TSINGHUA SCIENCE AND TECHNOLOGY
ISSNll1007-0214 10/10 pp630–644
DOI: 10.26599/TST.2018.9010098
Volume 24, Number 5, October 2019

VirtCO: Joint Coflow Scheduling and Virtual Machine Placement in Cloud Data Centers

Dian Shen, Junzhou Luo, Fang Dong*, and Junxue Zhang

Abstract: Cloud data centers, such as Amazon EC2, host myriad big data applications using Virtual Machines (VMs). As these applications are communication-intensive, optimizing network transfer between VMs is critical to the performance of these applications and network utilization of data centers. Previous studies have addressed this issue by scheduling network flows with coflow semantics or optimizing VM placement with traffic considerations. However, coflow scheduling and VM placement have been conducted orthogonally. In fact, these two mechanisms are mutually dependent, and optimizing these two complementary degrees of freedom independently turns out to be suboptimal. In this paper, we present *VirtCO*, a practical framework that jointly schedules coflows and places VMs ahead of VM launch to optimize the overall performance of data center applications. We model the joint coflow scheduling and VM placement optimization problem, and propose effective heuristics for solving it. We further implement *VirtCO* with OpenStack and deploy it in a testbed environment. Extensive evaluation of real-world traces shows that compared with state-of-the-art solutions, *VirtCO* greatly reduces the average coflow completion time by up to 36.5%. This new framework is also compatible with and readily deployable within existing data center architectures.

Key words: cloud computing; data center; coflow scheduling; Virtual Machine (VM) placement

1 Introduction

Modern virtualization-based cloud data centers, such as Amazon EC2[1], have become hosting platforms for a wide spectrum of big data applications[2, 3], including online data mining and social network analysis. Owing to their distributed nature, many of these emerging cloud applications introduce intensive network traffic between hosting Virtual Machines (VMs). Network transfer time may consume as much as 30%–50% of an application's execution time[4, 5]; thus, optimizing the network transfer between VMs is necessary not only to reduce the completion time of encapsulated applications but also to release space to accommodate requests from additional tenants.

Conventionally, data center administrators try to address this issue by scheduling network flows. To this end, coflow[6] is introduced to capture the traffic characteristics of emerging cloud applications, such as Mapreduce[7], Spark[8], and Pregel[9]. In the coflow semantic, a collection of parallel flows must be transferred between groups of servers, and applications complete the network transfer only after all flows have finished. Thus, optimizing the Coflow Completion Time (CCT), i.e., the time when all corresponding flows have finished, is critical for the performance of applications. Recent studies[6, 10, 11] have proposed effective algorithms for scheduling coflows to reduce the average CCT.

However, existing methods seldom consider the

• Dian Shen, Junzhou Luo, and Fang Dong are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. E-mail: {dianshen, jluo, fdong}@seu.edu.cn.
• Junxue Zhang is with the SING Group, Hong Kong University of Science and Technology, Hong Kong 999077, China. E-mail: jzhangcs@ust.hk.
∗ To whom correspondence should be addressed.
Manuscript received: 2018-04-02; accepted: 2018-05-01

server aspect and assume that the servers are Physical Machines (PMs). In fact, in the multi-tenant cloud environment, the resources are delivered in the form of VMs. The flexibility of VM placement allows further mitigation of the traffic traversing the data center fabric and reduction of the network transfer time of applications. Traffic-aware VM placement[12–14], for instance, places VMs with large mutual communication in the same PM.

Unfortunately, coflow scheduling and VM placement are traditionally conducted orthogonally. Existing solutions for coflow scheduling are unaware of VM placement and previous VM placement mechanisms are coflow-agnostic such that they do not account for collective behaviors of flows belonging to a coflow (Table 1). Nevertheless, the effects of these two Degrees Of Freedom (DOFs) are clearly coupled and optimizing on any one dimension alone is extremely restrictive (see Section 2 for a motivating example). According to our experimental results, applying the state-of-the-art coflow scheduling and traffic-aware VM placement independently could lead to a performance loss of up to 36.5% in terms of the average CCT.

Therefore, in this study, we aim to solve a joint coflow scheduling and VM placement problem, with the objective to further improve network efficiency in data centers. Although the joint optimization is intuitively beneficial, combining two mechanisms seamlessly in cloud data centers poses practical challenges in implementation and deployment.

First, network traffic information is required for both coflow scheduling and VM placement. Although this prerequisite has been mostly treated as an assumption in previous works, recent developments in data center network research have demonstrated the predictability of network traffic information. In cloud data centers, many jobs are recurring[3, 16] and have predictable resource requirements. This condition allows us to effectively predict their network traffic information with an accuracy of nearly 90%[17] using existing prediction[18] or profiling[3] techniques. By exploiting this characteristic, we can predetermine where and when VMs and the traffic between them should be placed and transferred.

Table 1 Summary and comparison of previous approaches to *VirtCO*.

Related work	Coflow-aware	Scheduling	VM placement
Varys[6], Aalo[10], Rapier[15]	Yes	Yes	No
Traffic-aware VM management[12, 14]	No	No	Yes
VirtCO	Yes	Yes	Yes

Second, without control over tenant VMs (end-hosts), the exertion of coordinated scheduling remains at the mercy of VMs' networking components, which are usually noncooperative. Current coflow scheduling mechanisms prioritize each flow in the end-hosts, such that cloud providers have no privileged control. Exploring how providers can regain authority over flow scheduling outside VMs is necessary. Thus, we target coflow scheduling for VMs at the hypervisor-level.

To address these challenges, we design and implement *VirtCO*, which jointly schedules coflows and places VMs with practical considerations. We first formulate the problem as a joint coflow scheduling and VM placement optimization problem to minimize the average CCT. Accordingly, we propose efficient heuristics to approximate a solution to this problem. Addressing the practical challenges, we implement the coflow scheduling in the virtual Switch (vSwitch) running on each hypervisor, without having control over the VMs or requiring modifications in network hardware. Thus, VM placement and coflow scheduling can be combined seamlessly in the same layer. We deploy *VirtCO* on an 18-server testbed in our data center with more than 200 VMs. Evaluations with traces of real-world applications show that compared to state-of-the-art methods, *VirtCO* can reduce the average CCT by 36.5%. Furthermore, the computation overhead of *VirtCO* is less than 3% of CPU occupation.

The main contributions of this study are the following:

• It explores the problem of joint coflow scheduling and VM placement with several key observations and practical challenges (Section 2).

• It proposes *VirtCO*, a novel practical solution for jointly scheduling coflows and placing VMs. Algorithms for joint optimization are then developed (Section 3).

• It formulates the joint coflow scheduling and VM management problem and computes the best VM placement and bandwidth allocation scheme by solving a Linear Programming (LP) problem (Section 4).

• It implements an OpenStack-based prototype system that is readily deployable within current data center architectures. Then extensive evaluations are conducted in a testbed with real-world traces (Section 5).

2 A Motivating Example

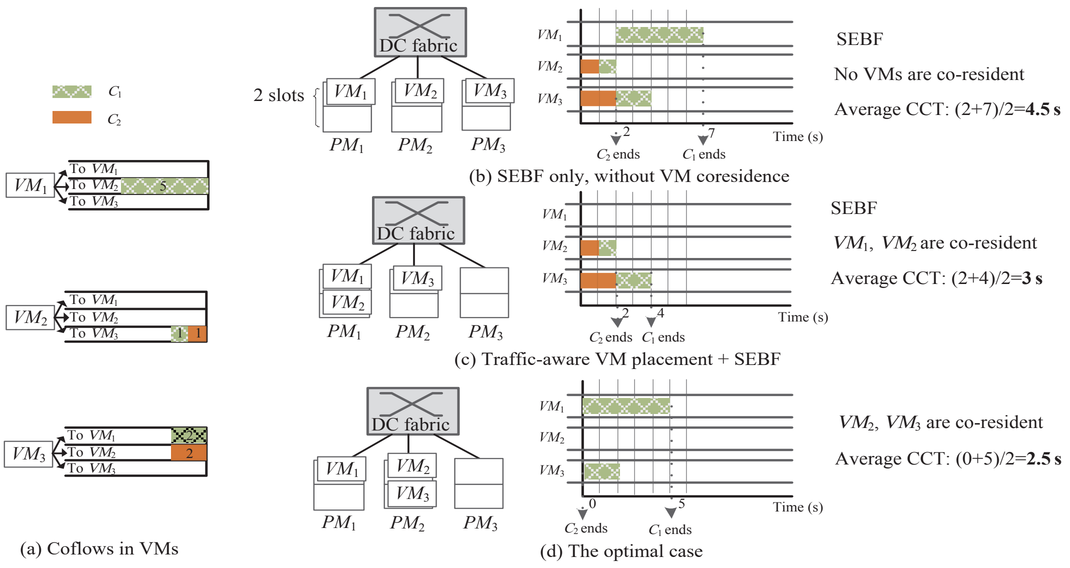
In this section, we illustrate the need for the joint design of VM placement and coflow scheduling through a motivating example in Fig. 1. As shown in Fig. 1a, two coflows exist: C_1 and C_2. C_1 has 3 flows transferred from VM_1 to VM_2, from VM_2 to VM_3, and from VM_3 to VM_1 with sizes of 5, 1, and 2 units, respectively. C_2 has 2 flows transferred from VM_2 to VM_3, and from VM_3 to VM_2 with sizes of 1 and 2 units, respectively. For VMs, we use a slot to represent one unit of static resource (CPU/memory/disk)[14]; the size of each VM in the example is 1 slot. The data center fabric is depicted as several PMs connected by a non-blocking switch[6], and each PM has 2 slots. The bandwidth of each PM is 1 unit/second. We approximate the intra-PM communication to be transient to facilitate our analysis. This approximation brings very slight and negligible bias because it is much faster than that between separate PMs. We measure the average CCT in different scenarios.

We consider the case in Fig. 1b, in which VMs are randomly placed, and no VMs are co-resident. In this case, the Smallest Effective Bottleneck First (SEBF)[6] method generates the optimal coflow scheduling such that the coflow with smallest bottleneck time is scheduled with the highest priority. The bottleneck flow of C_1 is 5 units and that of C_2 is 2 units. Thus, C_2 is scheduled before C_1. In this case, the average CCT is 4.5 s.

Then, we consider the situation in Fig. 1c. In this case, the coflows are scheduled with the SEBF algorithm. The traffic-aware VM placement is conducted independently. As VM_1 and VM_2 have the greatest mutual communication (5 units), they are placed in the same PM to localize the traffic. In this case, the average CCT is 3 s.

However, traffic-aware VM placement does not generate the optimal overall performance for applications, in terms of the average CCT. As shown in Fig. 1d, placing VM_2 and VM_3 together generates the optimal average CCT of 2.5 s, an improvement of 1.2×. Nevertheless, neither current VM placement nor coflow scheduling mechanisms alone could effectively generate the case in Fig. 1d.

From this example, we observe that coflow scheduling and VM placement are mutually dependent, and optimizing these two complementary DOFs orthogonally turns out to be suboptimal. Suboptimal VM placement and scheduling introduce unnecessarily prolonged overall execution time for applications, thereby impairing tenant satisfaction and system utilization. Having joint control over both aspects provides an opportunity to utilize the data center network efficiently.



C_1
C_2
VM_1
To VM_1
To VM_2
5
To VM_3
VM_2
1
1
VM_3
2
2
(a) Coflows in VMs
DC fabric
2 slots
PM_1
PM_2
PM_3
Time (s)
C_2 ends
C_1 ends
(b) SEBF only, without VM coresidence
SEBF
No VMs are co-resident
Average CCT: (2+7)/2=**4.5 s**
(c) Traffic-aware VM placement + SEBF
VM_1, VM_2 are co-resident
Average CCT: (2+4)/2=**3 s**
(d) The optimal case
VM_2, VM_3 are co-resident
Average CCT: (0+5)/2=**2.5 s**

Fig. 1 A motivating example. Two color-coded coflows (C_1 and C_2) reside in 3 VMs. The size of each VM is 1 slot, and each PM has 2 slots. C_1 has 3 flows: $VM_1 \rightarrow VM_2 = 5$, $VM_2 \rightarrow VM_3 = 1$, and $VM_3 \rightarrow VM_1 = 2$. C_2 has 2 flows: $VM_2 \rightarrow VM_3 = 1$ and $VM_3 \rightarrow VM_2 = 2$. (b)–(d) show the VM placement and scheduling on egress ports. (b) indicates the baseline in which no VMs are co-resident, and the coflow with the largest bottleneck (C_2) is scheduled first. (c) shows the result when VMs with the largest mutual communication ($VM_1 \rightarrow VM_2 = 5$) are placed together. The coflow scheduling (C_2 first) is conducted independently. (d) demonstrates the optimal case.

3 Design and Implementation of *VirtCO*

3.1 Design overview

Motivated by the potential gains of cooperative coflow scheduling and VM placement, we design *VirtCO* to jointly optimize these two aspects. Inspired by Refs. [6, 15], *VirtCO* works in a centralized manner. This approach is coherent with many recent centralized data center designs[19]. Given applications and their associated coflow information, such as flow sizes and sources/destinations (typically in the form of a traffic matrix), *VirtCO* determines *when* to start coflows, at *what rate* to serve them, and *where* to place the VMs to optimize the average CCT of all coflows. *VirtCO* has several advantageous features by design.

First, it bridges the gap between two mutually dependent problems despite their orthogonal treatment. Instead of scheduling network flows in an unknown (assumed to be fixed) environment, *VirtCO* enables the cloud provider to conduct intelligent coordinated scheduling with a panoramic view of its infrastructure. The cloud provider, being coflow-aware, has a better understanding of applications and can optimize their overall performance.

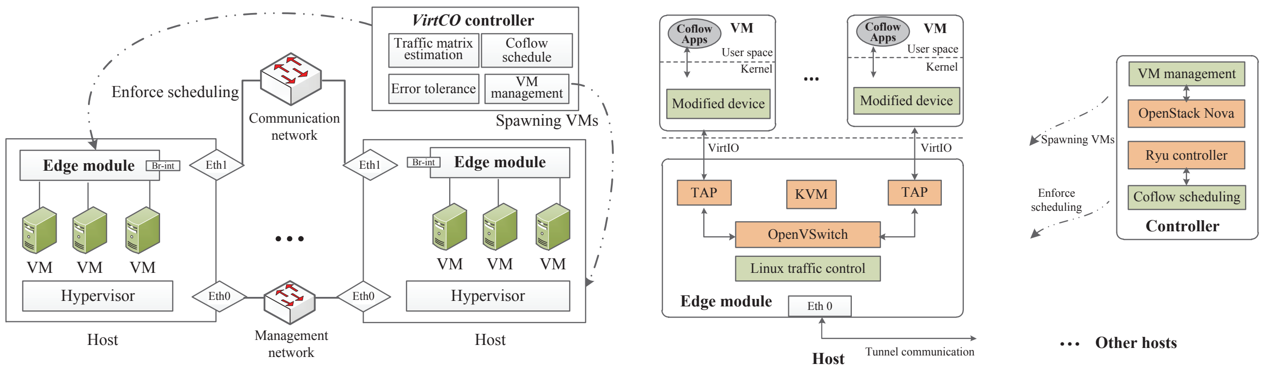
Another benefit of *VirtCO* is that it enables implementation of uniform scheduling throughout the data center, thereby ensuring fairness. Traditional flow priorities set inside VMs can become a "race to the top" in the cloud environment such that multiple co-resident VMs of different tenants may assume that their traffic is the most important and should receive the highest priority. Additionally, network components of different versions within VMs may not share the network fairly. *VirtCO* enforces fairness by regaining control over the network outside the VMs without hardware modification.

Finally, *VirtCO* is designed with consideration of scalability. The vSwitch on each hypervisor is only responsible for scheduling the VMs under its supervision. Network control is enforced at the granularity of coflows and hypervisors, rather than flows and VMs. According to our experiments, the extra CPU overhead on each hypervisor is less than 3%. Calculating joint placement and scheduling in the controller takes seconds. As we predetermine when VMs launch, the decision computation time is much shorter than the VM spawning time, which usually takes minutes. Distributing the controller's decision to tens of hypervisors takes approximately 4–5 ms, and to 10 000 (emulated) hypervisors takes less than 0.7 s.

3.2 Implementation

We implement *VirtCO* based on OpenStack[20] and integrate it with customized modules to schedule coflows in the hypervisor layer. *VirtCO* consists of a coflow traffic matrix estimation module that obtains the coflow traffic matrix, an error-tolerance scheduling module that reserve a fraction of resources, a controller that efficiently determines scheduling as requests arrive. The controller then communicates with distributed edge modules on each hypervisor to enforce scheduling. The architecture and implementation of *VirtCO* are depicted in Fig. 2.

Coflow traffic matrix estimation. As the predictability of data center jobs has been confirmed[3, 16], coflow information can be obtained by existing prediction[18] or profiling[3, 17] methods. *VirtCO* supports these existing methods and provides an interface with which users can specify their traffic matrix as scheduling hints when launching VMs. With this coflow traffic matrix information, *VirtCO*



(a) *VirtCO* overview (b) Implementation of edge module (c) Implementation of the controller

Fig. 2 Architecture and implementation of *VirtCO*.

predetermines joint scheduling when launching VMs.

Error tolerance. A key issue in traffic matrix estimation is that current methods inevitably cannot predict coflow information 100% correctly. Previous research[18] reported a 5%–10% error rate. CODA[17] identifies coflows with approximately 90% accuracy. Robust to inaccuracies, the error-tolerance module is designed to mitigate the impact of error estimation. According to our experimental analysis, underestimation can be worse than overestimation because it could lead to the "straggler" issue[17], which severely degrades the performance of coflow applications. Thus, we take the approach toward overestimation for all flow information. Each estimated coflow is attached to an overestimation factor β^*, and the overestimated traffic $E^*(C)$ for coflow C is $E^*(C) = E(C) \cdot (1 + \beta^*)$, where $E(C)$ is provided by users or the traffic matrix estimation module. β^* is inherently a tradeoff between utilization and error tolerance. Although a complex model can be used to compute β^*, *VirtCO* uses the flow-size distribution in the data center as an indicator, and lets β^* be the fraction of low-priority flows or flows without explicit completion time requirements, such as data backup flows. This simple approximation of β^* indicates that the overprovisioned resource can be used alternatively by other flows in a work-conserving manner.

The controller collects the global coflow information and uses the algorithms proposed in Section 4 to schedule coflows and VMs. The controller implementation is depicted in Fig. 2c. At the launch of a new set of VMs, *VirtCO* predetermines the VM locations, rates, and scheduling order for the coflow. *VirtCO* calls the API of OpenStack Nova to assign the locations of corresponding VMs and communicates with edge modules to enforce scheduling policies. When the scheduling of a coflow is decided, *VirtCO* assigns the VMs to corresponding PMs and updates the flow tables on corresponding hypervisors. To avoid starvation, *VirtCO* sets a threshold to ensure that any coflow does not starve for an arbitrarily long time.

Edge modules are installed on each hypervisor. An edge module is fully responsible for scheduling the aggregate flows of each coflow sent from VMs. The edge module installed on the hypervisor penetrates all outgoing packets from all VMs in the granularity of each coflow, queries the coflow table distributed by the controller for their priorities, and places these priorities in the corresponding queue. The edge module is implemented as an extension of Linux kernel module in the hypervisor. This module adopts Linux Traffic Control (TC) to enforce scheduling. As depicted in Fig. 2b, we use the queuing discipline (qdisc) mechanism offered by the Linux TC subsystem to rate-limit coflows with designated priorities. Our system is flexible and easy to deploy because TC modules require no kernel modifications and can be inserted and removed at runtime. The number of rate limiters is subjected to the number of coflows in each hypervisor, thereby making this solution scalable in the data center.

Usage scenario. In a multi-tenant cloud environment, tenants that need better network performance select *VirtCO*-marked VMs. When launching VMs, tenants provide the traffic matrix of their applications as a hint with which cloud providers can optimize performance. Otherwise, they allow cloud providers to profile their VMs' traffic for performance optimization. Thus, tenants can expect improved performance and reduced costs because current billing models charge tenants by usage time. Consequently, cloud providers can attract larger number of users to migrate their applications to the cloud, especially those who have concerns about cloud network performance. Moreover, by reducing the occupation time per tenant, cloud providers can utilize released resources to serve more tenants and benefit from the improved infrastructure utilization. Cloud providers can also increase the price per unit time, while reducing the total cost for a single tenant. Therefore, *VirtCO* usage is fully consistent with current cloud computing business models and leads to a win-win situation for cloud providers and tenants.

4 Core Algorithms of *VirtCO*

4.1 Global coflow scheduling and VM placement

We begin by modeling the joint coflow scheduling and VM placement problem to optimize the average CCT for all coflows. For the virtualization environment, we define H and h_m as the set of physical hosts and the m-th physical host within the data center, respectively. Applications request resources in the form of VMs. For each application, V and v_j denote the set of VMs and the j-th VM for the application, respectively. The VM placement is represented by an $H \times V$ matrix denoted by X. x_j^m denotes whether v_j is hosted by h_m. Specifically, $x_j^m = 1$ if v_j is hosted by h_m; otherwise, $x_j^m = 0$.

The coflow information C_i associated with an application is characterized by traffic matrix P_i, where

$p_{i,j,k}$ denotes the size of flow C_i to be transferred from v_j to v_k. In the example depicted in Fig. 1, coflow C_1 has the traffic matrix $\langle p_{1,1,2}=5, p_{1,2,3}=1, p_{1,3,1}=2\rangle$ and coflow C_2 has the traffic matrix $\langle p_{2,2,3}=1, p_{2,3,2}=2\rangle$. All incomplete coflows are included in the collection $\mathcal{C}$. To illustrate the network resource constraints, we define $\mathcal{B}$ as the residual bandwidth on PMs. $\mathcal{B}$ consists of the egress and ingress bandwidth, denoted by B_e^m and B_{in}^m, respectively. B_i denotes the rate allocated to C_i, which is composed of the egress and ingress rates allocated to C_i on h_m, denoted as e_i^m and in_i^m. VM size, denoted as $size_j$, is represented by several slots, and the number of available slots in h_m is $slot_m$. Under any placement, t_i denotes the CCT of C_i.

To optimize the average CCT, *VirtCO* leverages the minimum remaining time-first heuristic to schedule the coflow in $\mathcal{C}$ having minimum remaining CCT with the highest priority. We describe the main framework of *VirtCO* in Algorithm 1. The algorithm is invoked whenever a new application enters the data center. Specifically, when a new application associated with a coflow arrives, the algorithm is triggered to determine its VM placement, scheduling priority, and bandwidth allocation (allowing preemption). Line 4 of Algorithm 1 invokes another algorithm to compute the minimum CCT for the arriving coflow. In Section 4.2, we illustrate the details of minimizing the single CCT through VM placement and network scheduling. Among all incomplete coflows, the algorithm sets the priorities according to their remaining completion time in ascending order (lines 7–10). To avoid starvation, the algorithm checks the waiting time of coflows in $\mathcal{C}$ and sets a threshold to ensure that no coflow starves for an arbitrary period θ (lines 11–14). θ is usually in minutes and determined empirically by cloud providers.

Algorithm 1 ***VirtCO* main framework**

```
1:  procedure MRTF
2:      input: $\mathcal{C}$, arriving request $C_{new}$, residual slots $\mathcal{S}$, residual bandwidth $\mathcal{B}$, VM placement $X$
3:      output: void
4:      invoke minSingleCCT($C_{new}$,$\mathcal{S}$,$\mathcal{B}$,$X$)
5:      allocate $X_{new}$, $B_{new}$ and update $\mathcal{S}$, $\mathcal{B}$
6:      add $C_{new}$ to $\mathcal{C}$
7:      while $\mathcal{C} \neq \varnothing$ do
8:          $\forall$ $C_i$ in $\mathcal{C}$ update the remaining $t_i$
9:          sort $\mathcal{C}$ by the remaining $t_i$ in ascending order
10:         set the priority for all coflows in $\mathcal{C}$ in this order
11:         Check the waiting time $wait(C_i)$ in $\mathcal{C}$
12:         if $\exists wait(C_i) \geqslant \theta$ then
13:             schedule $C_i$ with the highest priority
14:         end if
15:     end while
16: end procedure
```

4.2 Minimizing the completion time of a single coflow

The problem of minimizing the CCT for a single coflow in the cloud environment is formally stated as follows: given the estimated traffic matrix of a coflow C_i, we try to find a feasible placement X_i of all its VMs onto PMs, allocating feasible bandwidth e_i^m and in_i^m for C_i on each h_m, to minimize its completion time t_i. Although single-flow optimization heuristics would allocate the entire bandwidth of the link to the scheduled flow, a desirable property of coflow is that completing any flow faster than the bottleneck in a coflow does not affect the CCT. Therefore, the minimum completion time of a coflow can be attained as long as all flows finish at the same time with the bottleneck flow. That is, all flows in C_i should finish at the CCT t_i.

Note that we enforce scheduling in the hypervisor layer with coflow granularity. Under placement X_i, the amount of data coflow that C_i sends from h_m is s_i^m, and the coflow C_i received from h_m is r_i^m. s_i^m and r_i^m are computed by

$$\begin{cases} s_i^m = \sum\limits_j \sum\limits_k x_j^m (1-x_k^m) p_{i,j,k}, \\ r_i^m = \sum\limits_j \sum\limits_k x_k^m (1-x_j^m) p_{i,j,k} \end{cases} \tag{1}$$

Corollary 1 Having allocated the bandwidth e_i^m and in_i^m to C_i on one PM h_m, we can determine the minimum CCT through weighted fair sharing among individual flows of C_i on h_m, with the weight

$$w_{i,j,k} = \frac{p_{i,j,k}}{s_i^m} \tag{2}$$

Proof For notational simplicity, we define h_m as the sender host and h_n as the receiver host. As all flows in a coflow are completed at the same time, we assume that $t_i = 1$. Then, the egress and ingress bandwidth allocated to each PM are equal to s_i^m and r_i^n, respectively. As the data sent must be equal to the data received, $\sum_m s_i^m = \sum_n r_i^n$. Let $p_{m,n} = \sum_{j\in m}\sum_{k\in n} p_{i,j,k}$ denote the aggregate flow sent from h_m to h_n. Thus, we have $\sum_m \sum_n p_{m,n} = \sum_n r_i^n$. Let $p_{m,n}$ be the bandwidth shared for flow $m \to n$, and then reduce $\sum_n$ from both sides. We then have $\sum_m p_{m,n} = r_n$. Thus, with $p_{m,n}$ as the bandwidth shared for all flows sent from m, the data transfer finishes exactly at $t_i = 1$. Therefore, to achieve the optimal time t_i, each $p_{m,n}$ shares the bandwidth with $w_{m,n} = p_{m,n}/s_i^m$. Furthermore, as $p_{m,n}$ is the aggregate flow of all $p_{i,j,k}$, each flow shares the bandwidth with the proportion $w_{i,j,k} = p_{i,j,k}/s_i^m$. ■

Corollary 1 indicates the theoretical feasibility of enforcing scheduling at the granularity of coflows and hypervisors. This control granularity is able to attain the same performance as the finer granularity of flows and VMs. This design allows fewer rate limiters than other implementations and reduces the computation intensiveness of core algorithms.

Given the above definitions, we formulate the problem of minimizing t_i as follows:

$$P = \min t_i \tag{3}$$

subject to the following:

$$\begin{cases} \frac{s_i^m}{e_i^m} = t_i, \\ \frac{r_i^m}{in_i^m} = t_i, \end{cases} \quad \forall m \tag{3a}$$

$$\begin{cases} e_i^m \leqslant B_e^m, \\ in_i^m \leqslant B_{in}^m, \end{cases} \quad \forall m \tag{3b}$$

$$\sum_m x_j^m = 1, \quad \forall j \tag{3c}$$

$$\sum_j x_j^m \leqslant slot_m, \quad \forall m \tag{3d}$$

Constraint Eq. (3a) enforces that the completion time of all flows is equal to the CCT. With the minimum bandwidth allocated, the residual bandwidth is released to admit other coflows. Equations (3c) and (3d) represent the placement constraints. However, the original problem (3) is difficult to solve directly because it is a nonlinear programming problem with binary integer variables. To facilitate the solution, we define $\alpha_i = 1/t_i$, then, Eq. (3) can be modified to

$$P' = \max \alpha_i \tag{4}$$

The boundary conditions of Eq. (4) are given by

$$\alpha_i \sum_j \sum_k x_j^m (1 - x_k^m) p_{i,j,k} \leqslant B_e^m, \quad \forall m \tag{4a}$$

$$\alpha_i \sum_j \sum_k x_k^m (1 - x_j^m) p_{i,j,k} \leqslant B_{in}^m, \quad \forall m \tag{4b}$$

$$\sum_m x_j^m = 1, \quad \forall j,$$

$$\sum_j x_j^m \leqslant slot_m, \quad \forall m.$$

Then, we introduce two variables: $W_{i,j,k}^m$ and $V_{i,j,k}^m$. Let $V_{i,j,k}^m = \alpha_i x_j^m (1 - x_k^m)$ and $W_{i,j,k}^m = \alpha_i x_k^m (1 - x_j^m)$. Substituting $V_{i,j,k}^m$ and $W_{i,j,k}^m$ into problem (4), and we can transform this problem into the following:

$$P' = \max \alpha_i \tag{5}$$

The boundary conditions of Eq. (5) are given by

$$\sum_j \sum_k V_{i,j,k}^m p_{i,j,k} \leqslant B_e^m, \quad \forall m \tag{5a}$$

$$\sum_j \sum_k W_{i,j,k}^m p_{i,j,k} \leqslant B_i^m, \quad \forall m \tag{5b}$$

$$\sum_m V_{i,j,k}^m \leqslant \alpha_i, \quad \forall j,k \tag{5c}$$

$$\sum_m W_{i,j,k}^m \leqslant \alpha_i, \quad \forall j,k \tag{5d}$$

Equation (5) represents an LP problem with the limited scale of variables and constraints that can be solved in a timely manner. We analyze the computation complexity of the algorithm from two perspectives: bandwidth allocation and VM placement. For the aspect of bandwidth allocation, as we only conduct rate limiting on each hypervisor, the variable scale is limited to the number of physical hosts $|H|$. For VM placement, each iteration of computation generates the VM placement for an application, and the variables' scale is bounded by $|V|$. The computation overhead to run the LP is bounded by $|H|$ and $|V|$. In practice, the available $|H|$ for hosting an application is usually limited to several adjacent racks, which are approximately dozens of hosts. The $|V|$ requested by an application is also extremely limited. Therefore, the small scale optimization can be computed efficiently, and with a small computation overhead.

After $V_{i,j,k}^m$ and $W_{i,j,k}^m$ are derived, x_j^m and x_k^m can be computed by

$$\begin{cases} x_j^m (1 - x_k^m) = \frac{V_{i,j,k}^m}{\alpha_i}, \\ x_k^m (1 - x_j^m) = \frac{W_{i,j,k}^m}{\alpha_i} \end{cases} \tag{6}$$

In the previous computation, as we relax the binary integral variables to fractional ones, the solution may contain some x_j^m that are decimal fractions. A fractional x_j^m means that we need to divide the VM to place it separately on different PMs, which is not practicable. Thus, we resort to rounding x_j^m by placing VM_j on PM m' where $m' = \max_m x_j^m$ and set $x_j^{m'} = 1$. In summary, the rationale for our algorithm is to integrate coflow scheduling and VM placement to achieve optimal scheduling, and then let VM placement cooperate to approximate this goal. The heuristic for pursuing a minimal CCT is summarized in Algorithm 2.

Finally, we analyze the performance of Algorithm 2 using the standard notion of approximation ratio. The

Algorithm 2 Minimizing single coflow completion time

```
1:  procedure MINSINGLECCT
2:      input: Coflow $i$ $C_i$, residual slots on PMs $\mathcal{S}$, residual bandwidth on PMs $\mathcal{B}$, current VM placement $X$
3:      output: $t_i$, placement and scheduling of $C_i$
4:      Retrieving the traffic matrix of $C_i$, store it in $p_i$
5:      Solve problem (5) with $p_i$, $\mathcal{S}$, $\mathcal{B}$ and $X$
6:      for each VM $j$ of $C_i$ do
7:          Find $m'$ where $m' = \max_m x_j^m$
8:          Set $x_j^{m'} = 1$
9:          Set other $x_j^m = 0$
10:     end for
11:     substitute $x_j^m$ to Eq. (4) and compute $\alpha_i$
12:     $t_i = 1/\alpha_i$
13:     return $t_i$, $X_i$, $B_i$
14: end procedure
```

approximation ratio of Algorithm 2 is defined as the supremum of $t_{\min}/t_i$, where $t_{\min}$ is the minimum CCT of C_i and t_i is CCT obtained by our algorithm.

Theorem 1 Algorithm 2 has the approximation ratio of 2, that is,

$$t_i \leqslant 2t_{\min}.$$

Proof To prove this theorem, the equivalent proposition is $\alpha_i \geqslant \alpha_{\max}/2$, where α_i and $\alpha_{\max}$ are the inverse of t_i and $t_{\min}$, respectively. We assume that the objective of problem (5) is α_{upper}. Problem (5) is the binary relaxation of problem (4) and the feasible solution of the linear formulation sets an upper bound $\alpha_{\text{upper}} \geqslant \alpha_{\max}$.

Multiplier $x_j^m(1-x_k^m)$ is mathematically equal to a graph cut where j and k are vertices and edge $\langle j,k\rangle$ cuts vertex m from $M-m$. The physical meaning of this cut is that we place VM j in host h_m with the proportion x_j^m. In our heuristic, we only pick the x_j^m with the largest value and discard the rest. Let $C_{(j,k)}^m$ be the weight of cut $\langle j,k\rangle$, and let the value of x_j^m be the value of $C_{(j,k)}^m$. Accordingly, let a_j^m be the value of cut $A_{(j,k)}^m$ in the optimal solution. Finally, we have a set of cuts C that separates every m from M, which is the final placement.

Note that each cut $C_{(j,k)}^m$ is an incident at two of these components. Each edge will be in two of the cuts, hence, $\sum w(C_{(j,k)}^m) = 2w(C)$. Therefore, we have

$$\sum_m x_j^m(1-x_k^m) \leqslant \sum_m w(C_{(j,k)}^m) \leqslant \sum_m w(A_{(j,k)}^m) = 2w(A),$$

$$\alpha_i = \frac{\sum\limits_m V_{i,j,k}^m}{\sum\limits_m (x_j^m(1-x_k^m))} \geqslant \frac{\sum\limits_m V_{i,j,k}^m}{2\sum\limits_m (a_j^m(1-a_k^m))} = \frac{\alpha_i^{\text{upper}}}{2} \geqslant \frac{\alpha_i^{\max}}{2}. \qquad \blacksquare$$

The analytical result provides the worst-case guarantee of this algorithm. Theoretically, a loose bound is derived, but in practice this algorithm works effetively in improving the CCT for real applications in the cloud environment.

5 Evaluation

5.1 Experiment setup

Testbed. The testbed is built using a leaf-spine architecture. We select 18 servers from 2 racks connected to 2 leaf top-of-rack switches with 1-Gbps links. Each leaf switch is connected to a spine switch using 10-Gbps links, ensuring full bisection bandwidth. The switches are configured as standard output-queued switches. The server is equipped with two 6-core Xeon 2.66 GHz processors, 24 GB memory, and a dedicated hard drive (a maximum of 12 VM slots). Each VM is configured with 2–4 GB memory, 100–200 GB hard disk, and 1–2 dedicated CPU cores (1–2 VM slots). The server and network oversubscription ratio are set to be 1.

Workloads. The real-world workloads of the applications we have deployed are as follows:

RECOMM is a Hadoop-based[7] application that conducts several rounds of collaborative filtering to recommend movies to users. We synthesized 1, 2, and 5 GB datasets from MovieLens.

PAGER is a Spark-based[8] application that computes page-rank values for an input graph. The input graphs generated by Spark has 0.5, 1, and 4 million nodes.

Given the combination of applications and input data sizes, we collect 6 sets of workload traces. Each of them is deployed on 10–40 VMs. We collect the statistics of these coflows' characteristics as defined in Varys[6]. As shown in Table 2, the workload traces cover a wide range of applications with various coflow characteristics.

We also deploy on 60 VMs with background traffic. These VMs communicate with a varying number of random VMs in a uniform distribution between 5 and 20. The data communication between a pair of VMs is uniformly distributed between 1 and 400 Mbps.

Comparison of schemes. We have implemented the following:

As introduced in Section 2, the SEBF algorithm is the optimal method of coflow scheduling. We implement

Table 2 Benchmarking applications and coflow characteristics.

Application		Coflow characteristics			
Name	Dataset	Size (GB)	Width	Length (GB)	Skew (GB)
PAGER-0.5M	0.5×10^6	2.98	10	0.72	0.3
PAGER-1M	1×10^6	4.45	20	0.98	0.45
PAGER-4M	4×10^6	19.53	40	2.55	0.92
RECOMM-1G	1 GB	7.68	10	4.88	1.5
RECOMM-2G	2 GB	16.21	20	8.54	3.78
RECOMM-5G	5 GB	33.37	40	12.4	4.21

the SEBF algorithm and use fair sharing of bandwidth as its counterpart. Regarding VM management, we compare traffic-aware VM management, a graph-cut-based implementation of which is demonstrated in Ref. [21], against random placement. We obtain the following 4 combinatorial mechanisms:

Baseline represents the naive situation in which no VMs are co-resident and flows are scheduled based on fair sharing.

SEBF-R represents the situation in which coflows are scheduled by SEBF method and VMs are randomly placed.

FS-TA represents the scenario with fair sharing among flows but without coflow-aware scheduling, and in which VMs are placed by the traffic-aware VM placement.

SEBF-TA represents the scheme in which the coflow scheduling with SEBF and traffic-aware VM placement are conducted independently.

The major competitor of *VirtCO* is **SEBF-TA** because these two state-of-the-art methods are most commonly used. By comparing these schemes, we can examine the benefits results from the two elements of *VirtCO*: VM placement and scheduling.

5.2 Experiment results

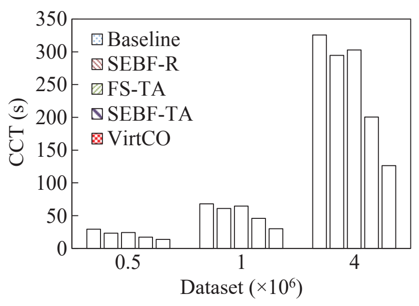
In this section, the experiment is organized to answer the following questions:

• How effective is *VirtCO* by jointly considering VM placement and coflow scheduling? (test cases 1–3)

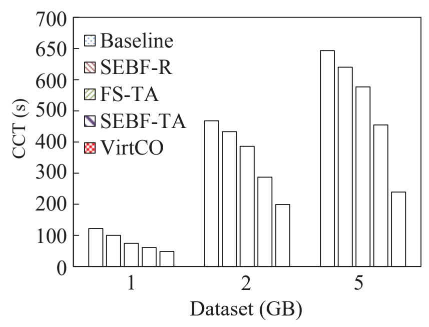
• When is *VirtCO* most beneficial in improving the average CCT? (test cases 4–6)

To define the effectiveness of *VirtCO*, we use the relative speed-up, which is defined as the amount of time that *VirtCO* saved (or added) to the completion time of an application. For instance, if an application runs for 100 s with SEBF-TA and 80 s with *VirtCO*, then the relative speed-up is $(100-80)/100=20\%$.

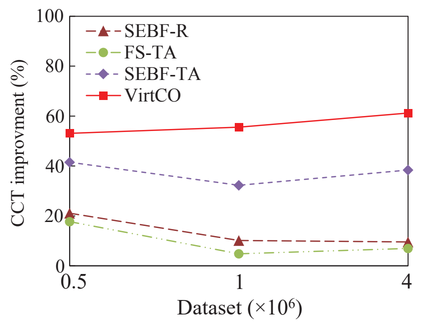
Test case 1 (**Minimizing the single CCT**). We compare the completion time of workloads under *VirtCO* and its competitors. The results are shown in Fig. 3. We observe that, for the PAGER application, *VirtCO* achieves a performance improvement of approximately 58.3% over Baseline. SEBF-TA



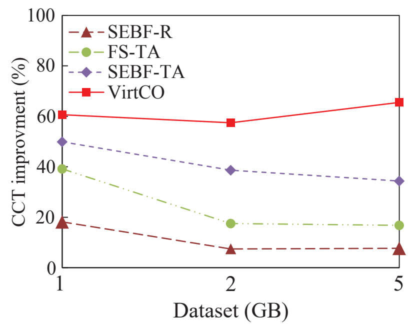
(a) CCT comparison of PAGER



(b) CCT comparison of RECOMM



(c) CCT improvement of PAGER compared with Baseline



(d) CCT improvement of RECOMM compared with Baseline

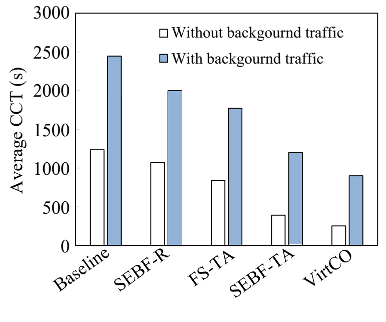
Fig. 3 Effectiveness of *VirtCO* in improving single CCT.

experiences a decrease in performance of approximately $16.7\% = 58.3\% - 41.6\%$ when compared with *VirtCO*. The improvement is more obvious for the RECOMM applications, the traffic skew of which is higher than that of other applications. Running with various dataset sizes, *VirtCO* improves CCT by more than 60% over Baseline. With the increasing dataset size, we observe larger performance losses by SEBF-TA because under *VirtCO*, the bottleneck flows can utilize the entire PM bandwidth to accelerate the CCT. The results of these real workloads indicate the effectiveness of *VirtCO* in reducing the completion time of a single coflow, mainly because of the ability of this framework to efficiently utilize the network.

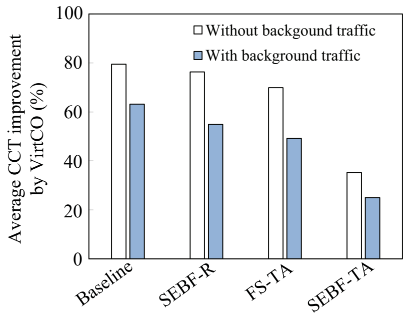
Compared with other schemes, SEBF-R is 3%–4% better than FS-TA for the PAGER applications. We find the opposite for RECOMM, for which FS-TA is up to 20% better. The takeaway is that VM placement contributes the most toward reducing CCT with a large skew, because when the traffic matrix is highly skewed, traffic-aware placement can significantly reduce data transfer. When the number of flows increases, scheduling plays a more important role.

Test case 2 (**Minimizing the average CCT**). The primary goal of our method is to reduce the average CCT so that the overall network efficiency of the data center can be improved. In this test case, we measure the average CCT when running all 6 benchmarking workloads concurrently. The results are presented in Fig. 4. Without background traffic, the improvement that *VirtCO* achieves over SEBF-TA is approximately 36.5%. The improvement over Baseline is as high as 79.1%. In the scenario where background traffic exists, the average CCT increases and the performance improvement decreases. As our framework has no control over the background traffic, it occupies available slots and reduces the optimization space. Although the background traffic restricts its benefits, *VirtCO* still achieves a 24.1% improvement over SEBF-TA in terms of average CCT.

Test case 3 (**Overhead**). The computation overhead is closely related to the efficiency and scalability of *VirtCO*. For the controller, the overhead we are concerned with is the time required to generate the joint scheduling at the arrival of each coflow. In previous test cases, we record the computation time of each round of scheduling and placement decisions. The results are shown in the Cumulative Distribution Function (CDF) figures in Fig. 5. We observe from Fig. 5a that, in our settings, the computation overhead is acceptable such that more than 80% of the decisions can be derived within 5.5 s. As our algorithm runs either in the VM launching or profiling phase, the extra computation time is negligible compared with the VM launch time (typically several minutes).



(a) Average CCT comparison under different schemes

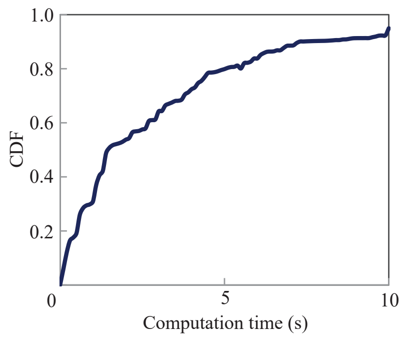


(b) Average CCT improvement achieved by *VirtCO* when compared to various competitors

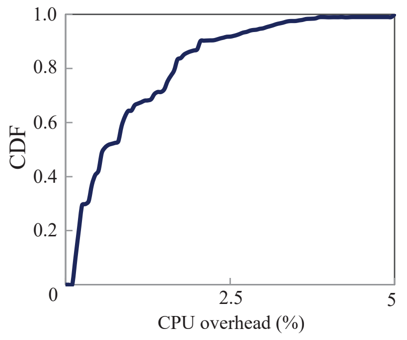
Fig. 4 Effectiveness of *VirtCO* in improving average CCT.

Regarding the overhead introduced by the edge module, we are mainly concerned with the extra CPU usage when enforcing scheduling policies. We continually measure the extra CPU usage during coflow transferring compared with the case in which no edge module is used (no rate limiting). As shown in Fig. 5b, the CPU overhead is less than 2% in 80% of the time (one core). The results confirm the practicability of the software and lightweight implementation of *VirtCO*.

Test case 4 (**Impact of coflow width**). In each round of the experiment, we send 20 coflows with the same width into the network. We observe from Fig. 6 that the performance improvement of SEBF-TA gradually degrades. When the coflow width is small, VMs are more likely to be placed in the same host; thus, the traffic could be reduced considerably, as could the CCT. With increasing coflow width, greater network collision occurs, especially in the virtualization environment.

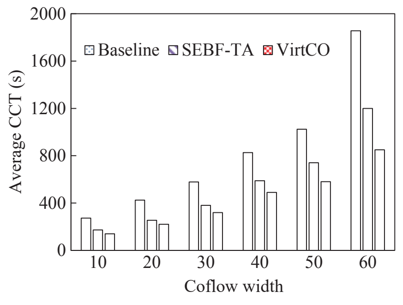


(a) Computation time of each round of decision

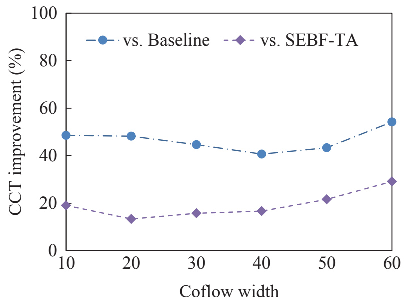


(b) CPU overhead on each hypervisor

Fig. 5 Overhead introduced by *VirtCO*.



(a) Average CCT comparison



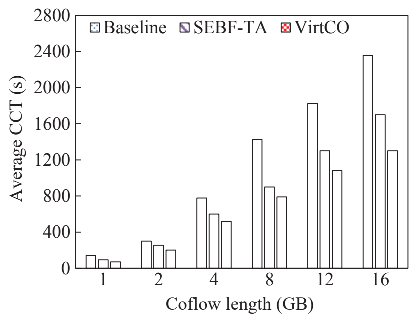
(b) Performance improvement achieved by *VirtCO*

Fig. 6 Impact of coflow width.

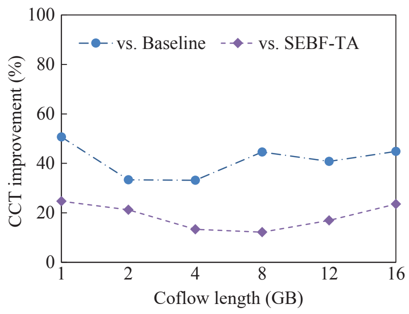
As flows are not aware of their co-residence, network collision may occur; thus the benefit of SEBF-TA decreases. When the coflow width is sufficiently small, the placement dominates the performance. However, as the width further increases, network collision dominates the performance degradation, where *VirtCO* demonstrates its benefits. *VirtCO* is highly competitive and is recommended for scenarios with large coflow widths.

Test case 5 (**Impact of coflow length**). Coflow length is the size of the largest flow of the coflows. As shown in Fig. 7, the coflow length dominates its CCT so that the CCT under Baseline increases nearly linearly. In this case, the improvement from using *VirtCO* is stable at 40%–50% of Baseline. Traffic-aware VM placement is the most beneficial in this scenario in which the co-resident VMs can considerably reduce traffic. Our method inherits this benefit, and makes further improvements by considering the coflow semantics of applications. In this case, *VirtCO* leads to a 17%–20% improvement over SEBF-TA in this test case. From Fig. 7b, minimal correlation can be observed between performance improvement and increasing coflow length.

Test case 6 (**Impact of coflow skew**). To investigate the impact of coflow skew, we send out coflows with skew in the range of 0.1–3 GB. Figure 8 shows that *VirtCO* outperforms other methods when the coflow skew increases. Initially, with a balanced dataset, the performance improvements of *VirtCO* over Baseline and SEBF-TA are 23.5% and 30.6%, respectively.

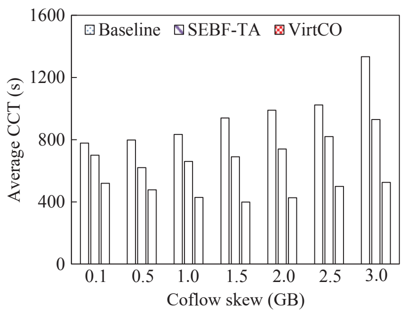


(a) Average CCT comparison

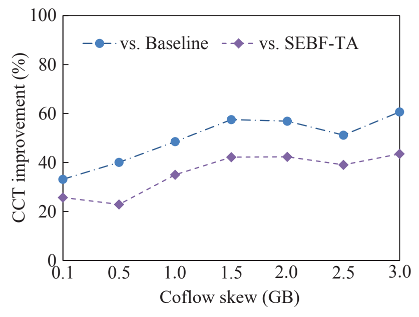


(b) Performance improvement achieved by *VirtCO*

Fig. 7 Impact of coflow length.



(a) Average CCT comparison



(b) Performance improvement achieved by *VirtCO*

Fig. 8 Impact of coflow skew.

The difference between *VirtCO* and SEBF-TA is small: SEBF-TA experiences a performance loss of only 7.1%. However, the improvement of *VirtCO* is particularly obvious when the skew is as large as 3 GB, with 61.2% and 40.5% improvements over Baseline and SEBF-TA, respectively. Skew is a characteristic that might significantly influence both the VM placement and coflow scheduling. The separate consideration of these two important gradients is insufficient, which agrees with the theoretical analysis of this problem. *VirtCO* is most beneficial for large coflow skews.

In summary, *VirtCO* significantly reduces the average CCT in the virtualization environment with limited overhead. *VirtCO* is most beneficial when the skew and width of coflows are large; these are typical characteristics of emerging big data applications and parallel processing frameworks[6].

6 Related Work

Flow scheduling: Coflow was introduced by Chowdhury et al.[6], they proposed the SEBF heuristic for scheduling coflows. Then Chowdhury and Stoica[10] introduced the non-clairvoyant coflow scheduling without priori knowledge of coflow information. Qiu et al.[11] formulated a discrete model to solve the coflow scheduling problem. Zhao[15] considered routing with scheduling and proposed a coflow-aware network optimization framework that integrates routing and scheduling for better application performance. Zhang et al.[17] proposed a method for identifying coflows through traffic analysis. Chen et al.[22] optimized CCTs with utility max-min fairness. Lu[23] designed an SDN-enhanced flow scheduling method to simultaneously meet deadlines and ensure the rates for non-deadline flows.

Network-ware task scheduling: Extensive research has been conducted on network-ware task scheduling with the same objective as ours. To name a few, Refs. [16, 24, 25] optimized the task placement with network flow considerations. Although these studies were aimed at task-level scheduling, they do not account for VMs. In the multi-tenant environment, virtualization prevails for the advantages in security and ease of management. To this end, instead of scheduling tasks in an unknown environment, *VirtCO* enables the data center provider to conduct wise and coordinated scheduling with a panoramic view of the infrastructure. Moreover, *VirtCO* does not contradict network-ware task scheduling and composing *VirtCO* and network-ware task scheduling could lead to an improvement in network efficiency.

Traffic-aware VM management: Traffic-aware VM management problem was first modeled by Meng et al.[13]; in this study, VMs with large mutual bandwidth usage are assigned to PMs in close proximity. The literature on this topic is vast, and we only name a few here. Li et al.[14] addressed the tradeoff between communication cost and resource utilization. Zhao et al.[26] optimized both VM placement and topology design to achieve higher traffic scalability. Wang et al.[27] addressed the issue of VM migration. Li et al.[28] proposed a VM allocation mechanism to minimize the sum of the VMs' network diameters for all tenants. Additionally, many VM management mechanisms account for other resources, such as CPU and memory. As these resources are explicitly requested when launching VMs, *VirtCO* considers them as constraints for VM placement. Furthermore, *VirtCO* is compatible with VM placement for various objectives.

7 Discussion and Conclusion

Network impact on application performance. Although some researchers have noted a limited effect of the network on the overall performance of big data applications[29], recent study[30] has demonstrated significant performance gains for applications moving from a 1 Gbps to a 10 Gbps network. Our work confirms the latter result; this bottleneck is only relieved when the network reaches 40 Gbps. In multi-tenant cloud environments, network resources are shared among tenants, and no single tenant can exclusively utilize high bandwidths. Therefore, optimizing network efficiency in this scenario could benefit application performance and release larger space to accommodate additional requests.

VM migration. In our experimental analysis, we find that VM migrations could further improve network efficiency by dynamically adjusting VM placement although the overhead incurred is extremely high. In the extreme case, memory pages and tens of gigabytes of local data are transferred during migration. Thus, in this study, we avoid migrations by determining VM placement ahead of VM launch. We plan to consider timely and cautious migration decisions in future work.

To conclude, this study addressed the problem of joint coflow scheduling and VM placement in cloud data centers. A practical solution, *VirtCO*, was designed and implemented to combine VM placement and coflow scheduling seamlessly. We formulated a joint coflow scheduling and VM placement problem and proposed efficient heuristics for solving it. Extensive evaluations with real-world applications showed that *VirtCO* preserves remarkable performance advantages over state-of-the-art mechanisms. Furthermore, *VirtCO* is practical and readily deployable within current data center architectures. Taking joint coflow and VM scheduling as a starting point, *VirtCO* demonstrates that VM management and flow scheduling can be jointly optimized for improved network infrastructure efficiency in data centers. Moreover, with the emergence of geo-distributed data centers[31], *VirtCO* provides a practical solution for managing the network-driven infrastructure.

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2017YFB1003000), the National Natural Science Foundation of China (Nos. 61572129, 61602112, 61502097, 61702096, 61320106007, and 61632008), the International S&T Cooperation Program of China (No. 2015DFA10490), the National Science Foundation of Jiangsu Province (Nos. BK20160695 and BK20170689), the Jiangsu Provincial Key Laboratory of Network and Information Security (No. BM2003201), the Key Laboratory of Computer Network and Information Integration of Ministry of Education of China (No. 93K-9), and partially supported by the Collaborative Innovation Center of Novel Software Technology and Industrialization and Collaborative Innovation Center of Wireless Communications Technology.

References

[1] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/, 2018.

[2] J. C. Mogul and L. Popa, What we talk about when we talk about cloud network performance, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'12)*, Helsinki, Finland, 2012, pp. 44–48.

[3] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, The only constant is change: Incorporating time-varying network reservations in data centers, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'12)*, Helsinki, Finland, 2012, pp. 199–210.

[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 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'12)*, Toronto, Canada, 2011, pp. 98–109.

[5] J. Jiang, S. Ma, B. Li, and B. Li, Symbiosis: Network-aware task scheduling in data-parallel frameworks, in *Proceedings of IEEE Conference on Computer Communications (INFOCOM'16)*, San Francisco, CA, USA, 2016, pp. 1–9.

[6] M. Chowdhury, Y. Zhong, and I. Stoica, Efficient coflow scheduling with varys, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'14)*, Chicago, IL, USA, 2014, pp. 443–454.

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

[8] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, MJ. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, San Jose, CA, USA, 2012, pp. 2–2.

[9] G. Malewicz, M. H. Austern, A. J. C. 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 International Conference on Management of Data (SIGMOD'10)*, Indianapolis, IN, USA, 2010, pp. 135–146.

[10] M. Chowdhury and I. Stoica, Efficient coflow scheduling without prior knowledge, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'15)*, London, UK, 2015, pp. 393–406.

[11] Z. Qiu, C. Stein, and Y. Zhong, Minimizing the

total weighted completion time of coflows in datacenter networks, in *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'15)*, Portland, OR, USA, 2015, pp. 294–303.

[12] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J. Kang, and P. Sharma, Application-driven bandwidth guarantees in datacenters, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'14)*, Chicago, IL, USA, 2014, pp. 467–478.

[13] X. Meng, V. Pappas, and L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in *Proceedings of IEEE Conference on Computer Communications (INFOCOM'10)*, San Diego, CA, USA, 2010, pp. 1–9.

[14] X. Li, J. Wu, S. Tang, and S. Lu, Let's stay together: Towards traffic aware virtual machine placement in data centers, in *Proceedings of IEEE Conference on Computer Communications (INFOCOM'14)*, Toronto, Canada, 2014, pp. 1842–1850.

[15] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Yang, D. Li, and S. Wang, Rapier: Integrating routing and scheduling for coflow-aware data center networks, in *Proceedings of IEEE Conference on Computer Communications (INFOCOM'15)*, Hong Kong, China, 2015, pp. 424–432.

[16] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, Network-aware scheduling for data-parallel jobs: Plan when you can, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'15)*, London, UK, 2015, pp. 407–420.

[17] H. Zhang, L. Chen, B. Yi, K. Chen, M Chowdhury, and Y. Geng, Coda: Toward automatically identifying and scheduling coflows in the dark, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'16)*, Florianopolis, Brazil, 2016, pp. 160–173.

[18] K. LaCurts, J. C. Mogul, H. Balakrishnan, and Y. Turner, Cicada: Introducing predictive guarantees for cloud networks, in *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'14)*, Philadelphia, PA, USA, 2014, pp. 14–19.

[19] J. Perry, H. Balakrishnan, and D. Shah, Flowtune: Flowlet control for datacenter networks. in *Proceedings of USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*, Boston, MA, USA, 2017, pp. 421–435.

[20] OpenStack Open Source Cloud Computing Software, https://www.openstack.org/, 2018.

[21] D. Shen, J. Luo, F. Dong, and J. Zhang, Appbag: Application-aware bandwidth allocation for virtual machines in cloud environment, in *45th International Conference on Parallel Processing (ICPP)*, Philadelphia, PA, USA, 2016, pp. 21–30.

[22] L. Chen, W. Cui, B. Li, and B. Li, Optimizing coflow completion times with utility max-min fairness, in *Proceedings of IEEE Conference on Computer Communications (INFOCOM'16)*, San Francisco, CA, USA, 2016, pp. 1755–1763.

[23] Y. Lu, Sed: An SDN-based explicit-deadline-aware TCP for cloud data center networks, *Tsinghua Science and Technology*, vol. 21, no. 5, pp. 491–499, 2016.

[24] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, Shufflewatcher: Shuffle-aware scheduling in multi-tenant MapReduce clusters, in *Proceedings of USENIX Annual Technical Conference (ATC'14)*, Philadelphia, PA, USA, 2014, pp. 1–12.

[25] A. Munir, T. He, R. Raghavendra, F. Li, and A. X. Liu, Network scheduling aware task placement in datacenters, in *Proceedings of the International Conference on Emerging Networking Experiments and Technologies (CoNEXT'16)*, Irvine, CA, USA, 2016, pp. 221–235.

[26] Y. Zhao, Y. Huang, K. Chen, M. Yu, S. Wang, and D. S. Li, Joint VM placement and topology optimization for traffic scalability in dynamic datacenter networks, *Computer Networks*, vol. 80, pp. 109–123, 2015.

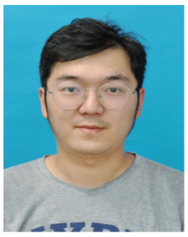
[27] H. Wang, Y. Li, Y. Zhang, and D. Jin, Virtual machine migration planning in software-defined networks, in *Proceedings of IEEE Conference on Computer Communications (INFOCOM'15)*, Hong Kong, China, 2015, pp. 487–495.

[28] J. Li, D. Li, Y. Ye, and X. Lu, Efficient multi-tenant virtual machine allocation in cloud data centers, *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 81–89, 2015.

[29] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B. G. Chun, Making sense of performance in data analytics frameworks, in *Proceedings of USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*, Oakland, CA, USA, 2015, pp. 293–307.

[30] A. Trivedi, P. Stuedi, J. Pfefferle, R. Stoica, B. Metzler, I. Koltsidas, and N. Ioannou, On the [ir] relevance of network performance for data processing, in *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'16)*, Denver, CO, USA, 2016, pp. 126–131.

[31] J. Zhang, J. Chen, J. Luo, and A. Song, Efficient location-aware data placement for data-intensive applications in geo-distributed scientific data centers, *Tsinghua Science and Technology*, vol. 21, no. 5, pp. 471–481, 2016.



Dian Shen is currently an assistant professor in the School of Computer Science and Engineering, Southeast University, China. He received the bachelor, master, and PhD degrees from Southeast University, China, in 2010, 2012, and 2018, respectively. His research interests include cloud computing, virtualization, and data center network.

Junzhou Luo is a full professor in the School of Computer Science and Engineering, Southeast University, China. He received the BS degree in applied mathematics from Southeast University in 1982, and then got the MS and PhD degrees in computer science both from Southeast University in 1992 and 2000, respectively. His research interests include network security, cloud computing, and wireless LAN.

Fang Dong is currently an associate professor in School of Computer Science and Engineering, Southeast University, China. He received the BS and MS degrees in computer science from Nanjing University of Science and Technology, China, in 2004 and 2006, respectively, and received the PhD degree in computer science from Southeast University in 2011. His current research interests include cloud computing, task scheduling, and big data processing.

Junxue Zhang is currently a PhD candidate in the Department of Computer Science and Engineering (CSE) at Hong Kong University of Science and Technology. He received the bachelor and master degrees from Southeast University, China, in 2013 and 2016, respectively. His current interest is data center networking.