2016)
5. Kavana, H.M., Kavya, V.B., Madhura, B., Kamat, N.: Load balancing using SDN methodology. Int. J. Eng. Res. Technol. **7**, 206–208 (2018)
6. Bholebawa, I.Z., Jha, R.K., Dalal, U.D.: Performance analysis of proposed OpenFlow-based network architecture using Mininet. Wirel. Pers. Commun. **86**, 943–958 (2016)
7. Kreutz, F.M., Ramos, V., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**, 14–76 (2014)
8. Chen-Xiao, C., Ya-Bin, X.: Research on load balance method in SDN. Int. J. Grid Distrib. Comput. **9**, 25–36 (2016)
9. Muthumanikandan, V., Valliyammai, C.: Link failure recovery using shortest path fast rerouting technique in SDN. Wirel. Pers. Commun. **97**, 2475–2495 (2017)
10. Chen, L., Qiu, M., Dai, W., Jiang, N.: Supporting high-quality video streaming with SDN-based CDNs. J. Supercomput. **73**(8), 3547–3561 (2016). https://doi.org/10.1007/s11227-016-1649-3
11. Karakus, M., Durresi, A.: A survey: control plane scalability issues and approaches in software-defined networking (SDN). Comput. Netw. **112**, 279–293 (2017)
12. Jha, R.K., Llah, B.N.: Software Defined Optical Networks (SDON): proposed architecture and comparative analysis
13. Sufiev, H., Haddad, Y., Barenboim, L., Soler, J.: Dynamic SDN controller load balancing. Future Internet **11**, 75 (2018)
14. Mehra, M., Maurya, S., Tiwari, N.K.: Load balancing in software defined network: a survey. Int. J. Appl. Eng. Res. **14**, 245–253 (2019)
15. Chahlaoui, F., Dahmouni, H.: A Taxonomy of load balancing mechanisms in centralized and distributed SDN architectures. SN Comput. Sci. **1**, 1–16 (2020)
16. Iyer, N., Hugar, N.S., Manjunath, M.N.: Load balancing using open daylight SDN controller: case study. Int. Res. J. Adv. Sci. Hub **2**, 59–64 (2020)

17. Jader, O.H., Zeebaree, S.R., Zebari, R.R.: A state of art survey for web server performance measurement and load balancing mechanisms. Int. J. Sci. Technol. Res. (IJSTR) **8**, 535–543 (2019)
18. De Rango, F., Inzillo, V., Quintana, A.A.: exploiting frame aggregation and weighted round robin with beamforming smart antennas for directional MAC in MANET environments. Ad Hoc Netw. **89**, 186–203 (2019)
19. Singh, N., Dhindsa, D.K.: Hybrid scheduling algorithm for efficient load balancing in cloud computing. Int. J. Adv. Netw. Appl. **8** (2017). 0975-0290
20. Osman, A.A.A.: Service based load balance mechanism using Software-Defined Networks. Doctoral dissertation, University of Malaya (2017)