DFFR: A flow-based approach for distributed load balancing in Data Center Networks

Chung-Ming Cheung, a,b Ka-Cheong Leung b

a Department of Computer Science, University of Southern California, Los Angeles, CA 90089, United States
b Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong, China

ABSTRACT

With the increasing need to support high performance and distributed cloud-based computing applications, data centers are employing commodity switches to build multi-rooted trees. An effective distributed, adaptive flow scheduling algorithm is needed to realize the full potential of the multiple parallel paths provided by such networks. The overall aim of this work is to design a load balancer that can maximize the aggregate network utilization. In this paper, the Distributed Flow-by-Flow Fair Routing (DFFR) algorithm is proposed for flow balancing in Data Center Networks. It is a scalable, distributed, and adaptive algorithm designed for maximizing network resources. Our analysis shows that the algorithm has proven theoretical performance bounds, which gives a low variance for the aggregate bandwidth utilization. A simulation study was conducted to compare the performance of the DFFR algorithm with other load balancing algorithms. Our simulation results reveal that the DFFR algorithm outperforms a static routing assignment protocol. It is also compared to Distributed Dynamic Flow Scheduling (DDFS), which is chosen because it is a distributed algorithm like DFFR. DFFR is shown to perform better for random traffic patterns than DDFS, but worse for patterns where hosts always send to the same receiver. The evaluation concludes that the DFFR is an effective load balancer for Data Center Networks with random traffic patterns.

1. Introduction

With the increasing importance of cloud computing for Internet-based applications, researchers look for ways to improve performance of cloud data centers. One such method is to improve load balancers in Data Center Networks (DCNs). To support high performance and distributed cloud-based computing applications, data centers employ commodity switches to build multi-rooted trees [1] as illustrated in Fig. 1. Such networks provide parallel paths for communication. Thus, there is a need for load balancers to direct traffic on these parallel paths efficiently.

Although load sharing or load balancing has been researched extensively in the past for grid networking and web servers, applying past algorithms in Data Center Networks is a new challenge [12]. The load balancer should be able to overcome the heavy workload of DCNs and also take advantage of the regular multi-rooted structure of the network.

There are many existing work on load balancers for DCNs. Hedera [2] makes use of a centralized controller to perform load balancing. Centralized solutions like them suffer from scalability problems [8], in particular, the inability to collect flow statistics from all links quick enough. To alleviate the problem, distributed solutions have been proposed. Some algorithms, such as DiFS [6], use flow rerouting in their approach. This kind of algorithms suffers from path oscillation problems, thereby requiring long time for convergence.

In recent years, Software Defined Networks (SDN), which are enabled by communication protocols such as Open Flow [19], are becoming popular in such load balancing algorithm designs. SDN provides an interface for the control plane to communicate with the data plane. Some functions offered by this control plane make use of the centralized flow statistics in the network so as to control forwarding tables of all switches. While SDN helps in both making and executing flow balancing decisions, in this work, we do not require SDN to be enabled by the DCN to allow the algorithm to be able to be deployed in both older and newer networks.

The aim of this work is to develop an efficient distributed load balancer to appropriately forward flows, so that the full potential of the network can be utilized. We would also like to show the efficiency of the algorithm through theoretical analysis and simulations of a simplified network model.

ARTICLE INFO

Keywords:
Data Center Networks
Load balancing

This work was done in part when Chung-Ming Cheung was with the University of Hong Kong.
Corresponding author.
E-mail addresses: chungmin@usc.edu (C.-M. Cheung), kcleung@ieee.org (K.-C. Leung).

https://doi.org/10.1016/j.comcom.2017.11.001
Received 17 March 2017; Received in revised form 29 September 2017; Accepted 4 November 2017
Available online 07 November 2017
0140-3664/ © 2017 Elsevier B.V. All rights reserved.
This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

1.1. Our contributions

In this paper, a load balancing system incorporating a novel Distributed Flow-by-Flow Fair Routing (DFFR) algorithm is proposed for implementation of flow scheduling in DCNs. This is a distributed and adaptive algorithm suitable for use in a Data Center Network environment. DFFR does this without performing flow rerouting and splitting, so that the performance of TCP is maintained. The algorithm is shown to guarantee a performance bound through theoretical analysis.

This work has implemented DFFR on an event-driven flow-based simulator. Simulation experiments have been carried out to compare DFFR with two other switch-based distributed scheduling algorithms. Our simulation results show that the DFFR algorithm outperforms Equal Cost Multiple Path, which is a static routing assignment protocol. DFFR is also able to maintain a more stable aggregate bisectional bandwidth compared to another distributed flow scheduling algorithm Distributed Dynamic Flow Scheduling (DDFS), showing that an adaptive distributed approach for load balancing can result in bandwidth gain while offering a more stable aggregate bisectional bandwidth with moderate additional cost in memory usage per switch.

The rest of the paper is organized as follows. Section 2 describes the existing literature on distributed flow scheduling. Section 3 discusses the network model that is adopted for this study. Section 4 explains the details of the algorithm used for the implementation of the load balancer for this work, the DFFR algorithm. Section 5 provides a theoretical analysis on the complexity and performance of DFFR. Section 6 gives and discusses the simulation results and their implications. Finally, Section 7 concludes the work.

2. Related work

In the past, a DCN uses a single-rooted tree network to support the network traffic by oversubscription, this design does not scale well. Thus, researchers have made several proposals to improve the performance of DCNs. Greenberg et al. [9] presented a Clos-based DCN. Al-Fares et al. [1] proposed that a fat-tree network, as a special case of a Clos-based network, can be used to provide many equal-cost paths while keeping a 1:1 oversubscription ratio by using a large number of roots. A load balancer is needed to take advantage of the parallel paths.

Hedera [2] presented two centralized flow balancing algorithms. Guo et al. [10] proposed a centralized algorithm that considers routing for both unicast and multicast traffic. These centralized solutions employ communication protocols, such as Openflow [19], to enable centralized access to forwarding tables and statistics gathering within the network. However, these centralized algorithms are difficult to scale well for DCNs, because of the sheer amount of traffic. Curtis et al. [8] have shown that Hedera’s claims were too optimistic, and that a centralized controller cannot pull flow statistics quick enough for making routing decision. DDFS [4] is a distributed flow balancing algorithm that tries to balance the workload sent to the aggregation switches by having each individual switch to route flows onto the least used link.

Another algorithm that makes use of rerouting is DiFS [5]. By rerouting, it causes packet reordering problems that lowers TCP performance. Sen et al. [20] proposed LocalFlow for flow balancing, which employed flow splitting. However, flow splitting causes packet reordering problems, as a result, the number of splits it performs has to be limited.

For DCN load balancing, static algorithms used in traditional networks like Equal-Cost Multiple Path (ECMP) [11] often serve as benchmarks for evaluating performance. Another one is Valiant Load Balancing (VLB), where flows are routed randomly on paths. VLB is suggested in [8] to be used in DCNs, but Mahapatra and Yuan [18] have shown that more intelligent algorithms that made use of knowledge of the network flows outperform VLB.

Another branch of load balancing work concerns the minimization of energy use, maximizing utilization of green energy, and reducing of electricity cost instead of having maximizing the throughput as the main goal [13,14]. Although such factors are not investigated in this paper, research in green data centers is also important for the long term development of data centers.

3. Network model

3.1. Fat-tree network topology

A fat-tree topology is made up of k-port switches organized in three layers. From the top to bottom, they are named core, aggregation, and edge. The bottom layer edge switches connects to hosts. Edge and aggregation switches are organized into structures called pods. There are k pods in a fat-tree topology. Each pod is a collection of \( \frac{k}{2} \) aggregation switches and edge switches, and the edge switch connects to every aggregation switch in the pod. Pods are inter-connected by \( \frac{k}{2} \) core switches. The core switches are grouped together for every \( \frac{k}{2} \) switches such that there are \( \frac{k}{2} \) groups in total. Each aggregation switch in a pod is connected to a certain group of \( \frac{k}{2} \) core switches and no two aggregation switches in the same pod connect to the same group of core switches.

When a flow is to be sent from a lower layer to an upper layer (e.g. edge layer to aggregation layer), it is called an upstream flow. When a flow is to be sent from an upper layer to a lower layer (e.g. core layer to aggregation layer), it is called a downstream flow.

While the application of DFFR is not limited to any kind of multi-rooted tree topology, in this work, fat-tree network topology [1] is used to illustrate its operations and effectiveness.

3.2. Classification of flows

Flows can be classified into elephant and mice flows according to their sizes, which is their total traffic load. Flows with sizes below a certain threshold is categorized as mice flows, flows with sizes above the threshold is categorized as elephant flows. In this paper, we follow the definition for large flows in [2] as flows that occupy more than 10% of the link’s full capacity.

It was shown in [3] that most of the data traffic in a DCN comes from the top 10% of large flows. Thus, like many existing work on flow scheduling [2,6,11], the proposed algorithm balances the traffic load from elephant flows only.

3.3. Simplifying assumptions

We have made the following simplifying assumptions on the network environment for the development of our load balancing algorithm:

- As stated in [18], small flows (mice flows) can be handled by simple hash-based routing methods like VLB. The load balancer only needs to handle large flows (elephant flows) which is the main causes of...
congestion.
- All communication links in the DCN are homogeneous.
  Heterogeneous links are left for future work.

3.4. Working condition

We require that the switches in the network would be able to access TCP header information. When a routing decision is made, an entry is added to the switch’s routing table to ensure all following packets from one flow uses the same path. In other words, a virtual circuit path is set up for each flow from its origin host to its destination host. Research has been done on implementing efficient scalable circuit-switched networks in DCNs [5].

4. DFFR design

4.1. Design objectives

To minimize congestion in DCN, the aim of DFFR is to route flows so that the traffic load is spread to all equal-cost paths as uniformly as possible. DFFR determines the outgoing link to forward a flow to an upper layer so that the deviation of the actual workload from the expected workload on all links is minimized. The expected workload of a link is the amount of workload that should be on the link for optimal flow routing, and the actual workload is the actual amount of workload currently on the link. By minimizing the deviation of the two, the network moves closer towards the optimal flow placement goal.

Flows forwarded to a lower layer has a deterministic route. Thus, they do not need to be considered. By minimizing the deviation, we expect the aggregate bandwidth in the DCN to be more stable since there will be less flow collisions which causes bandwidth drops.

There are several requirements on the algorithm:

1. DFFR should be distributed, as centralized controllers are not scalable.
2. No flow splitting and rerouting are allowed, in order to preserve the performance of TCP as these operations cause out-of-order packet problems.
3. No change is required on end-host applications, so that end-to-end principle holds.
4. Past and future traffic patterns should not be used in the algorithm for making routing decisions, as traffic patterns in DCNs vary over time.

Requirement 1 needs the algorithm to be distributed, because centralized controllers are not scalable. Requirement 2 guarantees the performance of TCP in the network because flow splitting and rerouting both cause out-of-order packet problems. While recent developments like SDNs and Multipath TCP (MP-TCP) aims to overcome this limitation of TCP, in this work, we keep the assumption that traditional TCP is used to allow the developed algorithm to be deployed in any existing DCN. Requirement 3 states that the algorithm can be applied to any end-host applications. Requirement 4 is desirable because the traffic patterns in Data Center Networks are unpredictable and varying continuously.

4.2. Workload definition of DFFR

In DFFR, workload is defined as the maximum bandwidth demand of a flow. It is proposed in [7] that elephant flows can be effectively detected by a special shim layer added in the operating system of every end host for monitoring the TCP socket buffers of the end-host applications.

By observing the rate at which the TCP buffer is filled, it is possible to measure the data production rate of an application, which is unaffected by the state of network congestion. This value can then be appended to the packets of the flow sent by this application as the current bandwidth demand for this flow, then be used by switches for routing decisions [17].

4.3. DFFR architecture

For each switch sending out a flow to another switch on the same layer, there are no direct links between them and the flow must be routed through an upper layer switch. In a fat-tree network, each switch is connected to \( \frac{k}{2} \) switches in the upper layer, we consider these \( \frac{k}{2} \) links as \( \frac{k}{2} \) different paths for the flow to be routed. For example, Fig. 2 shows that for \( k = 6 \), each edge switch can choose from Path 1, 2, or 3, which corresponds to the three aggregation switches that can route the flow.

The generalized load sharing model (GLS) [15] has been proposed to conceptualize how traffic is split ideally on a set of network paths. Inspired by GLS, we develop DFFR which attempts to distribute flows as evenly as possible in the context of DCNs. DCNs are special in that they have a tree-like structure with multiple layers that provide many parallel paths between any pair of hosts. The path chosen for sending a flow out of a switch would also determine the path used to enter the switch on the destination end. Thus, the design and analysis of DFFR should take these factors into account.

Each switch in the data center network can be modelled as a generalized flow-based traffic splitter [16] with equal path weightings for all paths.

Lemma 1. If all switches divide workload to the same destination edge switch via all paths equally, the network will be optimally balanced, where optimally balanced is the flow placement that allows for maximum bandwidth usage of all links’ capacity.

Proof: Consider the communication between any pair of edge switches that are not in the same pod. Suppose that the source switch receives a total workload \( x \) to be sent to the destination switch. The source switch assigns workload equally to each of the \( \frac{k}{2} \) aggregation switches it is connected, such that the workload of \( \frac{x}{k} \) is transmitted via each link. Each aggregation switch receives the workload of \( \frac{2x}{k} \) and divides it further into \( \frac{k}{2} \) equal portions to send to each core switch it is connected to. Thus, each aggregation-to-core link from this pod would have the total workload of \( \frac{4x}{k^2} \). The aggregation switch on the destination side would receive the workload of \( \frac{4x}{k^2} \) from each core switch and the destination edge switch would receive the workload of \( \frac{2x}{k} \) from each aggregation switch in the pod. Thus, all links connecting to the edge switch pairs and all links in the aggregation-core layer are utilized equally.

By Lemma 1, the switches on the destination pod would receive the workload from their incoming links equally, where the network traffic load is distributed evenly among the switches on the same pod. When
the network traffic from all sources with a total workload $x$ are evenly distributed, the expected workload on each of the $k$ links on the core-aggregation layer will be equal, i.e. $\frac{4x}{2k}$.

In DFFR, GLS is used to model the expected workload on each link in the best flow scheduling scheme.

Suppose that a source switch $S$ has a sequence of flows $f_1, f_2, \ldots$ to be sent to a destination edge switch $D$, where $b_i$ represents the maximum bandwidth demand of flow $n$. We use the maximum bandwidth demand because the current sending rate of a flow is not indicative of its natural bandwidth demand in a non-blocking network [3]. If we know that a flow is able to send at a higher rate, it should be allocated a less utilized path and deters more flows from being allocated to the same path, thereby allowing it to grow to its maximum sending rate. Let $P^D_i(m)$ and $P^D_j(m)$ be the expected workload and actual workload that is directed to $D$ to be sent on Path $i$ after the $m$th flow is sent. For $m > 0$, the paths are expected to have equal workload.

$$P^D_i(0) = P_j(0) = 0, \forall i = 1, 2, \ldots, k$$  \hspace{1cm} (1)

$$P^D_i(m) = \frac{2 \sum_{b=1}^{m} b}{k}, \forall i = 1, 2, \ldots, k$$  \hspace{1cm} (2)

We also define $R^D$ as the residual workload directed to $D$ on Path $i$, which is the amount of workload needed for a path $i$ to achieve the expected workload $P^D_i$ just before the routing decision is made.

$$R^D_i(m) = \begin{cases} P^D_i(m) - P^D_i(1), \text{ if } m = 1 \\ P^D_i(m) - P^D_i(m - 1), \text{ otherwise} \end{cases}$$  \hspace{1cm} (3)

Algorithm 1 summarizes the DFFR algorithm.

The residual workload to each path also needs to be updated whenever a flow terminates by calling Algorithm 2. The actions performed are to reverse the effects of Algorithm 1 called when the flow was added.

Algorithm 1

The DFFR algorithm.

```
DFFR(flown bandwidth $b_i$, destination id $D$)
1: for each path $i$ from 1 to $\frac{k}{2}$
2: $R^D_i \leftarrow R^D_i + \frac{b_i}{2}$; // Maintain correct value of $R^D_i$
3: Find $j$ s.t. $R^D_j \geq \text{max} R^D^j$; // Find path of max. $R^D_j$
4: Route flow to $D$ via Path $j$;
5: $R^D_j \leftarrow R^D_j - b_i$; // Maintain correct value of $R^D_j$
```

5. Theoretical analysis

5.1. Algorithmic complexity

**Lemma 2.** Given the number of paths $n_p$ and the number of destinations $n_d$, each run of DFFR has a computational complexity of $O(n_p \cdot n_d)$.

**Lemma 3.** Given the number of paths $n_p$ and the number of destinations $n_d$, the memory complexity of DFFR is $O(n_p \cdot n_d)$.

The computational complexities of an edge switch running DFFR are $O(k^2)$ and $O(k^3)$, respectively.

The algorithm requires the use of extra memory in each switch for storing the residual workload in each path. For a fat-tree network of size $k$, we need $O(k^2)$ memory for switches on the edge layer, and $O(k)$ memory for switches on the aggregation layer.

Therefore, in a fat-tree network of size $k$, the distributed algorithm has a linear time complexity in $k$. This means that for each incoming flow, the routing decision is made very quickly, making this algorithm scalable. The scalability of this algorithm also comes the fact that there is no central controller where the computation is concentrated in. Each switch has relatively low workload compared to a centralized algorithm.

5.2. Performance bounds

Assume there is no flow termination in these analysis. We denote $f_{\text{max}}$ as the maximum possible flow size. We do not make assumptions on what this number would be, and simply prove performance bounds in terms of this value.

We can prove bounds for the residual workload in links of the network bounded by applying Algorithm 1 on edge switches. Denote $P^s_i(m)$ and $\bar{P}^s_i(m)$ as the total expected and actual workload to all destinations $D$ on path $i$ respectively.

$$P^s_i(m) = \sum_{d \in D} P_i^D(m), \forall i = 1, 2, \ldots, k$$  \hspace{1cm} (4)

$$\bar{P}^s_i(m) = \sum_{d \in D} \bar{P}_i^D(m), \forall i = 1, 2, \ldots, k$$  \hspace{1cm} (5)

**Lemma 4.** For any path $i$, consider only flows going from edge layer to aggregation layer, denote $D$ as the set of destinations switches that flows on this path may be routed to, and $n_i$ as the number of destinations switches. For every positive integer $m$, the actual workload allocated $P_i(m)$ cannot exceed the expected workload by an amount equal to or more than $n_i f_{\text{max}}$. That is,

$$\bar{P}_i^s(m) - P_i(m) < n_i f_{\text{max}}$$  \hspace{1cm} (6)

**Lemma 5.** For any path $i$, consider only flows going from aggregation layer to edge layer, denote $S$ as the set of source switches that flows on this path may be routed from, and $n_s$ as the number of destinations switches. For every positive integer $m$, the actual workload allocated $P_i(m)$ cannot exceed the expected workload by an amount equal to or more than $n_s f_{\text{max}}$. That is,

$$\bar{P}_i^s(m) - P_i(m) < n_s f_{\text{max}}$$  \hspace{1cm} (7)

**Theorem 1.** For any link connecting an edge switch to an aggregation switch, its residual workload is bounded by $k(k - 1)f_{\text{max}}$.

**Proof:** First, consider flows on a link $i$ going towards the edge switch. By Lemma 5, the residual workload from source edge switches cannot exceed $n_s f_{\text{max}}$, since there are $\frac{k(k - 1)}{2}$ other pods that can send to this edge switch, the residual workload is bounded by $\frac{k(k - 1)f_{\text{max}}}{2}$. Next, consider flows on a link $i$ going out of the edge switch, there can be at most $\frac{k(k - 1)}{2}$ edge switches to send towards, so by Lemma 4, the residual workload is bounded by $\frac{k(k - 1)f_{\text{max}}}{2}$. Overall, the residual workload on $i$ is bounded by
Lemma 6. Denote $A_i(m)$ and $\bar{A}_i(m)$ as the expected workload and actual workload sent to an aggregation switch from edge switches in the same pod before the decision for the $m$th flow is made. For every positive integer $m$ and any aggregation switch $i$, the actual workload allocated cannot exceed the expected workload by an amount equal to or more than $\frac{k(k-1)}{4}f_{\max}$. That is,

$$\bar{A}_i(m) - A_i(m) < \frac{1}{4}k^2(k-1)f_{\max}$$

We can then give the upper bound on the residual workload of any links in the aggregate-core layer.

Theorem 2. For any link connecting an aggregation switch to a core switch, its residual workload is bounded by $(k + 2)(k-1)f_{\max}$.

Proof: First we consider flows on a link $i$ going from an aggregation switch to a core switch. Denote $A(m)$ and $\bar{A}(m)$ as the expected and actual workload routed via the aggregation switch respectively, after the $m$th flow routing decision is made. By the GLS model, this workload from the aggregation switch is split evenly among the $k/2$ links from the aggregation switch to core switches. Therefore,

$$\bar{P}_i(m) - P_i(m) = \bar{A}(m) - \frac{A(m)}{k} < \frac{k}{2} + 1(k-1)f_{\max}$$

Recall that an aggregation switch is connected to only $k - 1$ aggregation switches via the same group of core switches. This is similar to the case of edge switches sending to other edge switches, so we can apply Lemma 4 to show that the actual workload on link $i$ cannot exceed the expected amount when the received workload to split evenly on the links by $(k - 1)f_{\max}$.

$$\bar{P}_i(m) - \frac{A(m)}{k} < (k - 1)f_{\max}$$

By using Eqs. (9)-(11), we can evaluate a bound on the residual workload.

Next we consider flows on a link $i$ going to an aggregation switch from a core switch. Similar to the previous case, we can evaluate a bound on the residual workload.

$$\bar{P}_i(m) - P_i(m) < \frac{1}{2} + 1(k-1)f_{\max}$$

Thus, the overall residual workload upper bound on the link is the sum of its residual workload into and out of the link,

$$\bar{P}_i(m) - P_i(m) < (k + 2)(k-1)f_{\max}$$

6. Performance evaluation

In this section, we first describe two other load balancing algorithms used for performance studies. The methodology for conducting the simulations are then explained. The performance metrics used for evaluation are also defined. Afterwards, we describe how we synthesized the traffic traces used for the simulation. Finally, the simulation results are analyzed and its implications are discussed.

6.1. Load balancing algorithms

We study the performance of DFFR compared to two other algorithms used in DCNs, namely Equal-Cost Multiple Path (ECMP) [11] and Distributed Dynamic Flow Scheduling (DDFS) [4]. As described in Section 4, our proposed algorithm, DFFR, attempts to spread traffic load on links as uniformly as possible. ECMP is a static load balancing algorithm where the path assigned to a flow is decided by applying a hash function. DDFS chooses the outgoing link which is currently least utilized for each new incoming flow. The algorithm is also compared to the ideal case where all hosts are connected together by a single non-blocking switch.
6.2. Methodology

An event-driven simulator was developed to carry out a series of experiments to simulate the performance of DFFR running in a DCN. For simplicity, flows are stimulated to have uniform sending rate. The results were compared with the two other distributed load balancing algorithms DDFS and ECMP, as well as the ideal non-blocking switch case. The ideal case is used to provide an estimate of the best possible result for comparison.

As there are no publicly available DCN traffic traces, our simulation study follows the methodology used in [2,6], where traffic traces were generated to be used as the benchmark communication suite. Traffic traces following five types of patterns are synthesized to imitate different kinds of traffic patterns in data centers. The patterns are described as follows:

1. Stride(i): For any host with an identification number as x, it sends flows to the host with id = (x + i) mod nof hosts. A small value of i simulates intra-pod communication heavy pattern. A large value of i would stress out the inter-pod links. We take stride0, stride1, and stride2 as stride(1), stride(k), and stride(k²), respectively.
2. Staggered(P1, P2): For any host, it sends to another host under the same edge switch with probability of P1, sends to another host within the same pod with probability of P2, and sends to an edge switch outside the pod with probability of (1 − P1 − P2). We take stag0, stag1, and stag2 as staggered(0, 0), staggered(0, 0.5), and staggered(0.5, 0.3), respectively.
3. Rand: Destination of each flow is chosen randomly.
4. Randx(x): A host sends x flows consecutively to each randomly chosen destination. We take randx0 and randx1 as randx(2) and randx(4), respectively.
5. Randbij: Each host sends to another host by a bijective mapping of hosts generated randomly.

An all-to-all in-memory data shuffle experiment was also carried out. Data shuffle is a common and important operation for applications, like MapReduce and Hadoop for running parallel computations on many host servers situated in DCNs. In one operation, all hosts send a large amount of data to every other hosts. To model such an operation, a 16-host network is simulated where each host sequentially transfers 500 MB of data to all other hosts (a 120 GB shuffle). Ten separate runs were carried out for each algorithm.

A fat-tree network topology is employed for the DCN. All links have the same capacity of 1 GB/s. Delay is not modelled. For testing the benchmark communication suite, a network of 1024 hosts was simulated (a fat-tree network with k = 16) to show the scalability of DFFR. For the data shuffle operation test, a network of 16 hosts was simulated (a fat-tree network with k = 4) as in [6]. Since the algorithms only focus on balancing elephant flows, the flow size, which is the total amount of data to be sent by this flow, is assumed to have a minimum value of 100 MB and follows a uniform distribution. Flow bandwidth demand is uniformly distributed as suggested in [10]. In this work, its value falls in the range of (0,1] GB/s since the link capacity is 1 GB/s.

6.3. Evaluation metrics

The aggregate bisectional bandwidth, which is defined in [2] as the sum of the outgoing sending rates of all hosts, is employed as the system performance metric as it measures the total network throughput. The goal of a load balancer is to balance the flows and avoid congestion so as to maximize the throughput of the DCN. A higher aggregate bisectional bandwidth implies a higher throughput of the network, which directly reflects how well the load balancing method is performing.

In each benchmark communication suite simulation, following the approach in [2], we simulate a time period of 60 s of the network and the aggregate bisectional network bandwidth is recorded for every time interval of 0.01 s. The data points taken from the statistics collected based on the measurements within 10–50 simulated seconds of each run. Thus, the mean of these 4000 data points is the average bisectional bandwidth of the run.

The variance of these 4000 data points is also computed for comparing the stability of each algorithm. A low variance would imply that the throughput is stable throughout the experiment, meaning that congestion of links have rarely occurred. This is another measure that shows how well the load balancer is performing.

For data shuffle experiments, the maximum completion time taken among all hosts to finish their shuffling session and the completion time among the hosts are recorded. The goal of increasing throughput of the DCN is to allow jobs to finish as quickly as possible, a shorter completion time shows that jobs are completed with low latency time.

6.4. Results

Fig. 4 compares the average bisectional bandwidths of DFFR, DDFS, ECMP, and non-blocking switch for each type of traffic pattern. For most patterns, it can be seen that DFFR, DDFS, and non-blocking switch perform similarly. This is because there is minimal network congestion throughout the simulations of these three algorithms. For stride1 and stride2, DFFR performs noticeably worse than DDFS. A detailed explanation for this will be discussed in Section 6.5. ECMP performs much worse than all other algorithms in all cases but stag2. Overall, DFFR and DDFS perform equally well in terms of average bisectional bandwidth, with performance that is close to the non-blocking switch case. Both outperform ECMP by a significant margin, because ECMP is load-oblivious, making it prone to cause congestion in the DCN.

Regarding the variance of bisectional bandwidth, Fig. 5 shows that DFFR outperforms DDFS for most traffic patterns. In seven out of ten patterns, the variance of bisectional bandwidth of DFFR is significantly lower than that of DDFS, while in the other three cases, they are similar. This shows that DFFR is able to provide a more stable bandwidth. A DCN typically has a service level agreement (SLA) that promises compensation to customers if its service falls below a certain limit. Therefore, the ability to maintain a stable bandwidth of DFFR is a good quality for a practical load balancer.
every destination so that network congestion is less likely in the stride2, randbij. DFFR is designed to keep track of workload sent to is assigned one destination throughout the whole simulation (stride1, random switches with the congested outgoing ports. A switch may inform a flow through another path if its outgoing port’s usage is above a certain threshold. Another method is to utilize all arrays of counter values in path determination, so that a path is chosen not merely based on its maximum residual workload but also the residual workloads in other paths. The effectiveness of these approaches will be investigated as future work.

In addition, we have employed a simplified network model for our simulations. In particular, it is assumed that all network links are homogeneous. In the heterogeneous case, spreading the load onto all links evenly would not be the best solution. The extension of our proposed technique for heterogeneous links and different network topologies will be considered as part of the future work. Due to the symmetry of fat-tree networks, the ideal workload on all paths is to be spread equally, but this may not hold for other networks. DFFR can still be used but the expected workload of paths need to be defined differently. Our experiments have been performed using synthetic data. It would be interesting to see the performance of DFFR compared with other algorithms on real DCN data traces.

7. Conclusions

In this paper, we have proposed DFFR, a novel distributed and adaptive algorithm that can guarantee theoretical performance bounds so as to maintain a stable aggregate bisectional bandwidth in DCNs, while achieving bandwidth gains. Existing work shows that distributed algorithms typically suffer from path oscillation problems due to use of rerouting, thus requiring a long convergence period to stabilize after network state changes have occurred. DFFR overcomes this problem by maintaining information of outgoing traffic load on each outgoing link of every switch, and utilizes it for effective load balancing. DFFR achieves this effectively without performing flow rerouting and splitting, so that the performance of traffic flows, such as TCP flows, is maintained.

Our simulation results show that DFFR outperforms ECMP, demonstrating its effectiveness over static routing algorithms. DFFR can also maintain a more stable aggregate bisectional bandwidth than DDFS, another distributed flow scheduling algorithm, which helps achieve efficient and stable performance.

There are several ways that can improve the performance of DFFR. First, admission control can be used to prevent flows from being sent to switches with the congested outgoing ports. A switch may inform a lower layer switch to resend the flow through another path if its outgoing port’s usage is above a certain threshold. Another method is to utilize all arrays of counter values in path determination, so that a path is chosen not merely based on its maximum residual workload but also the residual workloads in other paths. The effectiveness of these approaches will be investigated as future work.

In addition, we have employed a simplified network model for our simulations. In particular, it is assumed that all network links are homogeneous. In the heterogeneous case, spreading the load onto all links evenly would not be the best solution. The extension of our proposed technique for heterogeneous links and different network topologies will be considered as part of the future work. Due to the symmetry of fat-tree networks, the ideal workload on all paths is to be spread equally, but this may not hold for other networks. DFFR can still be used but the expected workload of paths need to be defined differently. Our experiments have been performed using synthetic data. It would be interesting to see the performance of DFFR compared with other algorithms on real DCN data traces.

6.5. Data shuffle

A network topology with 16 hosts (fat-tree with \( k = 4 \)) was used. For each algorithm, there are ten runs for each simulation with different random seeds.

Fig. 6 shows the comparison of the mean of the maximum and mean completion time for ten data shuffle operations. The maximum completion time is the maximum time taken among all hosts to complete the process of data transmission and delivery. The mean completion time is the mean of the time taken for each host to complete its transmission process. Non-blocking takes the shortest time. DDFS takes slightly shorter time than DFFR. ECMP takes the longest time. The simulation results suggest that DDFS is better than DFFR, but it does not take the stability of the bandwidth usage into account.

6.6. Discussion

The aforementioned simulation results show that DFFR and DDFS outperforms each other in different situations.

DFFR performs relatively better for random patterns (random, randx0, randx1). In particular, the variance of bisectional bandwidth for the randx0 and randx1 patterns for DFFR are much lower than that of DDFS. This shows that DFFR is able to spread out the flows to the same destination on multiple paths by maintaining a separate record of residual workload on links for each destination, thereby allowing it to effectively prevent flow collisions under these traffic patterns.

DFFR performs relatively worse for traffic patterns where each host is assigned one destination throughout the whole simulation (stride1, stride2, randbij). DFFR is designed to keep track of workload sent to every destination so that network congestion is less likely in the destination pod. However, this advantage cannot be realized when the outgoing traffic from each host is directed to a single destination host for the simulation. In addition, when a link is empty or underloaded, the distributed load balancer in a switch or across different switches running DFFR may unintentionally put flows onto the same link synchronously, thus causing congestion.

The results of the Data Shuffle simulations mostly supports the observations from the synthesized data pattern experiments. DFFR and DDFS both completes the operation in around 20 s, outperforming ECMP and performs almost as well as a non-blocking switch. Fig. 6 shows that DFFR and DDFS has similar performance in a data shuffle operation. Both algorithms also achieve performance close to that of the ideal case of non-blocking switch.

7. Conclusions

In this paper, we have proposed DFFR, a novel distributed and adaptive algorithm that can guarantee theoretical performance bounds so as to maintain a stable aggregate bisectional bandwidth in DCNs, while achieving bandwidth gains. Existing work shows that distributed algorithms typically suffer from path oscillation problems due to use of rerouting, thus requiring a long convergence period to stabilize after network state changes have occurred. DFFR overcomes this problem by maintaining information of outgoing traffic load on each outgoing link of every switch, and utilizes it for effective load balancing. DFFR achieves this effectively without performing flow rerouting and splitting, so that the performance of traffic flows, such as TCP flows, is maintained.

Our simulation results show that DFFR outperforms ECMP, demonstrating its effectiveness over static routing algorithms. DFFR can also maintain a more stable aggregate bisectional bandwidth than DDFS, another distributed flow scheduling algorithm, which helps achieve efficient and stable performance.

There are several ways that can improve the performance of DFFR. First, admission control can be used to prevent flows from being sent to switches with the congested outgoing ports. A switch may inform a lower layer switch to resend the flow through another path if its outgoing port’s usage is above a certain threshold. Another method is to utilize all arrays of counter values in path determination, so that a path is chosen not merely based on its maximum residual workload but also the residual workloads in other paths. The effectiveness of these approaches will be investigated as future work.

In addition, we have employed a simplified network model for our simulations. In particular, it is assumed that all network links are homogeneous. In the heterogeneous case, spreading the load onto all links evenly would not be the best solution. The extension of our proposed technique for heterogeneous links and different network topologies will be considered as part of the future work. Due to the symmetry of fat-tree networks, the ideal workload on all paths is to be spread equally, but this may not hold for other networks. DFFR can still be used but the expected workload of paths need to be defined differently. Our experiments have been performed using synthetic data. It would be interesting to see the performance of DFFR compared with other algorithms on real DCN data traces.

8. References


