Contemporary Engineering Sciences, Vol. 9, 2016, no. 18, 881 - 888
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ces.2016.66105

Load Balancing of Software-Defined Network Controller Using Genetic Algorithm

Seung-Bo Kang and Gu-In Kwon[*]

Department of Computer Science and Engineering
Inha University, 100 Inharo, Nam-gu Incheon 22212, Korea
[*]Corresponding author

Abstract

A software-defined network (SDN) provides flexible network management using centralized and open control. It separates the control and data planes, and the controller is responsible for the control plane. Network administrators can apply new services to their network and expand or downscale it by programming the controller. Switches communicate with controllers through an OpenFlow protocol to synchronize and receive a flow table. When there are considerable requests for routing information from multiple switches to a controller, the controller overflows and the performance deteriorates. OpenFlow allows the use of multiple controllers in the network to solve this problem. All the controllers should be well balanced to avoid failure of a heavy-loaded controller. In this paper, we suggest a load-balancing algorithm based on a genetic algorithm. If controller load is not well balanced, our algorithm performs selection, crossover, and mutation to determine optimal load balancing.

Keywords: SDN, OpenFlow, genetic algorithm, load balancing, reliability

1 Introduction

It is difficult to manage a legacy network dynamically because its closed and distributed characteristics make it rigid and inaccessible. To solve these problems, a software-defined network (SDN) is proposed as a new paradigm. The key concept of SDN is to separate the control and data planes. A control plane is defined as software that controls network, and data plane is defined as a hardware that forwards packets. SDN switches are connected to a controller logically so that they can be controlled centrally. Network administrators can configure and manage devices easily by using a controller.

When there are several switches connected to a specific controller and the controller is overloaded, suboptimal performance problems can occur [6], which prevents controllers from accepting requests from switches. Therefore, OpenFlow is used, which supports multiple connections between switches and controllers [3]. Through this paper, we suggest a solution for distributing and balancing the load of controllers based on a genetic algorithm when imbalance is detected.

2 SDN controller load balancing

Switches send Packet in messages with the received packet to a controller to obtain flow tables if there is no routing information. When the controller receives the Packet_in messages, it creates a flow table through routing operations. The flow table is sent to the switches with a Packet_out message so that the switches transmit packets according to the flow table.

Current SDN controllers [1] [2] provide many service operations, such as routing topology processing, firewalls, and virtual private networks. When there are several switches connected to a particular controller, the amount of Packet_in messages and controller operations increase. This leads to suboptimal performance problems. The time required to receive a control message and flow table from the controller increases, or switches may not receive them at all. OpenFlow supports management of multiple controllers to distribute the load. In this paper, we propose an algorithm based on a genetic algorithm to distribute and rebalance load when a network is controlled by multiple controllers. This prevents suboptimal performance between the controller and switch in an SDN environment and increases the reliability of the network.

A genetic algorithm is a solving methodology based on evolutionary principles of nature [3]. This is based on a theory that mimics a natural phenomenon according to which the most appropriate objects and their genes survive and evolve in the environment [8]. It is used to solve routing and scheduling optimization problems [4]. A genetic algorithm has a certain set of solutions called population: a group of objects [9]. After generating the population, fitness values of each individual are calculated. These fitness values indicate whether the individual is fit for environment and can be calculated using a fitness function. Based on the calculated fitness values, the solutions can be obtained through selection, crossover, and mutation. These new solutions can be replaced with the individuals in the initial solution set, and the fittest individual can be an optimized solution [3]. In this paper, we propose a method to apply this genetic algorithm to the SDN environment so that the number of switches assigned to each controller is equally distributed.

3 Models for load balancing

3.1 Switches and controllers model

A set of controllers in the network can be expressed as $C = \{C_1, C_2, C_3, \cdots, C_M\}$, where M indicates the total number of controllers and C_i is the i-th controller. A set of switches under the control of the controllers is defined as $S = \{S_1, S_2, S_3, \cdots, S_N\}$, where N indicates the total number of switches and S_i is the i-th switch. We define the mapping solution of switches and controllers as M. It can be expressed as $M = \{M_1, M_2, M_3, \cdots, M_P\}$, where P is the number of mapping solutions.

3.2 Expression of load

Controller load can be measured by the amount of received messages from the switches, the amount of CPU resources and memory usage for routing operations and transmission of flow table, and network I/O [7]. In general, CPU utilization in routing operations and the amount of received Packet_in messages from the switches greatly influence controller throughput. Therefore, we considered these factors in the calculations of controller load in this study.

As many Packet_in messages that the switches sent may be assumed to increase the controller load, the sum of the Packet_in messages should be included for calculating the load. The average amount of Packet_in messages received at controller C_i during time T are expressed in equation (1), where $S_i(T)$ is the amount of messages sent to the controller from the switch S_i.

$$\overline{S_i(T)} = \frac{S_i(T)}{T} \tag{1}$$

In equation (2), $C(T)$ represents the controller load, whose value is calculated by considering the amount of received Packet_in messages and CPU utilization by the controller.

$$C(T) = \frac{\omega_1 \times \sum_{i=1}^{N} \overline{S_i(T)}}{C_{\mathrm{MAX}}(T)} \times 100 + \omega_2 \times U \tag{2}$$

where U represents CPU utilization, which has a value between 0 and 100 and $C_{\mathrm{MAX}}(T)$ is the maximum amount of Packet_in messages that the controller can process in time T. The ratio of $C_{\mathrm{MAX}}(T)$ to the amount of Packet_in messages during time T indicates the processing capability of the currently received message. The large amount of Packet_in messages received does not always affect CPU utilization because the controller may have the path information for the request. We use two weight values ω_1 and ω_2 to consider this weak correlation. These two values are computed from the experimental results. ω_1 is a value considering a case in which no routing calculation is required because the path information to its destination already exists, and ω_2 is a value considering a case in which CPU usage is not related to routing operations.

When some switches send numerous routing requests to a specific controller, or a specific controller experiences a rapid increase in the amount of computation, it leads to bottlenecks and load imbalance.

The standard deviation $\sigma_i\ (T)$ of each controller load in mapping solution M_i is defined as follows.

$$\sigma_i(T) = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(\overline{C_i(T)'} - \overline{C_j(T)})^2} \tag{3}$$

4 Load balancing strategy of an SDN controller

4.1 Population coding

We initiated a population to generate optimal mapping solutions by randomly assigning switches to controllers. Mapping solutions belonging to the population can produce a new good-mapping solution in which loads are well distributed to each controller through selection, crossover, and mutation according to the genetic algorithm. Conventional genetic theories represent genes in a chromosome structure as binary coding; however, the controller and switch have a one-to-many relationship. We represent the mapping solution of the controller and switch through a tree structure, shown in Figure 1. The root node of the tree is defined as a super controller node responsible for load distribution. Each internal node is a controller node and the leaf nodes represent switches [6].

4.2 Fitness function

The genetic algorithm uses a fitness function to determine the suitability of each solution. We use a fitness function $f(M_i, T)$ from an existing study of cloud computing [7] to obtain a fitness value of mapping solution M_i for time T.

$$f(M_i,\ T) = \frac{1}{A + B\ \times f_H}$$

$$f_H\ =\ \Phi(\sigma_i(M_i,\ T) -\ \sigma_0),\ \Phi(X) = \begin{cases} 1, & X \le 0 \\ r(> 1), & X > 0 \end{cases}$$

where σ_0 is the predefined standard deviation of the load and f_H is the difference between σ_0 and the standard deviation of the load in the current mapping solution. A and B are weighed values to tune the sensitivity of the standard deviation difference in the fitness function. If the current σ_i of M_i is less than the predefined σ_0, f_H = 1. If it is greater than σ_0, $f_H\ =\ r$, which is a value greater than 1. If the standard deviation of the controller loads is greater than the predefined value σ_0, the fitness value will be low. However, if the standard deviation of the load is lower than σ_0, the fitness value will be high [6].

4.3 Selection

After the fitness value is computed, the individuals are subjected to selection. A higher fitness value of each individual implies a higher probability of sharing information about the placement of switch to the next generation.

A roulette-wheel selection method is used to select the parent chromosome. This selection method allows to choose mapping solutions according to the ratio of the fitness value of each individual to the fitness value of all the individuals [5]. In this study, we calculate the probability as follows.

$$p(M_i) = \frac{f(M_i,\, T)}{\sum_{i=1}^{N} f(M_i,\, T)}$$

where $f(M_i, T)$ represents the fitness of each individual and $p(M_i)$ is the probability of being selected for M_i. After the selection, a new individual will be generated through a crossover operation.

4.4 Crossover operation

In a genetic algorithm, a better solution can be obtained by genetic recombination of the selected individuals through cloning and crossover operation. We can determine a better mapping solution in which loads are well distributed using a crossover operation. The following steps show the crossover operation for creating a new offspring M_0[6].

1. Calculate each individual’s fitness value and select the mapping solutions M_1 and M_2 through roulette-wheel selection.

2. Assign any switches that are duplicated from each parent mapping solution M_1 and M_2 to a new solution M_0.

3. Rank the remaining switches according to their load proportion to total load. A switch with the largest load will be assigned to the least loaded controller and the switch with the next largest load is assigned to the next least loaded controller. This assignment will continue until all switches are assigned to controllers.

4. If the standard deviation of a new mapping solution’s load is lower than its predefined value, we can use the mapping solution. Else, rerun this algorithm from step 2.

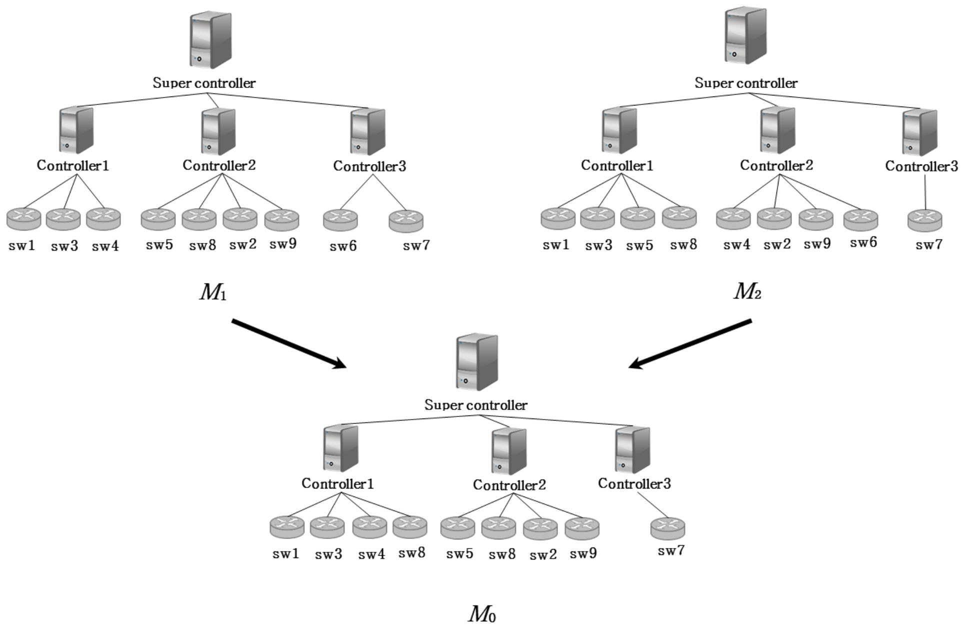

Fig 1. The offspring can be obtained by crossover of M_1 and M_2.

4.3 Mutation

The genetic algorithm performs mutation to vary the operational changes, preventing overly enthusiastic or potential loss of valuable genes after crossover operation [4]. Our algorithm performs mutation by selecting an individual randomly according to the predefined probability P_m in the selection method. It can widen the spectrum of the result [6].

5 Evaluation

An experiment was performed in a simulated environment similar to the actual SDN environment using Java language. The environment consists of one super controller to balance the load, four controllers for routing operations, and a hundred switches to transmit packets and generate Packet_in messages. The switches generate Packet_in messages in a static cycle according to a randomly given interval. The loads are calculated according to the previously defined model, and the load-balancing algorithm functions when the standard deviation of the calculated load on each controller exceeds the given threshold. In the simulation, the value of the load is 1,000–1,500 and the experiment is designed to utilize the load-balancing algorithm for each given interval.

In the genetic algorithm, the population with 30–200 individuals is considered to perform best [4]. In this experiment, we generated 200 individuals. Mutation probability is 1%, which is well-known as the best performance.

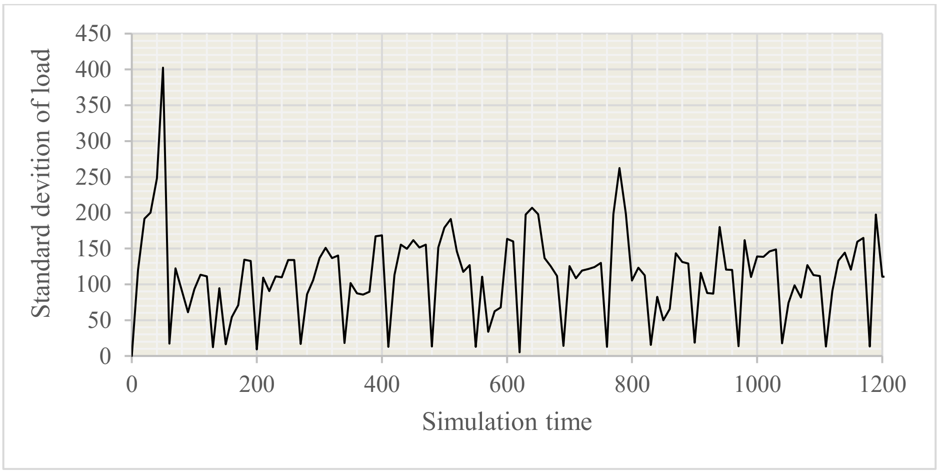

Fig 2. A super controller periodically calculates controller load. If the deviation of load exceeds the predefined threshold 50, it considers that imbalance occurred and the load-balancing algorithm is implemented

Figure 2 shows the change in standard deviation of the controller load according to time elapsed from the initial mapping solution. The controller load is increased by exchanging Packet_in messages between switches and controllers, and the change in standard deviation occurs according to the difference between the amounts of messages in each controller. The threshold at which the imbalance will be detected was set at 50. If the value of the standard deviation is more than 50, the load-balancing algorithm is applied to determine an appropriate mapping solution. The new mapping solution has a standard deviation that does not exceed the threshold, and when unbalance is detected, the load-balancing algorithm is performed again.

6 Conclusion

In this paper, we modeled controller load and proposed a method for load balancing by using a genetic algorithm. We calculated controller load with CPU utilization and Packet_in messages. Further, when its standard deviation exceeded a predefined threshold, we distributed the load using the algorithm. We simulated an SDN environment and proved that an optimized mapping solution can be determined through the proposed algorithm. If the switches generate high-complexity Packet_in messages in this simulation, similar performance can be revealed in the real network.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by Korea government (MSIP) (no. NRF-2015R1A2A2A01003501).

References

[1] Floodlight. http://www.projectfloodlight.org

[2] Opendaylight. https://www.opendaylight.org

[3] D. E. Golberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addion-Wesley, 1989.

[4] Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, Springer Science & Business Media, 2013.

[5] B. L. Miller and D. E. Goldberg, Genetic algorithms, selection schemes, and the varying effects of noise, *Evolutionary Computation*, **4** (1996), no. 2, 113-131. http://dx.doi.org/10.1162/evco.1996.4.2.113

[6] S. B. Kang and G. I. Kwon, Load Balancing Strategy of SDN Controller Based on Genetic Algorithm, *Advanced Science and Technology Letters*, **129** (2016), 219-222. http://dx.doi.org/10.14257/astl.2016.129.43

[7] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, R. Kompella, Towards an elastic distributed SDN controller, *ACM SIGCOMM Computer Communication Review*, **43** (2013), no. 4, 7-12. http://dx.doi.org/10.1145/2534169.2491193

[8] A. Abu-Srhan and A. D. Essam, A hybrid algorithm using a genetic algorithm and cuckoo search algorithm to solve the traveling salesman problem and its application to multiple sequence alignment, *International Journal of Advanced Science and Technology*, **61** (2013), 29-38. http://dx.doi.org/10.14257/ijast.2013.61.04

[9] H. Heidari and A. Chalechale, Scheduling in multiprocessor system using genetic algorithm, *International Journal of Advanced Science and Technology*, **43** (2012), 81-93.

Received: May 1, 2016; Published: August 11, 2016