doi: 10.1093/comjnl/bxz142

Hidden Markov Model-based Load Balancing in Data Center Networks

Binjie He, Dong Zhang* and Chang Zhao

College of Mathmatics and Computer Science, Fuzhou University, China
******Corresponding author: zhangdong@fzu.edu.cn*

Modern data centers provide multiple parallel paths for end-to-end communications. Recent studies have been done on how to allocate rational paths for data flows to increase the throughput of data center networks. A centralized load balancing algorithm can improve the rationality of the path selection by using path bandwidth information. However, to ensure the accuracy of the information, current centralized load balancing algorithms monitor all the link bandwidth information in the path to determine the path bandwidth. Due to the excessive link bandwidth information monitored by the controller, however, much time is consumed, which is unacceptable for modern data centers. This paper proposes an algorithm called hidden Markov Model-based Load Balancing (HMMLB). HMMLB utilizes the hidden Markov Model (HMM) to select paths for data flows with fewer monitored links, less time cost, and approximate the same network throughput rate as a traditional centralized load balancing algorithm. To generate HMMLB, this research first turns the problem of path selection into an HMM problem. Secondly, deploying traditional centralized load balancing algorithms in the data center topology to collect training data. Finally, training the HMM with the collected data. Through simulation experiments, this paper verifies HMMLB's effectiveness.

Keywords: hidden Markov Model (HMM); load balancing algorithm; data center

Received 15 July 2018; revised 13 August 2019; editorial decision 22 October 2019; accepted 22 October 2019
Handling editor: Alan Marshall

1 INTRODUCTION

With the development of networks and applications, it has come increasing demands on the performance of data centers. For example, the growth of cloud computing accompanied by the emergence of applications such as MapReduce [1] require a high volume of data center throughput to process and send computational data. At the same time, more and more applications of companies rely on public data centers. The need for high quality services for those users imposes new challenges and requirements on the development of modern data centers. In response, new data center architectures such as fat-tree [2], VL2 [3] and DCell [4] have emerged. Their symmetric architecture can provide multiple peer-to-peer shortest paths between nodes, thereby increasing the data center network throughput to deal with the new demands. However, how to reasonably distribute data flows among multiple peer-to-peer shortest paths has become an urgent problem.

Load balancing can be used to distribute data flows among multiple paths and balance network load rationally. Load balancing algorithms can be divided into two categories, distributed and centralized control [5]. The distributed load balancing algorithm makes modifications and decisions on path selection through information of each switch. The centralized load balancing algorithm can utilize the network center controller to collect information about the entire network and use the information to make decisions; thus, its path selections are usually more reasonable. However, the controller requires time to monitor the global network information, so the time delay is substantial. Kandula and Greenberg [6] pointed out that a data center with 1500 servers directs 1×10^5 data flows per second into the data center network. Large-scale data flow requires that the load balancing algorithm cannot consume too much time, or it will be unable to handle such a large data flow. (A flow carrying a small amount of data called 'mice flow' and carrying a large amount of data called 'elephant flow'.)

Hedera [7] is a centralized load balancing algorithm. To reduce the time delay, it classifies the data flow as a mice flow or an elephant flow in the edge switch. For a mice flow with a low-latency requirement, it uses equal-cost multipath routing (ECMP) to select a path. For an elephant flow with a high throughput requirement, it utilizes the controller to collect information on all links to choose a path. Compared with ECMP, Hedera can effectively improve the bisection bandwidth, and it consumes less time than a centralized control load balancing algorithm that does not classify data flow. In a later study, Mahout [8] optimizes the data flow classification by categorizing the data flow in the data sending buffer of terminal, further reducing time consumption.

However, the time consumption of the centralized control load balancing algorithm comes mainly from the controller monitoring link. To reduce part of the time consumption, Hedera and Mahout use data flow classification to make mice flow routes through ECMP, which does not require link information. However, for an elephant flow, the controller must still monitor the links on all parallel paths.

This paper proposes an algorithm called hidden Markov Model-based load balancing (HMMLB) for reducing the link monitoring time consumption caused by the path selection of an elephant flow. HMMLB uses the hidden Markov Model (HMM) [9] to choose paths for data flows. This research deploys traditional centralized load balancing algorithms in a data center topology and records data such as link load and path selection information as training samples. Through supervised learning, the HMMLB can achieve the same network throughput rate as traditional centralized load balancing and decrease the number of monitored links and the amount of time to select a path. Meanwhile, we are aware that the bandwidth between centralized controllers (e.g. software-defined network (SDN) controllers) and the underlying switches is limited and valuable, and a large number of link load data transmissions will undoubtedly consume bandwidth resources between controllers and switches [5]. The HMMLB can save those bandwidth resources by reducing the number of monitored links.

In our research, the following questions must be resolved: which links should be monitored and how to turn the path selection problem into an HMM problem. For the first question, the architecture of a modern data center provides parallel paths for end-to-end communication. There are parallel links in different parallel paths. For a given set of parallel links, we need to choose only one or several monitored links as a basis for determining the load status. For the second question, the problem can be expressed as 'How to determine the observable states and hidden states of the HMMLB.' For the observable states, the proposed work uses the bandwidth of the monitored link usage. The switches in the path are used as hidden states. The hidden states corresponding to the observable states were selected, and the data flow transmission path was finally formed.

Section 2 introduces the background of this research and related work. Section 3 proposes a model of the path selection problem for a centralized load balancing algorithm, and an objective function to measure performance. Section 4 elaborates on the principles and details of the HMM training and application modules and architecture of HMMLB. Section 5 documents the results of the simulation experiment and demonstrates the effectiveness of the algorithm. Section 6 summarizes this paper.

2 BACKGROUND AND RELATED WORK

2.1 Traffic pattern in a data center network

Previous researches [10, 11] show that applications deployed in data centers are becoming increasingly diverse over time, resulting in data flows with different requirements; there are delay-sensitive data flows and throughput-sensitive data flows. Delay-sensitive data flows are usually small, approximately a few kilobytes, and generated by interactive applications or network protocols. Throughput-sensitive data flows are generally larger than delay-sensitive flows and are typically created by applications such as MapReduce [1], virtual machine migration and data backup. Kandula and Greenberg[6] pointed out that in modern data centers, 90% of data flows carry less than 1 MB of data, and 90% of those data are generated by data flows that carry more than 100 MB of data.

2.2 Load balancing of centralized mechanisms

This section describes related work on centralized load balancing algorithms.

ECMP [12] is an earlier noncentralized load balancing algorithm. It selects the path for the target flow through a hashing algorithm. It does not take into account how the different alternative paths differ. It only chooses the transmission path based on hashing results. Thus, ECMP may lead to a congested path with packet loss, and the remaining paths may still have much available bandwidth, which is contrary to the original design of modern data center architecture [13]. A centralized load balancing algorithm can solve the problems faced by ECMP. The centralized load balancing algorithm can collect global topology link information through the controller and improve the rationality of data flow path selection, thereby increasing the data center throughput [5]. However, that algorithm also has its disadvantages. It can increase the throughput of the data center, but there is a huge time cost for the controller to collect the global topology link information. Therefore, researchers need to solve the problem of how to minimize the latency caused by the centralized load balancing algorithm.

In this paper, load balancing algorithms are classified into two categories according to whether there are parameter adjustments during the running of the algorithms. The static class (e.g. Hedera *et al.* [14]) is where the algorithm runs according to set parameters and does not adjust settings while running. The dynamic class (e.g. Flow Distribution Aware Load Balancing (FDALB) [15] and Freeway [16]) is where the algorithm adjusts parameters while running according to the network conditions to fix the attributes of path or data flows.

Hedera [7]: Hedera reduces the overhead incurred by the network controller from monitoring underlying links by using different path selection strategies for different types of data flows. It divides flows into elephant flows and mice flows in edge switches by using their flow rate. ECMP allocates mice flows a path. Elephant flows are allocated a path by First Fit or Simulated Annealing (SA) according to path bandwidth information. Hedera's contribution lies in the way it proposes to classify data flows into elephant and mice flows, and because mice flows are not sensitive to throughput rates, it uses ECMP to choose paths, which reduces the time overhead. However, Hedera still monitors many links for elephant flows, which still consumes much time.

Mahout [8]: Mahout's central idea, like Hedera, is to reduce the network controller's monitoring overhead by using different selection strategies for different types of data flows. Therefore, Mahout is also required to monitor many links to make path selection decisions for elephant flows. However, it uses the socket buffer on the terminal, instead of edge switches, to determine the type of data flow. This method reduces the time spent in judging the types of data flows.

Fastpass [14]: Fastpass is different from Mahout and Hedera. It does not distinguish between elephant and mice flows but uses a mixture of transport protocols and traffic distribution policies to achieve zero-queuing in the data center. A Fastpass control protocol (FCP) determines when each packet can be transmitted. At the same time, Fastpass controls the transmission path of data flows by monitoring the link information. The Fastpass contribution is that due to the presence of the FCP, it reduces the queuing delay in the network and speeds the transmission of data packets. However, on the problem of path selection, it does not classify the data flow or optimize the link monitoring time cost.

FDALB [15]: FDALB researches data flow classification. Its core mechanism is similar to those of Mahout and Hedera. For a long flow, it uses the information of the links monitored by the controller as a basis for selection, and for a short flow, it uses a static selection algorithm such as ECMP. Unlike Mahout and Hedera, its threshold for long flows and short flows is constantly adjusted and changed according to network conditions. FDALB makes the data flow classification more reasonable and enables the network to achieve higher throughput rates. However, the path selection method for elephant flows is not changed; FDALB does not significantly improve time performance.

Freeway [16]: depending on the bandwidth of the transmission path, Freeway classifies the path as either high throughput or low latency. For an elephant flow, Freeway uses the link information monitored by the controller to make a selection decision for a high-throughput path, and for a mice flow uses ECMP to make a selection decision for a low-latency path. Freeway ensures that the transmission paths of elephant and mice flows do not affect each other, and can improve the network throughput to some extent. However, the disadvantage of Freeway is that it requires link bandwidth information to classify paths. Also, Freeway cross-converts high-throughput paths and low-latency paths at regular intervals depending on the bandwidth of the path. The disadvantage increases Freeway's time overhead, so it does not do well in time performance.

The article [17] proposed a centralized load balancing. It uses path computation element (PCE) to calculate the best path between end to end. PCE uses PCE protocol to collect and deliver link usage state. It not only considers the link usage state but also takes the resource consumption into account. However, it needs to monitor all links in the network. The time performance is ordinary.

The article [18] improves backpressure-based algorithms. There are two shortcomings of the backpressure-based algorithm. Firstly, it may choose a long path for a flow, which is not necessary. Secondly, it consumes a lot of memory to store path selection information. The article forces every flow to transfer by the shortest path and apply other paths only when the shortest path congest happens. The paper also decrease memory consumption by reducing the path information store. However, the article determines the link state by observing the switch queue length without considering the link usage. Thus, the throughput performance is not very good.

The paper [19] is a survey of nature-inspired meta-heuristic load balancing algorithms. There are five kinds of nature-inspired meta-heuristic algorithms. The ant colony optimization technique is inspired by the ant colony. It has an excellent performance of throughput, but a local optimum might occur in the algorithm. Genetic Algorithm (GA) is inspired by the gene. GA load balancing can be used in SDN. However, it cost a lot of time in training and path selection. Particle swarm optimization (PSO), which stems from the simulation of birds flock. PSO load balancing also can be used in SDN, but the scalability is very low. The greedy algorithm is widely used by the load balancing algorithm, but it has a high time cost. SA can reduce the time consumption of path selection, but the performance of throughput is undesirable.

The related researches on centralized control load balancing show that to reduce the time overhead, the methods above classify the data flow mainly through the features of the data center flow, so only a few parts of flows require the global network information. However, for a path selection problem that requires global network information, the controller still must monitor many links.

Moreover, there are some distributed load balancing algorithms are worth referencing. Multi-level dynamic traffic load-balancing (MDTLB) [20] is a distributed load balancing in which the link state is determined by observing the round-trip time (RTT). It divides links to five grades: very good, good, grey, bad and very bad. MDTLB uses a rerouting algorithm to reroute the flows that were blocked. When a new flow enters the network, MDTLB will select the path according to the link state level. Because MDTLB is a distributed mechanism, the collection of link state is not very accurate, and the throughput performance is low. The candidate path table of MDTLB needs to be updated and traversed constantly, thus time performance is general. Luopan [21] is also a distributed load balancing in which the link state is determined by sampling. Luopan divides elephant flow to flow cells and send flow cells to different paths. Because Luopan uses probes to take links sample, the time cost is very high. Besides, Luopan needs to sample all links in the network which cut down the time performance.

3 MODELING THE PATH SELECTION PROBLEM IN THE CENTRALIZED MECHANISM

In centralized load balancing algorithms, path selection is a core problem. Because a centralized data center network can obtain the link information of the entire network, the algorithm utilizes link information to decide on a path and to improve the rationality of the selection decision. Path selection of a centralized load balancing algorithm comprises two main steps: first, a centralized controller of the data center collects link bandwidth information; second, the centralized load balancing algorithm uses a path selection strategy (e.g. First Fit [22], SA [23], greedy algorithm) to select a path.

Therefore, this paper decides to use the path selection model of a traditional centralized load balancing algorithm. In the path selection model, the input is link bandwidth information. The processing module calculates the path bandwidth by using the link bandwidth information and chooses a path selection strategy to select the transmission path for data flow; the output of the model is a transmission path.

3.1 Re-forming the path selection model with HMM

In the common HMM, the input is the chain of observable states, and the processing module uses the three matrices of the HMM (EmitProbMatrix, TransProbMatrix, InitStatus) to infer the most probable chain of hidden states corresponding to the chain of observable states.

Before re-forming the path selection model, the network parameters represented by the observable and hidden states must be determined. In this research, link bandwidth information is taken as an observable state, because it can be observed. The switch is treated as a hidden state because a path is made up of a series of switches.

So the input of the new model can be formed from a series of link bandwidths (a chain of observable states). The processing module would use the three matrices of the HMM to infer the path with the highest probability (a chain of hidden states) that corresponded to the link bandwidth information. The output would be the transmission path of the data flow.

3.2 Objective functions

This paper proposes two objective functions to evaluate the performance of the new model.

The throughput rate of the network can be measured by the actual bisection bandwidth [24], thus defining the percentage of the ideal bisection bandwidth $bisection_I$ of the network that is taken up by the actual bisection bandwidth as a measure of the network throughput rate. The ideal bisection bandwidth can be derived from the network topology. Proposed work uses a cross section to divide the network into two equal subnets. The ideal bisection bandwidth ($bisection_I$) is the sum of the theoretical bandwidth of the link cut by the cross section ($bandwidth_{link}$). $bisection_P$ means the practical bisection bandwidth. So there is the first objective function to measure network throughput:

$$P = \frac{bi\sec tion_P}{bi\sec tion_I} = \frac{bi\sec tion_P}{\sum_1^m bandwidth_{link}}. \tag{1}$$

m is the number of links cut by the cross section. When P = 1, the network achieved the best throughput. The second objective function measured the time delay required by the network controller to monitor the links. This paper assumes that there are several parallel paths in end-to-end communication, which consist of n links. Several of these links must be monitored. The time required to monitor several of the links can be expressed as $\sum t_j (j \in \textit{the sequence number of observed link})$. The time required to monitor all links can be expressed as $\sum_1^n t_i$. Then there is the second objective function to measure time:

$$T = \frac{\sum t_j}{\sum_1^n t_i}. \tag{2}$$

When T = 1, the centralized load balancing algorithm monitors all links. In this instance, the information must have been the most comprehensive, but the time consumption was also the largest. Therefore, the goal of using the centralized load balancing algorithm is to increase the network throughput as much as possible; that is, to maximize P, and to reduce the algorithm processing delay as much as possible; that is, to minimize T.

4 HMMLB DESIGN

This paper utilizes the HMM to re-form the path selection model, but some problems need to be solved: firstly and most importantly, to make up the chain of observable states, we needed to know which links should be monitored. Secondly, the HMM training algorithm should be proposed. Thirdly, when the three matrices of HMM concrete numerical value was determined, and the model had been trained, the HMM should select a path for data flow [25]. The first and second problems will be answered in Section 4.2. The third problem will be solved in Section 4.3.

To enable the answer of the first problem to be easier understood, there is a simple example that should be put forward in advance. This paper believes that in end-to-end multipath communication, there are parallel links between parallel paths. In Fig. 1, a1 and a2 in the simple topology

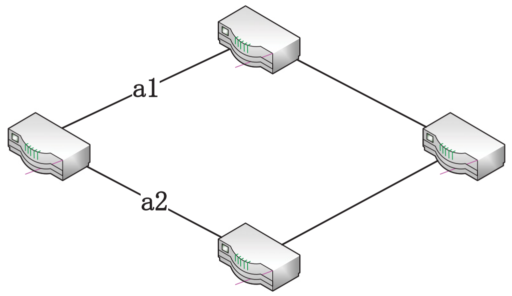

FIG. 1. Simple topology of switches.

shown are parallel links. There is a relation between parallel links, which is called 'as one falls, other rises'. It can be expressed in a general way; in end-to-end communication, when a data flow is allocated to one of the links, the corresponding parallel link saves this part of the bandwidth. Then when the next data flow comes, it is more probable that the link that saved part of the bandwidth will be chosen, therefore choosing one of the parallel links as a monitored link. A series of monitored links composes the chain of observable states.

4.1 The HMM

The HMM is a machine learning algorithm; it is also a statistical model. The HMM is used to describe a Markov process with hidden, unknown parameters. Its core use is to determine the hidden parameters of the process from observable parameters.

The HMM contains N possible sets of hidden states: $Q = \{q_1, q_2, \dots q_N\}$, and M possible sets of observed states: $V = \{v_1, v_2, \dots v_M\}$. The HMM also contains three elements: (i) a TransProbMatrix, $A = [a_{ij}]_{N*N}$. a_{ij}, is the probability that at any time t and state q_i conditions occur, the next time t+1 state is q_j; (ii) an EmitProbMatrix, $B = [b_{ij}]_{N*M}$, where b_{ij} is the probability of generating observation v_j under any condition of time t and state q_i; and (iii) an initial state probability vector, $\overrightarrow{\pi} = \pi_1, \pi_2, \pi_3 \dots \pi_N$; π_i is the probability of state q_i under the condition that the initial time t=1 [26]. In summary, an HMM can be described as (3)

$$\lambda = (A, B, \vec{\pi}). \tag{3}$$

For HMM, given the observation sequence $O = (o_1, o_2 \dots o_T)$, it had to estimate the parameters of model $\lambda = (A, B, \vec{\pi})$ so that $P = (O|\lambda)$ was the maximum. In order to estimate the parameters, $(O_1, I_1) \dots (O_S, I_S)$ is used to be the training set. Then the maximum likelihood estimation [27] is used to estimate the model parameters. The TransProbMatrix $A = [a_{ij}]_{N*N}$, the EmitProbMatrix $B = [b_{ij}]_{N*M}$ and the initial state probability vector $\overrightarrow{\pi}$ can be calculated by Equations (4), (5) and (6), respectively. A_{ij} is the frequency of state q_i at time t converted to state q_j at time t+1. B_{ij} is the frequency of observation v_j generated by the state value q_i in the sample, and η_i is the frequency of the initial state q_i in the sample.

Moreover, for HMM, given the model $\lambda = (A, B, \vec{\pi})$ and the observable state sequence $O = (o_1, o_2 \dots o_T)$, it had to infer the hidden state sequence $I = (i_1, i_2 \dots i_T)$ of this observable state sequence with the greatest probability; that is, the I can maximize $P = (I|O)$. The solution is explained by application module in Section 4.3.

$$\hat{a}_{ij} = \frac{A_{ij}}{\sum_{j=1}^{N} A_{ij}} \tag{4}$$

$$\hat{b}_{ij} = \frac{B_{ij}}{\sum_{j=1}^{N} B_{ij}} \tag{5}$$

$$\hat{\pi}_i = \frac{\eta_i}{\sum_{1}^{N} \eta_i} \tag{6}$$

4.2 Instantiation of the HMM in the path selection problem

To make the HMM available in the load balancing algorithm, the corresponding parameters of the HMM's observable and hidden states in the network should be specified. Because the ultimate goal of the algorithm is to select a path, and the path is composed of a sequence of switches, the switches are used as the hidden states of the HMM.

To define the observable state in the HMM, the following example is illustrated: Modern data center architectures provide several parallel paths for end-to-end communication. Assuming that a parallel path consists of N switches, $\{n_1, n_2, n_3 \dots n_N\}$, then position n_i can select one of the alternative switch sets $S_i = \{s_1, s_2, s_3 \dots s_k\}$ for padding. Note that the value of k also changes due to different locations. For example, in a fat-tree topology whose pod = 4, two hosts with different pods communicate with each other; the parallel path topology is shown in Fig. 2. There are four parallel paths from Host a to Host b, and the number of switches on the parallel paths is N = 5. There is only one unique designated switch, *Switch*1, in the location of n_1, and the position of n_2 can be selected from the set of alternative switches $S_2 = \{Switch2, Switch3\}$. This paper refers to switches in the same set as equivalent switches, and the equivalent switch is selected based on its corresponding monitored link bandwidth. In the example shown in Fig. 2, selecting the equivalent switch in set S_2 at position n_2 requires the bandwidth conditions of a1 and a2, which called parallel links. In fact, it is believable that the network can select the appropriate equivalent switch by monitoring only a portion of the parallel links based on the principle 'as one falls, other rises.' That is, some are selected as monitored links in the parallel links, and the bandwidth status of the monitored links is used as the observable state. Each observable state; in other words, the monitored links' bandwidth usage, corresponds to a hidden state; in other words, the selected switch number.

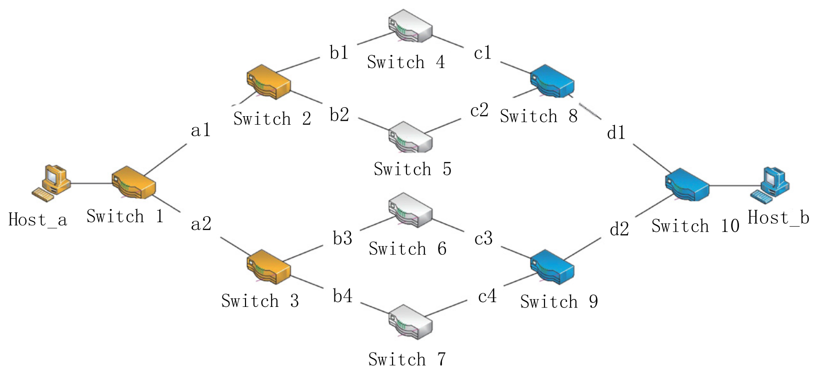

FIG. 2. A unit of fat-tree.

TABLE 1 Quantization of monitored link bandwidth utilization.

Monitored link's bandwidth utilization	Quantized value
Parallel link failure	0
Less than 20%	1
Between 20% and 50%	2
More than 50%	3
Observed link failure	4

Note that, according to the above description, in the process of selecting a path P, N switches must be selected in sequence. If a switch position A has only one switch to choose from, then it has no corresponding monitored link; this eliminates the need for bandwidth monitoring and allows direct selection of the only switch. To facilitate the calculation, the observable state should be quantified; the provisions are shown in Table 1. In order to cope with link failure, the quantized states 0 and 4 are set. The method of dealing with link failure will be presented in 4.5.

Now that the instantiated HMM model has been established, so the model could be trained according to Equations (4), (5) and (6). The training algorithm is shown in Algorithm 1.

4.3 The application module

The work of the application module was to solve the second basic problem of the HMM; that is, given the model $\lambda = (A, B, \vec{\pi})$ and the observable state sequence $O = (o_1, o_2 \ldots o_T)$, it was inferred that the hidden state sequence $I = (i_1, i_2 \ldots i_T)$ of the observable state sequence could be generated with the greatest probability; that is, the I that can maximize $P = (I|O)$. The Viterbi algorithm [28] is generally used to solve this problem. $\delta_t(i)$ is the maximum probability in all paths with state i at time t, expressed as

$$\delta_t(i) = \max P(i_t = i, i_{t-1}, \ldots i_1, o_t, \ldots o_1|\lambda). \tag{7}$$

Algorithm 1 Training Algorithm

```
S:the total number of switchs in Path K
datamatrix[M][N] = read from Train data
for i=0;$i < M$;i++; do
  update Ematrix[ ][ ]
  update Tmatrix[ ][ ]
  update InitStatus[ ]
end for
for traverse the Ematrix do
  $Ematrix[i][j] = \frac{Ematrix[i][j]}{\sum_{j=0}^{N-1} Ematrix[i][j]}$
end for
for traverse the Tmatrix do
  $Tmatrix[i][j] = \frac{Tmatrix[i][j]}{\sum_{j=0}^{N-1} Tmatrix[i][j]}$
end for
for traverse the Initialvector do
  $InitStatus[i] = \frac{InitStatus[i]}{\sum_{i=0}^{S} InitStatus[i]}$
end for
return Ematrix,Tmatrix,InitStatus
```

In addition, $\psi_t(i)$ is defined as the previous maximum probability state at time t with state i. Then it takes four steps to calculate the best path. Firstly, initialized by Equations (8) and (9), b_i is the set of all elements in row i of EmitProbMatrix $B = [b_{ij}]_{N*M}$:

$$\delta_1(i) = \pi_i b_i(o_1), \qquad i = 1, 2, \ldots, N \tag{8}$$

$$\psi_1(i) = 0, \qquad i = 1, 2, \ldots, N. \tag{9}$$

Secondly, recursion. For t = 2,3..., T has Equations (10) and (11):

$$\delta_t(i) = \max_{1 \le j \le N} [\delta_{t-1}(j) a_{ji}] b_i(o_t), \qquad i = 1, 2, \ldots, N \tag{10}$$

$$\psi_t(i) = \arg\max_{1 \le j \le N} [\delta_{t-1}(j) a_{ji}], \qquad i = 1, 2, \ldots, N. \tag{11}$$

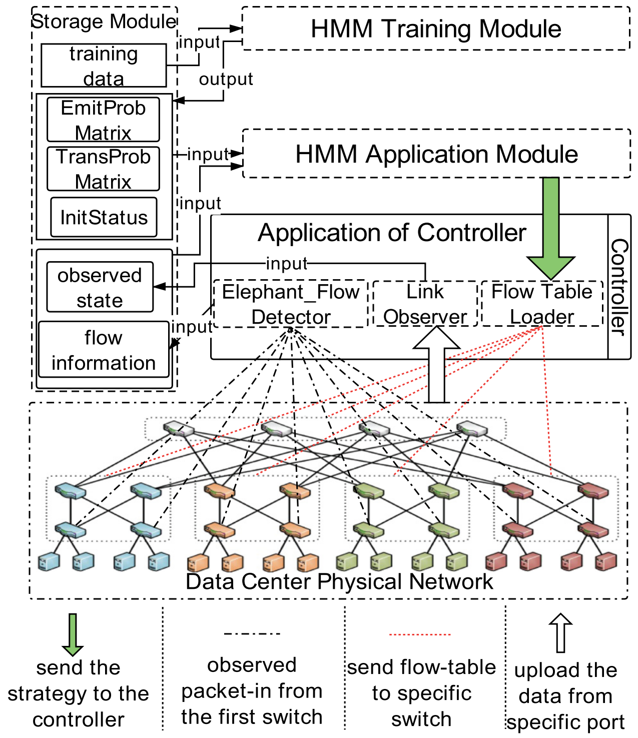

FIG. 3. Structure of HMMLB.

Thirdly, termination. This can be expressed as Equations (12) and (13):

$$P^* = \max_{1 \leq j \leq N} [\delta_T(i)] \tag{12}$$

$$i_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)]. \tag{13}$$

Finally, an optimal path backtracking is done, and there is Equation (14) for t = T-1, T-2..., 1:

$$i_t^* = \psi_{i+1}(i_{i+1}^*) \tag{14}$$

finding the optimal path $I^* = (i_1^*, i_2^*, ..., i_T^*)$.

To comply with the requirements of the Viterbi algorithm, this paper made some changes; that is, it is not necessary to choose the backtracking path of maximal probability. Instead, the path was selected based on the path probability. This kind of change can, to a certain degree, circumvent prediction errors caused by fewer monitoring points and increase the fault tolerance of the algorithm.

4.4 The structure of HMMLB

The HMMLB's structure shows in Fig. 3; the following terms are used:

Storage module: The module stores training data, three basic matrices of HMM and underlying links and data flow information. The training data are the data generated by running the sample load balancing algorithm.

HMM training module: As shown in Fig. 3, this module reads the training data from the storage module. It trains the HMM through supervised learning.

HMM application module: The module reads related information from the storage module. Finally, the results of the selected path are generated by the Viterbi algorithm, and the selected path result and data flow information are transferred to the flow table loader.

Applications of controller: Controller applications include the elephant flow detector, the link observer and the flow table loader. The elephant flow detector captures large flows and stores large flow information in the storage module. The specific workflow is that it collects the flow information from the first switch that each data flow passes through. The link observer monitors the link's bandwidth condition. The flow table loader receives the selected path and flow information from the application module and deploys it by delivering a flow table to the switches on the path.

Distribut information collection: As shown in the Fig. 3, it is an SDN architecture. The controller is able to monitor the switch and link by OpenFlow protocol. To be specific, controller will recognize the elephant flow while it enters the network. And then Link Observer will be activated, which could gain the monitored switch ports' current flow status. Finally, the link usage will be calculated by its corresponding port's flow status.

Controller synchronization mechanism: The controller will input the monitored link usage information into the Storage Module. HMM will get the information from Storage Module and use it to make path selection decisions. After the decisions were made, it will be synchronized with Flow Table Loader by controller's Representational State Transfer (REST) Application Programming Interface (API). The Flow Table Loader converts the path decision into OpenFlow Table and sends it to the switches through the OpenFlow protocol.

4.5 Link failure

This paper uses the keep-alive mechanism to monitor link status in the topology. Link Observer is used to monitoring the link status. Link Observer gets the switch ports information to determine if there is a link failure every 10 seconds. To cope with link failure, the 0 and 4 quantized states were set to tackle two situations of link failure.

The first link failure situation: there is a failure in the HMM monitored link. HMM will set the link quantized state to 4, that is, HMM will recognize the link as a failure link. The controller will send OpenFlow tables to the corresponding switches and guide the flows, which are in the failure link to the parallel link. When the next elephant flow comes to the network, the HMM will remove the failure link from the backup link set and choose the parallel link for the elephant flow. For example, in the Fig. 2, HMM selects monitoring a2, b2, b4, c2 and c3 as the path selection reference. The keep-alive mechanism will monitor all link status every 10 seconds. If the link a2 fails, HMM will set a2 quantized state to 4. The controller will send OpenFlow tables to the corresponding switches and guide the flows to a1. If a new elephant flow enters the network, since the state of a2 is 4, HMM will not select a2 as the transmission path but choose the parallel link a1 as the path. When the link returns to normal, the quantized state value of a2 is set again according to the link usage.

The second link failure situation: there is a failure in the ordinary link, which is not monitored by HMM. HMM will set the quantized state of the parallel link, which is monitored by HMM to 0. The controller will send OpenFlow tables to the corresponding switches and guide the flows, which are in the failure link to the parallel link. When the next elephant flow enters the network, to avoid the failure link, HMM will unconditionally select the link, which is tagged 0 as the part of the transmission path. For example, in the Fig. 2, HMM selects monitoring a2, b2, b4, c2 and c3 as the path selection reference. If the link a1 fails, HMM will set a2 quantized state to 0. The controller will send OpenFlow tables to the corresponding switches and guide the flows to a2. If a new elephant flow enters the network, since the state of a2 is 0, HMM will unconditionally select the link a2 as the part of the transmission path. When the link returns to normal, the quantized state value of a2 is set again according to the link usage.

5 EXPERIMENTS AND EVALUATION

In this section, Hedera is selected as the comparison of HMMLB, because Hedera is a classical algorithm in centralized load balancing research. Hedera uses ECMP to choose a path for a mice flow and uses a specific path selection strategy for an elephant flow. HMMLB follows the same idea, using ECMP for a mice flow and using the HMM to select a path for an elephant flow. This section shows that HMMLB and Hedera make the same mice flow path selection, but HMMLB shows better time performance than Hedera in an elephant flow's path selection without reducing the throughput rate of the network.

This paper designs simulation experiments to verify the performance of HMMLB. Two different selection strategies were chosen, First Fit and Greedy, as Hedera's path selection strategy. Two topologies, fat-tree and VL2, were used as experimental topologies.

Hedera first deployed and ran on a specific experimental topology. Then the hosts in the topology were divided into pairs of communication groups and generated random flows to be transmitted between communication groups. The relevant training data in the experiment process was recorded, including monitored links' bandwidths and the result of the selected path for each data flow. At the end of the experiment, performance data was recorded which includes the actual bisection bandwidth of the topology and the time spent by Hedera to distribute the elephant flow to measure its load balancing and time cost performances. HMMLB used data from Hedera to train itself. HMMLB was then deployed on the same network topology. The same data flows as Hedera was used to transfer between communication groups. At the end of the experiment, the performance data was recorded. Mininet [29] was used as a topology simulator, and Ryu [30] was used as an SDN [31] controller.

5.1 Experiment environment in fat-tree

Our simulation experiments were done in an SDN environment. Our test bed consisted of 16 hosts interconnected using a fat-tree of 20 four-port gigabit Ethernet switches, as shown in Fig. 4. Hedera (ECMP for mice flow, First Fit or Greedy for elephant flow) and ECMP (ECMP for both mice flow and elephant flow) were used as HMMLB's (ECMP for mice flow, HMM for elephant flow) comparison. First Fit selected the first matching feasible path, and Greedy traversed all paths to choose the feasible path with the largest remaining

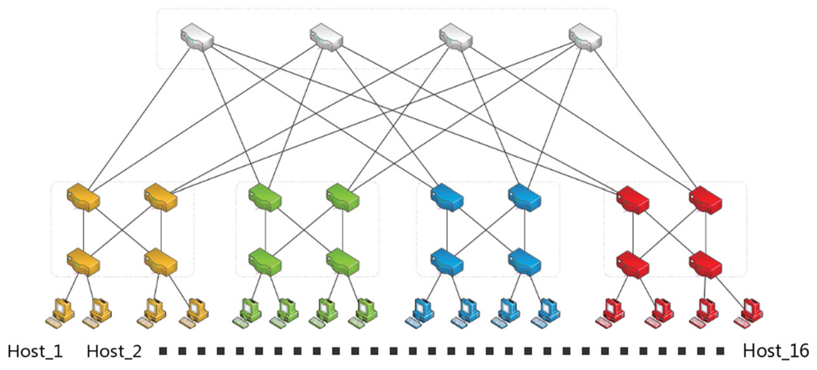

FIG. 4. Fat-tree topology.

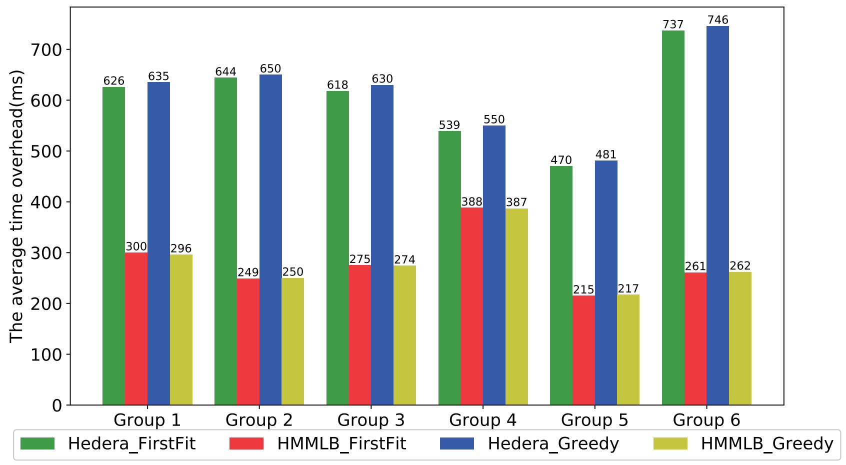

FIG. 5. Average time of each elephant flow overhead in fat tree.

amount of path bandwidth. The performance parameters for comparison are time performance and bisection bandwidth performance.

Communication group: host_x and host_y formed a communication group, $y = (x + 8) \bmod 16$. For example, host_1 and host_9 formed a communication group, so there were eight communication groups in the above topology.

Random-flow generation rules: A set of random data flows were generated for each host for transmission. To make the topology close to a full load, each random data flow was 20% elephant flow, whose transmission rate was 100 to 150 Mbps, and 80% mice flow, whose transmission rate was 0 to 100 Mbps. The total bandwidth occupied by data flows generated by a single host was as close as possible to 1 Gbit/s.

HMM training: Hedera used First Fit and Greedy for elephant flow selection strategies to deploy in the fat-tree. Training data was recorded and inputted to HMMLB's training module. According to Algorithm 1 presented in Section 4.2, the HMM's three core matrices were obtained, EmitProbMatrix, TransProbMatrix and InitStatus.

Because the topology of each communication group was the same, one of the communication groups was selected as a description of the monitored link selection in the fat-tree topology, as shown in Fig. 2. By the monitored link selection rule proposed in Section 4, the links a2, b2, b4, c2, c3 and d2 were chosen as the monitored links. Also, because there were common monitored links in the eight communication groups, in this experimental topology, HMMLB was required to monitor a total of 12 links, and Hedera was needed to monitor the bandwidth of all 32 links in the topology.

5.2 Time overhead in fat-tree

In terms of time performance, no comparison of ECMP was made because ECMP did not need to monitor the links, so there was no extra time consumption and no comparability. Six sets of experiments were set up. The experimental results were shown in Fig. 5. The HMMLB_FirstFit and HMMLB_Greedy in the figure meant that using Hedera_FirstFit or Hedera_Greedy in the fat-tree for training HMM. After training, HMMLB uses HMM to generate the path for a flow. The ordinate is the average time, which was made up largely of monitoring time for each flow. It can be seen that HMMLB had a clear advantage over Hedera regarding time performance, whether Hedera was based on Greedy or First Fit. The main reason for this performance was that HMMLB had fewer monitoring links than Hedera.

According to the objective function of measuring time Equation (2), it can be concluded that Hedera was required to monitor all 32 links regardless of the First Fit or Greedy strategy, so there is an equation:

$$T_{H\_FirstFit} = T_{H\_Greedy} = \frac{\sum_{1}^{32} t_j}{\sum_{1}^{32} t_i} = 1.$$

In the six sets of experimental data, the average time spent by Hedera_FirstFit assigned to each flow was 605.7 ms, and the average time spent by HMMLB_FirstFit assigned to each flow was 281.3 ms. HMMLB was required to monitor only 12 links in the fat-tree topology, so there is an equation:

$$T_{HMM\_FirstFit} = \frac{\sum_{1}^{12} t_j}{\sum_{1}^{32} t_i} = \frac{281.3(\text{ms})}{605.7(ms)} \approx 0.464.$$

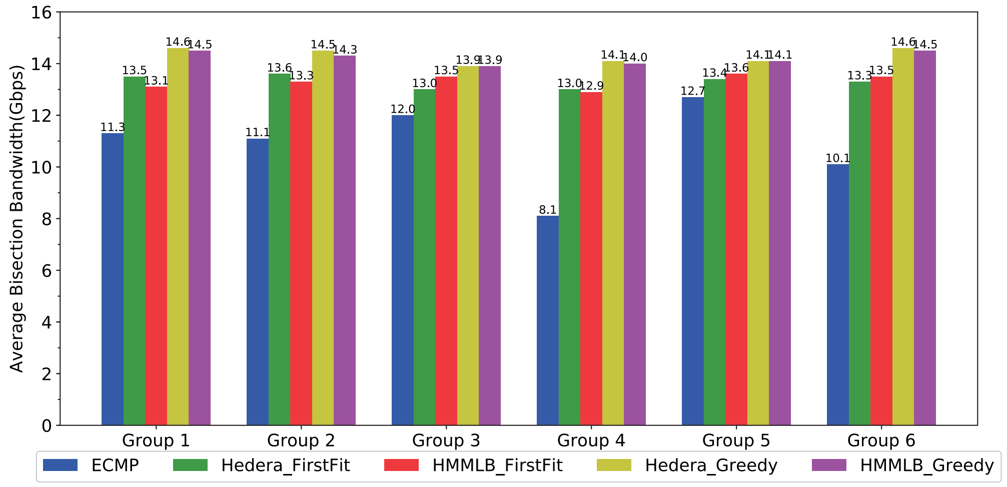

FIG. 6. Comparison of average bisection bandwidth in the fat tree.

Hedera_Greedy averaged 615.3 ms for each flow, and HMMLB_Greedy averaged 281.0 ms for each flow:

$$T_{HMM\_Greedy} = \frac{\sum_{1}^{12} t_j}{\sum_{1}^{32} t_i} = \frac{281.0(\text{ms})}{615.3(ms)} \approx 0.457.$$

Two conclusions can be drawn through data analysis. First, because the values of $T_{HMM\_FirstFit}$ and T_{HMM_Greedy} were both approximate 0.5, when using the same strategy in the fat-tree topology, HMMLB can decrease the time spent by nearly half compared to Hedera. Second, HMMLB's time consumption is not related to the chosen strategy. We found that the average time of each flow that HMMLB_FirstFit and HMMLB_Greedy allocated was approximate the same, but there was $T_{HMM\_FirstFit} > T_{HMM\_Greedy}$, indicating that HMMLB's time consumption was roughly the same, whereas Hedera's adoption of the Greedy strategy resulted in higher time consumption than using First Fit under different path selection strategies. The explanation for this phenomenon is that no matter which strategy was adopted as the training object, HMMLB always converted it to the HMM model and used the Viterbi algorithm to solve the feasible path. In Hedera, the Greedy strategy required the traversal of all feasible path bandwidths to find the optimal feasible link, which consumed more time than the First Fit strategy.

5.3 Performance of bisection bandwidth in fat-tree

Figure 6 showed the actual bisection bandwidth level in the fat-tree topology that ECMP, Hedera and HMMLB achieved. It was clear that both Hedera and HMMLB were superior to ECMP in the bisecting bandwidth level, regardless of whether the First Fit or Greedy strategy was used in the fat-tree topology. From the six sets of experimental data, the average bisection bandwidth level is calculated: ECMP was 10.88 Gbps, the Hedera_FirstFit was 13.30 Gbps, the HMMLB_FirstFit was 13.31 Gbps, the Hedera_Greedy was 14.30 Gbps and the HMMLB_Greedy was 14.21 Gbps. According to the objective function of network throughput Equation (1), the theoretical bisection bandwidth of the experimental topology was 16.00 Gbps, so for ECMP:

$$P_E = \frac{10.88(Gpbs)}{16.00(Gpbs)} = 0.680;$$

for Hedera_FirstFit:

$$P_{H\_FirstFit} = \frac{13.30(Gpbs)}{16.00(Gpbs)} \approx 0.831;$$

for HMMLB_FirstFit:

$$P_{HMM\_FirstFit} = \frac{13.31(Gpbs)}{16.00(Gpbs)} \approx 0.832;$$

for Hedera_Greedy:

$$P_{H\_Greedy} = \frac{14.30(Gpbs)}{16.00(Gpbs)} \approx 0.894;$$

and for HMMLB_Greedy:

$$P_{HMM\_Greedy} = \frac{14.21(Gpbs)}{16.00(Gpbs)} \approx 0.888.$$

The above data showed that using the same strategy, Hedera and HMMLB can achieve similar bisection bandwidth levels.

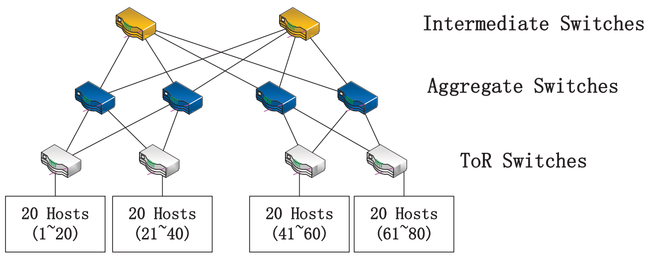

FIG. 7. VL2 topology.

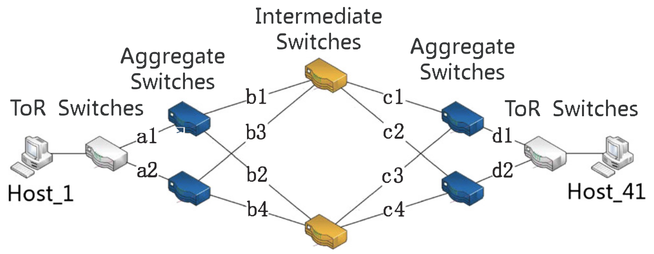

FIG. 8. Example of a communication group in VL2.

5.4 Experiment environment in VL2

In addition to experimenting under the fat-tree topology, HMMLB was deployed on the VL2 topology to demonstrate that it can operate effectively in different topologies. The VL2 topology had three layers. From the upper layer to the lower layer were intermediate switches, aggregate switches, and top-of-rack (ToR) switches. Twenty hosts were connected to each ToR switch. The VL2 topology used in our experiment is shown in Fig. 7. Intermediate switches and aggregate switches were connected through 1-Gbps ports, ToR switches and aggregate switches were connected through 1-Gbps ports and each ToR was connected to its 20 hosts through 100-Mbps ports. The theoretical bisection bandwidth of this VL2 topology was 8 Gbps.

Communication groups: Host_x and Host_y formed a communication group, $y = (x + 40) \bmod 80$. For example, Host_1 and Host_41 formed a communication group, so in the above topology, there were 40 communication groups.

Random-flow generation rules: A random flow of data was generated for a host. To make the topology close to a full load, each random data flow was 20% elephant flow, whose transmission rate was from 10 Mbps to 15 Mbps, and 80% mice flow, whose transmission rate was from 0 to 10 Mbps. The total data flow rate generated for a single host was as close as possible to 100 Mbps.

HMM training: As in the fat-tree topology experiment, Hedera deployed First Fit and Greedy for elephant flow selection strategies in the VL2 topology. The training data was recorded and inputted to HMMLB's training module. Because the topology of each communication group in the VL2 topology was the same, one of the communication groups was selected as the description of the selection of the monitored links in the VL2 topology, as shown in Fig. 8. By the monitored links selection rule proposed in Section 4, a2, b2, b4, c2, c4 and d2 were chosen as the monitored links. Also, because there were common monitored links among the 40 communication groups, HMMLB was required to monitor 8 links in the VL2 topology, and Hedera was needed to monitor the bandwidth of all 16 links in the topology.

5.5 Time overhead in VL2

To compare time performances, there were six groups of experiments. The results of the experiments were shown in Fig. 9. It can be seen that in the VL2 topology, HMMLB had apparent advantages over Hedera in terms of time performance. The main reason for this result is that HMMLB had fewer monitoring links than Hedera had.

According to the objective function of measuring time Equation (2), it can be concluded that Hedera was required to monitor all 16 links regardless of the First Fit or Greedy strategy, so there is an equation:

$$T_{H\_FirstFit} = T_{H\_Greedy} = \frac{\sum_{1}^{16} t_j}{\sum_{1}^{16} t_i} = 1.$$

In the six sets of experimental data, the average time spent by Hedera_FirstFit assigned to each flow was 503.5 ms, and the average time spent by HMMLB_FirstFit assigned to each flow was 219.2 ms. HMMLB was required to monitor only 12 links in the fat-tree topology, so there is an equation:

$$T_{HMM\_FirstFit} = \frac{\sum_{1}^{8} t_j}{\sum_{1}^{16} t_i} = \frac{219.2}{503.5} \approx 0.435.$$

Hedera_Greedy averaged 518.2 ms for each flow, and HMMLB_Greedy averaged 215.8 ms for each flow:

$$T_{HMM\_Greedy} = \frac{\sum_{1}^{8} t_j}{\sum_{1}^{16} t_i} = \frac{215.8}{518.2} \approx 0.416.$$

This paper drew two conclusions from the data analysis. First, because the values of $T_{HMM\_FirstFit}$ and T_{HMM_Greedy} were both less than 0.5, when using the same selection strategy in the VL2 topology HMMLB took half the time of Hedera. Second, HMMLB's time consumption was not related to the chosen strategy, which was the same as in the fat-tree topology. It showed that the average time of each flow

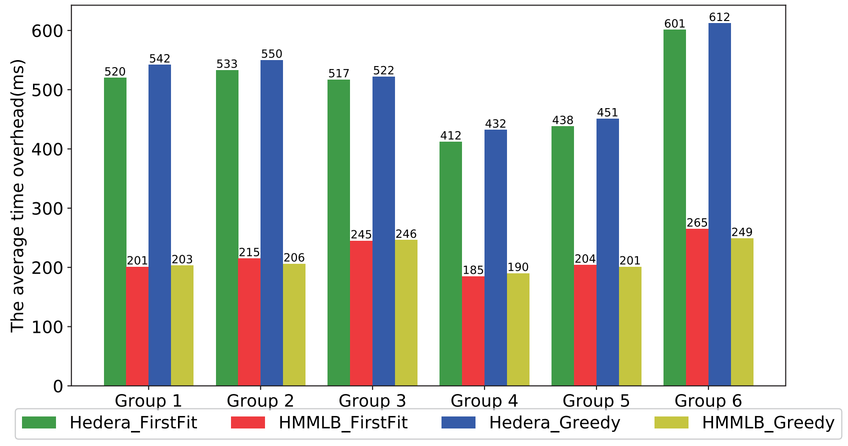

FIG. 9. Average time of each elephant overhead in VL2.

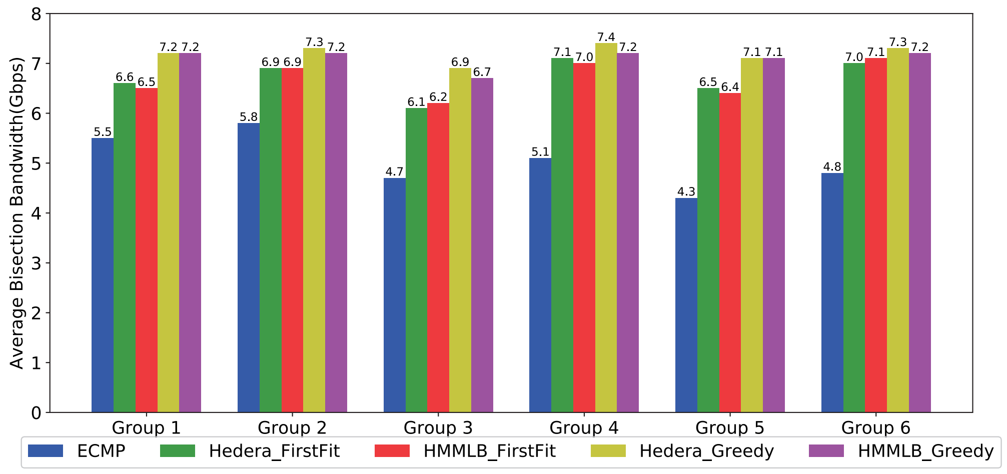

FIG. 10. Comparison of average bisection bandwidth in VL2.

that HMMLB_FirstFit and HMMLB_Greedy allocated was approximate the same as that under the VL2 topology, but there was $T_{HMM\_FirstFit} > T_{HMM\_Greedy}$, indicating that under different strategies, HMMLB's time consumption was approximately the same. Hedera's adoption of the Greedy selection strategy, however, resulted in greater time consumption than using First Fit.

5.6 Performance of bisection bandwidth in VL2

We recorded the actual bisection bandwidth level in the VL2 topology that ECMP, Hedera and HMMLB achieved (Fig. 10). It was clear that both Hedera and HMMLB were superior to ECMP in the bisecting bandwidth level, regardless of whether the First Fit or Greedy strategy was used in the fat-tree topology. From the six sets of experimental data, the average bisection bandwidth level was calculated which reached by ECMP was 5.03 Gbps, by Hedera_FirstFit was 6.70 Gbps, by HMMLB_FirstFit was 6.68 Gbps, by Hedera_Greedy was 7.2 Gbps and by HMMLB_Greedy was 7.1 Gbps.

According to the objective function of network throughput Equation (1). The theoretical bisection bandwidth of the experimental topology was 8.00 Gbps, so for ECMP:

$$P_E = \frac{5.03\ (Gpbs)}{8.00\ (Gpbs)} = 0.629;$$

for Hedera_FirstFit:

$$P_{H\_FirstFit} = \frac{6.70\ (Gpbs)}{8.00\ (Gpbs)} \approx 0.838;$$

for HMMLB_FirstFit:

$$P_{HMM\_FirstFit} = \frac{6.68(Gpbs)}{8.00(Gpbs)} = 0.835;$$

for Hedera_Greedy:

$$P_{H\_Greedy} = \frac{7.20\ (Gpbs)}{8.00\ (Gpbs)} = 0.900;$$

and for HMMLB_Greedy:

$$P_{HMM\_Greedy} = \frac{7.10\ (Gpbs)}{8.00\ (Gpbs)} \approx 0.888.$$

6 CONCLUSION

This paper introduces HMMLB, which uses the HMM to select paths for data flows in data center. Proposed work transforms the path selecting problem into an HMM problem and uses the HMM to re-form the path selecting model of the traditional centralized load balancing algorithm. HMMLB is compared with the traditional centralized load balancing algorithm, which can only be used under the condition of monitoring some of the links in the topology. Both algorithms were run in data centers with similar throughputs. Therefore, it can be said that HMMLB effectively reduces the time cost of a centralized load balancing algorithm and ensures that the data center throughput rate did not decrease. In addition, in the simulation experiment, this paper choose different path selecting strategies to train HMMLB and deploys it to different data center topologies. The experimental results show that HMMLB has excellent performance in different data center topologies, demonstrating its adaptability.

Acknowledgements

This work is supported in part by the CSC Scholarship 201806655007, in part by the Major Scientific Project of Zhejiang Lab under Grant 2018FD0ZX01, in part by the Natural Science Foundation of Fujian Province under Grant 2018J01796 and 2018Y0029.

FUNDING

Natural Science Foundation of Fujian Province, China.

REFERENCES

[1] Dean, J. and Ghemawat, S. (2008) Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51, 107–113.

[2] Alfares, M., Loukissas, A. and Vahdat, A. (2008) A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.*, 38, 63–74.

[3] Greenberg, A., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P. and Sengupta, S. (2009) Vl2: a scalable and flexible data center network. *Commun. ACM*, 54, 95–104.

[4] Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y. and Lu, S. (2008) Dcell: a scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM Comput. Commun. Rev.*, 38, 75–86.

[5] Zhang, J., Yu, F.R., Wang, S., Huang, T., Liu, Z. and Liu, Y. (2018) Load balancing in data center networks: a survey. *IEEE Commun. Surv. Tut.*, 20, 2324–2352.

[6] Kandula, S. and Greenberg, A. (2009) The Nature of Data Center Traffic: Measurements and Analysis. In *2009 Internet Measurement Conference*, Chicago, Illinois, USA, November 4–6, pp. 202–208. Association for Computing Machinery, Inc., New York.

[7] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N. and Vahdat, A. (2010) Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. 7th USENIX Conf. Networked Systems Design and Implementation*, San Jose, CA, USA, April 28–30, pp. 19–19. USENIX Association, Berkeley, CA, USA.

[8] Curtis, A.R., Kim, W. and Yalagandula, P. (2011) Mahout: Low-Overhead Datacenter Traffic Management using End-Host-based Elephant Detection. In *2011 Proc. IEEE INFOCOM*, Shanghai, China, April 10–15, pp. 1629–1637. IEEE Computer Society, Los Alamitos, CA, USA.

[9] Meyer, I.M. (2004) *Hidden Markov Model (HMM)*. John Wiley & Sons, Inc., Hoboken, New Jersey.

[10] Benson, T., Anand, A., Akella, A. and Zhang, M. (2009) Understanding Data Center Traffic Characteristics. In *Proc. 1st ACM Workshop on Research on Enterprise Networking*, Barcelona, Spain, August 21, pp. 65–72. ACM, New York, NY, USA.

[11] Liu, C.H., Kind, A. and Vasilakos, A.V. (2013) Sketching the data center network traffic. *IEEE Netw.*, 27, 33–39.

[12] Dzida, M., Zagozdzon, M., Pioro, M. and Tomaszewski, A. (2006) Optimization of the Shortest-Path Routing with Equal-Cost Multi-Path Load Balancing. In *2006 Int. Conf. Transparent Optical Networks*, Nottingham, UK, June 18–22, pp. 9–12. IEEE Computer Society, Los Alamitos, CA, USA.

[13] Hopps, C. (2000) *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC Editor, USA.

[14] Perry, J., Ousterhout, A., Balakrishnan, H., Shah, D. and Fugal, H. (2014) Fastpass: a centralized "zero-queue" datacenter network. *ACM SIGCOMM Comput. Commun. Rev.*, 44, 307–318.

[15] Wang, S., Zhang, J., Huang, T., Pan, T., Liu, J. and Liu, Y. (2016) FDALB: flow distribution aware load balancing for datacenter networks. In *2016 IEEE/ACM 24th Int. Symposium on Quality of Service (IWQoS)*, Beijing, China, June20–21, pp. 1–2. IEEE Computer Society, Los Alamitos, CA, USA.

[16] Wang, W., Sun, Y., Zheng, K., Kaafar, M. A., Li, D. and Li, Z. (2014) Freeway: adaptively isolating the elephant and mice flows on different transmission paths. In *2014 IEEE 22nd Int. Conf. Network Protocols*, Raleigh, NC, USA, October 21–24, pp. 362–367. IEEE Computer Society, Los Alamitos, CA, USA.

[17] Zhang, P., Xie, K., Kou, C., Huang, X., Wang, A. and Sun, Q. (2019) A practical traffic control scheme with load balancing based on PCE architecture. *IEEE Access*, 7, 30648–30658.

[18] Yin, P., Yang, S., Xu, J., Dai, J. and Lin, B. (2019) Efficient traffic load-balancing via incremental expansion of routing choices. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 4, 1–35.

[19] Akbar Neghabi, A., Jafari Navimipour, N., Hosseinzadeh, M. and Rezaee, A. (2019) Nature-inspired meta-heuristic algorithms for solving the load balancing problem in the software-defined network. *Int. J. Commun. Syst.*, 32, e3875.

[20] Memon, S., Huang, J., Saajid, H., Khuda Bux, N., Saleem, A. and Aljeroudi, Y. (2019) Novel multi-level dynamic traffic load-balancing protocol for data center. *Symmetry*, 11, 145.

[21] Wang, P., Trimponias, G., Xu, H. and Geng, Y. (2019) Luopan: sampling-based load balancing in data center networks. *IEEE Trans. Parallel Distr. Syst.*, 30, 133–145.

[22] He, J., Brandtpearce, M. and Subramaniam, S. (2011) Analysis of blocking probability for first-fit wavelength assignment in transmission-impaired optical networks. *IEEE/OSA J. Optical Commun. Netw.*, 3, 411–425.

[23] Fan, Z., Shen, H., Wu, Y. and Li, Y. (2013) Simulated-annealing load balancing for resource allocation in cloud environments. In *Proc. 2013 Int. Conf. Parallel and Distributed Computing, Applications and Technologies*, Washington, DC, USA, December 16–18, pp. 1–6. IEEE Computer Society, Los Alamitos, CA, USA.

[24] Surhone, L.M., Tennoe, M.T. and Henssonow, S.F. and Berformannce (2010) *Bisection Bandwidth*. Betascript Publishing, Riga, Latvia.

[25] Winstein, K. and Balakrishnan, H. (2013) TCP ex machina: computer-generated congestion control. *ACM SIGCOMM Comput. Commun. Rev.*, 43, 123–134.

[26] Beal, M.J., Ghahramani, Z. and Rasmussen, C.E. (2002) The infinite hidden Markov model. *Adv. Neural Inf Process Syst.*, 14, 577–584.

[27] Paszek, E. (2005) Maximum likelihood estimation (MLE). *BMS Bull. Sociol. Methodol.*, 17, 147–147.

[28] Lou, H.L. (1995) Implementing the Viterbi algorithm. *IEEE Signal Process Mag.*, 12, 42–52.

[29] Kaur, K., Singh, J. and Ghumman, N.S. (2014) Mininet as software defined networking testing platform. In *Int. Conf. Communication, Computing & Systems (ICCCS)*, Chennai, India, February20–21, pp. 139–42. IEEE Computer Society, Los Alamitos, CA, USA.

[30] Kubo, R., Fujita, T., Agawa, Y. and Suzuki, H. (2014) Ryu SDN framework-open-source SDN platform software. *NTT Technical Rev.*, 12, 1–5.

[31] Zhang, C.-K., Cui, Y., Tang, H.-Y. and Wu, J.-P. (2015) State-of-the-art survey on software-defined networking (SDN). *J. Softw.*, 26, 62–81.