Managing Flow Transfers in Enterprise Datacenter Networks with Flow Chasing

Cheng Ren[1,2] and Sheng Wang[1]
[1] Key Lab of Optical Fiber Sensing and Communication, Education Ministry of China
University of Electronic Science and Technology of China, Chengdu, P. R. China
[e-mail: wsh_keylab@uestc.edu.cn]
[2] School of Electrical Engineering and Information, Southwest Petroleum University, Chengdu, P.R. China
[e-mail: rencheng@swpu.edu.cn]
*Corresponding author: Sheng Wang

Received August 18, 2015; revised November 2, 2015; revised December 3, 2015; accepted January 10, 2016; published April 30, 2016

Abstract

In this paper, we study how to optimize the data shuffle phase by leveraging the flow relationship in datacenter networks (DCNs). In most of the clustering computer frameworks, the completion of a transfer (a group of flows that can enable a computation stage to start or complete) is determined by the flow completing last, so that limiting the rate of other flows (not the last one) appropriately can save bandwidth without impacting the performance of any transfer. Furthermore, for the flows enter network late, more bandwidth can be assigned to them to accelerate the completion of the entire transfer. Based on these characteristics, we propose the flow chasing algorithm (FCA) to optimize the completion of the entire transfer. We implement FCA on a real testbed. By evaluation, we find that FCA can not only reduce the completion time of data transfer by 6.24% on average, but also accelerate the completion of data shuffle phase and entire job.

Keywords: Datacenter networks, transfer completion time (TCT), flow chasing

This work is partially supported by China's 973 Program (2013CB329103), NSFC Fund (61271165, 61301153), Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT), the 111 Project (B14039), Shanghai Oriental Scholar Program and Key Project of Sichuan Education Department(13ZA0185).

http://dx.doi.org/10.3837/tiis.2016.04.003 **ISSN : 1976-7277**

1. Introduction

Recent years, increasingly suppliers of processing virtualization and storage virtualization software have begun to flog “Big Data” in their presentations. These data sets are so large that it becomes difficult to process using on-hand database management tools or traditional data processing applications. Accordingly, more and more enterprises deploy clustering computation framework (e.g. MapReduce [1], Dryad [2], CIEL [3], and Spark [4]) in datacenter networks (DCNs) to deal with such a massive amounts of data.

Due to the high cost of datacenter operation, enterprise datacenter operators aim to maximize the network utilization and complete more jobs in a given time. To achieve this goal, several solutions have been proposed to reduce the job completion time (JCT) [5], [6], [7]. Most of these works focus on optimizing the computation and storage resource utilization, but ignoring the network resource, though the data transmission is still not a negligible part during a job execution. Some real traces show that transferring data between successive stages accounts for 33% of the running times of jobs with the reduce phase [8].

In clustering computation frameworks like MapRedcue [1] and BSP [9] (see detail in Section 3.1), a stage cannot complete (or sometimes even start) before all the date from the previous stage are received. Accordingly, we should minimize the completion time of the entire *transfer*, which is defined as a group of flows that can enable a computation stage to start/finish [8]. In the network's point of view, the minimal average transfer completion time (TCT) is the objective to pursue. To achieve this objective, we can limit the rate of a flow if its completion does not result in the completion of its transfer. The remaining network resources can be used to accelerate other flows without increasing the completion time of this transfer though the flow completion time (FCT) is prolonged. Furthermore, the bottleneck flow of a transfer, i.e. the flow completing last in a transfer, should have the priority to get more bandwidth if it does not impact the performance of other transfers. This is another way to reduce the average TCT in the network.

Though the flow relationship is referred by many works [8], [10], [11], [12], to the best of our knowledge, Orchestra [8] is the only existing work concretely focusing on optimizing the transfer completion time. It designs scheduling schemes to control the rate of each flow to minimize the TCT. However, it assumes that all the flows belonging to a transfer are sent out simultaneously, which is an impractical assumption in real-world (see Section 3.3 for details). Therefore, it is difficult to work in practice. Furthermore, it pursues the fair sharing among different transfers. In an enterprise datacenter, the operator should focuse on maximizing the job throughput, i.e. the number of jobs completed in a given time period, to reduce the operation cost. To this end, we should forget the fairness among jobs/transfers (see Section 3 for details). In this case, the algorithms proposed in [8] does not work any more. In this paper, we are to design a practical flow scheduling mechanism to minimize the average transfer completion time, and further reduce the job completion time in the enterprise datacenter networks. As far as we know, we are among the first to optimize the average TCT in enterprise datacenter networks.

The main contributions of this work are

- We study the flow characteristics in DCNs through real traces and analyze the properties that an algorithm should have in order to pursue the minimal average TCT in the network.

- We propose the flow chasing algorithm (FCA) which is practical in real-word DCNs to pursue the minimal average TCT.
- We implement FCA on a 36-node testbed. The evaluation results show that FCA reduces the average TCT in the network by about 6.24%.

This paper is organized as follows. In Section 2, we first briefly review previous works on traffic optimization in DCNs. After that, we present some statistic results which guide our work and analyze the algorithm requirements to pursue a good performance in Section 3, which is followed by the flow chasing algorithm in Section 4. The implementation and evaluation results are presented in Section 5. Finally, we conclude this paper in Section 6.

2. Related Work

Our work is to optimize the data transfer in the network to minimize the average TCT. To achieve this goal, there are largely three ways. The first one is controlling the flow routing to pursue load balance and fully utilize the network resource. The representatives adopting this way are Hedera [13], SWAN [14] and MicroTE [15]. Hedera proposed an algorithm to estimate the traffic matrix and use the Simulated Annealing algorithm to find the near optimal routing solution. SWAN and MicroTE leveraged the short term traffic forecasting to optimize the traffic routing.

The second way to optimize the flow transmission in the network is flow scheduling, which controls the flow sending sequence and minimize the average flow completion time (FCT). The representatives of this method are pFabric [16] and PDQ [10]. The main idea of both works is sending out the flow with the minimal remaining size first, which can get the minial average FCT theoretically. Unfortunately, both pFabric and PDQ require some switch modifications and hence they are not easy to deploy in the network. More importantly, neither of them consider the relationship among different flows. Only optimizing the average FCT may incur an even larger average TCT in some cases.

The last way to optimize the transfers in DCNs is controlling the available bandwidth of each flow. This is the easiest way to take the flow relationships into account. Orchestra [8] is the representative of this solution. It guarantees that at least one link is fully utilized among all the used links of a transfer and limits the rates of flows that will finish earlier than the last flow in the transfer. Orchestra is the most relative work to our study, however, it assumes that all the flows belonging to a transfer are arriving/sent out at the same time. This assumption is impractical in real networks. On the other hand, Orchestra pursues the fairness among different transfers, and hence it cannot minimize the average transfer completion time in the network, which is a more reasonable objective in enterprise networks. Due to the different objectives to pursue, Orchestra cannot be transplanted into enterprise datacenter networks where the minimal TCT is objective, with a simple modification. FCA proposed in this paper will remove the impactical assumption and utilize the minimal remaining time first principle to reduce the average TCT in the enterpirse datacenter networks.

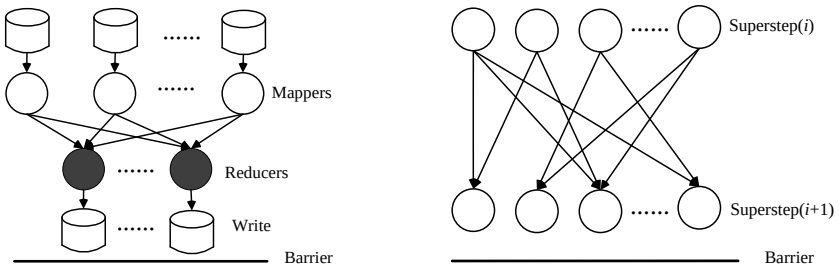
3. Problem Analysis

In this section, we deeply analyze the problem to optimize the data transfer in DCNs. At first, we briefly discuss two popular cluster computing frameworks, MapReduce and Bulk Synchronous Parallel (BSP) in Section 3.1. Based on these computing framework discussions,

we formulate the problem to solve in this paper and discuss the challenges to solve this problem in Section 3.2. Then some statistic results that motivate our study are shown in Section 3.3. After that we analyze the main properties a practical algorithm should have in order to minimize average TCT in Section 3.4.

3.1 Common Cluster Computing Framework

In this section, we briefly review the most common cluster computing frameworks MapReduce and Bulk Synchronous Parallel (BSP). Based on the characteristics of these cluster computing frameworks, we will propose the problem we should optimize in cluster computing. These two computing frameworks are shown in **Fig. 1**.



(a) MapReduce (b) Bulk Synchronous Parallel (BSP)

Fig. 1. Common cluster computing frameworks

Fig. 1(a) shows MapReduce framework. In this framework, each mapper reads its input from the Distributed File System (DFS), and performs user-defined computations. When a mapper finishes the computation, it asks the central controller to inform the corresponding reducers to pull intermediate data from itself. After collecting all the required data, a reducer merges the data from mappers and writes its output to the DFS. In this framework, there are two explicit barriers. One is at each reducer: it starts merging data when all the required data are collected. And the other one is at the end of the job: only when all the reducers finish their data merging and write the outputs to the DFS, the job is completed. When focusing on the data transfer, we should pay attention to the barrier at each reducer.

Bulk Synchronous Parallel (BSP) shown in **Fig. 1(b)** is another well-known framework in cluster computing. In this framework, a job is executed in a serial of supersteps. Each superstep contains three ordered stages: *concurrent computation*, *communication* between VMs and *barrier synchronization*. In concurrent computation stage, the input of each processor is stored in the local memory of this processor and the computations are independent with other processors. Then the processors exchange data between themselves in the communication stage. When a process arrives at the barrier synchronization stage, it waits until all other processors have finished their communication actions. The start of next superstep is determined by the finish of the last flow in the communication stage.

3.2 Problem Formulation and Discussion

Based on above discussion, we know that the flows in the computation cluster are dependent on each other and the completion of a single flow does not make too much sense. Accordingly, we should try to speed up the completion of the entire group of flows that

determines the start/finish of its next computation stage, i.e. to speed up the completion of a transfer. In the network wide, we should minimize the average TCT. Formally, the problem can be formulated as following optimization problem:

$$\text{minimize} \quad \sum_{T \in \boldsymbol{\Gamma}} (F_T - S_T) \tag{1}$$

$$\text{Subject to:} \quad S_T \le s_f \quad \forall T, f \in T \tag{2}$$

$$F_T \ge t_f \quad \forall T, f \in T \tag{3}$$

$$\int_{s_f}^{t_f} b_f(t)dt \ge v_f \quad \forall f, t \tag{4}$$

$$u_l(t) = \sum_{f: l \in P_f} b_f(t) \quad \forall l \in \boldsymbol{E}, t \tag{5}$$

$$u_l(t) \le c_l \quad \forall l \in \boldsymbol{E}, t \tag{6}$$

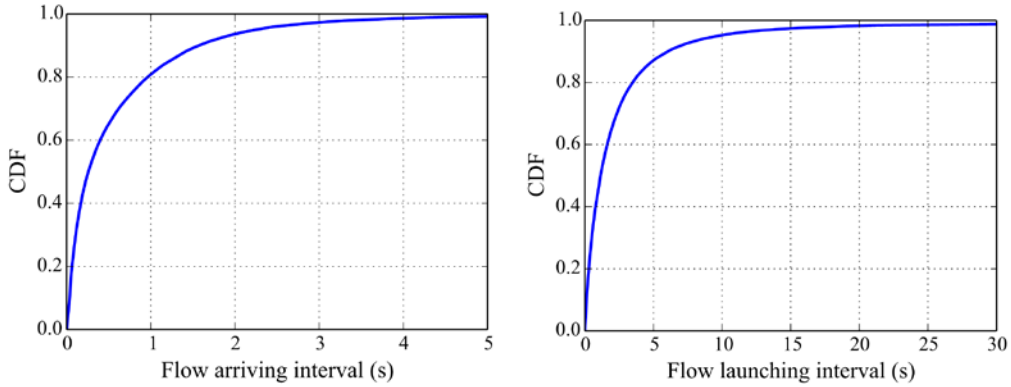
In above formulation, $\boldsymbol{\Gamma}$ is the set of all the transfers in the enterprise datacenter networks, F_T and S_T are the finish time and start time of transfer T, respectively. Hereby, (F_T - S_T) is the completion time of transfer T. The objective (1) is to minimize the average completion time (i.e. minimize the sum of completion time of all the transfers) of all the transfers in the network. In (2) and (3), s_f and t_f are the start and finish time of flow f, respectively. These two constraints say that a transfer starts before all the flows in it begin, while finishes after all the flows in it complete. In (4), $b_f(t)$ is the bandwidth assigned to flow f at time t and v_f is the volume of flow f. This constraint is used to ensue all the flow data is served during s_f to t_f. In (5) and (6), $u_l(t)$ is the total traffic rate on link l, c_l is the capacity of link l, and P_f is the set of link on the path that traversed by flow f. (5) calculates the traffic rate on link l, while (6) indicates that the traffic rate on each link cannot exceed the link capacity.

This problem is difficult to solve, since (I) the arriving time and volume of each flow are random variables in real system, and the programming constraint is for infinite time instants. Therefore, we cannot really get above programming problem even for analysis purpose; (II) the flows in each transfer may not arrive simultaneously. Therefore, we cannot optimize the completion time for each single transfer; (III) even the completion time of each transfer is obtained, how to scheduling these transfer in an online system is still an NP-hard problem [17]; (IV) we do not know the number of transfers in the network. Due to these challenges, we only try to design an online heuristic to optimize the average TCT in the network.

3.3 Statistic Results

In this section, we present some statistic results based on the real trace by running a terasort job (with size 200G) in our Hadoop system. The testbed to generate this trace uses the same configuration detailed in Section 5.1.

3.3.1 Flow Arriving Interval



(a) Flow arriving interval of a reducer (b) Flow arriving interval of a mapper

Fig. 2. Statistic results of flow arriving interval

Ideally, all the flows belonging to a transfer are arriving simultaneously. In this case, we can easily calculate the bandwidth assigned to each flow to optimize the TCT. Unfortunately, this is not the case. **Fig. 2** shows the CDF of flow arriving intervals in a transfer. If we treat all the flows to a reducer as a transfer (see **Fig. 2(a)**), about half of the flow arriving intervals are larger than 200 ms, which is a large time interval and cannot be ignored in a high throughput low latency network (the RTT is usually less than 400 as in [18]). If we treat all the flows from a mapper as a transfer (see **Fig. 2(b)**), 20% of the flow arriving intervals are larger than 2 s. Even worse, some flow arriving intervals are more than 30 s. To hold flows in the network for such a long time is impractical, though the last flow blocks the completion of the whole transfer.

3.3.2 Flow Completion Time

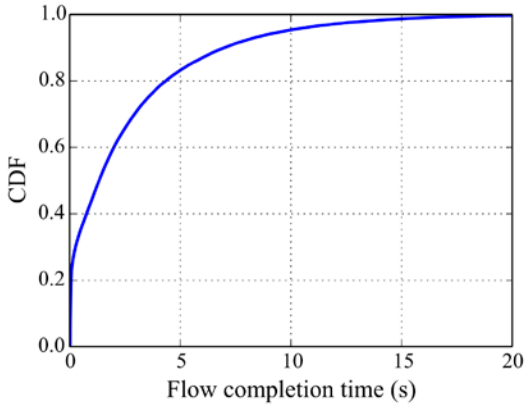


Fig. 3. CDF of flow completion time

Fig. 3 shows the CDF of the flow completion time. From this figure we see that about 60% of the flows finish in 2 s. In other words, most of the flows in DCNs complete very quickly. If we hold a flow to wait for the coming of next flow in the same transfer, in 30% of the cases, about 10% of the flow completion time (say half of the flows wait for at least 200 ms) is

wasted on waiting, which is non-ignorable for a flow, let alone the case that every flow waits for all the flows in the same transfer.

3.3.3 Traffic Volume for a Transfer

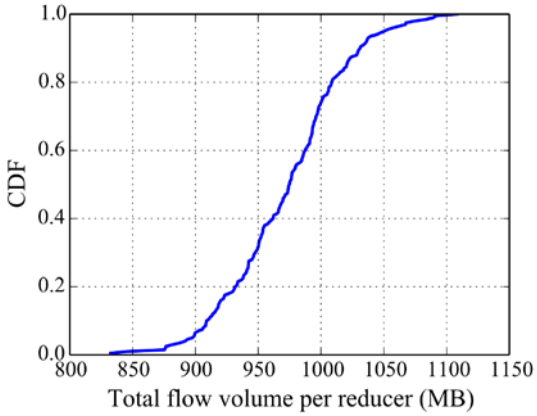


Fig. 4. CDF of total flow volume of a reducer

Fig. 4 shows the CDF of the total traffic volume that a reducer receives before it starts its computation. It implies that most of the reducers receive more than 900 MB before they start working. When all the mappers are evenly distributed in the datacenter, it costs at least 24 s to transfer the data for a reducer even if all the bandwidth is occupied by the flows to this reducer. If we run a larger job, the condition will be even worse. Accordingly, if we collect all the flows belonging to a transfer and send them out simultaneously, it would bring network congestions and incur TCP retransmissions.

3.4 Optimality Analysis

Based on the discussion above, we identify the necessary characteristics of an algorithm to optimize the data transfer phase:

● *Necessity to be an online algorithm*. In DCNs, most of the flows are bursting and the traffic rate on a link is only relatively stable in 1 s scale [14]. As the decomposition of the traffic rate on each link, the traffic matrix is more dynamic. Therefore, it is difficult to forecast the traffic matrix in DCNs. On the other hand, as shown in Fig. 2, even the arriving intervals of flows from the same mapper or to the same reducer are distributed in a large range. Accordingly, we cannot assume that we have all the flow information of every transfer before its flows enter the network, in which case we can optimize the data transfer statically.

● *Work conserving property*. There is idle bandwidth only when there is no traffic to send. For any flow, without degrading the performance of other flows, it should send out bytes as fast as possible. This is a way to fully utilize the network resource and reserve more resource for the further flows.

●*Focus on the bottleneck flow in a transfer*. As discussed in Section 3.1, in those cluster computing frameworks with explicit barriers, the start of some computing stage is determined by the completion of the bottleneck flow in the transfer. Without any information in the transfer level, we can only treat the coming flow as the last flow of a transfer. Under this

assumption, more bandwidth should be assigned to the bottleneck flow to accelerate the transfer completion.

●*Minimal remaining transfer first principle*. As studied in [8, 10, 16, 19], the minimal remaining flow first achieves the minimal average flow completion time. This conclusion is easily extended to the transfer level, i.e. the minimal remaining transfer first principle would achieve the minimal average TCT. In other words, if all the flows belonging to a transfer arrive at the same time, the transfer with the minimal remaining traffic volume should have the higher priority to preempt the network resource.

4. Algorithm Design

In this section, we present Flow Chasing Algorithm (FCA) to optimize the data transfer in detail. In Section 4.1, we first identify the key idea of FCA and show the algorithm framework. After that, the rate control algorithms for intra- and inter- transfer flows are introduced in Section 4.2 and Section 4.3, respectively. At last, we present some discussions on how to implement our framework in the real system in Section 4.4.

4.1 Flow Chasing Framework

The main idea of FCA is to limit the rate of flows that are not bottleneck flows, while assign more bandwidth to the bottleneck flows. **Fig. 5** shows why this method works. In this example, there are two transfers (say T_a and T_b) and each of them has two flows. All the flows have 30 Mbit to transmit and the route of each flow is as shown in the figure. Suppose all the flows enter in a very short time slot and all the link capacity is 30 Mbps except (s_1, d_1) whose capacity is 5 Mbps, (s_2, d_2) is 10 Mbps and (s_3, d_3) is 15 Mbps. On the common link (s_3, d_3), if these two flows share the bandwidth fairly, the completion time of the two transfers is 6 s (T_a) and 4 s (T_b), respectively. However, if the traffic rate of f_{a2} is limited to 5 Mbps, and the remaining bandwidth is assigned to flow f_{b2}, the completion time of two transfers are 6 s and 3 s, respectively, which is better than that uner the fair sharing policy.

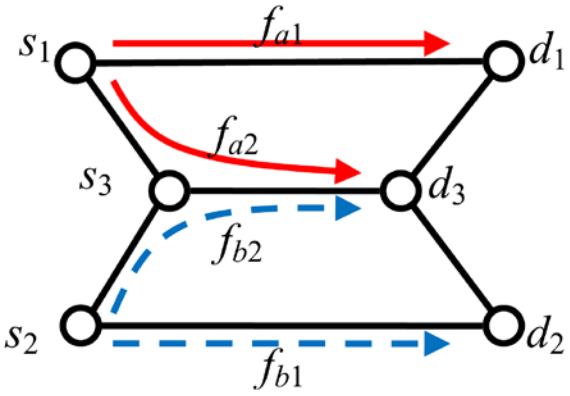


Fig. 5. Why rate control works

The performance improvement originates from the fact that the bottleneck of transfer T_a is f_{a1}, complete f_{a2} earlier cannot improve the performance of the entire transfer. Accordingly, we can limit its rate and leave more bandwidth to f_{b2}, the bottleneck of transfer T_b, to speed up the transfer. Therefore, limit the rate of flows that are not bottleneck flows and assign more bandwidth to the bottleneck flows can reduce the average TCT in the network.

In addition, consider another case that transfer T_a arrives at time 0 s, while transfer T_b arrives at time 2 s, and the capacity of link (s_2, d_2) is 15 Mbps. When transfer T_a arrives, f_{a1} is the bottleneck of transfer T_a. We can myopically limit the traffic rate of flow f_{a2} to be 5 Mpbs. If so, when the transfer T_b arrives, there are only 10 Mbps capacity leaving to flow f_{b2} on link (s_3, d_3) and flow f_{b2} becomes the bottleneck of transfer T_b. In this case, the completion time of two transfers are 6 s and 3 s, respectively. If we let f_{a1} to monopolize all the capacity on link (s_3, d_3) from time 0 s to 2 s, it can finish at time 2 s. Though this scheme cannot reduce the completion time of transfer T_a, f_{b2} monopolizes link (s_3, d_3) when tranfer T_b arrives, which speed up the completion of transfer T_b. By this method, the completion time of two transfers are 6 s and 2 s, respectively. The performance improvement is due to the fact that the second scheme pursues work conserving when assigning the network bandwidth, which can bring benefit to the flows/transfers that will come later.

Based on these discussions, the key points of FCA are:

● When a flow enters, try to assign enough bandwidth to it with the purpose to catch up with the completion of the bottleneck flow in its transfer. (This is why we call the algorithm as flow chasing.)

● If the new coming flow cannot catch up with the bottleneck flow in its transfer, limit the rates of other flows that are not transfer bottleneck flows in the transfer to save bandwidth.

● Flows should compete for the residual bandwidth. It is to pursue the work conserving property and further optimize the average TCT in the network by bringing benefit to the flows/transfer coming later.

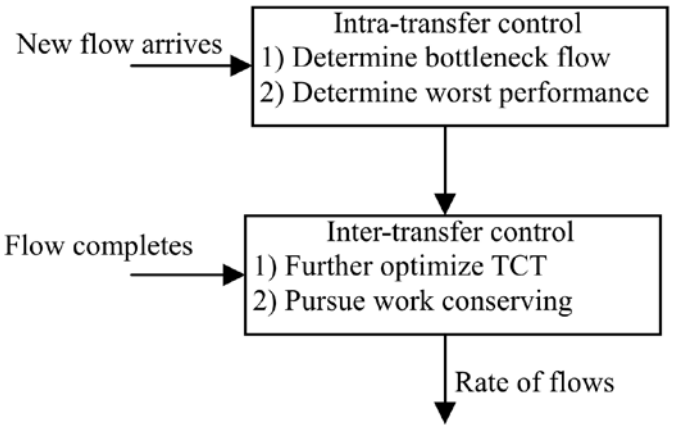


Fig. 6. Framework of FCA

The first two bullets are for the intra-transfer rate control while the last bullet is for the inter-transfer scheduling. Therefore, the framework of FCA can be shown as in **Fig. 6**. When a flow enters, FCA first does the intra-transfer control to determine the worst completion time of its transfer and to shrink traffic rates to save bandwidth. After that, inter-transfer control is triggered to assign the residual bandwidth to different transfers/flows and further optimize average TCT in the network. When a flow completes, only the inter-transfer control should be triggered to fully utilize the bandwidth released by the completed flow. In the next two subsections, we introduce intra- and inter- transfer rate control of FCA in detail.

4.2 Intra-transfer Flow Rate Control

Algorithm 1: Intra-transfer rate control

Require: Coming flow f, all flows in the network $\{f\}$ and transfers $\{T\}$

Ensure: Flow rate of all flows in the network $\{r_f\}$

1: **for** all transfer T in the network **do**
2: $g = \arg\max_{h \in T} h.completionTime$
3: **for** all flows h in T **do**
4: $r_h = \frac{h.remainingVolume}{g.completionTime - SYSTEMTIME}$, $h.completionTime = g.completionTime$
5: **end for**
6: **end for**
7: Calculate reduce bandwidth on each link $\{b_l\}$
8: $b_{fmax} = \min_{l \in P_f} b_l$
9: **if** $g.completionTime < SYSTEMTIME + \frac{f.volume}{b_{fmax}}$ **then**
10: $//g$ is a flow in the same transfer as f.
11: $r_f = b_{fmax}$, $f.completionTime = SYSTEMTIME + \frac{f.volume}{b_{fmax}}$
12: Update $r_g = \frac{g.remainingVolume \times b_{fmax}}{f.volume}$, $g.completionTime = f.completionTime$ for all $g \in T$
13: **else**
14: $r_f = \frac{f.volume}{g.completionTime - SYSTEMTIME}$
15: **end if**
16: Update residual bandwidth on each link
17: **return** $\{r_f\}$

The algorithm to control the rates of flows in a transfer when a flow enters is shown in Algorithm 1. The key sprite of this algorithm is: (I) first minimize the completion time of the transfer of the coming flow with the assumption that the coming flow is the last flow of its transfer, and (II) then minimize the bandwidth consumption without enlarging the TCT. Line 1 to Line 6 are used to shrink the flow bandwidth to leave more bandwidth for the coming flow. After the operations in Line 1-6, all the flows in a transfer should get the bandwidth that makes them complete simultaneously, i.e. the bandwidth of all the flows that will complete before the bottleneck flow is shrank. The “if” expression from line 9 to line 15 is the core of Algorithm 1. If the coming flow is the bottleneck (as the “if” expression shows, it cannot complete before its transfer), it gets as much residual bandwidth as possible without impacting the performance of other flows. In this case, we further shrink the bandwidth of other flows in the same transfer to match the completion time of the coming flow (Line 12). Otherwise, the coming flow only gets the appropriate bandwidth to chase the bottleneck flow of its transfer (Line 13). It is worth noting that after intra-transfer rate control, the transfer containing the coming flow achieves the minimal TCT under current network utilization condition, while other transfers hold their performance.

4.3 Inter-transfer Flow Rate Control

Intra-transfer flow rate control determines the lower bound of the completion time of each transfer, however, such a performance can be further improved since there is residual bandwidth, especially when some flow completes and release its network resources. Furthermore, pursuing the work conserving property could speed up the completion of some flows and benefit the later flows.

There are two steps to pursue the work conserving. The first step is to check whether any transfer's completion can be sped up by utilizing the residual bandwidth, while the second step is to speed up flows to fully utilize the network resource. Due to the minimal remaining transfer first principle, the transfer with the smaller remaining size has the higher priority to preempt the residual bandwidth in both steps.

In the first step, we pick out the unchecked transfer with the minimal remaining volume, say transfer *T*, to calculate its minimal completion time by preempting residual bandwidth. To this end, we solve the following programming:

$$\text{Objective:} \quad \text{minimize} \quad t \tag{7}$$

$$\text{Subject to:} \quad u_l = \sum_{f \in \boldsymbol{T}: l \in P_f} \left(\frac{r_f}{t} - b_f\right) \quad \forall l \in \boldsymbol{E} \tag{8}$$

$$u_l \le c_l \quad \forall l \in \boldsymbol{E} \tag{9}$$

where P_f is the set of links on f's path, r_f is the remaining volume of f, b_f is the bandwidth of f determined by the intra-transfer rate control and c_l is the residual bandwidth on link l. Though this is not a standard convex optimization problem, it can be solved by water filling algorithm [20] easily. Then the rate of each flow f belonging to transfer $\boldsymbol{T}$ can be increased by $r_f / t - b_f$.

In the second step, we allocate the residual bandwidth to the flows in the network though it cannot reduce the TCT further, however, it guarantees that the network resource is fully utilized and benefits the flows coming later. Accordingly, we check each flow, say flow *f*, in the network if there is any residual bandwidth for it to complete earlier by calculating

$$BF = \min_{l \in P_f} b_l \tag{10}$$

If $BF>0$, we increase f's rate by BF. All the flows are checked in the order of the minimal remaining transfer first, and the minimal remaining volume flow first in each transfer.

Whenever a flow completes, more residual bandwidth is released and we assign it to other online flows in the network for the work conserving purpose. To this end, only the inter-transfer rate control needs to be triggered. To reduce the computation complexity, we only consider the transfers that have common links with the completed flow (an alternative improvement is recalling the inter-transfer rate control in batch).

4.4 Discussions

To implement FCA in a real system, there are remaining some issues. The first one is on the flow volume. Since in real system, some of the flows are sent out while part of the data are still being generated. In other words, we cannot get the flow volume information exactly. To solve this problem, we assume the flow volume is only as large as the amount that is already generated. When there are new data generated, we treat the new generated data as a new arriving flow and trigger FCA to update the bandwidth allocation.

Another issue is that there are too many mice flows (usually less than 50 KB) in the network. If we trigger FCA whenever such flows arrive, it may be a too large system overhead. Consider that most of these flows are delay sensitive, we can only set a higher priority to these mice flows and let them be served first. Only when the volume of arriving flow is larger than a predefined threshold, we use FCA to optimize the average TCT in the network.

5. Implementation and Evaluation

In this section, we evaluate FCA by implementing it on our testbed. In Section 5.1, we first show the configuration of our testbed. Then, we present the technical details of FCA's implementation in Section 5.2. In the end, we evaluate FCA in Section 5.3.

5.1 Testbed

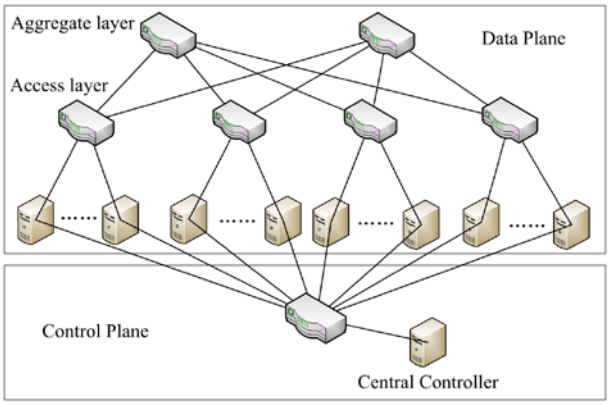


Fig. 7. Simulation and implementation topology

Our testbed is formed by 37 servers and 7 switches as **Fig. 7** shows. In the data plane, 6 of the switches form a leaf-spine topology (2 spine switches and 4 leaf/ToR switches), while there are 9 servers connected to each leaf/ToR switch. The remaining server and switch are used to form the control plane to run FCA. All the 37 servers are HP PowerEdge R320 with a quad core Intel Xeon E5-1410 2.80GHz CPU and 8GB memory and are running Debian GNU/Linux 6.0. To collaborate with our low-throughput HDDs, the bandwidth capacity of each link is limited to 300Mbps. All the switches are Pronto 3295 switches, which run PicOS 2.0.4 in OpenFlow mode. All the evaluations are based on the results of real job executions in a 0.20.2 Hadoop system.

5.2 Implementation

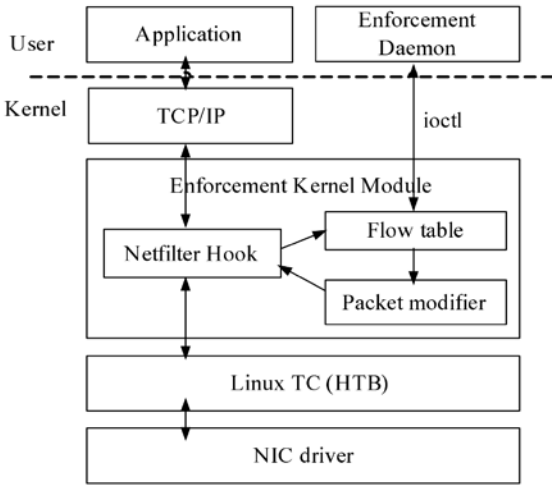


Fig. 8. Software stack of FCT's bandwidth enforcement

For load banlancing purpose, we randomly select a route for each flow in the testbed and use Openflow based explict routing to control the traffic. With this method, the routing result will be similar to the ECMP. The architecture of FCT's bandwidth enforcement is shown in **Fig. 8**. The enforcement daemon at the user space communicates with the kernel module via ioctl to manage the flow table. The kernel module, locating between TCP/IP stack and TC, intercepts all outgoing packets and modifies nfmark field of socket buffer based on the rules in flow table. The modified packets are then delivered to TC for rate limiting. We leverage two-level Hierarchical Token Bucket (HTB) in TC: the root node classifies packets to their corresponding leaf nodes based on nfmark field and the leaf nodes enforce per-flow rates.

Whenever a flow enters, its source host sends its flow information to the central controller. During the three-way handshaking of each flow, central controller can execute FCA and update the rate of all the flows in the network. Unfortunately, due to the enforcement delay and flow rate unstablity, we cannot exactly maintain the remaining volume of each flow. To simplify the implementation, we use the total volume of a transfer/flow to determine which transfer/flow has the higher priority to preempt the bandwidth instead of their estimated remaining volumes, when executing the inter-transfer rate control. Similarly, when a flow completes, its source host should report its finish to the central controller to trigger the inter-transfer rate control procedure.

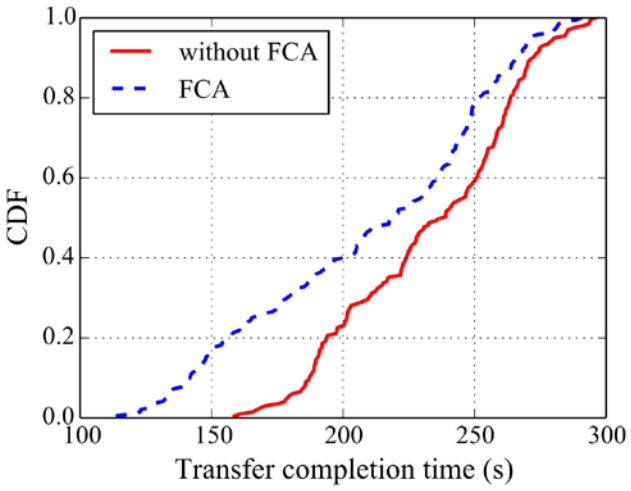


Fig. 9. CDF of TCT

5.3 Performance Evaluation

In this subsection, we evaluate the flow chasing algorithm (FCA) in three ways by running a 200G terasort job on our testbed: to speed up the reducer completion, to speed up the shuffle phase and to speed up the job completion.

5.3.1 Reducer Completion

In our evaluation, a transfer is defined as all the flows to a reducer since a reducer cannot enable its merging operation before all the data it requires are collected. Accordingly, the reducer completion time directly reflects the performance of FCA. Since we cannot guarantee a reducer executing the same computation have the same ID even if we run the same job twice, we only count the CDF of the transfer completion time before and after FCA is enabled in the network. **Fig. 9** shows the performance of FCA. From the figure we see that the CDF curve of

TCT in the FCA-enabled network is on the left of that without FCA, it implies that FCA reduces the average TCT in the network. The reason is that FCA can save bandwidth without degrading the performance of any transfer. Actually, the average TCT in the network with FCA implemented is 217.52s, while the baseline is 232.00s. About 6.24% performance improvement is brought by FCA.

5.3.2 Shuffle Completion

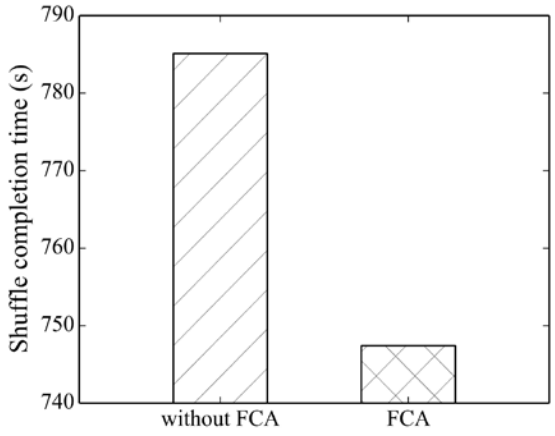


Fig. 10. Shuffle Completion Time

Intuitively, optimizing the completion of each transfer further shrinks the shuffle phase of a job. **Fig. 10** shows the shuffle completion time of a job with and without FCA. It implies that FCA saves about 4.8% (about 38 s) of the shuffle phase.

5.3.3 Job Completion

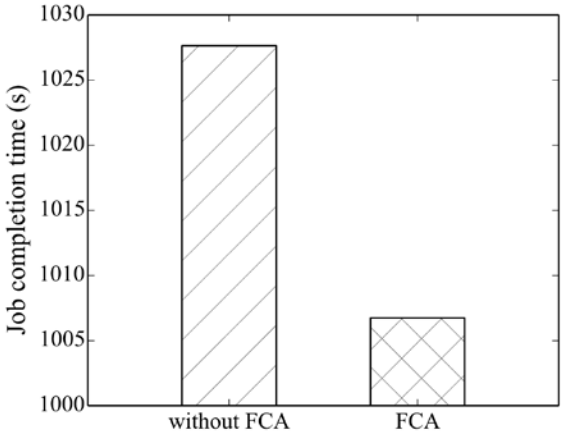


Fig. 11. Job Completion Time

Fig. 11 shows the benefit FCA brought to the job completion time (JCT). From the figure, we find that our algorithm can improve the job completion time by 2.00% (about 21 s). It is very strange that the absolute time saved in a job is even less than that saved in the shuffle

phase. This is because that the CPU is not powerful enough to deal with the data as soon as possible though the data transfer is completed.

6. Conclusion

We studied the flow transfer management problem in datacenter networks in this paper. An efficient flow chasing algorithm (FCA) is proposed to optimize the flow transfer. Since the completion of the flow transfer is limited to the completion of the bottleneck flow, we shrink the rates of non-bottleneck flows to save bandwidth and accelerate the transmission of the latter flow of a transfer to reduce its transfer completion time. We also implement FCA on a real testbed. By evaluation, we find that FCA can not only reduce the average transfer completion time (TCT) in the network, but also further reduce the shuffle phase and speed up the job completion.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, 107-113, 2004. Article (CrossRef Link)

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," in *Proc. of ACM EuroSys*, Lisboa, Portugal, pp. 59-72, 2007. Article (CrossRef Link)

[3] D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand, "CIEL: a universal execution engine for distributed data-flow computing," in *Proc. of USENIX conference on Networked systems design and implementation*, pp. 113-126, 2011. Article (CrossRef Link)

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proc. of USENEX HotCloud*, Boston, MA, USA, 2010. Article (CrossRef Link)

[5] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, *et al.*, "Reining in the outliers in map-reduce clusters using Mantri," in *Proc. of the 9th USENIX conference on Operating systems design and implementation*, Vancouver, BC, Canada, 2010. Article (CrossRef Link)

[6] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proc. of the ACM SIGOPS 22nd symposium on Operating systems principles*, Big Sky, Montana, USA, 2009. Article (CrossRef Link)

[7] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," in *Proc. of ACM EuroSys*, Paris, France, pp. 265-278, 2010. Article (CrossRef Link)

[8] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in *Proc. of ACM SIGCOMM*, Toronto, Ontario, Canada, 2011. Article (CrossRef Link)

[9] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, *et al.*, "Pregel: a system for large-scale graph processing," in *Proc. of the 2010 ACM SIGMOD International Conference on Management of data, Indianapolis*, Indiana, USA, 2010. Article (CrossRef Link)

[10] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing Flows Quickly with Preemptive Scheduling," in *Proc. of ACM SIGCOMM*, Helsinki, Finland, 2012. Article (CrossRef Link)

[11] M. Chowdhury and I. Stoica, "Coflow: A Networking Abstraction for Cluster Applications," in *Proc. of ACM Hotnets*, Seattle, WA, USA, 2012. Article (CrossRef Link)

[12] ND. Han, Y. Chung, M. Jo, "Green data centers for cloud-assisted mobile ad hoc networks in 5G," *IEEE Network*, Vol.29, No.2, pp. 70-76, April 2015. Article (CrossRef Link)

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. of USENIX NSDI*, Berkeley, CA, USA, 2010. Article (CrossRef Link)

[14] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, *et al.*, "Achieving High Utilization with Software-Driven WAN," in *Proc. of ACM SIGCOMM*, Hong Kong, 2013. Article (CrossRef Link)

[15] Theophilus Benson, Ashok Anand, Aditya Akella, and M. Zhang, "MicroTE Fine Grained Trafic Engineering for Data Centers," in *Proc. of ACM CoNEXT*, 2011. Article (CrossRef Link)

[16] M. Alizadeh, S. Yang, M. Sharif, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal Near-Optimal Datacenter Transport," in *Proc. of ACM SIGCOMM*, 2013. Article (CrossRef Link)

[17] Mosharaf Chowdhury, Yuan Zhong, and I. Stoica, "Efficient Coflow Scheduling with Varys," in *Proc. of ACM SIGCOMM*, Chicago, IL, USA, 2014. Article (CrossRef Link)

[18] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Datacenter Traffic: Measurements & Analysis," in *Proc. of ACM IMC*, Chicago, Illinois, USA, pp. 202-208, 2009. Article (CrossRef Link)

[19] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-Aware Datacenter TCP (D2TCP)," in *Proc. of ACM SIGCOMM*, Helsinki, Finland, pp. 115-126, 2012. Article (CrossRef Link)

[20] J. Walrand and S. Parekh, *Communication Networks: A Concise Introduction*, 2010. Article (CrossRef Link)

Cheng Ren received her master degree in circuit and system from University of Electronic Science and Technology of China (UESTC), Chengdu, China in 2006. Now, she is a Ph.D. candidate in Communication Engineering in UESTC. Her research interests include software-defined networking and network resource allocation.

Sheng Wang received the B.S., M.S., and Ph.D. degrees in Electrical Engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 1992, 1995, and 1999, respectively. He is now a Professor and a Research Group Leader in UESTC. His research interests include network design, optical switching, and next generation networks.