Multi-Threshold SDN Controllers Load Balancing Algorithm Based On Controller Load

Jia-qi LI$^{1,2,3}$, En-chang SUN$^{1,2,3,*}$ and Yan-hua ZHANG$^{1,2,3}$

$^1$Beijing Advanced Innovation Center for Future Internet Technology, Beijing University of Technology, Beijing, 100124, China

$^2$Beijing Laboratory of Advanced Information Networks, Beijing, 100124, China

$^3$Faculty of Information Technology, Beijing University of Technology, Beijing, 100124, China

*Corresponding author

Keywords: SDN, Control plane, Multi-thresholds, Load balance, Switch migration.

Abstract. SDN (Software-Defined Networking) centralized control features make it suffer from performance bottlenecks and therefore need to be overcome using a distributed control plane. However, the imbalanced load among multiple controllers will also lead to a decrease in overall system performance. In order to solve the conflict between inefficient nodes and overloaded nodes, a more efficient and flexible load balancing algorithm is needed to ensure the efficient operation of the SDN network. The algorithm firstly accurately selects the controller load description parameters, then uses multiple thresholds to distinguish the controller status, and at the same time more effectively selects the switch to migrate to the idle controller. The algorithm determines the state of the controller more precisely while trying to make each controller work in the normal mode, to avoid the controller in a long period of high load mode. The experimental results show that the algorithm can effectively improve the load balancing degree and throughput recovery speed of SDN control plane.

Introduction

Since 2006, Stanford University has proposed Software-Defined Network (SDN) technology to deal with the explosive growth of wireless communications and traditional networks which are difficult to make technological breakthroughs. The SDN technology separates the control functions such as routing decisions in the router from the equipment, and make them controlled by the software of the central controller, achieving the separation of control and forwarding [1]. At the same time, the controller also has the functions of topology sensing and routing decision [2], which makes the equipment simpler and easier to control.

SDN (Software-Defined Network) centralized control functions are plagued by performance bottlenecks and therefore need to be overcome using distributed control planes. However, the unbalanced load among multiple controllers will also lead to a decrease in overall system performance. In order to solve the problem of node inefficiency, the only way is to design an efficient and flexible load balancing algorithm to implement the remapping of controllers and switches, scalability and reliability ensure that SDN runs large complex networks efficiently.

Research Status

For the problem of load imbalance among multiple controllers, the researchers proposed several frameworks to deal with.

DIFANE$^{[3]}$ aims to improve the performance of the control plane by reducing the communication between the switch and the controller and reducing the controller load. DIFANE reduces the load of the controller by keeping the traffic at the data level as much as possible to improve the performance of the control plane. However, the structure of the authoritative switch in the DIFANE system is relatively complex, which is not conducive to the deployment of large-scale networks.
Devo Flow\(^4\) reduces the number of switch requests to reduce the communication between the data plane and the control plane, in order to improve the performance of the controller. The Devo Flow system needs to modify the flow table structure of the switch and the hardware structure of the switch. At this stage, it has not been actually deployed and is not conducive to the interaction of heterogeneous networks.

Eugen\(^5\) proposed a multi-controller deployment optimization solution, focusing on factors such as controller load balancing and inter-controller delay. It weighed several conflicting optimization goals, but did not require the number and location of migration switches. It is easy to cause multiple switch migration conflicts.

The literature\(^6\) proposes the ASIC. ASIC puts forward a concept of “load balancing layer” and solves the above problems through technologies such as data sharing, load balancing, and parallel processing.

The above algorithm has opened up ideas for related research on load balancing in the control plane, and has also aroused the interest of scholars in this area. The DALB algorithms\(^7\) proposed by ZHOU and YU\(^8\) proposed load balancing algorithms based on multiple distributed controller load notification policies, all reducing the delay caused by network transmission. But in these algorithms, switch migration will be triggered after the threshold has been reached in the controller load. And the switch with the largest load will be directly migrated. This can easily overload the new controller. The literature\(^9\)[10] are all based on the single-layer controller load balancing scheme. The literature\(^9\) mainly focuses on the scalability of the control plane. The literature\(^10\) uses the expansion server to dynamically distribute the traffic. Although the above two methods can mitigating controller load, but they cannot effectively solve the load balancing problem between the various controllers. So they are difficult to meet the needs of large networks. The literature\(^11\) proposed a load balancing algorithm based on the system balance degree to allow the controller with higher performance to distribute more loads. However, this algorithm chooses to use the round trip time as the main parameter when migrating the switch. Therefore, it is easy to create multiple controllers overload at the same time. The goals of literature\(^12\) and\(^13\) are to plan the optimal path for outbound traffic. However, when the load is balanced, there are few paths to choose, and the granularity of the flow division is too coarse to achieve optimal results.

The algorithm proposed by Dixit et al.\(^14\) increases or decreases the number of controllers according to the load status of the control plane. However, there is no specific description of the evaluation of the load status of the controller and the selection of the target controller. Correspondingly, Yazici et al.\(^15\) also proposed an algorithm about dynamically adding or removing controllers and a symbolic description of controller and switch mapping models. This model is not based on the Open Flow role mechanism, but based on IP aliases. In the event of a controller failure, the transfer will make the configuration of the physical network more complicated. Literature\(^16\) also proposes a flexible method that can dynamically change the number of controllers and their locations under different network conditions, thus realizing the redeployment of controllers and switches. However, this method does not explain the factors that need to be considered for migration. Literature \(^17\) proposed a Load Balancing Algorithm (LBA) with a minimum utilization rate migration strategy, which solves the dynamic migration of switches and controllers. The core idea is to select the node with the lowest controller usage rate as the receiving node during each switch migration. This method is simple to implement and the complexity of the algorithm is low, but it does not consider the overload of the target node and the cost of each migration.

From the above studies, it can be seen that the existing controller load balancing strategy is mainly based on switch migration. Although the current research on dynamic load balancing algorithms has made some progress. However, due to the different perspectives and optimization goals, each has its own applicable conditions and shortcomings. For example, some algorithms can achieve good results when the load is light, but they do not perform well when the load is heavy. In addition, most of the algorithms mainly study how to obtain a better load balancing effect and ignore the impact of system cost and other factors on the system. In other algorithms, switch migration objects are rigidly selected.
Migration strategy calculations require a large amount of computer memory and lots of time. They do not give much consideration to the issue of multi-switch migration conflicts.

In view of the above issues, this paper comprehensively considers the selection strategies of switch and controller, and proposes an optimized load migration algorithm. The algorithm uses the resource utilization of nodes, the performance of controller nodes and some other indicators to assign tasks. This algorithm can achieve a better load balancing effect while minimizing the cost of system migration and improving the overall performance of the system.

Multi-Threshold Control Plane Load Balancing Algorithm

Definition and Calculation of Load

There are many ways to collect and calculate the load status of controllers, but the controller's working ability and real-time load status must be comprehensively considered. There are generally two ways to collect the load status of the SDN control plane controllers: centralized load collection algorithm and distributed load collection algorithm. The centralized load collection algorithm uses a load balancer to collect the load information of the controller in the network. However, as the sole carrier and regulator of network load balancing, the Load balancer performance will be insufficient. Therefore, centralized load collection algorithms are generally applicable to smaller networks. In the distributed load collection algorithm, all controller nodes can be used as a load balancer, which can collect and calculate their respective loads. Compared with the centralized load collection algorithm, this algorithm is more applicable, and it can effectively collect the load information of the controller no matter how large the network is. Therefore, this paper will use a distributed load collection algorithm, each controller is responsible for collecting its own load information (CPU usage, memory usage, etc.).

The description of the controller load is usually measured in two ways: switch input metrics and performance metrics. The input metrics of the switch generally use parameters such as the number of switches and the average message arrival rate of the switch to describe the load level of the controller. However, the use of switch input metrics to describe the controller load is only applicable to switches and controllers in the network are approximately same. Performance metrics generally use controller current CPU usage, memory usage, and other parameters to describe controller load conditions. The performance metric can accurately describe the load status of the controller. Therefore, this paper uses performance metrics as the load description of the controllers.

For parameter selection, this paper selects the controller's CPU usage and memory usage as the load description of the controller. Some articles select the controller connection request number or controller bandwidth usage and other parameters as the load description of the controller, but these parameters will eventually affect the CPU and memory usage rate. Therefore, the usage rate of CPU and memory of the controllers are selected to describe the load status of the controllers intuitively and accurately. Each controller is equipped with a module for obtaining and sharing its own load information.

Set the number of controllers in the control plane is i (i=0,1,...,n), the load of the controller i in the control plane is L(i), the CPU usage rate of the controller is L_C(i), and the controller memory usage rate is L_M(i)., then

\[ L(i) = r_1 L_C(i) + r_2 L_M(i) \]  

(1)

r_1 and r_2 are weights, which are used to balance the influence of different parts on the load. r_1+r_2=1. In general, r_1 and r_2 need to be manually set according to the situation of the controller.

Controller State Determination

When assessing the load status of a node in the controller cluster, different evaluation criteria will produce different load status results. Therefore, objective and accurate load status assessment is an
important basis for load balancing algorithms. Judging the state of the controller based on the
threshold method is the most used classic method. However, this method has two major drawbacks.
One is that the current threshold needs to be artificially set. Second, only one threshold cannot
accurately distinguish the various loading status of the controllers.

For this situation, this paper proposes an algorithm based on multi-threshold to determine the load
status of the controllers better. In this paper, the algorithm sets four thresholds. In a real environment,
the number of thresholds can be increased or decreased according to specific requirements to more
accurately distinguish the various states of the controllers.

First, set the highest threshold $T_{\text{Max}}(i)$ and the lowest threshold $T_{\text{Min}}(i)$. Then set the threshold
percentage parameter $P$ ($0 < P < 1$). In the actual environment, the two thresholds and the threshold
parameter $P$ depend on the average performance and working requirements of the entire control plane
and can be adjusted according to specific requirements. then,

$$T_0(i) = (1 - P) \times T_{\text{Max}}(i)$$  \hspace{1cm} (2)

$$T_{2r}(i) = P \times T_{\text{Max}}(i)$$  \hspace{1cm} (3)

There are different operations with the different situations of load $L(i)$,

- **Full Overload Mode**
  $$L(i) \geq T_0(i)$$

- **Overload Mode**
  $$T_{\text{Max}}(i) \leq L(i) < T_0(i)$$

- **High Load Mode**
  $$T_{2r}(i) < L(i) < T_{\text{Min}}(i)$$

- **Normal Mode**
  $$T_{\text{Min}}(i) < L(i) \leq T_{2r}(i)$$

- **Idle Mode**
  $$L(i) \leq T_{\text{Min}}(i)$$

The algorithm divides the load state of the controller into five states: full overload mode, overload
mode, high load mode, normal mode, and idle mode. A more detailed breakdown has been made of
the problems that have been mentioned above in the controller load balancing problem.

The full overload mode corresponds to the overload mode in the general algorithm. Once the load
status touches the threshold, it indicates that the current controller load has completely exceeded its
own working capacity. If the switch is not migrated, it will cause congestion, downtime and other
consequences. The switch migration operation needs to be triggered immediately.

Overload mode indicates that the controller is in abnormal working condition and is in overload
state, but it can still work for a period of time. If the controller is still in the overload state in the
second measurement, the switch is migrated.

The high load mode indicates that the current load of the controller is at the highest acceptable
point, and the controller can work normally under this condition. However, the controller is
overloaded and the loss is too fast in this case, which will also make the controller cope with sudden
drop in capability. IF controller work in this mode, the control plane should search a normal mode
controller. If so, switch migration is triggered to make the load of the control plane more balanced and
the work of the entire system more stable.

The normal mode indicates that the controller can work normally and still have the ability to handle
some unexpected situations. The controller's working condition is the most perfect. Controllers that
work in normal mode will receive a lower priority than idle mode controller in the choice of the target
controller. Normal mode is the ideal controller operating mode.
The idle mode controller's transaction is far from its processing power, and the controller still has the ability to "lead" more switches. The idle controller acts as the highest priority for receiving switches. Once a switch is migrated, it will migrate to the idle controller first.

If the load of two or more controllers simultaneously exceeds its load threshold, it will cause them to perform the migration operation at the same time. If they simultaneously migrate the switch to the same target controller, it may cause the target controller to be overloaded. Therefore, each time a switch migration is triggered, priority is given to processing the switch with the most overloaded controller. After the migration completed, the controller with the highest overload is selected again in the remaining overload controllers until the load is balanced.

\[
D(i) = \frac{UL(i) - T(i)}{T(i)}
\]  

(5)

D(i) is the overload ratio of the overload controller. The larger D(i) indicates that the overload of the controller C(i) is more severe and will obtain higher priority when the overload controller is selected.

**Selection of Migration Switch**

Selecting different switches for migration has a significant impact on load balancing. Choosing to migrate to a switch with a lower load on the controller can reduce the processing delay, but requires frequent migration; selecting a switch with a higher load on the controller for migration may cause the new host controller to be overloaded and big costly. In addition, the round-trip time for the switch to interact with the controller is also an important factor that affects the load. If the round trip time is long after the switch is migrated, the controller's ability to control the switch will be reduced, leading to more problems. Therefore, this paper selects switches for migration by taking the controller load and the interaction time between the switch and the controller as parameters.

If the switch occupy the controller load is \(LS(j)\), where \(j\) is the switch number (\(j=0,1,\ldots,m\)), then,

\[
GL(j) = \frac{|LS(j) - [L_C(i) + T(j)]|}{T(j)}
\]

(6)

GL(j) is the load score of the switch S(j) relative to the controller C(i), which is determined by the difference between the switch occupying the controller load and the controller exceeding the normal mode load part. The smaller the difference, the easier controller work in normal mode after the switch migrate.

Set the round trip time of the switch S(j) to the controller of it, then,

\[
G(j) = \alpha GL(j) + \beta R(j)
\]

(7)

G(j) is the score of the switch, where \(\alpha\) and \(\beta\) are adjustable coefficients. The coefficient should be set according to the topology of the controller and switch in the network. Each switch can obtain its own score G(j) according to the above algorithm. According to the G(j) value, the switches controlled by the overload controller are sorted in ascending order, and the switch with the smallest G(j) value is preferentially selected for migration. If the switch with the smallest G(j) value can reduce the overload controller load below the normal mode threshold, only the switch is selected for migration; if not, G(j) is calculated again for the remaining switch after the first switch is migrated to select the switch that needs to be migrated.

**Target Controller Selection**

After selecting the switch to be migrated, we need to select the target controller for migration. According to the contents mentioned in Section 2, the controller's load condition is used as one of the parameters for selecting the controller, and the controller in the idle mode is preferentially selected. In addition, the greater round trip time between the switch and the target controller, the lower efficiency of the migration. Therefore, when selecting the target controller, the target controller should be selected considering the load conditions of the controller and the round trip time for each switch to interact with the domain controller.
Controller load rating use the formula (1). The round-trip time of the selected switch $S(j)$ to the target controller $C(i)$ is $R_{i}(j)$, and the round-trip time to the original controller is $R(j)$. Then,

$$G_{R}(j) = \frac{R_{i}(j) - R(j)}{R(j)}$$

(8)

$GR(j)$ is the round-trip time score of the switch to be migrated to the target controller. Then,

$$G_{j}(i) = \gamma L(i) + \delta G_{R}(j)$$

(9)

Among them, $\gamma$ and $\delta$ are adjustable coefficients, setting according to different conditions of the network, and $G_{j}(i)$ is the overall score of the target controller for the switch to be migrated. After selecting the switches to be migrated, the selected target controllers are all calculated with the score $G_{j}(i)$ and the controllers are sorted in ascending order according to $G_{j}(i)$, and the controller with the lowest $G_{j}(i)$ value is preferentially selected for migration.

**Entire Description of the Algorithm**

The overall flow of the algorithm is as follows: Each controller collects its own load information according to the load information collection cycle $Z$ (Setting according to network load conditions), including CPU usage, memory usage, and the switch load information controlled by itself, including the switch occupies its own load, and the switch and its own round-trip time. Any controller will only keep the load information twice (for monitoring the controller in overload mode or high load mode). The controller determines which state it is in based on its own load $L(i)$. If multiple controllers are overloaded at the same time, the controller with the most severe overload degree is selected according to the overload ratio $D(i)$ to perform the switch migration operation. After triggering the switch migration operation, first select the switch with the lowest $G(j)$ among the switches of the controller and migrate to the controller with the lowest $G_{j}(i)$ score in the other controllers. After the migration is completed, if there are any remaining overload controllers, switch migration will continue until the load is balanced. After all migration operations are completed, wait for the next load information collection.

**Simulation Experiments and Analysis**

This article adopts Ryu controller to carry on the experimental test on the Mininet test platform. The Ryu controller is written in Python language and supports the OpenFlow protocol version 1.0-1.5. Ryu controllers are deployed on different physical devices. It is divided into controllers A, B, C, and D. The experimental machines of controllers A, C, and D are configured as Intel Core i5-4570 3.20GHz, RAM is 4GB, and the experimental machine of controller B is configured as Intel Core i7-3960 3.30GHz, RAM: 8GB.

Ryu[19] is an open source SDN controller project led by NTT (Nippon Telegraph & Telephone). Ryu means “streaming” in Japanese. Ryu is developed by the Python language and supports OpenFlow1.0-1.5.

Mininet[20] is a lightweight test platform for software-defined network. It supports various protocols such as OpenFlow, OpenvSwitch, etc. Mininet can also emulate a complete web host, link, and switch on the same computer and facilitate interactive development, test and demonstration.

The simulation experiment uses three modes respectively for testing: (1) Dynamic Adaptive Algorithm (DALB). This method periodically obtains controller load information and then migrates the switches on the overloaded controller to the lightest loaded controller. (2) Load balance algorithm based on the system balance degree (PALB). This method uses the round trip time of the exchange between the switch and the controller as the main parameter of the selected switch, and the controller selection is based on the system balance. (3) The algorithm flow in Section 3.5 to load balance the controller.
There are four controllers A, B, C, and D in this experiment (Figure 1), connected to switch 1-12 respectively. Controller A connects to switches 1 and 2, controller B connects to switch 3-7, controller C connects to switches 8 and 9, and controller D connects to switch 10-12. To simplify the experiment, all switches use homogeneous switches. Four controllers are connected in pairs and the switch is connected to each controller. In order to make the picture not too complicated, only the controllers of each switch and the management itself are connected in the topology diagram, and other connection lines that do not upload data are not drawn. The experimental topology is as the Figure 1.

![Figure 1. Experimental topology.](image1.png)

![Figure 2. Controller load of the proposed algorithm.](image2.png)

Set the load threshold of each controller as $T_{Max}=85\%$, $T_{Min}=30\%$, threshold auxiliary parameter $p=0.9$, load calculation weight $(r_1, r_2)=(0.5, 0.5)$, and the switch rating parameter to be migrated $(\alpha, \beta) = (0.8, 0.2)$, target controller scoring parameters $(\gamma, \delta) = (0.8, 0.2)$. The load determination period $Z = 10s$.

As the Figure 2 shows. The message request rate of each switch is 3000 at beginning. After acquiring the load information for three times, the request rate of the switch 3-7 message is changed to $(3000, 6000, 12000, 15000, 3000)$, and the controller B is in full overload mode, a switch of controller B is migrated to controller C. After the migration, increase the message request rate of switch 5 to 15000. Then controller B works in high load mode. The first time that controller B is detected to work in high load mode does not change, and in second phase, controller B is still in high load mode. Controller A in the system is found to be working in an idle state, which triggers switch migration and migrates another switch of controller B to controller A. The principle of selecting the switch twice to migrate follows the migration so that controller B can work in normal mode.

![Figure 3. Controller load of DALB.](image3.png)

![Figure 4. Controller load of PALB.](image4.png)

Comparing the algorithm proposed by this paper with the DALB algorithm (Figure 3) and PALB algorithm(Figure 4), we can see that the average load of the controller of this algorithm is lower, and the load of each controller is similar, and the load balancing effect is good; DALB algorithm makes the load of two controllers high, and controller C is in idle state, there is no reasonable use of resources. Because the DALB algorithm simply migrates the most heavily loaded switch on the overloaded controller to the lightest controller, without allowing the controller to maintain its normal
mode; Because of the simple method of selecting switches, the switch waiting to be migrated is selected based on the round trip time between the switch and the controller. PALB algorithm improves the load of controller C and does not effectively reduce the load of controller B. In summary, the algorithm of this paper has an excellent load balancing effect.

Figure 5. Throughput of the three algorithms.

According to the observation of the throughput of the three algorithms in Figure 5, the algorithm proposed in this paper is not only better than the other two algorithms in the final throughput, and the throughput recovery speed is faster, because the algorithm makes the controller work in normal mode as much as possible. The DALB algorithm has a large throughput jitter during the restoration process. This is because the load balancing effect of the algorithm is relatively general, and therefore, stable throughput cannot be obtained. The PALB is superior to the DALB algorithm and its growth rate is faster than that of the DALB, and the growth is relatively stable. This is because the PALB algorithm is based on the degree of system equalization in the selection of the controller and can better balance the controller load compared to the DALB algorithm. The algorithm proposed in this paper is superior to the above two algorithms in terms of both the final throughput and the speed of throughput growth.

Conclusion

To solve the problem of load imbalance in the SDN control plane, this paper proposes a more effective multi-threshold load balancing algorithm. The algorithm firstly accurately selects the controller load description parameters, then uses multiple thresholds to distinguish the controller status, and at the same time more effectively selects the switch to migrate to the idle controller. The algorithm determines the state of the controller more precisely while trying to make each controller work in the normal mode, to avoid the controller in a long period of high load mode. Compared with the DALB and PALB algorithms, this algorithm makes the load of the controller more balanced, and at the same time, it can effectively improve the throughput recovery speed and greatly improve the ability of the control plane to cope with unexpected situations. In the future work, it will study how to reduce the communication overhead of the algorithm and the decision delay caused by the information transmission more effectively.

Acknowledgement

This work was financially supported by the National Natural Science Foundation of China under Grant 61571021 and 61671029, the Science and Technology Development Project of Beijing Municipal Education Commission under Grant KM201610005007.

References


