Dynamic Load Balancing Algorithm Based on Flow Scheduling Selection for DCN: Postprint

Authors: Li Songzhou, Shu Yong' an

Date: 2018-05-20T00:00:00+00:00

Abstract

To address the load imbalance issues arising from dynamic scheduling in data center networks (DCN), this paper proposes a Dynamic Load Balancing based on Flow Scheduling Selection (DLBFSS) algorithm. The algorithm first computes the equal-cost shortest paths for each elephant flow on congested links and eliminates paths that fail to meet the flow bandwidth requirements; subsequently, it calculates the available throughput of the remaining paths and selects the path with the maximum available throughput as the optimal scheduling path; finally, it defines a scheduling congestion probability based on the bandwidth of elephant flows and the load of the optimal path, using this congestion probability as the criterion for elephant flow scheduling decisions. Experimental results demonstrate that, compared with traditional ECMP (Equal-Cost Multi-Path) routing and existing elephant flow scheduling algorithms, DLBFSS effectively reduces network latency, enhances flow bandwidth utilization, and ensures superior load balancing.

Full Text

Preamble

Dynamic Load Balancing Algorithm for DCN Based on Flow Scheduling Selection

Li Songzhou, Shu Yongan
College of Computer Science & Technology, Anhui University, Hefei 230601, China

Abstract: To address the load imbalance problems caused by dynamic scheduling in data center networks (DCN), this paper proposes a Dynamic Load Balancing algorithm based on Flow Scheduling Selection (DLBFSS). The algorithm first calculates the equal-cost shortest paths for each large flow on congested links and eliminates paths that cannot meet the flow' s bandwidth requirements.

It then computes the available throughput of each remaining path and selects the path with maximum available throughput as the optimal scheduling path. Finally, it defines a congestion probability for scheduling based on the large flow' s bandwidth and the optimal path' s load, using this probability as the basis for large flow scheduling decisions. Experimental results demonstrate that compared with traditional ECMP (Equal-Cost Multi-Path) routing and existing large flow scheduling algorithms, DLBFSS can reduce network delay, improve flow bandwidth utilization, and ensure better load balancing.

Keywords: data center network (DCN); dynamic load balancing; software defined network (SDN); scheduling selection; congestion probability

0 Introduction

With the rapid development of computer networks and data center networks (DCN), new technologies are needed to manage and monitor network infrastructure. Software Defined Networking (SDN), as a novel network architecture, enables centralized management and global information monitoring through the decoupling of control and forwarding planes [1]. DCNs typically belong to a single owner, which aligns with SDN' s centralized control philosophy [6]. Through centralized control, SDN controllers can provide a global network view and real-time network status, creating favorable conditions for routing algorithm design and network congestion control. DCNs host numerous delay-sensitive and high-throughput applications [2], and to meet the demands for high bandwidth and low latency, most DCN architectures adopt hierarchical multi-root topology models such as Folded-Clos [3] and Fat-Tree [4]. These network models feature multipath characteristics that provide multiple equivalent paths for data flow routing, resulting in abundant bandwidth resources within DCNs. However, due to routing protocol limitations, these resources cannot be effectively utilized, making routing protocols a primary cause of inefficient bandwidth utilization [5]. Therefore, effective routing algorithm design and congestion control mechanisms can achieve network load balancing, reduce link congestion, and improve bandwidth resource utilization. Leveraging the advantages of SDN' s centralized control, this paper proposes the DLBFSS algorithm, which primarily aims to solve congestion control problems in DCNs, achieve better dynamic load balancing, and enhance network resource utilization.

1 Related Research

Load balancing algorithms in SDN-based data center networks can be categorized into dynamic and static approaches. In static load balancing, flow paths cannot be changed during transmission, whereas dynamic load balancing allows flows to dynamically change their transmission paths during transfer, enabling the scheduling of partial traffic from congested paths to idle paths to achieve dynamic network load balancing.

Static Load Balancing Algorithms: ECMP (Equal-Cost Multi-Path) [7] is a typical load balancing method that utilizes network multipath characteristics to statically map data flows to various equivalent paths through hash functions. However, it fails to consider network status and flow size, easily causing network congestion. The DLB algorithm proposed in [8] employs a greedy strategy, using depth-first search to select the link with maximum free bandwidth at each hop. Since it does not consider global network load, it cannot guarantee the quality of selected paths. CAMOR [9] is a congestion-aware routing algorithm that selects the path with maximum available throughput and no congestion among multiple equivalent shortest paths between nodes, but it neglects the minimum residual bandwidth of paths, potentially causing link overload.

Dynamic Load Balancing Algorithms: LABERIO [10] is a single-hop algorithm that judges network balance degree by setting a network equilibrium threshold. When the balance degree exceeds this threshold, it schedules the largest flow on the most heavily loaded link. Wile Sehery et al. [3] proposed SRL and FlowFit algorithms. SRL serves as a routing initialization algorithm that randomly selects two equivalent shortest paths between node pairs and chooses the one with minimum load as the route. When a link becomes congested, FlowFit sequentially schedules the flows with maximum impact on load until the link load falls below the congestion threshold. However, both LABERIO and FlowFit only select the currently largest flow for scheduling without considering other large flows. This approach may not yield optimal results and can easily cause high load or congestion on scheduling paths, which is detrimental to network load balancing. To address this limitation, this paper proposes the DLBFSS algorithm, which calculates the congestion probability for each large flow on congested links based on flow bandwidth and scheduling path load, using this probability to select flows for scheduling. This prevents inappropriate large flow scheduling and improves the effectiveness of dynamic load balancing.

2 Dynamic Load Balancing Scheme Based on Flow Scheduling Selection

This dynamic load balancing scheme consists of three modules: traffic measurement, routing, and congestion scheduling. These modules are implemented as part of the Floodlight controller [11] and rely on services provided by the controller to operate, as shown in [Figure 1: see original paper].

2.1 Routing Module

To compare the congestion scheduling effectiveness of FlowFit and DLBFSS, this paper uses the same SRL algorithm [3] as the routing module for DLBFSS to initialize routing and select routes for each newly arrived flow. The process is as follows: First, all equivalent shortest paths for the flow are calculated based on network topology information. Then, two different hash functions are applied to the flow header information, and the hash values are used to select

two equivalent shortest paths. Finally, the path with minimum load is selected as the route using link load information provided by the traffic measurement module.

2.2 Traffic Measurement Module

This module periodically sends statistical request messages to switches in the network to obtain traffic statistics, which are used to calculate network load status and traffic distribution, providing decision-making information for other modules.

Key Functions: Calculate link load and utilization, identify large flows on links, detect link congestion, and invoke the congestion scheduling module when congestion is detected.

Notation: This paper uses a graph to represent network topology, where $V = \{v_1, v_2, ..., v_x\}$ denotes the set of switches, $E = \{e_1, e_2, ..., e_y\}$ denotes the set of links, $e_{ij} \in E$ represents the link between node v_i and node v_j, and T denotes the period.

Link Load: Link load is the occupied bandwidth of a link at a given moment, which can be measured by the number of bytes transmitted and received at connected ports within a unit time. If the number of bytes transmitted and received at connected ports during period T are denoted as R and S respectively, the load of link e_{ij} is defined as:

$$B_{ij} = \frac{R + S}{T}$$

Link Utilization: Link utilization is the ratio of link load to link bandwidth, denoted as B_c in the experiments. Based on the known link load B_{ij}, the utilization is defined as:

$$u_{ij} = \frac{B_{ij}}{B_c}$$

Large Flow Detection: In SDN networks, the OpenFlow protocol maintains counters for each flow table entry, which can be used to count the number of bytes received and estimate flow bandwidth. This paper uses B_{th} to denote the threshold for distinguishing large flows from small flows, and b_f to represent flow bandwidth. If the number of bytes transmitted by a flow at times $t - T$ and t are b_s and b_t respectively, the flow bandwidth can be defined as:

$$b_f = \frac{b_t - b_s}{T}$$

When $b_f > B_{th}$, the flow is considered a large flow; otherwise, it is a small flow.

Link Congestion Detection: In the experiments, a certain proportion of link bandwidth (denoted by β) is used as the congestion threshold B_{th}, defined as $B_{th} = \beta \times B_c$. When $u_{ij} > \beta$ is detected, link e_{ij} is considered congested, and the congestion scheduling module is invoked.

2.3 Congestion Scheduling Module

In data center networks, there are large flows with high bandwidth requirements and small flows that are delay-sensitive [12]. When network congestion occurs, scheduling small flows increases delay overhead, while large flows are less sensitive to delay, making them suitable for scheduling.

Key Functions: Select optimal scheduling paths for each large flow on congested links, calculate congestion probabilities for large flow scheduling, and use these probabilities to select flows for scheduling. This method considers the impact of large flow scheduling on scheduling paths, prevents inappropriate large flow scheduling, and facilitates network load balancing.

Notation: If K large flows are detected on congested link e_{ij}, they are denoted as $F = \{f_1, f_2, ..., f_K\}$. Flow f_k has M equivalent shortest paths, with the m-th path denoted as $p_{km} = \{e_{km1}, e_{km2}, ..., e_{kmi}\}$, where e_{kmi} represents the i-th link on path p_{km}, B_{kmi} represents the load of link e_{kmi}, b_{fs} represents the statistical bandwidth of flow f_k, and the allocatable bandwidth of flow f_k is denoted as b_{fk}.

Candidate Path Calculation: First, the shortest path algorithm is used to calculate all equivalent shortest paths for large flows on congested links. Then paths that do not satisfy the flow bandwidth transmission requirement are removed, meaning each path must satisfy:

$$B_c - B_{kmi} \geq b_{fk}$$

The resulting candidate path set for flow f_k is denoted as:

$$P_k = \{p_{k1}, p_{k2}, ..., p_{kN}\}$$

Optimal Scheduling Path Selection: This paper selects the path with maximum available throughput that satisfies flow bandwidth requirements as the optimal scheduling path. For flow f_k' s candidate path set P_k, the available throughput of candidate path p_{kn} is defined as the sum of residual bandwidth of each link on the path, denoted as $T(p_{kn})$:

$$T(p_{kn}) = \sum_{i=1}^{I_k} (B_c - B_{kmi})$$

From this definition, the larger the path' s available throughput, the lower the congestion degree, which is more favorable for flow transmission. The candidate path with maximum $T(p_{kn})$ value is selected as the optimal scheduling path, denoted as p_{ko}:

$$p_{ko} = \arg\max\{T(p_{kn}) \mid n = 1, ..., N\}$$

Flow Scheduling Selection: To make large flow scheduling more accurate and effective, this paper adopts a worst-fit approach to define the congestion

probability for large flow scheduling, considering the impact of large flow scheduling on the scheduling path. The ratio of flow f_k' s bandwidth to the bottleneck bandwidth of its optimal scheduling path is defined as the scheduling congestion probability, denoted as p_k:

$$p_k = \frac{b_{fs}}{\min\{B_c - B_{kmi} \mid e_{kmi} \in p_{ko}\}}$$

where $\min\{B_c - B_{kmi} \mid e_{kmi} \in p_{ko}\}$ is the minimum residual bandwidth of path p_{ko}, i.e., the bottleneck bandwidth of p_{ko}. From this definition, the smaller the p_k value, the lower the congestion degree of path p_{ko} after scheduling, and the more balanced the network load.

All large flow scheduling congestion probabilities are then calculated, and flows are sorted by this probability to obtain the scheduling set:

$$Flowlist = \{f_k \mid k = 1, ..., K\}_{sort}$$

where flows are stored in ascending order of p_k value. The smaller the congestion probability, the higher the scheduling priority. Large flows are selected for scheduling in ascending order of congestion probability until the link load falls below the congestion threshold.

DLBFSS Algorithm Based on Flow Scheduling Selection:

```
if (u_{ij} >  ) then
    F = e_{ij}.largeflowlist();
    K = F.length();
    for (f_k in F, k = 1 to K) do
        p_{ko} = f_k.optimalpath();
        p_k = p_{ko}.getprobability();
    end for
    Flowlist = F.sort(p_k);
    for (f_k in Flowlist) do
        reroute(f_k, p_{ko});
        updatelinkload();
        if (u_{ij} <  ) then
            break;
        end if
    end for
end if
```

The `reroute()` function schedules flow f_k to path p_{ko}, and `updatelinkload()` updates link loads after scheduling. Scheduling stops when link load falls below the congestion threshold.

Due to the development of virtualization technology and cloud computing, east-west traffic between servers in data center networks has increased significantly [13], easily causing network congestion during operation. In mechanisms lacking

dynamic traffic scheduling, some links become congested while others remain idle, resulting in unbalanced traffic load, increased network transmission delay, reduced flow bandwidth utilization, and degraded overall network performance. In mechanisms with dynamic traffic scheduling, if the impact on scheduling paths is not considered and inappropriate flows are selected for scheduling, new network congestion can also occur, leading to unbalanced traffic load and hindering overall network performance improvement. Therefore, this paper introduces a large flow scheduling selection algorithm based on congestion probability. As described above, this method minimizes the load on scheduling paths after large flow scheduling while reducing traffic load on congested links, preventing congestion on scheduling paths. During dynamic changes in network load status, it ensures balanced traffic load across all links, facilitates flow transmission, reduces network transmission delay, increases flow bandwidth utilization, and improves overall network performance, as validated through experiments below.

3 Simulation Experiments

3.1 Simulation Scenario Description

SDN Environment Setup: The experiments use the Floodlight controller as the control plane and Mininet [14] to simulate the data plane. Mininet can create a data plane comprising Open vSwitch switches, links, and end hosts. The experimental environment is a 64-bit Ubuntu 14.04 operating system.

Data Center Network Model: Mininet is used to simulate a Fat-Tree topology as the research object for data center networks. When a Fat-Tree topology has k pods, it contains $\frac{k^3}{4}$ core switches, $\frac{k^2}{2}$ aggregation switches, $\frac{k^2}{2}$ edge switches, and $\frac{k^3}{4}$ hosts. This paper uses $k = 4$, creating a data center network topology with 4 pods, 20 switches, and 16 hosts, including 4 core switches, 8 aggregation switches, and 8 edge switches, as shown in [Figure 2: see original paper].

Experimental Parameters: Link bandwidth is set to 1 Gbps. The Iperf traffic generation tool is used to send traffic of different sizes. To simulate realistic data center network scenarios, a random traffic model (where any host sends data flows to other hosts with equal probability) is employed. The threshold for distinguishing large flows from small flows is set to 0.5% of link capacity [3], i.e., $B_{th} = 5$ Mb/s. Based on traffic characteristics in data center networks, 90% of flows are configured as small flows and 10% as large flows. Network performance is tested by gradually changing traffic load, comparing algorithm performance under different loads. Traffic load is defined as the ratio of average occupied bandwidth on host links to link capacity, ranging from [0,1]. Nine load levels are tested, from 0.1 to 0.9.

3.2 Load Balancing Evaluation Metrics

Average bandwidth utilization and average delay are important metrics for evaluating network performance and reflecting network load balancing degree.

a) Average Bandwidth Utilization: The average of bandwidth utilization across all flows, where single flow bandwidth utilization is the ratio of bandwidth received at the server to bandwidth sent by the client.

b) Average Delay: The average end-to-end transmission delay of data packets.

3.3 Simulation Results and Analysis

To validate the superior performance of the proposed congestion probability-based large flow scheduling selection algorithm, DLBFSS is compared with traditional ECMP and the existing FlowFit large flow scheduling algorithm. ECMP is a widely deployed equal-cost multipath load-sharing routing algorithm in enterprise and data center networks that uses flow-based hashing to statically map data flows to equivalent paths for traffic load balancing. FlowFit [3] is a new typical data center network traffic dynamic scheduling algorithm that reduces network congestion by scheduling some of the largest flows from congested links to less loaded paths, achieving dynamic load balancing during flow transmission. DLBFSS improves upon FlowFit' s limitation of only selecting the largest flow by considering the impact on overall network performance after scheduling. All three algorithms are implemented in the Floodlight controller, with simulation results shown in [Figure 3: see original paper] and [Figure 4: see original paper].

[Figure 3: see original paper] shows the average bandwidth utilization of the three algorithms under different loads. As traffic load increases, DLBFSS' s average bandwidth utilization is significantly higher than ECMP and FlowFit. ECMP has the lowest average bandwidth utilization. When traffic load reaches 0.9, DLBFSS' s average bandwidth utilization is 3.8% and 11.9% higher than FlowFit and ECMP, respectively. ECMP does not consider network load during routing, easily causing link congestion, and lacks dynamic flow scheduling mechanisms, resulting in more packet loss and lowest average bandwidth utilization. Both FlowFit and DLBFSS use SRL as the routing initialization algorithm, consider network load, and have dynamic flow scheduling mechanisms, reducing packet loss and achieving higher average bandwidth utilization than ECMP. When link congestion is detected, FlowFit always schedules the currently largest flow without considering other large flows, potentially causing congestion on scheduling paths. DLBFSS considers all large flows, defines a congestion probability for large flow scheduling, and selects the flow with minimum congestion probability for scheduling, reducing the likelihood of scheduling path congestion and facilitating network load balancing. This results in fewer packet losses and higher average bandwidth utilization than FlowFit.

[Figure 4: see original paper] describes the change in average delay under different traffic loads. As traffic load increases, the average delay of all three

algorithms continues to rise. Under the same load, ECMP has the highest average delay, while DLBFSS has the lowest. When traffic load reaches 0.9, DLBFSS' s average delay is 22 s lower than FlowFit and 74 s lower than ECMP. ECMP statically assigns traffic through hashing, easily routing flows to heavily loaded paths. When links become congested, it cannot dynamically schedule flows, resulting in longer data packet transmission times and highest average delay. Both FlowFit and DLBFSS consider network load during routing initialization and can change flow paths on congested links, scheduling flows to idle paths, reducing network congestion and decreasing data packet transmission time, thus achieving lower average delay than ECMP. During congestion scheduling, DLBFSS considers the bandwidth of all large flows and link load, selecting among large flows to be scheduled and avoiding FlowFit' s limitation of only selecting the largest flow. This reduces network congestion degree and achieves lower average delay than FlowFit.

4 Conclusion

This paper proposes the DLBFSS algorithm, which improves upon existing dynamic load balancing scheduling algorithms that only select the largest flow, optimizing the selection method for large flow scheduling. When network congestion is detected, DLBFSS comprehensively considers the bandwidth of large flows on congested links and the load of equivalent paths, calculates the optimal scheduling path and congestion probability for each large flow, and sequentially schedules large flows with minimum congestion probability until link load falls below the congestion threshold. This minimizes the load on scheduling paths after large flow scheduling, avoiding new network congestion problems. Simulation results show that DLBFSS achieves better network performance than existing load balancing algorithms ECMP and FlowFit, solves FlowFit' s limitation of only selecting the largest flow, improves the accuracy and effectiveness of large flow scheduling, and achieves better dynamic load balancing.

References

[1] Veena S, Rustagi R P, Murthy K N B. Network management and performance monitoring using Software Defined Networks [C]// Proc of 20th Annual International Conference on Advanced Computing and Communications. 2014: 29-31.

[2] Gholami M, Akbari B. Congestion control in Software Defined Data Center Network through Flow Rerouting [C]// Proc of the 23rd Iranian Conference on Electrical Engineering. 2015: 654-657.

[3] Sehery W, Clancy T C. Load balancing in data center networks with folded-clos architectures [C]// Proc of the 1st IEEE conference on network softwarization. 2015: 1-6.

[4] Jo E, Pan Deng, Liu J, et al. A simulation and emulation study of SDN-based multipath routing for fat-tree data center networks [C]// Proc of Winter Simulation Conference. 2014: 3072-3083.

[5] Wojcik R, Domzal J, Dulinski Z. Flow-Aware Multi-Topology Adaptive Routing [J]. IEEE Communications Letters, 2014, 18 (9): 1539-1542.

[6] 杨洋, 杨家海, 秦董洪. 数据中心网络多路径路由算法 [J]. 清华大学学报: 自然科学版, 2016, 56 (3): 262-268.

[7] Chiesa M, Kindler G, Schapira M. Traffic engineering with equal-cost-multipath: an algorithmic perspective [J]. IEEE//ACM Trans on Networking, 2017, 25 (2): 779-792.

[8] Li Yu, Pan Deng. OpenFlow based load balancing for Fat-Tree networks with multipath support [C]// Proc of the 12th IEEE International Conference on Communications. 2013: 1-5.

[9] Shah S A R, Seok W, Kim J, et al. CAMOR: congestion aware multipath optimal routing solution by using software-defined networking [C]// Proc of the International Conference on Platform Technology and Service. 2017: 1-6.

[10] Long Hui, Shen Yao, Guo Minyi, et al. Dynamic load-balanced Routing in OpenFlow-enabled Networks [C]// Proc of the 27th International Conference on Advanced Information Networking and Applications. 2013: 290-297.

[11] Project Floodlight. Floodlight [EB/OL]. [2017-06-05]. http://www.projectfloodlight.org/.

[12] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild [C]// Proc of the 10th ACM SIGCOMM Conference on Internet Measurement. 2010: 267-280.

[13] Deng Gang, Gong Zhenghu, Wang Hong. Characteristics research on modern data center network [J]. Journal of Computer Research and Development, 2014, 51 (2): 395-407.

[14] Mininet Team. Mininet [EB/OL]. [2017-06-05]. http://mininet.org/.

Note: Figure translations are in progress. See original paper for figures.