RAPIER: Integrating Routing and Scheduling for Coflow-aware Data Center Networks

*Yangming Zhao (UESTC), Kai Chen, Wei Bai, Minlan Yu (USC), Chen Tian (HUST),
Yanhui Geng (Huawei), Yiming Zhang (NUDT), Dan Li (Tsinghua), Sheng Wang (UESTC)
SING Group, HKUST

***Abstract*—In the data flow models of today's data center applications such as MapReduce, Spark and Dryad, multiple flows can comprise a *coflow* group semantically. Only completing all flows in a coflow is meaningful to an application. To optimize application performance, routing and scheduling must be jointly considered at the level of a coflow rather than individual flows. However, prior solutions have significant limitation: they only consider scheduling, which is insufficient.**

To this end, we present RAPIER, a coflow-aware network optimization framework that seamlessly integrates routing and scheduling for better application performance. Using a small-scale testbed implementation and large-scale simulations, we demonstrate that RAPIER significantly reduces the average coflow completion time (CCT) by up to 79.30% compared to the state-of-the-art scheduling-only solution, and it is readily implementable with existing commodity switches.

I. INTRODUCTION

Cluster computing frameworks such as MapReduce [1], Dryad [2], Spark [3] and so on have become the mainstream platforms for data processing and analysis in today's cloud services. A common feature of these different computing paradigms is that they all implement a data flow computing model, in which a group of data flows need to pass through a sequence of intermediate processing stages before generating the final results. These intermediate flow transfers can account for more than 50% of job completion time [4], and have a significant impact on job performance. Therefore, optimizing such flow transfers is important for applications.

The term *coflow* is defined as the set of all flows transferring data between two stages of a job [5]. To optimize application performance, we need to optimize flow transfers at the level of coflow rather than individual ones. This is because the job completion time depends on the time it takes to complete the entire coflow, instead of the time to complete individual flows composing it. For example, in MapReduce [1] and BSP [6], a stage cannot complete, or sometimes even start, before it receives *all* the flows in a coflow from the previous stage. From an application's perspective, when a stage is pending for the input data, the CPU often sits idle or is under-utilized. As a result, reducing the coflow completion time (CCT) can further improve CPU utilization, maximizing application performance and job throughput in a given time period.

To minimize average CCT, both routing and scheduling must be considered simultaneously (see Section II for details).

*This work was performed when Yangming Zhao was an intern student at the SING Group of HKUST under supervision of Prof. Kai Chen.

However, prior solutions for network flow optimization such as [4, 7]–[13] have significant limitations (Table I): some of them (e.g., [7]–[11]) are ineffective, because they are coflow-agnostic that do not account for collective behaviors of flows belonging to a coflow; existing coflow-aware solutions (e.g., [4, 12, 13]) are insufficient, because although these approaches improve by starting to consider coflow level semantics, they only focus on scheduling while neglecting an indispensable component—routing. We show in Section V that this can directly lead to 63.37% performance loss.

Related work	Coflow-aware	Routing	Scheduling
pFabric [8], PDQ [9], Pase [10], D3 [11], etc.	No	No	Yes
Varys [12], Baraat [13] Orchestra [4]	Yes	No	Yes
RAPIER	Yes	Yes	Yes

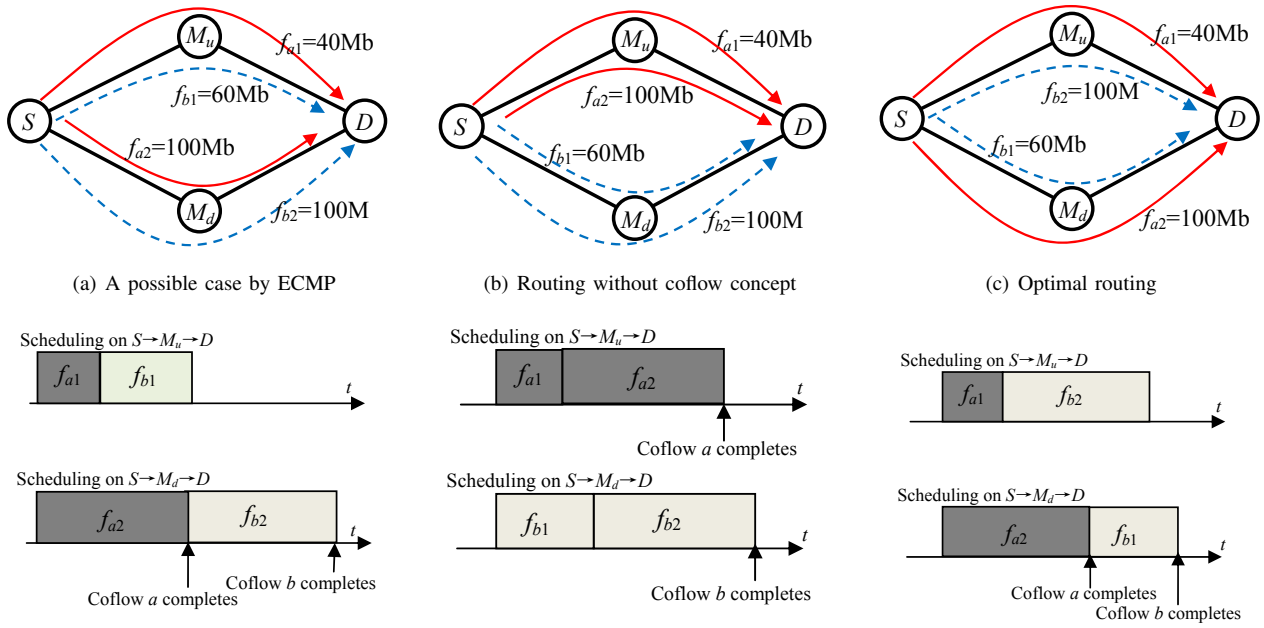
TABLE I
SUMMARY OF PRIOR SOLUTIONS AND COMPARISON TO RAPIER

Motivated by this situation, we design and implement RAPIER, a coflow-aware network optimization system for data center networks (DCNs). To improve average CCT, RAPIER seamlessly combines routing and scheduling together by formulating it as a joint optimization model. This model is a nonlinear programming and contains integer variables; it is impossible to be directly solved. Accordingly, we propose an efficient heuristic to approximately solve this problem based on the relaxation of the model.

We evaluate RAPIER using a small-scale testbed implementation as well as large-scale simulations. Our evaluation results show that RAPIER can reduce average CCT by up to 79.30% and 60.43%, compared to the scheduling-only and routing-only schemes respectively. Our implementation verifies that RAPIER can be readily implementable with existing commodity switches.

In summary, the main highlights of this paper include:

- A key observation that both routing and scheduling must be jointly considered for optimizing average CCT. (Section II)
- A coflow-aware network optimization solution, RAPIER, which takes into account routing and scheduling simultaneously for the first time. In the course of system design, we also develop fast and efficient online algorithms to approximately solve theoretical NP-hard problems. (Section III and Section IV).



(a) A possible case by ECMP (b) Routing without coflow concept (c) Optimal routing

(d) Optimal scheduling up on (a) (e) Optimal scheduling up on (b) (f) Optimal scheduling up on (c)

Fig. 1. A motivating example, where (a)∼(c) show different routing schemes, and (d)∼(f) show the optimal scheduling schemes for (a)∼(c).

- A real testbed implementation and extensive large-scale simulations. (Section V)

II. A Motivating Example

In this section, we make key observations through a motivating example in Fig. 1. In this example, there are two coflows: Coflow a has flows f_{a1} and f_{a2} with the sizes of 40Mb and 100Mb respectively; Coflow b has flows f_{b1} and f_{b2} with the sizes of 60Mb and 100Mb respectively; and the link bandwidths are all 100Mbps. As a reference point, the optimal average CCT of this example should be 1.3s.

The first takeaway is that *scheduling alone is not sufficient to optimize average CCT*. When the routing is fixed, good scheduling can minimize the average CCT by determining the sequence of flows to send out traffic. Fig.1(a) shows a case of randomized routing by equal-cost multipath (ECMP). With a naive scheduling such as fair sharing, both coflows are dominated by path $S \rightarrow M_d \rightarrow D$ hence their CCT are both 2s. If using the optimal scheduling shown in Fig.1(d), the CCTs for two coflows become 1s and 2s respectively; apparently, scheduling does play a critical role. However, the average CCT (in this routing) is only 1.5s, which still has a 0.2s gap to the real optimal value 1.3s. It is clear that routing should also play a critical role: the loads of two paths in Fig.1(a) are severely unbalanced, where path $S \rightarrow M_d \rightarrow D$ has a traffic load doubles that of path $S \rightarrow M_u \rightarrow D$.

The second takeaway is that *considering routing and scheduling separately cannot optimize average CCT*. As an example in Fig.1(b), a load-balancing routing results in: both flows of Coflow a are routed on $S \rightarrow M_u \rightarrow D$ while the flows of Coflow b on $S \rightarrow M_d \rightarrow D$; now the network is more balanced. However, the optimal CCTs for coflows a and b in this case are 1.4s and 1.6s respectively (see Fig.1(e)); the average CCT 1.5s is still not optimal. The reason is that flows of the same coflow are routed through the same path, which leaves little space for scheduling to take effect for reducing the average CCT.

The conclusion is that *both routing and scheduling must be jointly considered in order to optimize average CCT*. In our example, the minimal average CCT can be achieved by combining the routing in Fig.1(c) and scheduling in Fig.1(f). In this case, the CCTs of two coflows are 1s and 1.6s respectively, and the average CCT is minimized. This motivates our design below.

III. Design Overview

Rapier optimizes the average CCT in data-intensive DCNs by coordinating routing and scheduling flows in the networks. Given each coflow with information about its individual flows, such as flow sizes, and sources/destinations, Rapier determines *which paths* to carry these flows, *when* to start them, and *at what rate* to serve them, in order to optimize the average CCT of all the coflows in the networks.

Inspired by [4, 12], we design Rapier to work in a centralized, cooperative manner. This decision is also coherent with many recent centralized data center designs such as [1, 7, 14]–[18], etc. As prior works [8, 9, 11]–[13], Rapier assumes that the information about a coflow can be readily derived from upper layer applications [5] or using state-of-the-art prediction techniques [19].

A. Desirable Properties

We identify the following goals when designing Rapier.

- *Scalability*: Rapier is necessarily an online system. Up on a new incoming coflow, the Rapier algorithms must

Algorithm 1: The RAPIER Framework

```
1:  Procedure MinimizeMeanCCT(Coflows Ω, Bandwidth R)
2:  Sort all the coflows in Ω non-increasingly according to their waiting time; Ω* ← Ω
3:  while Ω* ≠ Φ do
4:    T_min ← ∞, C_min ← Φ;
5:    for C ∈ Ω* do
6:      T_C=MinimumCCT(C, R); /* compute the minimum completion time for coflow C, and the corresponding routing and rate allocation */
7:      if C.waitTime() > δ then
8:        T_min ← T_C, C_min ← C;
9:        break;
10:     end if
11:     if T_C < T_min then
12:       T_min ← T_C, C_min ← C;
13:     end if
14:   end for
15:   Ω* ← Ω* \ C_min;
16:   Assign all the flows in coflow C_min using routing and rates computed in Line 6, and then update R;
17: end while
18: DistributeBandwidth(Ω, R); /* distribute the remaining bandwidth for work conservation */
19: end procedure
```

be able to quickly and efficiently decide the routing paths, rates, and scheduling orders for all individual flows in the coflow. For this purpose, these algorithms must run in real-time with low time complexity.

- *Starvation-free*: As RAPIER allows bandwidth preemption, we must ensure that any coflow should not starve for an arbitrarily long period, though this might benefit the average CCT in the network.
- *Work-conserving*: Work-conservation means that the network resource sits idle only if there is no traffic demand in the network. We require RAPIER to be work-conserving to fully utilize network capacity and to minimize CCT.
- *Readily deployable*: The system should be readily implementable with existing commodity switches and easy to deploy without modifying any network devices.
- *Ensure coexistence*: The system must be able to work with all types of traffic. Especially, latency-sensitive interactive traffic must be delivered without any delay.

B. RAPIER *in a Nutshell*

At a high level, to achieve scalability, RAPIER mainly orchestrates large coflows of data-intensive applications, while latency-sensitive individual flows and small coflows are treated as background traffic; background traffic can be sent directly and routed over the network using ECMP. A site broker periodically predicts the usage of background traffic in each link, and derives the residual bandwidth for coflow scheduling.

We describe the coflow optimization framework of RAPIER with Algorithm 1, which is invoked whenever a new coflow comes or an existing coflow finishes. More specifically, when a new coflow arrives, RAPIER is triggered to compute the routing and the transmission rate for each individual flow (we allow bandwidth preemption as below). When an existing coflow finishes and network resource is released, we also need to trigger RAPIER to determine which coflows should take up the released bandwidth. The underlying scheduling policy RAPIER assumes is the well-known minimum remaining time first (MRTF) [9, 12].

As the input of Algorithm 1, all the coflows that are not completed should be included in Ω. In this case, even if a coflow is occupying the bandwidth in the network, it may be preempted if a "smaller" coflow comes. On the other hand, if part of a coflow is served, its remaining volume information should be updated when we recompute the coflow order. To prevent starvation, RAPIER prioritizes coflows which are waiting for a time longer than a user-defined threshold to schedule (Line 7-10). Other than that, it is the turn for the coflow with the minimum completion time (Line 3-17). When a coflow is selected to send, RAPIER updates the bandwidth utilization and continues to find the next coflow with the next minimum completion time (through Line 5-14). After the schedule order is determined, the remaining bandwidth is distributed to different coflows for the work-conservation purpose (Line 18).

Note that there are two key algorithms in RAPIER. The first one is to calculate the minimum completion time for each coflow given the information of all individual flows in this coflow and the network resource that can be used (Line 6). The other one is to distribute the remaining bandwidth for work-conservation (Line 18). Designing these two complex algorithms is challenging.

IV. ALGORITHM DETAILS

In this section, we present the details for the two key algorithms in RAPIER. In Section IV-A, we discuss how to calculate the minimum completion time for a single coflow by jointly optimizing routing and scheduling. After that, we analyze the approximation ratio of our algorithm in Section IV-B. In Section IV-C, we present the heuristic algorithm in RAPIER to distribute the remaining bandwidth to flows for work-conservation.

A. Minimize Single Coflow Completion Time

Given the information of all the flows in a coflow, such as flow volume, source, destination, and network resource (the residual bandwidth on each link), we can formulate the problem to minimize the CCT of a coflow i as follows:

$$\text{minimize} \quad t_i \tag{1}$$

Subject to:

$$\frac{v_{ij}}{b_{ij}} = t_i, \quad \forall j \tag{1a}$$

$$\sum_j \sum_{k:l \in p_{ij}^k} b_{ij} x_{ij}^k \leq R_l \quad \forall l \tag{1b}$$

$$\sum_k x_{ij}^k = 1, \quad \forall j \tag{1c}$$

$$x_{ij}^k \in \{0,1\}, \quad \forall j,k \tag{1d}$$

t_i in the objective is the completion time of coflow i, and hence it should be minimized. Symbol v_{ij} is the flow volume of the j^{th} flow in coflow i, while b_{ij} is the bandwidth assigned to this flow. With constraint $(1a)$, we directly enforce that the completion time of all the flows to equal to the CCT, since it is reasonable to have all the flows in a coflow to have the same completion time (aka, the bottleneck's completion time) in the optimum solution [4, 5, 12]. Let x_{ij}^k indicate whether the j^{th} flow in coflow i uses its k^{th} path (the link set of this path is denoted by p_{ij}^k), the left-hand term of $(1b)$ calculate the capacity that is used by coflow i on link l, which should be less than the residual capacity of link l, denoted by R_l. $(1c)$ and $(1d)$ require that a flow only chooses one routing path.

It is impossible to solve problem (1) directly, since this programming not only is nonlinear, but also has binary integer variables. This problem is an integer multi-commodity flow problem that is proven to be NP-hard [20]. Therefore, we resort to designing an efficient heuristic to solve this problem.

Based on constraint $(1a)$, we know that the rate of each flow b_{ij} is directly proportional to its volume v_{ij}, i.e., $b_{ij} = \alpha_i v_{ij}$. The larger α_i means more bandwidth is obtained by flows in coflow i, and hence the smaller completion time is required. In fact, we have $\alpha_i = 1/t_i$. The programming model (1) can be modified as follows:

$$\text{maximize} \quad \alpha_i \tag{2}$$

Subject to:

$$\sum_j \sum_{k:l \in p_{ij}^k} \alpha_i v_{ij} x_{ij}^k \leq R_l \quad \forall l \tag{2a}$$

$$(1c),\ (1d)$$

However, there are still binary variables x_{ij}^k in programming model (2), which leads the problem to be intractable on large scale systems. Therefore, we further relax the binary constraint and obtain:

$$\text{maximize} \quad \alpha_i \tag{3}$$

Subject to:

$$\sum_j \sum_{k:l \in p_{ij}^k} v_{ij} (\alpha_i x_{ij}^k) \leq R_l \quad \forall l \tag{3a}$$

$$x_{ij}^k \geq 0 \quad \forall l \tag{3b}$$

$$(1c)$$

It should be noted that there is a product of two variables (i.e., α_i and x_{ij}^k) in constraint $(3a)$. It makes the problem difficult to solve since it is a concave optimization. To solve this problem, variables m_{ij}^k are introduced to substitute this product and we obtain:

$$\text{maximize} \quad \alpha_i \tag{4}$$

Subject to:

$$\sum_j \sum_{k:l \in p_{ij}^k} v_{ij} m_{ij}^k \leq R_l \quad \forall l \tag{4a}$$

$$\sum_k m_{ij}^k = \alpha_i \quad \forall j \tag{4b}$$

$$m_{ij}^k \geq 0 \quad \forall l \tag{4c}$$

Now, problem (4) becomes a linear programming that has only $\sum_j n_{ij} + 1$ variables and $3L + F_i$ constraints (n_{ij} is the number of candidate path for j^{th} flow in coflow i, L is the number of links, and F_i is the number of flows in coflow i). This is a small scale linear programming and can be solved in a timely manner. However, since we relax the binary integer constraint, the solution may be that some x_{ij}^k are decimal factions. To solve this problem, we route the j^{th} flow in coflow i to a path k' such that $m_{ij}^{k'} = \max_k m_{ij}^k$.

When the path of each flow is determined, i.e., x_{ij}^k is fixed, we can go back to problem (2). We substitute in the obtained x_{ij}^k values, and make (2) a linear programming problem and solve it. Given the path of each flow, the minimum CCT is exactly the inverse of the objective in (2).

In summary, the heuristic is: integrate scheduling and routing in the optimization together and let scheduling "guide" the routing selection; after fixing the routing with the approximation, the optimal scheduling is then derived. The heuristic to pursue the minimum completion time of a coflow is summarized as Algorithm 2.

B. Approximation Bound Analysis

We have presented a heuristic to pursue the minimum CCT by relaxing problem (1). In this part, we show how good the performance of the algorithm is through theoretical analysis.

Algorithm 2: Minimize Coflow Completion Time

1: **Procedure** MinimumCCT(Coflow C_i, Bandwidth $\boldsymbol{R}$)
2: Solve problem (4) with coflow and network resource utilization information
3: **for** all flow j in coflow C_i **do**
4: Initialize $x_{ij}^k \leftarrow 0$ for all k
5: $k' \leftarrow \arg_k \max m_{ij}^k$
6: $x_{ij}^{k'} \leftarrow 1$
7: **end for**
8: Solve (2) by fixing x_{ij}^k to be the result obtained from Line 3-7
9: $b_{ij} \leftarrow \alpha_i v_{ij}$, where α_i is the objective in Line 8
10: $t_i \leftarrow \frac{1}{\alpha}$
11: **return** t_i
12: **end procedure**

Theorem 1: Assume the minimum CCT is t_{min} and t_{alg} is the CCT obtained by Algorithm 2, then

$$t_{alg} \leq K t_{min}$$

where K is the number of candidate paths for each flow.

Proof: To prove this theorem, the equivalent proposition is

$$\alpha_{alg} \geq \frac{\alpha_{max}}{K}$$

where α_{alg} and α_{max} are the inverse of t_{alg} and t_{min}, respectively.

Assume the objective of problem (4) is α_{upper}, there must be

$$\alpha_{upper} \geq \alpha_{max} \tag{5}$$

From Algorithm 2, we route each flow to the path with maximum m_{ij}^k, and hence we have

$$m_{ij}^k \geq \frac{\alpha_{upper}}{K} x_{ij}^k \tag{6}$$

for any k. Substitute (6) into the constraints of problem (4), we have

$$\sum_j \sum_{k:l \in p_{ij}^k} \frac{\alpha_{upper}}{K} v_{ij} x_{ij}^k \leq \sum_j \sum_{k:l \in p_{ij}^k} m_{ij} v_{ij} \leq R_l$$

Combine with the fact that the constraint $(1c)$ and $(1d)$ are guaranteed by Algorithm 2, we know that $\frac{\alpha_{upper}}{K}$ is a feasible solution to problem (2) for the the given x_{ij}^k. It means that

$$\alpha_{alg} \geq \frac{\alpha_{upper}}{K} \geq \frac{\alpha_{max}}{K} \tag{7}$$

■

It is worth noting that although the theoretical bound is loose, in practice our implementation obtains very good results.

C. Distribute Bandwidth for Work-conservation

In Section IV-A, RAPIER only allocates minimal bandwidth to each flow, such that all the flows in a coflow are completed simultaneously. However, there may be some remaining bandwidth that can be used to serve more flows. We pursue work-conserving property by distributing the remaining bandwidth to flows in RAPIER to optimize the overall system performance.

The key point in distributing bandwidth is how to determine the order of flows to preempt the bandwidth. At first, for the coflows that have already been scheduled, more bandwidth for any flow in it cannot improve its CCT. Therefore, among all coflows, the coflows that have not been scheduled should have higher priority to use the remaining bandwidth; this also helps prevent starvation. Within a coflow, we prefer to allocate more bandwidth to the larger flows than the smaller ones. This is because the flows with larger traffic volume are more likely to be the bottleneck of a coflow, i.e., complete last if all the flows are served by best-effort delivery. Based on all these considerations, we design Algorithm 3 to distribute bandwidth to flows for work conserving purpose.

Algorithm 3: Distribute Remaining Bandwidth

```
1:  Procedure DistributeBandwidth(Coflows Ω, Bandwidth R)
2:  Non-increasingly sort all the coflows in Ω in terms of their minimal CCT
3:  for all C ∈ Ω do
4:      Non-increasingly sort all the flows in C in terms of flow volume
5:      for all f ∈ Ω do
6:          AssignBandwidth(f,R)
7:      end for
8:  end for
9:  end procedure

10: Procedure AssignBandwidth(Flow f, Bandwidth R)
11: maxBandwidth ← 0
12: for All the candidate paths for f, p do
13:     pathBandwidth ← ∞
14:     for All the links l ∈ p do
15:         if R_l < pathBandwidth then
16:             pathBandwidth ← R_l
17:         end if
18:     end for
19:     if pathBandwidth > maxBandwidth then
20:         maxBandwidth ← pathBandwidth
21:     end if
22: end for
23: return maxBandwidth
24: end procedure
```

In Line 4, we sort the coflows non-increasingly in terms of their minimal CCT. In this case, the coflows with infinite CCT, i.e., that are not scheduled, can get higher priority to preempt the bandwidth. The procedure AssignBandwidth() assign the bandwidth to corresponding flows. Note that the procedure AssignBandwidth() will route a flow to the path that can provide it with the maximum bandwidth.

V. EVALUATION

We evaluate RAPIER through a small-scale testbed emulation as well as large-scale simulations.

Schemes to compare: We compare the following schemes with RAPIER.

- **Baseline:** all the flows are routed by ECMP and all of them fairly compete for bandwidth.
- **Scheduling-only (Varys):** routes all the flows by ECMP but schedules them according to MRTF, which is conceptually equivalent to the state-of-the-art Varys [12].
- **Routing-only:** routes all the flows to pursue load balancing but all the flows should fairly compete for bandwidth.

Through comparison with the last two schemes, we can inspect the benefits brought by the two ingredients of RAPIER: routing and scheduling, respectively.

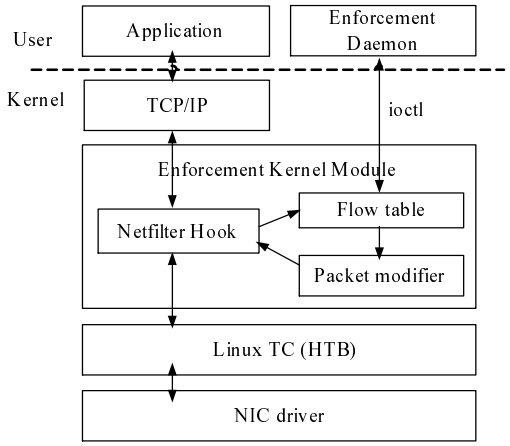


Fig. 2. Software stack of RAPIER's bandwidth enforcement.

Metrics: In this section, we define the performance of scheme 1 compared to scheme 2 as $\frac{CCT_2-CCT_1}{CCT_2}$, where CCT_1 and CCT_2 are the average CCT derived by scheme 1 and scheme 2, respectively. Without declaration, the performance is compared to baseline scheme.

Summary of the main results is as follows:

- Through the experiment on the small-scale leaf-spine testbed (Fig. 3), we can see that 48.6% and 28.22% of the average CCT can be reduced by RAPIER, compared to the baseline and routing-only schemes respectively.
- Results from simulations repeatedly indicate that RAPIER can reduce the average CCT by up to 79.30%, 60.43%, 90.79%, compared to state-of-the-art scheduling-only(e.g., Varys [12]), routing-only, and baseline schemes in different scenarios.
- When network load changes, the performance of RAPIER is relatively stable; as a comparison, the performances of routing-only and scheduling-only schemes vary a lot.
- When inter-coflow arrival interval is large, RAPIER consistently shows very high performance gain. However, even if all the coflows arrive at the same time (the smallest arrival interval), RAPIER can still achieve 53.28% performance improvement in Fattree and 39.86% in VL2.

A. Implementation and Testbed Emulations

Implementation: The RAPIER prototype system consists of the central controller and end host enforcement modules. For routing enforcement we use the Software-defined Networking (SDN) technology to enable explicit routing. For bandwidth enforcement, we leverage Linux Traffic Control (TC) to perform per-flow rate limiting.

The architecture of RAPIER's bandwidth enforcement is shown in Fig. 2. 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.

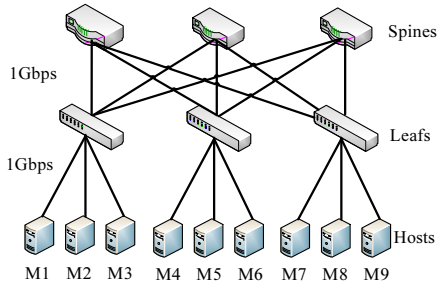


Fig. 3. Testbed topology.

Testbed: We build a leaf-spine topology as shown in Fig. 3. It interconnects 9 hosts through 3 leaf (ToR) switches connected to 3 spine switches using 1Gbps links, resulting in a non-blocking fabric. We use Pronto 3295 48-port Gigabit Ethernet switch with PicOS 2.04 system that supports both Layer 2/3 and OpenFlow. Each server has a 4-core Intel E5-1410 2.8GHz CPU, 8G memory, 500GB hard disk and 1G Ethernet NICs. The OS of servers is Debian 6.0 64bit version with Linux 2.6.38.3 kernel. The CPU, memory or hard disk is not a bottleneck in the experiments. We use iperf to generate TCP flows. The base round-trip time in our testbed is around 100us.

Experiment: In our experiment, we inject 3 coflows into the network to evaluate the performance of RAPIER. As a comparison, we also evaluate the cases of baseline and routing-only schemes. We do not include scheduling-only because we cannot get the exact flow paths with ECMP on the testbed. All the information of this experiment is summarized in Table II. It should be noted that the performance of baseline scheme is averaged by 20 tries, due to the randomness of ECMP. From this experiment, we can see that RAPIER can save $\frac{228.6-117.5}{228.6} = 48.6\%$ of the average CCT compared to the baseline scheme, and it can reduce the average CCT by $\frac{163.7-117.5}{163.7} = 28.22\%$ compared to the routing-only scheme.

Overhead: To make sure that the overhead of the enforcement module is negligible, we measured the extra CPU usage introduced by RAPIER's enforcement module. We generated more than 900Mbps of traffic with more than 100 flows on a rack server (with 4-core Intel E5-1410 2.8GHz CPU). The extra CPU overhead introduced was around 3% (one core) compared with the case that RAPIER's enforcement module was not used (no rate limiting). The throughput remained same in both cases. Actually, we note that, apart from the software solutions, some recent hardware solutions [21] can also be used to achieve precise rate enforcement especially at high link speeds, offloading some work from the CPU.

B. Large Scale Simulations

Simulation methodology: Existing packet-level simulators such as ns-2 are not suitable to our case due to their high overhead [7]. Similar to [7, 12], we develop our own flow-level simulator. The simulator accounts for the flow arrival events and departure events, rather than packet sending and receiving

Coflow Id#	Flow Id#	Source	Destination	Volume (GB)	Coflow Completion Time (s)		
					RAPIER	Routing-only	Baseline
1	1 2 3	M1 M2 M3	M4 M5 M9	3.17 5.29 5.29	50.6	84.1	107.1
2	4 5	M8 M6	M6 M5	10.6 5.29	100.9	203	289.5
3	6 7	M7 M9	M4 M6	17.9 10.6	201.1	204.1	289.2

TABLE II
SUMMARY OF TESTBED EXPERIMENT: THE AVERAGE CCT OF RAPIER IS 117.5S, ROUTING-ONLY IS 163.7, AND BASELINE IS 228.6.

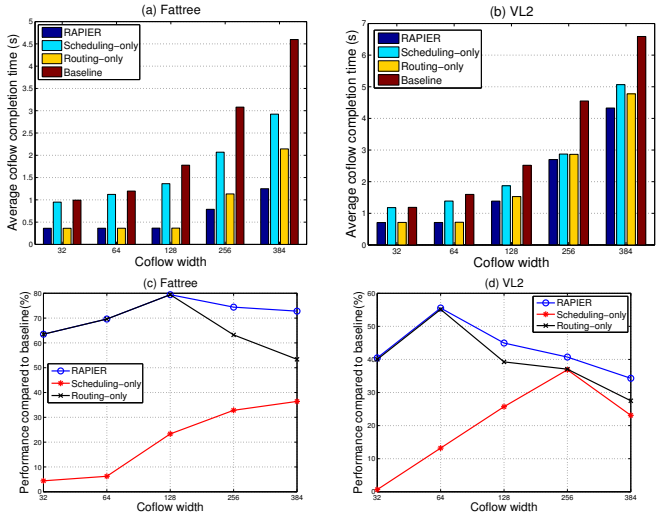


Fig. 4. The impact of coflow width.

events, to reduce the simulation complexity. It updates the rate and remaining volume of each flow when an event occurs. To solve linear programming in RAPIER, we embed the API provided by CPLEX 10.0 into our simulator.

In the simulations, we use many-to-many communication pattern within a coflow and assume the inter-coflow arrival rate follows a Poisson distribution. We mainly evaluate 3 aspects that may affect the performance of RAPIER: the width of a coflow (i.e., the number of flows within a coflow), the number of coflows in the network, and the inter-coflow arrival interval.

For reasonable simulation time, we choose 512-server Fattree [16] and VL2 [17] as topologies. We also compared the results on 512-server Fattree with that on 8192-server Fattree (on which the simulator runs much slower, over 8 hours for just one try) and observed similar performance. In the simulations, each of our results is an average of 20 tries. The overall simulation results are shown in Fig. 4-6. In general, we can see that RAPIER outperforms all other schemes in all scenarios.

Impact of coflow width: In each round of simulations, we send 20 coflows with the same width into the network. Fig. 4 shows the simulation results. From this figure, we make the following observations.

Firstly, as shown in Fig. 4(a) and (b), the absolute average CCT is increased with the coflow width. Compared to the baseline scheme, RAPIER can reduce average CCT by up to 79.44% in Fattree, and 55.55% in VL2. Without routing, the scheduling-only scheme would loss up to $69.61\% - 6.24\% = 63.37\%$ (see Fig. 4(c) at width of 64) of the performance in Fattree.

Secondly, in Fig. 4(c) and (d), we observe a trend that the relative performance of scheduling-only scheme almost increases with the coflow width on both topologies. The reason is that when the coflow width is relatively small, all the coflows are distributed at different parts of the network. In this case, coflows are unlikely to compete for bandwidth with each other. As a result, the scheduling does not have much benefit, and routing-only scheme achieves almost the same performance as RAPIER. With the increase of coflow width, different coflows will interleave with each other. Then, scheduling can effectively reduce average CCT by controlling the flow transmission rates.

Again, in Fig. 4(c) and (d), the performance of routing-only scheme increases with the coflow width at first, but then decreasing with it. This is because the flow collision probability (multiple flows are concurrently active at the same link) increases with the flow number in the network. When the coflow width is relatively small, routing scheme can get good performance as it solves such collision. However, when the coflow width is relatively large, the routing-only scheme cannot avoid such collision and hence show poor performance.

Thirdly, comparing Fig. 4(c) with 4(d), the routing-only scheme has better performance in Fattree than that in VL2. The reason is that in Fattree, more link-disjointed paths can be found for different flows if they are from different source-destination pairs, which is not the case in VL2. Accordingly, routing has more optimization space to improve the average CCT in Fattree.

Fourthly, there is an anomaly in VL2 when the coflow width is 384 (see Fig. 4(d)). We should have expected that the relative performance of scheduling-only scheme would increase with the flow number in the network. However, this expectation goes against the actual simulation results. We note in Fig. 4(b) that large average CCT is caused by many flows in the network, which makes a large denominator in the performance definition. This accounts for the drop.

Impact of coflow number: To evaluate how the performance of RAPIER is influenced by the coflow number in the network, we fix the coflow width to be 128. From the results in Fig. 5, we make the following observations.

Firstly, as shown in Fig. 5(a) and (b), the average CCT is increased with the number of coflows in the network. RAPIER always outperforms routing-only and scheduling-only schemes obviously. We can see from Fig. 5(a) that in Fattree the performance of RAPIER is up to $\frac{1.7039-1.0426}{1.7039} = 38.81\%$ compared to scheduling-only scheme (with 10 coflows) and

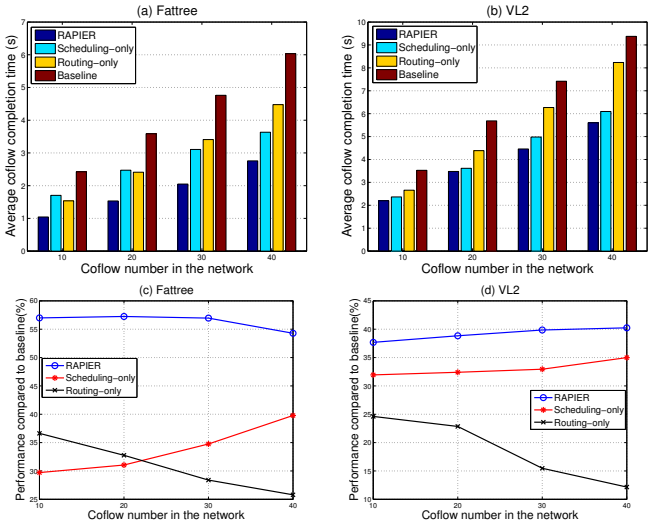


Fig. 5. The impact of coflow number.

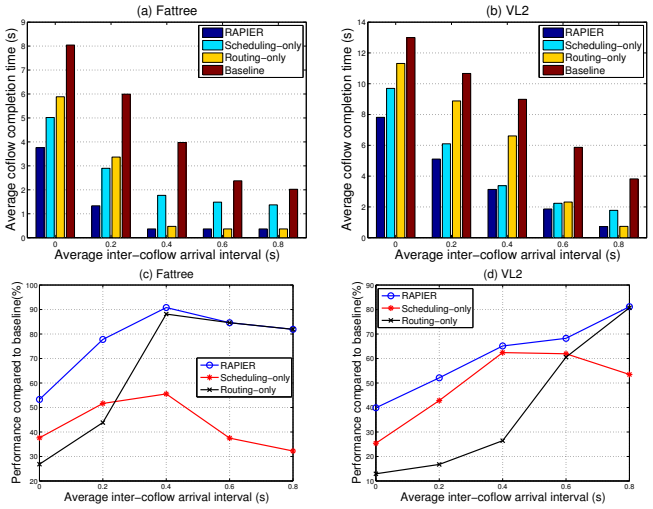


Fig. 6. The impact of inter-coflow number arrival interval.

$\frac{3.4095-2.0495}{3.4095} = 39.89\%$ compared to routing-only scheme (with 30 coflows).

Secondly, from Fig. 5(c) and 5(d), we can see that, in both topologies, RAPIER keeps relatively stable performance with different coflow number. The stable performance of RAPIER comes from its combination of routing and scheduling. When routing makes less contribution to RAPIER with the increase of coflow number, scheduling can contribute more to compensate the performance loss.

Thirdly, we find that the scheduling-only scheme always outperforms routing-only scheme in VL2 (Fig. 5(d)), and scheduling-only scheme is more effective in VL2 than in Fattree (Fig. 5(c) and (d)). Actually, when there are more coflows competing with each other on the same link, scheduling makes more contribution to RAPIER than routing does. Compared to Fattree, there are fewer up links from ToRs in VL2, so it is likely that more flows will interleave with each other in VL2 than in Fattree. Hereby, scheduling is more efficient for VL2 than for Fattree.

Impact of inter-coflow arrival interval: To investigate the impact of the inter-coflow arrival interval, we send out 30 coflows with the width of 128 into the network, and observe the relationship between the system performance and arrival interval of sequential coflows. Note that the larger average inter-coflow arrival interval indicates the lower coflow arrival rate. Zero arrival interval means that all the coflows arrive at the same time. We set the largest average inter-coflow arrival interval to be 0.8s, since each coflow may complete in at most 1s if it monopolizes the network in our simulations. From Fig. 6, we make the following observations.

Firstly, as shown in Fig. 6(a) and (b), the average CCT is decreased with the increase of average inter-coflow arrival interval. This is obvious because, as explained above, larger inter-coflow arrival interval means lower coflow arrival rate. Furthermore, from Fig.6(c) and Fig.6(d), we find that with different inter-coflow arrival intervals, RAPIER can reduce CCT by up to 90.79% in Fattree and 81.14% in VL2 compared to baseline scheme. Even compared to the routing-only scheme and scheduling-only scheme, in Fattree (Fig.6(a)) the performance of RAPIER can be up to $\frac{3.366-1.332}{3.366} = 60.43\%$ (when arrival interval is 0.2s) and $\frac{1.7673-0.3659}{1.7673} = 79.30\%$ (when arrival interval is 0.4s), respectively.

Secondly, the performance of scheduling-only scheme may firstly increase as the average coflow arrival interval increases, and then decrease if the average inter-coflow arrival interval continue increasing after a certain point (see Fig.6(c) and 6(d)). When the inter-coflow arrival interval is small, many coflows should wait for the completion of other coflows. Hence, the baseline is large and it results in bad performance (see Fig. 6(a) and 6(b)). When the inter-coflow arrival interval is large, the later coflows may come when the previous coflows almost complete. In this case, the scheduling scheme does not take effect to reduce the average CCT, since only a few coflows are in network at the same time.

Thirdly, in Fig.6(c) and 6(d), we can see that the performance of RAPIER has the same trend as scheduling-only scheme when the inter-cotask arrival interval is small, while has the same trend as routing-only scheme when the inter-cotask arrival interval is large. This is also because that scheduling is not effective to reduce the average CCT when only a few coflows are active in network concurrently, while routing does not take effect when too many coflows in the network.

Takeaways: *For reducing average CCT, routing contributes more when the network is relatively light loaded, since routing can reduce unnecessary flow collisions. As a comparison, scheduling is more critical when network load increases, and coflows interleave with each other. The success of* RAPIER *is that it integrates both routing and scheduling, hence always outperforms other schemes regardless of the network status.*

VI. Related Work

RAPIER contains two parts: routing and scheduling. There is a large spectrum of related work along either routing or scheduling. We only review some closely related ones here.

Flow routing in DCNs: Traditional traffic engineering solutions inside a data center [22] or across data centers [23, 24] focus on improving the network resource utilization while not reducing the average CCT. They leverage the short-term traffic predictability in DCNs to improve the system performance. In another work, zUpdate [25] applies to the scenario where some network components face failure, while Hedera [7] and Duet [26] focus on how to distribute flows to balance the traffic load in the network.

Relative to them, RAPIER investigates how to distribute the flows belonging to the same coflow evenly into the network so that the average CCT can be further reduced by scheduling.

Individual flow scheduling in DCNs: There are also many existing work on optimizing network utilization and reducing average flow completion time (FCT) by using scheduling methods, such as PDQ [9] and pFabric [8]. Both PDQ and pFabric are flow scheduling schemes to minimize FCT by tagging priority on the packets. Unfortunately, neither of them can be implemented using existing commodity switches, and hence they are not easy to widely deploy. Furthermore, they do not take into account the flow dependency semantics and thus are coflow-agnostic.

Coflow scheduling in DCNs: Orchestra [4] is perhaps the first work that take the semantics among flow into account when optimizing the flow transfers in data center clusters. After that, the work [5] summarizes the traffic patterns and flow dependency in DCNs and explicitly proposes the concept of coflow. Then, recent solutions (e.g., Varys [12] and Barrat [13]) start to apply the coflow concept (or task-aware) in their network optimizations, however they only focus on scheduling while neglecting an indispensable part—routing, which make these solutions insufficient.

VII. CONCLUSION

RAPIER is a system which optimizes average coflow completion time in DCNs by integrating routing and scheduling. To the best of our knowledge, RAPIER is the first work that proposes and proves the position that routing and scheduling must be jointly considered for optimizing the average CCT. Through real implementation and extensive simulations, we demonstrate that RAPIER works with existing commodity switches and preserves remarkable performance advantages over the scheduling-only or routing-only solutions.

Acknowledgements This work was supported in part by HKRGC-ECS 26200014, National Basic Research Program of China (973) under Grant 2011CB302601, 2013CB329103, 2014CB340303, 2014CB347800, Huawei Noah's Ark Lab, NSFC Fund (61202107, 61271165, 61379055, 61271171, 61201129, 61301153 and 91438117), and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys*, 2007, pp. 59–72.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the USENIX HotCloud 2010*.

[4] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra." in *Proceedings of the ACM SIGCOMM 2011*, pp. 98–109.

[5] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 31–36.

[6] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the ACM SIGMOD 2010*, pp. 135–146.

[7] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, 2010.

[8] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of the ACM SIGCOMM 2013*, 2013, pp. 435–446.

[9] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proceedings of the ACM SIGCOMM 2012*, pp. 127–138.

[10] A. Munir, G. Baig, S. Irteza, I. A. Qazi, F. Dogar, and A. Liu, "Friends, not Foes - Synthesizing Existing Data Center Transport Strategies," in *Proceedings of the ACM SIGCOMM 2014*.

[11] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 50–61, Aug. 2011.

[12] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the ACM SIGCOMM 2014*.

[13] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proceedings of the ACM SIGCOMM 2014*.

[14] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the SOSP 2003*, pp. 29–43.

[15] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "Generic and automatic address configuration for data center networks," in *Proceedings of the ACM SIGCOMM 2010*.

[16] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[17] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Aug. 2009.

[18] K. Chen, A. Singlay, A. Singhz, K. Ramachandranz, L. Xuz, Y. Zhangz, X. Wen, and Y. Chen, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," in *Proceedings of the USENIX NSDI 2012*, pp. 18–29.

[19] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu, "Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing," in *Proceedings of the IEEE INFOCOM 2014*.

[20] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Foundations of Computer Science, 1975., 16th Annual Symposium on*, Oct 1975, pp. 184–193.

[21] S. Radhakrishnan, Y. Geng, V. Jeyakumar, A. Kabbani, G. Porter, and A. Vahdat, "Senic: Scalable nic for end-host rate limiting," in *Proceedings of the USENIX NSDI 2014*, pp. 475–488.

[22] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the ACM CoNEXT 2011*.

[23] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, and V. Gill, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013*, pp. 15–26.

[24] S. Jain, A. Kumar, S. M, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. W, J. Zhou, M. Zhu, J. Zolla, U. Hlzle, S. Stuart, A. Vahdat, and G. Inc, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013*.

[25] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz, "zupdate: updating data center networks with zero loss," in *Proceedings of the ACM SIGCOMM 2013*, pp. 411–422.

[26] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud scale load balancing with hardware and software," in *Proceedings of the ACM SIGCOMM 2014*, pp. 27–38.