Efficient Scheduling of Weighted Coflows in Data Centers

Zhiliang Wang, Han Zhang, *Member, IEEE*, Xingang Shi, Xia Yin, Yahui Li, *Member, IEEE*, Haijun Geng, Qianhong Wu, and Jianwei Liu

Abstract—Traditional network resource management mechanisms are mainly flow or packet based. Recently, coflow has been proposed as a new abstraction to capture the communication patterns in a rich set of data parallel applications in data centers. Coflows effectively model the application-level semantics of network resource usage, so high-level optimization goals, such as reducing the transfer latency of applications, can be better achieved by taking coflows as the basic elements in network resource allocation or scheduling. Although efficient coflow scheduling methods have been studied, in this paper, we advocate to schedule *weighted coflows* as a further step in this direction, where weights are used to express the importances or priorities of different coflows or their corresponding applications. We propose the Weighted Coflow Completion Time (WCCT) minimization problem and a $(2-\frac{2}{n+1})$-approximate optimal offline algorithm, where n is the concurrent number of coflows. We then design an information-agnostic online algorithm named IAOA to dynamically schedule coflows according to their weights and the instantaneous network condition. We also design and implement a coflow scheduling system named FlyTransfer, which can use the online algorithm as its scheduling method. We test the performance of FlyTransfer by trace-driven simulations as well as real deployment in openstack. Our evaluation results show that, compared to the latest information-agnostic coflow scheduling algorithms, FlyTransfer can reduce more than 40 percent of the WCCT, and more than 30 percent of the completion time for coflows with above-the-average level of importance. It even outperforms the most efficient clairvoyant coflow scheduling method by reducing around 30 percent WCCT, and 25- 30 percent of the completion time for coflows with above-the-average importance, respectively.

Index Terms—DataCenter, coflow, weight, CCT, WCCT

1 Introduction

Nowadays, low latency [29], [36] and high throughput [30] are required by data center applications, for example, those with map-reduce style data-intensive computing [20], and those using distributed file storage [21], [32], etc. To meet these requirements, the data center network infrastructures have been specially tailored, with a tremendous amount of effort in topology design, routing schemes, as well as transport schemes.

Among these efforts, flow level scheduling methods try to schedule arriving packets according to the characteristics of the corresponding flows. For example, PDQ [24] and pFabric [13] approximate the *shortest job first* (SJF) policy to let shorter flows preempt the bandwidth of longer ones. As a result, the average Flow Completion Time (FCT) tends to be reduced, so does applications' transport latency. However, since the communication of an application cannot finish until all its flows have completed their transmission, the overall transport performance of an application may not be effectively improved. Indeed, the performance may be impaired by scheduling individual flows without considering their inter-relationship.

Recently, coflow has been proposed as a new abstraction to capture the communication patterns in a rich set of data parallel applications. According to [17], a coflow is *a collection of flows between two groups of machines with associated semantics and a collective objective*. Those flows share a common performance goal, e.g., minimizing the completion time of the slowest flow in the collection, or ensuring that flows in the collection meet a common deadline. Coflows effectively model the application-level semantics of network resource usage, so high-level optimization goals (i.e., reducing the transfer latency of applications) can be better achieved by taking coflows as the basic elements in network resource allocation or scheduling. Efficient coflow scheduling methods on minimizing average Coflow Completion Time (CCT) have been studied. Varys [19] proposes the *Smallest-Effective-Bottleneck-First* (SEBF) heuristic to determine the scheduling order of coflows and uses *Minimum-Allocation-for-Desired-Duration* (MADD) to compute the

• H. Zhang, Q. Wu, and J. Liu are with the School of Cyber Science and Technology, Beihang University, Beijing 100191, China. E-mail: {zhhan, qianhong.w, liujianwei}@buaa.edu.cn.
• Z. Wang and X. Shi are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the Beijing National Research Center for Information Science and Technology (BNRist), Beijing 100084, China. E-mail: {wzl, shixg}@cernet.edu.cn.
• X. Yin and Y. Li are with Beijing National Research Center for Information Science and Technology (BNRist), Beijing 100084, China. E-mail: yxia@tsinghua.edu.cn, li-yh15@mails.tsinghua.edu.cn.
• H. Geng is with the School of Software Engineering, Shanxi University, Taiyuan 030006, China. E-mail: genghaijunzt@163.com.

Manuscript received 1 Aug. 2018; revised 3 Mar. 2019; accepted 5 Mar. 2019. Date of publication 18 Mar. 2019; date of current version 7 Aug. 2019. (Corresponding author: Han Zhang.) Recommended for acceptance by Y. Yang. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2019.2905560

bandwidth to be allocated. However, Varys is a clairvoyant method which determines the priority of coflows depending on flow information (flow size, flow arrival time, etc). This may limit its applicability since these information may not be provided beforehand. To solve this problem, non-clairvoyant methods that are flow information agnostic, such as Aalo [18], Barrat [22], sunflows [25] and CODA [48] have also been studied.

In this paper, we advocate to schedule *weighted coflows* as a further step in this direction, where weights are assigned to coflows to express the importance of coflows or their corresponding applications. For example, supporting the internal communication requirement of a search engine with intense internal computation should be more important than supporting the file backup on a distributed storage system. When scheduling coflows, we would like to take this difference into consideration. To accomplish this, we assign different weights to them, i.e., coflows of search engine have a higher weight, while coflows of file backup have a lower weight, and try to minimize the Weighted Coflow Completion Time (WCCT), which is the weighted summation of the completion time of all coflows.

We formulate an Idealized Weighted Coflow Completion Time Minimization (IWCCTM) problem, and prove that one of its special cases is equivalent to minimizing the weighted job completion time in the concurrent open shop problem [35], which is known to be NP-hard. We derive for IWCCTM a non-preemptive scheduling algorithm that is ($2-\frac{2}{n+1}$)-approximate optimal, where n is the concurrent number of coflows. We convert the non-preemptive scheduling algorithm to an **i**nformation-**a**gnostic **o**nline **a**lgorithm, which is named IAOA. IAOA introduces only minor performance loss, but can schedule coflows online without knowing their size or arrival time a priori. We further design and implement a scheduling system named FlyTransfer that can readily be deployed in a productional cloud environment like openstack. We test the performance of IAOA by both trace-driven simulations and testbed evaluations. We then compare it against state-of-the-art coflow scheduling algorithms such as Varys [19], Aalo [18], Barrat [22], Sincronia [11] and the polynomial-time deterministic algorithm [42]. The results show that IAOA performs quite effectively in reducing the average weighted coflow completion time. Our contributions in this paper are summarized as follows:

- We propose the scheduling of *weighted coflows* as an essential network resource management scheme in data centers hosting multiple kinds of applications. We collect real application traffic from a medium-sized data center and make an in-depth analysis to reveal the importance of weighted coflow scheduling.
- We formulate an Idealized Weighted Coflow Completion Time Minimization problem, study its hardness, and derive a non-preemptive scheduling algorithm and prove it is ($2-\frac{2}{n+1}$)-approximate optimal, where n is the concurrent number of coflows.
- We further design a practical information-agnostic online algorithm named IAOA and build the FlyTransfer system to minimize WCCT in real environment. FlyTransfer can readily be deployed in a cloud environment like openstack.
- Evaluation results show that, compared to the latest information-agnostic coflow scheduling algorithms, FlyTransfer with IAOA can reduce more than 40 percent of the average WCCT, and more than 30 percent of the completion time for coflows with above-the-average level of importance. It even outperforms the most efficient clairvoyant coflow scheduling method by reducing around 30 percent WCCT, and 20-30 percent of the completion time for coflows with above-the-average emergence, respectively.

The rest of the paper is organized as follows. Section 2 introduces the related work and our motivation. Section 3 formulates the WCCT minimization problem and discusses its hardness. Section 4 presents the offline algorithm and the information-agnostic online coflow scheduling algorithm IAOA in detail. Section 5 shows the design details of the coflow transfer system FlyTransfer. Section 6 evaluates IAOA against several coflow scheduling methods, and Section 7 concludes.

2 Background and Motivation

Data center is now becoming an important facility for hosting a large number of services and applications. To meet applications' demands of high throughput and low latency, a tremendous amount of research effort has been devoted to network resource allocation. For example, DCTCP [12], D^2 TCP [44], L^2 DCT [37], LPD [49], D3 [47], PIAS [14] and PDQ [24] are flow level rate control or scheduling schemes aiming to minimize the average Flow Completion Time or to meet strict flow deadlines. The concept of coflow [17] has recently been proposed to meet the application performance requirement at a higher level, where a coflow is a collection of flows between two groups of machines with associated semantics and a collective objective. [45] makes a survey on coflow scheduling and divides coflow scheduling methods into two categories: information-aware methods and information-agnostic methods. Varys computes network bottleneck based on flow size and applies its *Effective-Bottleneck-First* heuristic, while MCS [46], Aalo [18], sunflows [25] and CODA [48] try to "guess" various characters of coflows in different ways to help scheduling. The former methods are often called clairvoyant methods which schedule coflows with the help of flow information like size and arrival time, while the latter four are flow information agnostic. D-CAS [33] and Stream [43] schedule coflows in decentralized manner. Rapier [51] and OMCoflow [31] consider both routing and scheduling.

Although these coflow level scheduling methods indeed effectively improve the communication or data transfer performance, they treat different coflows or applications as equally important. As a result, the difference of performance improvement between different coflows or applications depends on the specific scheduling heuristics. For example, one scheduling scheme may favor short flows, while another may favor coflows with larger width, i.e., consisting of a large number of flows, sometimes even unpredictable or inconsistent. In this sense, we argue that coflows (or applicaitons) often have priorities or emergency levels, and should be explicitly taken into consideration when they are scheduled. For example, supporting the internal communication requirement of a search engine with intense internal computation

TABLE 1
Applications and Their Levels of Importance

App Name	Type	width	length (MB)	Importance
Event	communication	20	5	significant
vRouter	communication	8	3	significant
Druid	interactive	6	18	important
Hadoop	computation	5	42	normal
Web	interactive	3	5	normal
VoltDB	background	4	21	normal
Hive	background	7	32	unimportant
Redies	background	2	30	unimportant
data-backup	background	3	124	lax
data-dist	background	5	93	lax

TABLE 2
Coflows in Hardoop and Their Levels of Importance

Coflow Type	width	length (MB)	Importance
index-sort	10	3	significant
db-analysis	3	12	important
index-count	6	20	normal
log-analysis	6	31	unimportant
crawler	4	12	unimportant
word-count	3	11	unimportant

should be more emergent than supporting the file backup on a distributed storage system. Although in this example the more emergent coflows mainly consist of short flows and tend to have a large width, this is not always the case, and in general emergency is orthogonal to other physical characteristics. Thus, the scheduling methods introduced above lack the capability to improve the performance of coflows with the strongest demand.

To make this argument more convincing, we now use real network traffic from Equinix's medium-sized data center located in Washington, D.C. More than 100 applications run on about 3000 servers simultaneously in this data center. After talking with the network operators and software development engineers about the emergence of different applications, we make an in-depth analysis of the traffic collected from 720 servers deployed in 60 racks.[1] The traffic lasts about one month, and the basic statistics of the top ten network-demanding applications are summarized in Table 1. There are typically five importance levels, i.e., significant, important, normal, unimportant and lax. For example, the event application and the vRoute application are responsible for communication of critical components, and are of the highest level of importance, while data-backup and data-dist are applications running in the background and are of the lowest level of importance. We can see applications consisting of a larger number of flows or smaller flows are not necessarily more important. For example, the average number of flows in Hive (i.e., 7) is larger than that of Web (i.e., 3), but Hive is less important than Web. Another example is that, the average flow size of Web (i.e., 5) is smaller than that of Druid (i.e., 18), but Web is of less important than Druid. This supports our former statement that in general, importance is orthogonal to other physical characteristics of coflows. Table 2 further shows the details of different coflows in the hadoop application, from which we can make the similar conclusions.

To illustrate the ineffectiveness of existing coflow scheduling methods, we plot the average Coflow Completion Time of different coflows, grouped by their levels of importance. Fig. 1a shows the direct measurement result of all applications, while Fig. 1b shows the simulation result when Varys is used to schedule them. Comparing the two plots, we can see that Varys clearly reduces the completion time of less emergent coflows, such as Redies, data-backup and data-dist, at the expenses of the same of even longer completion time of more emergent coflows, such as Event and vRouter. A similar effect can be observed in Figs. 1c and 1d, where the measured CCTs of Hadoop coflows are compared to the scheduling results of Varys. To find the reasons for this, we show a small case where coflow C1 and coflow C2 arrive simultaneously. From Fig. 2a, we can see C1 has 3 flows and C2 has 2 flows. C2 has higher priority than C1. Fig. 2b shows under the premise of using TCP, C1's completion time is 4 and C2's completion time is 6. From Fig. 2c, we can see C1's completion time is 3 and C2's completion time is 6. C1's completion time is reduced by 1 and C2's completion time remains unchanged, while C2 is more important than C1. The reason for this is that Varys adopts *Smallest-Effective-Bottleneck-First* (SEBF) but ignores the importance of coflow. This motivates us when scheduling coflows, the importance of coflows should also be taken into consideration.

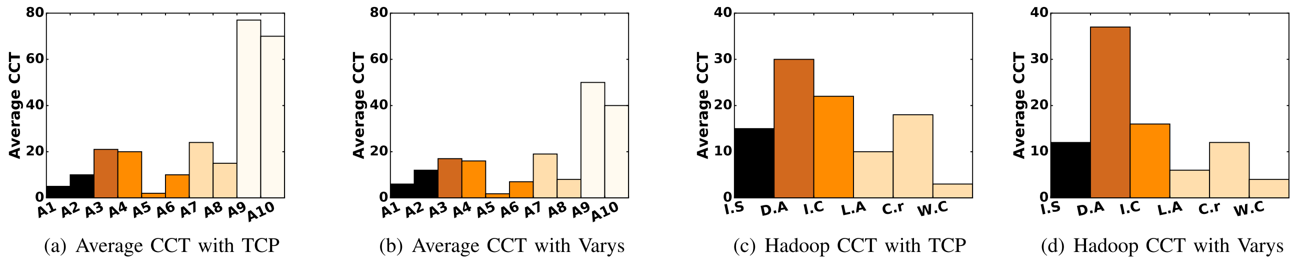
Importance-Priority-First is also not a good choice since it ignores the network condition. Assume coflow C1 and C2 contend at port L1, L2, L3. C1 has 3 flows, whose lengths are 5, 1, 2 and C2 has 2 flows, whose lengths are 1, 1, respectively. C1 is more important than C2. Fig. 3 (a) shows the result of importance priority first. Fig. 3b shows the result of scheduling with coflow importance and network condition. With importance priority first policy, completion time of C1 is 5 and the completion time of C2 is 6. When scheduling with coflow importance and network condition, completion time of C1 is 6 and completion time of C2 is 1. Comparing the two methods, C1 slows down $\frac{6-5}{5} = 20\%$ and C2 accelerates $\frac{6-1}{1} = 5\times$. This motivates us both the importance of coflow and the network condition be taken into consideration when scheduling coflows. In this case, we advocate to optimize the average Weighted Coflow Completion Time (Weight$\times$CCT), where weight denotes the importance of coflows, and the important coflows have larger weight than the unimportant ones.

Since now, some existing literatures [11], [15], [27], [42] have already considered average weight coflow completion optimizaiton. However, these methods always try to solve a convex optimization problem, which may cause long computation time and it is impossible to use in the real world.

3 Model and Problem Formulation

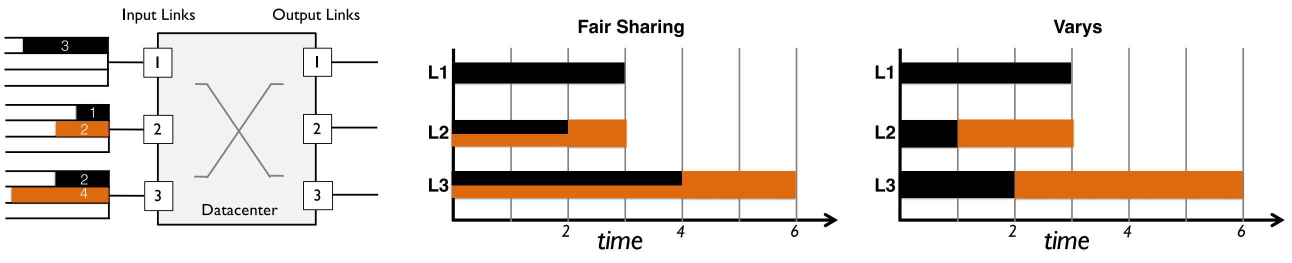
In this section, we first introduce the non-blocking fabric model to reasonably simplify the coflow scheduling

1. We developed a questionnaire to rate the applications of the data center. In the questionnaire, applications can be graded 1, 2, 3, 4, 5, which denote significant, important, normal, unimportant and lax, respectively. We sent the questionnaire to all the operations staff, network administrator and engineers. At last, we compute the average scores of all the applications and get the importance of applications.



(a) Average CCT with TCP (b) Average CCT with Varys (c) Hadoop CCT with TCP (d) Hadoop CCT with Varys

Fig. 1. Normalized Coflow Completion Time, grouped by the level of importance. A1-A10 denote applications (Event - data-dist) in Table 1 and I.S (index-sort), D.A (db-analysis), I.C (index-count), L.A (log-analysis), C.r(crawler), W.C (word count) are abbreviations of hadoop applications.



(a) Coflows contend at the ports (b) Schedule with TCP (c) Schedule with Varys

Fig. 2. C1 contains 3 flows, whose lengths are 3, 1, 2. C2 has 2 flows, whose lengths are 2, 4. C2 has higher importance level.

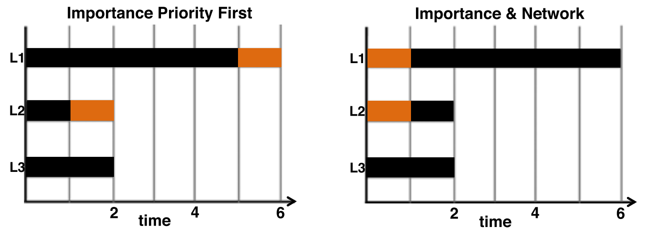
problem in data center networks. Then we formulate an Idealized Weighted Coflow Completion Time Minimization problem and discuss its hardness.

3.1 Non-Blocking Fabric Model for Data Center Networks

Recent researches [13], [18], [19], [25] regard a data center network as a big giant non-blocking switch which interconnects all machines. In this mode, all machines' ports have the same normalized unit capacity(*1*), and flows compete bandwidth only at these ports, i.e., the ingress and egress ports of the big fabric. Such an abstraction is reasonable and matches with the full bisection bandwidth topologies widely used in productional data centers, e.g., the non-blocking Clos topology. In this paper, we use this model and assume congestions occur only at the ingress and egress ports.

3.2 Problem Formulation and Hardness Proof

We consider the scheduling of n coflows in a data center network modeled as a non-blocking fabric with m hosts, i.e., there are m ingresses and m egresses. A coflow is a collection of flows sharing a common performance goal, e.g., minimizing the completion time of the latest flow or



(a) Importance Priority First (b) Schedule with Coflow Importance and Network Condition

Fig. 3. C1 (black) and C2 (orange) contend at port L1, L2, L3. C1 has 3 flows, whose length are 5, 1, 2 and C2 has 2 flows, whose length are 1, 1. Link capacity is 1.

ensuring that flows meet a common deadline [18], [19]. We extend the definition of coflow and the k-th coflow $F^{(k)}$ in the non-blocking data center network can be defined as:

Definition 1. $F^{(k)} = \{f_{i,j}^k | 1 \leq i \leq m, 1 \leq j \leq m\}$, *where* k *is the sequence number assigned to the coflow, and* $f_{i,j}^k$ *is the normalized size of the flow (in this coflow* $F^{(k)}$*) sent from host* i *to host* j*. If there is no flow from host* i *to host* j*, then* $f_{i,j}^k = 0$.

Since in the non-blocking data center, every port has a unit capacity, so that the transfer time of the flow (i.e., the time the flow actually occupies the port) $f_{i,j}^k$ is also $f_{i,j}^k$. In this section, we simply assume all flows arrive simultaneously, and we will remove this assumption when designing the online scheduling algorithm in the next section. We use w_k to denote the weight, i.e., the importance of each coflow $F^{(k)}$, so that its completion time C_k can be optimized with regard to its weight. If only non-preemptive scheduling is taken, the completion time of a coflow is the finishing moment. Main notations used in the paper is shown at Table 3. Then the Idealized Weighted Coflow Completion Time Minimization problem is defined as follows:

$$\text{minimize} \quad \sum_{k=1}^{n} w_k C_k \tag{1}$$

$$\text{s.t.} \qquad \forall k, j : \sum_{\forall l: C_l \leq C_k} \sum_{i=1}^{m} f_{i,j}^{(l)} \leq C_k \tag{2}$$

$$\forall k, i : \sum_{\forall l: C_l \leq C_k} \sum_{j=1}^{m} f_{i,j}^{(l)} \leq C_k. \tag{3}$$

Our goal is to minimize the sum of the weighted coflow completion times. The constraints (2) and (3) are due to the competition on the capacity of each port. For a coflow $F^{(k)}$ with completion time C_k, consider the set of coflows that finish before it, i.e., $F^{(l)}$: $C_l \leq C_k$. For any ingress port i

(or egress port j), the total flow transfer time of this coflow on this port is at least $\sum_{j=1}^{m} f_{i,j}$ (or $\sum_{i=1}^{m} f_{i,j}$, correspondingly), which must be no greater than C_k.

Consider a simplified version of this problem, where we want to minimize the total completion time of flows competing a single link. It is well known that *shortest job first* is the optimal non-preemptive scheduling policy, while *short remaining time first* is the optimal preemptive one. However, our problem for scheduling weighted coflows over multiple links is much harder, as indicated by the following proposition

Proposition 1. *If only non-preemptive scheduling is allowed, the IWCCTM problem is equivalent to the problem of minimizing the sum of weighted completion times in a concurrent open shop [35] and it is NP-hard.*

Proof. The concurrent open shop problem [35] can be described as a set of machines $M = \{1, \dots, m\}$, with each machine capable of processing one operation type. A set of jobs $N = \{1, \dots, n\}$, with each job requiring specific quantities of processing for each of its m operation types. Each job $j \in N$ has a weight $w_j \in R > 0$, and the processing time of job j's operation on machine i is $p_{ij} \in R \geq 0$. A job is completed when all its operations are completed. In particular, operations from the same job can be processed in parallel. For the $m \times m$ network fabric, a coflow $F^{(k)}$ has $2m$ transfer tasks, each transfer task $L_l^{(k)} (0 \leq l < 2\text{m})$ is defined as:

$$L_l^{(k)} = \begin{cases} \sum_{i=1}^{m} f_{i,l}^{(k)} & 0 \leq l < m \\ \sum_{j=1}^{m} f_{l-m,j}^{(k)} & m \leq l < 2m. \end{cases} \tag{4}$$

Eq. (4) describes the lth $(0 \leq l < 2m)$ transfer task of each coflow, where $\sum_{i=1}^{m} f_{i,l}^{(k)} (0 \leq l < m)$ shows the computation of the first m transfer task and $\sum_{j=1}^{m} f_{l-m,j}^{(k)} (m \leq l < 2m)$ shows the second m transfer task. Each of the first m transfer task is flows that are aggregated from the ingress ports, while each of the second m transfer task is flows that are aggregated from the egress ports. Consider the concurrent open shop scheduling problem with $2m$ machines of the same capacity and n jobs, where each job has $2m$ types of operations on the $2m$ machines. There is a one-to-one correspondence between a coflow and a job, and between one type of operation on a machine and one transfer task on a port. It has been proved that minimizing the average weighted job completion time is NP-hard [16], [35], [39], [40], so the IWCCTM problem is also NP-hard. □

4 Scheduling Algorithms

In this section, we first propose for the IWCCTM problem a non-preemptive scheduling algorithm. Then we simplify the algorithm and propose a simple offline algorithm. At last, we remove the assumption that all coflows arrive at the same time and flow sizes are known a priori, and derive an information-agnostic preemptive scheduling algorithm that can work in an online fashion.

4.1 Non-Preemptive Scheduling Algorithm for IWCCTM

For minimizing the average weighted job completion time in the concurrent open shop problem, the best known theoretical result is a 2-approximation greedy algorithm [28], [35]. Since our IWCCTM problem has a close relationship with it, we can derive a similar non-preemptive scheduler for IWCCTM, as shown in Algorithm 1.

TABLE 3
Main Notation Used in This Paper

Name	Description
m	Number of hosts of DC fabric
$F^{(k)}$	The kth coflow
$f_{i,j}^{k}$	Normalized size of flow (in coflow $F^{(k)}$) sent from host i to j
w_k	Weight of coflow k
C_k	Completion time of coflow $F^{(k)}$

Algorithm 1. 2-Approximate Algorithm for IWCCTM

```
Input: Coflow List $\mathcal{F} = \{F^{(k)}\}$, Weight List $\mathcal{W} = \{\overline{w_k} = w_k\}$,
       $1 \leq k \leq n$;
Output: a permutation $\gamma$ of $\{1, \dots, n\}$;
1:  $\mathcal{P} \leftarrow \{1, \dots, 2m\}$, $\mathcal{R} \leftarrow \{1, \dots, n\}$;
2:  $L_i^{(k)} = \sum_{j=1}^{m} f_{i,j}^{(k)}$ for $1 \leq k \leq n$ and $i \leq m$;
3:  $L_{j+m}^{(k)} = \sum_{i=1}^{m} f_{i,j}^{(k)}$ for $1 \leq k \leq n$ and $j \leq m$;
4:  $L_p = \sum_{1 \leq k \leq n} L_p^{(k)}$ for each $p \in \mathcal{P}$;
5:  for $i$ from $n$ to 1 do
6:      $p^* = \arg\max_{p \in \mathcal{P}} L_p$;
7:      $\gamma[i] = r^* = \arg\min_{r \in \mathcal{R}} \overline{w_r} / L_{p^*}^{(r)}$;
8:      $\theta = \overline{w_{r^*}} / L_{p^*}^{(r^*)}$;
9:      $\mathcal{R} = \mathcal{R} \setminus \{r^*\}$;
10:     $\overline{w_r} = \overline{w_r} - \theta \times L_{p^*}^{(r)}$ for all $r \in \mathcal{R}$;
11:     $L_p = L_p - L_p^{(r^*)}$ for all $p \in \mathcal{P}$;
12: return $\gamma$;
```

Algorithm 1 takes a list $\mathcal{F}$ of n coflows and a list $\mathcal{W}$ of n weights as its input, where the k-th coflow in $\mathcal{F}$ is defined as $F^{(k)} = \{f_{i,j}^{k} | 1 \leq i \leq m, 1 \leq j \leq m\}$, whose weight is w_k, the k-th weight in $\mathcal{W}$, as already explained in Section 3. It outputs γ, a permutation of $\{1, \dots, n\}$ that indicates the scheduling order of the n coflows.

The Algorithm first composes a port list $\mathcal{P} = \{1, \dots, 2m\}$, corresponding to m ingress and m egress ports, and computes the total load on each port (line 1-4). Then its iteratively finds the coflow to be scheduled in the i-th round, nonetheless in a reverse order. In each iteration, it finds the port (p^*) with the heaviest load, chooses the coflow who has the minimal ratio of coflow weight to its load on that port, and saves its index (r^*) in $\gamma[i]$ (line 6-7). It then updates the set of coflows remaining to be scheduled as well as their weights, and the port load (line 8-11), before it goes to the next iteration.

Different from Varys, Algorithm 1 is a port-based scheduling methods. Fig. 4 shows comparison of 2-approximate algorithm against Varys and TCP. We can see that both average CCT and average weight CCT for the 2-approximate algorithm is better than Varys.

Algorithm 1 generates the permutation γ as the scheduling order within $O(n(m+n))$ operations. Indeed Algorithm 1 is $(2\text{-}\frac{2}{n+1})$-approximate optimal, where n is the concurrent number of coflows. F denotes id set of coflows to be scheduled and the kth coflow $F^{(k)}$'s subtask denotes as $L_l^{(k)}(0 \le l < 2m)$. $L_l^{(k)}$ has the following useful property, first proved by Schulz [41] for other scheduling problems and used by Mastrolilli [35] to prove the approximation of minimize total weighted completion time of jobs.

Lemma 1. *For any $j \le 2m$ and $S \subset F$, we have $(\sum_{k\in S} L_j^{(k)})^2 \le (2-\frac{2}{n+1})\left[\frac{1}{2}\sum_{k\in S}(L_j^{(k)})^2 + \frac{1}{2}(\sum_{k\in S} L_j^{(k)})^2\right]$.*

Lemma 1 describes the property of $L_l^{(k)}$. With the help of Lemma 1 , we can prove the following theorem.

Theorem 1. *Algorithm 1 is (2-$\frac{2}{n+1}$)-approximate optimal, where n is the concurrent number of coflows.*

Proof. According to Chen [16] and the relationship between IWCCTM and Concurrent Open Shop Problem, IWCCTM has the following constrains:

$$\text{minimize} \sum_{k=1}^{n} w_k C_k \tag{5}$$

$$\text{s.t. } \forall j \le 2m, S \subset F : \sum_{\forall k\in S} L_j^{(k)} C_k \ge \frac{1}{2}\sum_{\forall k\in S}(L_j^{(k)})^2 + \frac{1}{2}\left(\sum_{\forall k\in S} L_j^{(k)}\right)^2. \tag{6}$$

We call this IWCCTM-C. Based on this, we get IWCCTM-C's dual function IWCCTM-D:

$$\text{maximize} \quad \sum_{j=1}^{2m}\sum_{S\subset F}\left[\frac{1}{2}\sum_{\forall k\in S}(L_j^{(k)})^2 + \frac{1}{2}\left(\sum_{\forall k\in S} L_j^{(k)}\right)^2\right] y_j^S \tag{7}$$

$$\text{s.t.} \quad \forall k \le n : \sum_{j=1}^{2m} L_j^{(k)} \sum_{S\subset F} y_j^S = w_k \tag{8}$$

$$\forall y_j^S \ge 0. \tag{9}$$

Let $p(i)$ denote the ports that are selected at Line 6, and $\theta(i)$ denote the computation result of θ at Line 8. $\overline{w_r(i)}$ represents adjust weight of i-the iteration that is computed at Line 10. Let $\gamma(i)$ represent the coflow set that is not scheduled at the start of the i-th cycle and we have $\gamma(i) = \{r^*(1), r^*(2)...r^*(i)\}$, where $r^*(i)$ denote coflow's id. Define the following solution of problem IWCCTM-D:

$$y_j^S = \begin{cases} \theta(i) & \text{if } j = p(i),\ S = \gamma(i), i = 1,2,3...n \\ 0 & \text{other.} \end{cases} \tag{10}$$

Eq. (10) satisfies constrains (8), because for any coflow that is to be scheduled in the ith iteration $r^*(i)$ (i = 1, 2, 3...n), we have:

$$\begin{aligned} \sum_{j=1}^{2m} L_j^{r^*(i)} \sum_{S\subset F} y_j^S &\overset{(a)}{=} \sum_{l=i}^{n} L_{p(l)}^{r^*(i)} y_{r^*(l)}^S \\ &\overset{(b)}{=} \sum_{l=i}^{n} L_{p(l)}^{r^*(i)} \theta(l) \\ &\overset{(c)}{=} w_{r^*(i)} - \overline{w_{r^*(i)}} \\ &\overset{(d)}{=} w_{r^*(i).} \end{aligned} \tag{11}$$

(a) holds since Algorithm 1 loops from n to 1, so the ith iteration can be computed according to the result from i to n. (b) holds because the solution (10). (c) holds since in the ith iteration, weight is adjusted as

$$w_r^{(i)} = w_r - \sum_{l=k}^{n} L_{p(l)}^r \theta(l). \tag{12}$$

(d) holds because at the end of i-th iteration $\overline{w_{r^*(i)}} = 0$

Line 7 chooses coflow with smallest $\overline{w_r}/L_{p^*}^{(r)}$, so (9) holds. As a result, (10) satisfies both constrains (8) and (9), so it is the solution of problem IWCCTM-D. In Algorithm 1, the completion time satisfies $C_{\gamma[1]} \le C_{\gamma[2]} \le ...C_{\gamma[n]}$. According to Line 6 and 7, for i = 1, 2,..n, we have $C_{\gamma[i]} = \sum_{j=1}^{i} L_{p^*(i)}^{r^*(j)}$. Let $(C_k^{CP})_{k\in\mathcal{F}}$ be the optimal solution and $(C_k^*)_{k\in\mathcal{F}}$ be an optimal completion time vector, we have:

$$\begin{aligned} &\sum_{k=1}^{n} w_k C_k \\ &= \sum_{k=1}^{n}\left(\sum_{j=1}^{2m} L_j^{(k)} \sum_{S\subset F} y_j^S\right) C_k \\ &= \sum_{j=1}^{2m}\sum_{S\subset F} y_j^S \sum_{k\in S} L_j^{(k)} C_k \\ &= \sum_{l=1}^{n} y_{p^*(l)}^{\gamma(l)} \sum_{g=1}^{l} L_{p^*(l)}^{g} C_g \\ &= \sum_{l=1}^{n} y_{p^*(l)}^{\gamma(l)} \sum_{h=1}^{l} L_{p^*(l)}^{r^*(h)} C_{r^*(h)} \\ &\overset{(e)}{\le} \sum_{l=1}^{n} y_{p^*(l)}^{\gamma(l)} C_{r^*(l)} \sum_{h=1}^{l} L_{p^*(l)}^{r^*(h)} \\ &\overset{(f)}{=} \sum_{l=1}^{n} y_{p^*(l)}^{\gamma(l)} \left(\sum_{h=1}^{l} L_{p^*(l)}^{r^*(h)}\right)^2 \\ &\overset{(g)}{\le} \left(2-\frac{2}{n+1}\right)\sum_{l=1}^{n} y_{p^*(l)}^{\gamma(l)} \left\{\frac{1}{2}\sum_{h=1}^{l}\left[L_{p^*(l)}^{r^*(h)}\right]^2 + \frac{1}{2}\left[\sum_{h=1}^{l} L_{p^*(l)}^{r^*(h)}\right]^2\right\} \\ &\overset{(h)}{\le} \left(2-\frac{2}{n+1}\right)\sum_{k=1}^{n} w_k C_k^{CP} \\ &\le \left(2-\frac{2}{n+1}\right)\sum_{k=1}^{n} w_k C_k^*. \end{aligned} \tag{13}$$

(e) holds since for h = 1, 2,...l, we have $C_{r^*(h)} \le C_{r^*(l)}$, (f) holds since $C_{r^*(l)} = \sum_{h=1}^{l} L_{p^*(l)}^{r^*(h)}$, (g) holds since lemma 1. (h) holds y is the optimal solution for IWCCTM-D.

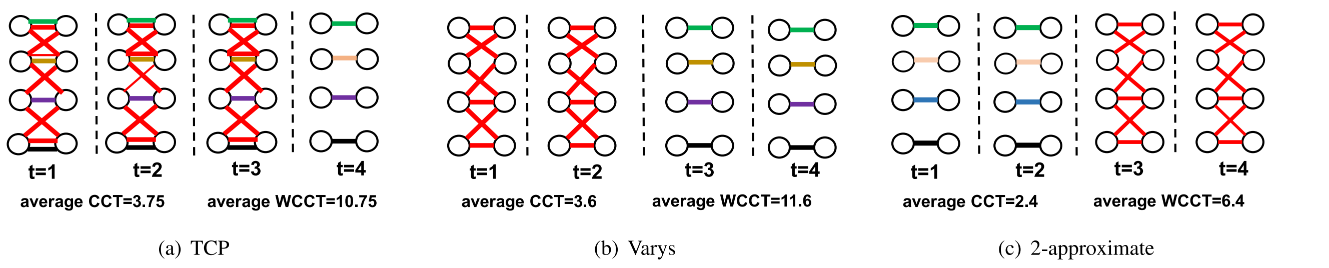


Fig. 4. Comparison of 2-approximate algorithm against Varys and TCP. 5 coflows contend at the 4 ingress and 4 egress ports. C1 contains 9 flows, whose length are all 1. C2, C3, C4, C5 only have 1 flow whose length is 2. Weight of C1, C2, C3, C4, C5 are 1, 2, 3, 4, 5, respectively.

As a result, Algorithm 1 is ($2-\frac{2}{n+1}$)-approximate optimal, where n is the concurrent number of coflows. □

However, since IWCCTM makes some idealized assumptions, the algorithm cannot be straightforwardly used in real environment.

Algorithm 2. Coflow Size Based Offline Algorithm

```
Input: Coflow List $\mathcal{F} = \{F^{(k)}\}$, Weight List $\mathcal{W} = \{w_k\}$, $1 \le k \le n$;
Output: a permutation $\gamma$ of $\{1, \ldots, n\}$;
1: $\mathcal{R} \leftarrow \{1, \ldots, n\}$;
2: for $r \in \mathcal{R}$ do
3:     $L^r = \max(\max_i \sum_{j=1}^m f_{i,j}^{(k)}, \max_j \sum_{i=1}^m f_{i,j}^{(k)})$;
4: for $i$ from $n$ to 1 do
5:     $\gamma[i] = r^* = \arg\min_{r \in \mathcal{R}} w_r / L^r$;
6:     $\mathcal{R} = \mathcal{R} \setminus \{r^*\}$;
7: return $\gamma$;
```

Algorithm 3. Information-Agnostic Online Algorithm (IAOA)

```
Input: Coflow list $\mathcal{F} = \{F^{(k)}\}$, weight list $\mathcal{W} = \{w_k\}$, $1 \le k \le n$;
Output: Coflow list $\omega$;
1: if a new coflow arrives or a coflow ends or duration since last call this algorithm $> T$ then
2:     update the active coflow list $\mathcal{F} = \{F^{(k)}\}$;
3:     $n = |\mathcal{F}|$;
4:     for $1 \le k \le n$ do
5:         update $s_{i,j}^{(k)}$, the cumulative volume of the traffic sent from i to j, for $1 \le i, j \le m$;
6:         $L_i^{(k)} = \sum_{j=1}^m s_{i,j}^{(k)}$ for $1 \le i \le m$;
7:         $L_{j+m}^{(k)} = \sum_{i=1}^m s_{i,j}^{(k)}$ for $1 \le j \le m$;
8:         $l^{(k)} = \max_{1 \le i \le 2m} L_i^{(k)}$;
9:         $\alpha^{(k)} = l^{(k)} / w_k$;
10:    $\omega$ = sort the list of $\{\alpha^{(k)}\}$ in nondecreasing order;
11:    return $\omega$;
```

4.2 Information-Agnostic Algorithm for Minimizing WCCT

The key idea of Algorithm 1 lies in line 7, where it tries to prioritize coflows with large weight but small load on the most loaded port p^*. We regard Algorithm 1 as *port-based* method, since it ranks coflows according to the port traffic load. However, in practice, flow size is rarely known by the scheduler before it finishes, and flows arrive at any time, which makes the port-based algorithm unrealistic. On the other hand, although the load on each port varies with time, on average, ports will have comparable loads since jobs are effectively load balanced across a data center (with technologies such as [20], [22], [23], [26], [34]). With this in mind, the schedule process does not need to take the load diversity of ports into account. We make a simplification that ignores the port load difference, and just prioritize coflows with large weight but small size. By replacing the prioritizing rule in Algorithm 1 with this heuristic, we get the *coflow size based* offline scheduling Algorithm 2.

In the coflow size based Algorithm 2, we first compute the load of each coflow (line 3). Then every literation, we choose the coflow which has the minimal ratio of coflow weight to its load (line 4-6). Algorithm 2 can't be used directly, since it should know flow size beforehand. In practice, volume can be measured cumulatively in realtime, so we replace flow length with the volume of traffic it has sent. This is particularly useful for scenarios where dynamic-sized flows are generated, for example, in hadoop or some other map-reduce style applications. Algorithm 3 shows the process of IAOA.

Algorithm 3 is called by a scheduler when a flow starts or finishes or duration since last calls the algorithm is larger than the time threshold T. When a coflow starts or ends, the active coflow list is updated (line 1-2).[2] For each active coflow in the active coflow list, we update $s_{i,j}^{(k)}$ which denotes the cumulative volume of the traffic sent from i to j and compute the cumulative volume of traffic of port i, j (line 6-7). We use the cumulative volume of traffic to estimate the length of coflow (line 8) and compute the priority of coflow (line 9). Then we update the order of coflow (line 10) of the active coflow list. For each active coflow, we allocate bandwidth to flows according to the coflow priority. Algorithm 3 is an information-agnostic online algorithm that dynamically schedules coflows according to their weights and the instantaneous network condition. Compared to the 2-approximate algorithm, it inevitably introduces some performance loss, but our evaluation shows there is only a small gap, as will be seen in the next part. There are also some other information-agnostic algorithms such as PIAS [14], L^2 DCT [37], FDRC [50], Aalo [18]. PIAS and Aalo implement Multiple Level Feedback Queue (MLFQ) to achieve information-agnostic scheduling, in which a flow gradually demotes from higher-priority queues to lower-priority queues based on the bytes

2. A coflow set $\mathcal{F}$ and the corresponding schedule order are maintained in the whole life of the scheduler. The prioritization procedure is bypassed, and the schedule order remains unchanged if no new coflow arrives.

TABLE 4
Coflow Types Divided by Length and Width

Coflow Type	S-N	L-N	S-W	L-W
Length	short	long	short	long
Width	narrow	narrow	width	width
% of Coflows	52%	16%	12%	20%
% of Bytes	0.01%	0.65%	0.33%	99.01%

it has sent. They emulate Shortest Job First with the help of router and TOS (Type of Service) of IP packets. This method can provide clear priority level; however, there are only 8 priorities because of the 3-bit width of TOS. FDRC accumulates flow duration time to estimate flow length and it is also not a good choice, since flow duration time in data center is always less than 100 ms [12], which is not easy to measure accurately. Algorithm 3 is similar to L^2 DCT, which uses the idea of using the number of bytes a flow has already sent as an estimator for its pending data. Different from L^2 DCT, Algorithm 3 is a task-level method, which employs the longest flow to compute the priority. This approach can provide abundant and flexible priority choices and doesn't need to configure router.

4.3 How Far is IAOA from the Optimal

Two relaxations are adopted in the information-agnostic online algorithm (IAOA). Simple-offline algorithm assumes data center generally assigns jobs with load balancing and ignores load difference of each port. The online algorithm further assumes that larger coflows that last long time and have low priority. The two relaxations can absolutely lose the performance. In this part, we analyze the performance gap from both theory and simulations.

We use a two coflow model to show the performance loss of IAOA. Assume there are two coflows $F1$ and $F2$, and each coflow has $m \times m$ flows, whose length are all L_1 and L_2. We further assume all the flows belong to the same coflow start simultaneously. If $L_1 < L_2$ and coflow $F1, F2$ start at 0. Let W_1 and W_2 denote the weight of coflow $F1$ and $F2$, and $W_1 > W_2$.

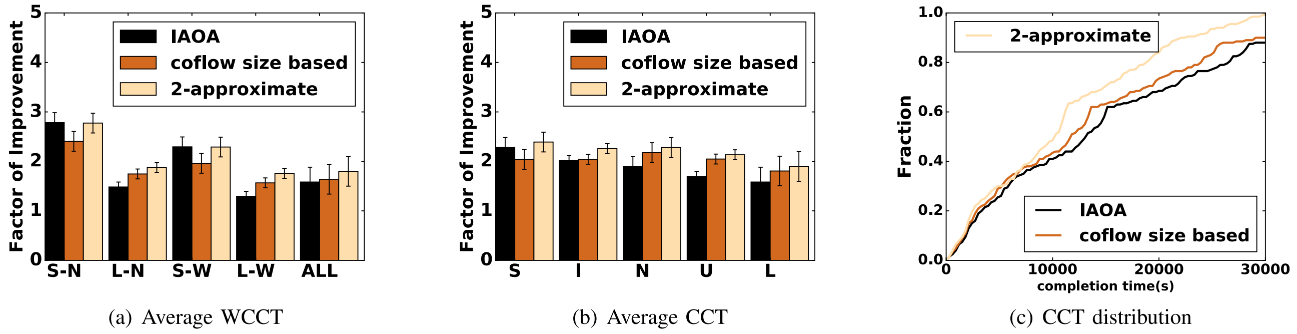
Lemma 2. *$C(OPT)$ and $C(IAOA)$ denotes the weighted coflow completion time of the optimal solution and the worst case of IAOA. For the non-blocking DC fabric, when $T < L_1 < 2T$, we have $C(IAOA) - C(OPT) < W_1 \times T$*

Proof. As $F1$ and $F2$ arrive simultaneously, and $W_1 > W_2, L_1 < L_2$, so that optimal WCCT is $C(OPT) = W_1L_1 + W_2(L_1 + L_2)$. For IAOA, it has two cases, $F1$ starts earlier or $F2$. If $F1$ starts earlier than $F2$, then after T, $F2$ has higher priority than $F1$. As IAOA schedules coflows every T and it priories short and important coflows, so that in the following T time, $F1$ has higher priority again. Indeed, they alternate higher priorities. If $F1$ starts earlier, $C(IAOA) = W_1(L_1 + T) + W_2(L_1 + L_2)$. If $F2$ starts earlier, $C(IAOA)$ is also $W_1(L_1 + T) + W_2(L_1 + L_2)$. The worst case for IAIA is $C(IAOA) = W_1(L_1 + T) + W_2(L_1 + L_2)$. The gap between IAOA and the optimal solution is $W_1 \times T$. □

We can see the performance gap has positive correlation with T. In reality, if the network operator sets a small value of T, the performance of IAOA could be close to the optimal solution. We now explore the performance gap between IAOA and the 2-approximate offline algorithm. We consider 100 coflows served by a small-scale cluster consisting of 60 hosts. We write a program [3] to generate coflows as Narrow&Short (N-S), Narrow&Long (N-L), Wide&Short (W-S), and Wide&Long (W-L), where a coflow is considered to be short if its length is less than 100 MB, and narrow if it involves at most 20 flows. Type percentage and flow bytes are shown at Table 4. In our simulation, flow length and width subject to exponential distribution. Let all the coflows arrive at t = 0 and set weight within 1, 2, 3, 4, 5 with uniform distribution. Re-generate all the coflows 100 times and Fig. 5 shows the result. In our experiment, error bar paints the mean and mean ± standard deviation value. We use the improvement in average CCT and WCCT as our primary metric and the improvement factor is defined as:

$$Factor\ of\ improvement = \frac{results\ with\ TCP}{results\ with\ current\ method}.$$

Fig. 5a shows that factor of improvement for WCCT of the online method is 1.6. While that for simple-offline method is 1.7 and for the 2-approximate method is 1.8. Comparing with the 2-approximate algorithm, the online algorithm has about 10 percent performance loss on minimizing average WCCT. Note, the gap between the 2-approximate and the online algorithm is small for N-S and W-S coflows, this is because the online algorithm prefers the short coflows. Fig. 5b shows the performance for different important level of coflows. We can see that online method factor of improvement for significant, important coflows are 2.2 and 2.0 on average. The 2-approximate methods are about 2.3, 2.2, which indicates that the



(a) Average WCCT (b) Average CCT (c) CCT distribution

Fig. 5. [Simulation] Comparison between the online and offline algorithms, TCP is selected as the baseline. Note in the picture, L, U, N, I, S are abbreviations of Lax, Unimportant, Normal, Important, Significant.

TABLE 5
Comparison between the Three Algorithms

#Scheme	Mode	Procedure	Performance	information
2-approximate	offline	Complex	High	clairvoyant
Simple-offline	offline	simple	High	clairvoyant
IAOA	online	simple	High	non-clairvoyant

online algorithm has less than 10 percent performance loss for the above-the-average coflows. While for the below-the average coflows, IAOA has about 30 percent performance loss. This indicates that IAOA has better performance for the important coflows. Fig. 5c shows the distribution for CCT. We can see more than 80 percent CCT for 2-approximate method is within 20000s, while that for simple-offline and online methods are 70 and 60 percent. Table 5 shows the comparison between the methods. Although the online method has some performance loss, it is non-clairvoyant method. As flow size for some applications can not be known beforehand, so that non-clairvoyant method could be more widely used in practice.

5 FlyTransfer System

To make our scheduling algorithm practical, we design and implement a scheduling system named FlyTransfer. FlyTransfer can be readily deployed in productional cloud environment like openstack and employs the actor model [1]. It uses Akka [2] for message transferring and kryo [7] for object serialization. It consists of about 6000 lines of scala code, and can be downloaded at [5].

5.1 Coflow Scheduling and Flow Rate Control

FlyTransfer adopts centralized method to schedule coflows. We use centralized methods, because comparing with decentralized scheduling method, the centralized method can decide coflow priority based on the full knowledge of networks and coflows. To achieve, we implement a system called FlyTransfer. Fig. 6 shows details of FlyTransfer System. FlyTransfer consists of three types of nodes, i.e., master, worker and monitor. The master node is the brain of the system, which plays the role of controller. It collects coflow information from workers, performs coflow prioritization and bandwidth computation. The work nodes run two daemons: interact daemon and rate control daemon. The rate control daemon is used to throttle flow to the value that is

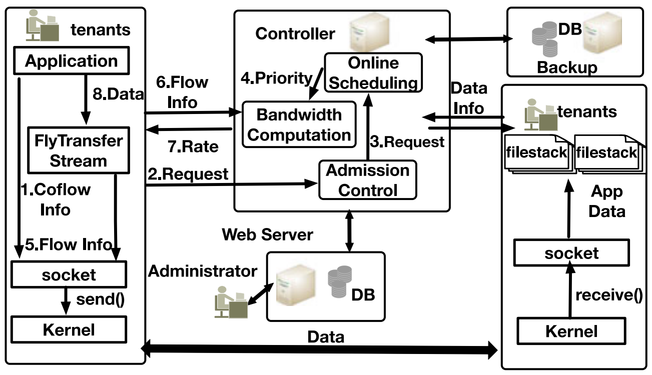


Fig. 6. FlyTransfer system.

TABLE 6
Coflow API

Methods	Paramters	Description
registerCoflow	numflows, weight	Register a coflow
unregisterCoflow	coflowId	Remove a coflow
flowPut	coflowId	Start a flow
flowGet	flowId	Get a flow information
coflowGet	coflowId	Get a coflow's information
schedulerChoose	ScheduerId	Choose a scheduler

computed by the controller. The interact daemon communicates with the controller with a set of APIs shown at Table 6.

First, applications should register coflow information to controller. The register information includes number of flows in the coflow and weight of the coflow. If the coflow is admitted by the controller, the scheduling module belonging to the controller computes the priority of this coflow. Then the controller produces an identifier that presents the coflow and notifies the id as well as the priority to all the nodes.

Second, after admitting the flow, controller computes the bandwidth of the flow and sends this back to the sender. Instead of evaluating rate limit at kernel level, FlyTransfer extends the OutputStream [9] at the user state to throttle flow rates. We choose this mode for two reasons. On the one hand, the user state evaluation mode is easy and has similar effect. On the other hand, the user state evaluation mode can control sending rate of both TCP and UDP. However, the kernel state evaluation mode such as changing congestion window only applies to TCP. Then data is sent through socket.

Note, when scheduling each coflow, tenant should pass the coflowId to the driver to tell the controller that flow has started. After the controller admits the start signal, it decreases the counter of flows. During the sending process, tenants can also get the state of flow.

The Backup node in FlyTransfer System backups the coflow scheduling state of the master node in real time, so that if the master node crashes, the monitor can restart the master and restore its state. The Administrator monitors the system through the web server.

5.2 About Scalability

The system introduced above is a coflow scheduling framework and the controller can choose scheduler with different scheduling algorithms. Algorithm 1 has the best performance in theory, however, it needs to know each flow size and assume all coflows start simultaneously. However, these flow information are hard to be known beforehand for some applications (hadoop, spark, etc). Algorithm 2 is also not a good choice since coflows may arrive at any time. FlyTransfer evaluates Algorithm 3, because it can keep high performance and doesn't need to solve the optimization problem.

After deciding the priority of coflows, we should decide the rate of flows. Algorithm 4 shows the process of flow rate control. Bandwidth is allocated to flows according to the priority of coflows (line 3). Then it uses greedy method to allocate bandwidth to flows as much as possible in each ports (line 4-8). If there is spare bandwidth, it allocates remaining bandwidth equally to all flows. Recall that, Varys need the length of each flow and it schedules coflow every σ time slot. Different from Varys, IAOA doesn't need flow length and it

TABLE 7
Comparison of Scalability

	Required knowledge	Schedule comparison	Schedule frequency	Reliability
FlyTransfer	No flow length	Importance & Network	On coflow Arrivals	Backup node guarantees
Varys	Flow length is needed	Network	Every σ timeslot	Single point of failure

schedules coflow when a new coflow arrives, so that the timeliness of scheduling can be guaranteed. Besides, IAOA schedules coflow according to network and coflow importance and the back up node can guarantee the reliability of the system. Varys suffers from the problem of single point of failure. Table 7 shows the comparison of scalability.

Algorithm 4. Greedy Flow Rate Control

```
    Input: Coflow list $\omega$;
    Output: flow rate of each active coflow
 1: $\gamma =$ index set of $\omega$;
 2: initialize the bandwidth to be allocated on each ingress and
    egress port: $I_p = O_p =$ port physical bandwidth, for
    $1 \leq p \leq m$;
 3: for $\ell$ from 1 to $n$ do
 4:   schedule coflow $F = F^{(\gamma[\ell])}$;
 5:   $r$ = min $\{I_p, O_q\}$, subject to $F$ still need to send traffic to
      port $p$ or receive traffic from port $q$;
 6:   for each unfinished flow $F_{i,j}$ in $F$ do
 7:     allocate bandwith of $r$ to $F_{i,j}$;
 8:     update $I_i$ and $O_j$ by deducting $r$ from them;
 9: allocate remaining bandwidth equally to all flows;
10: return flow rate of each active coflow
```

6 Evaluation

In this section, we thoroughly evaluate the performance of IAOA and FlyTransfer. In the evaluation part, default value of T is 50 ms. We run trace-based simulations using real traffic trace large scale and medium-sized data centers. We also conduct experiments on a small scale openstack [8] testbed, where there are 80 virtual machines (with 2 Cores and 4 GB Memory) run simultaneously. As the facebook trace [19] doesn't contain weight information, we set weight to coflows within $\{1, 2, 3, 4, 5\}$ under uniform distribution by default. The default scheduling algorithm is IAOA. The main results of our evaluation are summarized as follows:

- For the trace of facebook [19] with random weight within 1, 2, 3, 4, 5 under uniform distribution, IAOA improves about 20, 10, 50 percent, 3 $\times$ better than Sincronia, Varys, Approximate method, Barrat on reducing the average WCCT. For coflows with above-the-average level of importance, IAOA performs about 20 percent better than Varys.
- For the trace of our data center, compared to Varys, the best performing one among existing coflow scheduling algorithms, IAOA reduces about 30 percent of the average WCCT, and reduces about 25-30 percent of the WCCT of the most important coflows.
- The average computation time required by FlyTransfer for each round of scheduling is about about 20 percent smaller than Varys .
- The simulation results match well with the openstack testbed results and this verifies the performance of IAOA from another view.

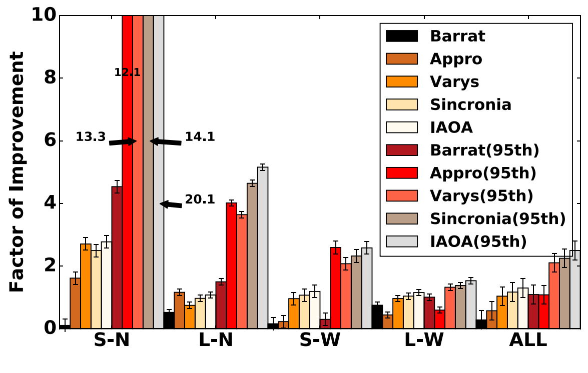
6.1 Trace-Driven Simulation Methodology

The simulations are based on two traffic traces. The first one is collected from a data center of Facebook with a cluster of 3000 machines in 150 racks [19]. The second one is collected from a medium sized data center where around 100 applications run simultaneously among 720 servers deployed in 60 racks. The traffic is classified into 5 categories according to the importance level of different applications, i.e., significant, important, normal, unimportant, and lax, with default weight of 5, 4, 3, 2, and 1, respectively. Since the trace of facebook does not reveal the types of applications, we randomly assign weights to applications with uniform distribution. What's more, we also develop a coflow generator [4] to generate coflows according to coflow length, coflow width and coflow arrival time distribution.

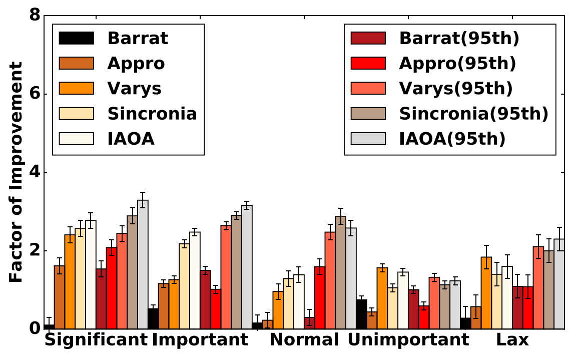
We use two metrics to evaluate the effectiveness of different scheduling methods. The first metric is the Coflow Completion Time, while the second is the Weighted Coflow Completion Time. We consider both the average CCT or WCCT of all coflows, as well as the CCT or WCCT of specific categories of coflows. We compare IAOA against three typical coflow scheduling algorithm Varys [19], Barrat [22], Approximate method [42], Sincronia [11]. Varys is an clairvoyant method, so it has the best performance among them. Barrat works in a distributed fashion, while the Approximate method [42] and Sincronia [11] try to solve a convex optimization problem. There are other coflow scheduling algorithms like sunflows [25] and CODA [48]. However, sunflows targets a special communication pattern, and CODA focuses on a learning method, so in general they perform no better than Varys. To ease the comparison, we often use TCP as the baseline, and examine the ratio of improvement, i.e., $\frac{CCT\ by\ TCP}{CCT\ with\ current\ method}$, and $\frac{WCCT\ by\ TCP}{WCCT\ with\ current\ method}$. For a thorough comparison, coflows are further classified according to their length (i.e., the volume of the largest flow in a coflow) and width (i.e., the number of parallel flows in a coflow). The types include Narrow&Short (N-S), Narrow&Long (N-L), Wide&Short (W-S), and Wide&Long (W-L), where a coflow is considered to be short if its length is less than 100 MB, and narrow if it involves at most 20 flows. The source code of all our simulations can be download at [6], so that interested readers can repeat them.

6.2 Simulation by Traces with Facebook Traffic

In this section, we use trace of facebook [19] to test the performance of IAOA. As the original facebook trace doesn't include weight information, we set random weight within 1, 2, 3, 4, 5 with uniform distribution to present the importance



(a) Average WCCT



(b) Average CCT

Fig. 7. [Simulation] Average WCCT and average CCT comparison for facebook trace, TCP is selected as the baseline.

of coflows. Repeat the process 100 times and Fig. 7 shows the performance of IAOA, Varys [19], Barrat [22], Approximate method [42], Sincronia [11] and TCP. Note TCP is selected as the baseline in this group and the Y-axis denote $\frac{CCT\ by\ TCP}{CCT\ with\ \ current\ method}$ or $\frac{WCCT\ by\ TCP}{WCCT\ with\ \ current\ method}$. Repeat the weight setting process 100 times, where the error bar paints the mean and mean $\pm$ standard deviation value. To eliminate the influence of extreme value, we remove the top and last 2.5 percent result of each methods and recompute the factor of improvement, so the 95th cases are also shown in the picture.

Fig. 7a implies that IAOA greatly reduces the average WCCT across all coflow types. For the 95th of all flows, IAOA performs about 20, 10, 50 percent, 3× better than Sincronia, Varys, Approximate method, Barrat. Specially, for the 95th Short-Narrow (S-N) case, IAOA performs about 43, 54, 66 percent, 3.5× better than Sincronia, Varys, Approximate method, Barrat. IAOA prefers S-N case, because it priors short coflows. Fig. 7b shows the average CCT of coflows with different importance. We can see IAOA performs about 20 percent better than Varys for the coflows with Significant and Important level, but for the less important ones, Varys perform about 10 percent better than IAOA. The reason for this is that Varys schedules coflows only accords to the network condition, while IAOA considers both network condition and importance of coflows. Under IAOA, important coflows may have higher priority, thus the average coflow completion time will reduce.

We can see that Varys and Sincronia performs better than Barrat and the approximate method on minimizing average CCT and average WCCT. In the following experiment, we mainly use Varys and Sincronia as the reference method to show the performance of IAOA.

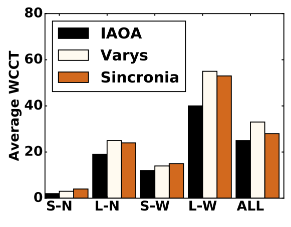
6.3 Simulation by Traces with Medium-Sized Datacenter Traffic

In this section, we solve the problem that is proposed at Section 2. Applications have 5 level of importances in our data center and we use 5, 4, 3, 2, 1 to present the Significant, Important, Normal, Unimportant and Lax level of importance. Run the trace of the medium-sized datacenter and Fig. 8 shows the result.

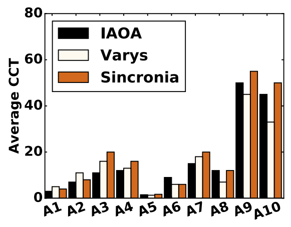
Fig. 8a shows average WCCT for all coflows. IAOA performs about 20 percent better than Varys on minimizing average WCCT. From Fig. 8b, we can see that for vRouter and event which own significant level importance, Varys performs almost the same as Sincronia, while IAOA performs about 20 percent better than Sincronia and Varys. For Hadoop and Druid (Important), IAOA performs about 30 percent better than Varys. While other unimportant applications Varys perform about 10-20 percent better than IAOA. This is because IAOA takes the importance of coflows into consideration. Fig. 8c shows Hadoop average WCCT and Fig. 8d shows Hadoop CCT. For index sort and db analysis which are important in our data center, Varys performs worse than Sincronia. However, using IAOA, it performs about 30-40 percent better. But for the unimportant and lax coflows, Varys performs best. For average WCCT, IAOA performs about 20 percent better than Varys and Sincronia.

6.4 The Effect of Weight Setting

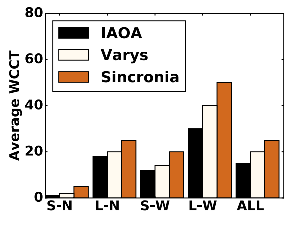
Weight plays an important role in IAOA scheduling algorithm. In the previous experiments, we set weight within $\{1, 2, 3, 4, 5\}$ with uniform distribution to present the importance of coflows. We use 5 importance in the previous experiments and the weight difference is 1. In this part, we



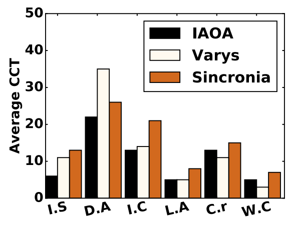
(a) Average WCCT



(b) Average CCT



(c) Hadoop WCCT



(d) Hadoop CCT

Fig. 8. [Simulation] Average CCT (Normalized) and Average WCCT for all applications and Hadoop. A1-A10 denote applications (Event - data-dist) in Table 1 and I.S (index-sort), D.A (db-analysis), I.C (index-count), L.A (log-analysis), C.r (crawler), W.C (word count) are abbreviations of hadoop applications.

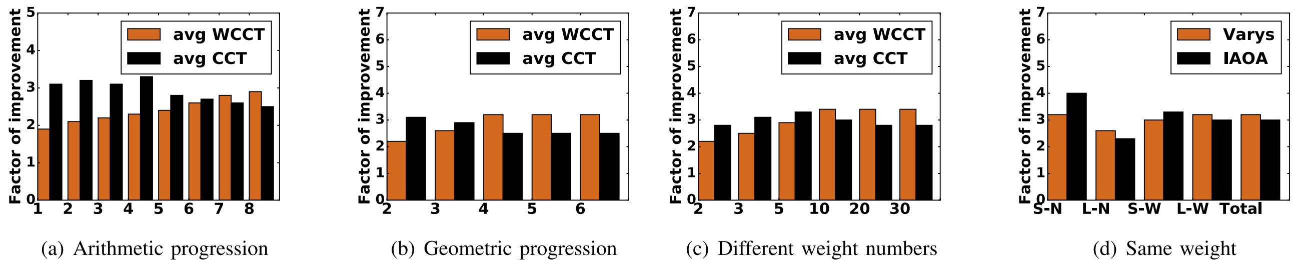


Fig. 9. [Simulation] Performance under different magnitude of weights.

use the trace of our data center to test the performance of IAOA under different magnitude of weights.

Fig. 9a shows the performance of IAOA, when weights are arithmetic progression and the common difference for the 5 importance level ranges from 1 to 8 (for example, if the common difference is 2, the weight set is $\{1, 3, 5, 7, 9\}$). We can see that factor of improvement for average WCCT increases and average CCT decreases when the common difference becomes larger. This is because when weight difference becomes larger, the influence of network is smaller and weight becomes the dominant.

Fig. 9b shows the performance of IAOA, when weights are set with geometric progression and the common ratio for the 5 importance level ranges from 2 to 6 (for example, if the common ratio is 2, weight set is $\{1, 2, 4, 8, 16\}$). We can see that factor of improvement for average WCCT increases and then stays almost the same when common difference becomes larger. This is because when weight difference is large enough, influence of weight becomes the key component and network condition is almost useless.

In practice, the number of importance levels can be decided by the network administrator. In most of our experiments, we use 5 level of importance. It is necessary to explore the performance of IAOA with different number of weights. Fig. 9c shows the comparison of different number of weights, when weights are arithmetic progression (common difference is 1 and coflows choose weight with uniform distribution). We can see factor of improvement for average WCCT increases and then stays almost the same when there are more weights choices. This motivates us that we can set more accurate important level for coflows when using IAOA in practice. We think 5 level of importance is good enough for both average CCT and average WCCT optimization, so that we use the 5 level of importance settings in the following experiments.

Fig. 9d shows the factor of improvement over average CCT of IAOA and Varys when all weights are the same. On the whole, Varys performs about 5 percent better than IAOA. However, IAOA performs better than Varys for short coflows. This is because IAOA prefers short coflows and Varys tries to minimize average coflow completion time according to network condition.

6.5 Other Comparison

It is hard to find the optimal schedule, so that we find an LP-based solution [38] whose approximation ratio is $\frac{67}{3}$ to compare with. The LP-based solution can not be used directly, because it has to solve a LP problem, which is time-consuming. Besides, it should know all the information. Run the media size data center traffic and Fig. 10 shows the result. We can see that the gap between IAOA and the LP-based solution is about 10 percent on average. However, for the short coflows, IAOA performs about 20 percent better. From the measurement of our media data center and facebook, we can see that most coflows are the short ones, so that IAOA is a good choice with only small performance loss.

After deciding the priority of coflows, Algorithm 4 uses the greedy algorithm to allocate rate to flows. There are also other rate control methods such as Minimum-Allocation-for-Desired-Duration and TCP. Use the media data center trace and set weight with arithmetic progression and the common difference for the 5 importance ranges from 1 to 8 (for example, if the common difference is 2, weight set is 1, 3, 5, 7, 9), then Fig. 11 shows the performance comparison between greedy, MADD and TCP. We can see the greedy algorithm performs about 10 and 20 percent better than MADD and TCP, with the same coflow priority settings.

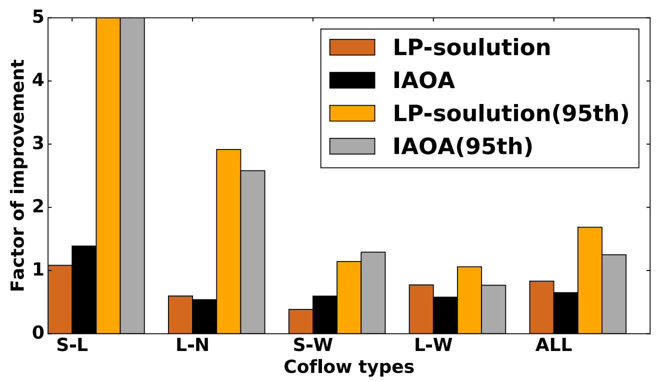


Fig. 10. [Simulation] Performance comparison between the LP-based solution and IAOA.

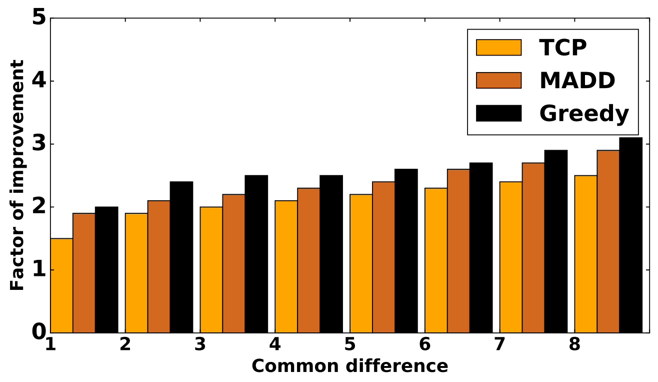


Fig. 11. [Simulation] Performance comparison between different rate control methods.

TABLE 8
Application Information

Coflow Type	mean width	mean length (MB)
map-reduce	10	300
file-copy	3	800
file-distribute	6	400
data-backup	6	380
data-distribute	4	120

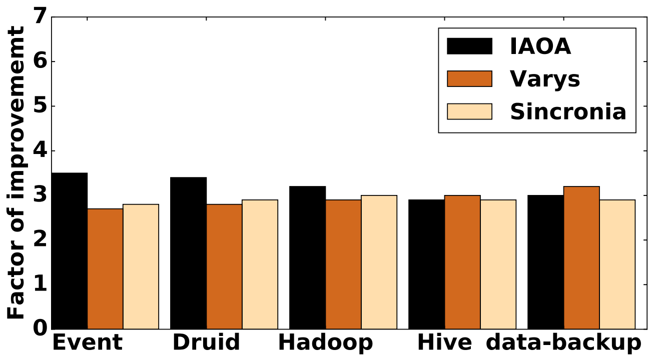


Fig. 12. [Testbed] Performance under 5 common applications in openstack.

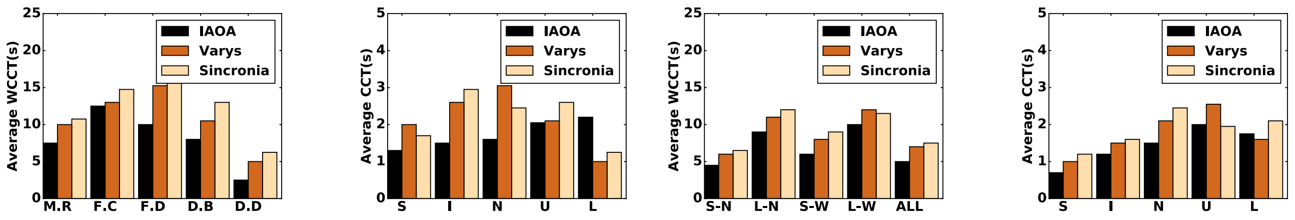
6.6 Real Testbed Evaluation

We deploy FlyTransfer at our private openstack platform which can start at most 80 virtual machines (2 Cores, 4 GB Mem) simultaneously. Operation system of each virtual machine is Ubuntu16.04. With the help of Traffic Control module [10], we constrain the maximum bandwidth of each VM's NIC to 1 GB/s. Default coflow importance is set within 1, 2, 3, 4, 5 with uniform distribution.

We can't deploy all the applications of the medium size data center, so we choose 5 common ones in our data center: Event, Druid, Hadoop, Hive and data-backup. In order to be as close as possible to the actual scenario, length and width of coflows are generated with exponential distribution, whose mean length and width are set according to Table 1. To ease the comparison, TCP is set as the baseline and Fig. 12 shows the results. We can see for Event (Significant) and Druid (Important), IAOA performs about 30 percent better than Varys and Sincronia. For Hadoop (Normal), Hive (Unimportant) and data-backup (Lax), IAOA's performs close to Varys. The results match the simulation.

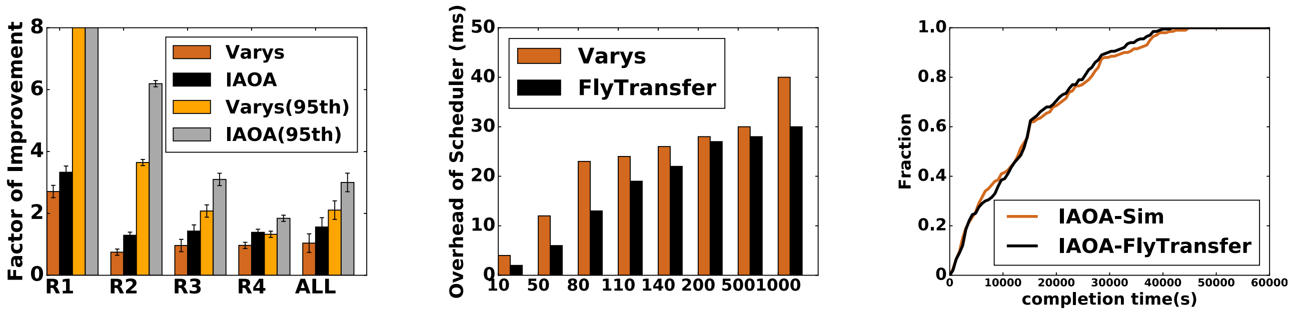
We then deploy another 5 applications: map-reduce, file-copy, file-distribute, data-backup, data-distribute in our platform. Length and width of coflows are generated with exponential distribution according to Table 8. Coflow importance is set within $\{1, 2, 3, 4, 5\}$ with uniform distribution and coflow starts randomly between 0 s and 20 s. We repeat the process 100 times and Fig. 13a shows the average WCCT of applications. We can see IAOA performs about 30 percent better than Varys and Sincronia. Fig. 13b shows the average CCT with different level of importance. We can see that for the significant, important, normal and unimportant applications, IAOA performs better than Varys and Sincronia. However, for the lax applications, IAOA performs worse than Varys and Sincronia. Fig. 13c shows the average WCCT of data distribution and Fig. 13d shows the average CCT. The result is similar to the case of all applications.

Then for the file distribution application, we let each virtual node constantly construct files whose size ranging from 1 KB to 200 MB. Destinations are nodes which are randomly chosen from the total nodes set. TCP-fair is chosen as the baseline and Fig. 14 shows performance comparison between IAOA and Varys. We repeat the process 10 times and the error bar indicates the max, min and average factor of improvement. From Fig. 14a, we can see in total,



(a) Average WCCT (b) Average CCT (c) Data-Distribute WCCT (d) Data-Distribute CCT

Fig. 13. [Testbed] Performance under different applications in openstack and M.R (Map-Reduce), F.C (File-Copy), F.D (File-Distribute), D.B (Data-Backup), D.D (Data-Distribute) are abbreviations. L, U, N, I, S denote Lax, Unimportant, Normal, Important, Significant



(a) Average WCCT with different file size (b) Overhead with different active coflows (c) Testbed and Simulator comparison

Fig. 14. [Testbed] Some details of FlyTransfer. R1([0.01,1]), R2([1,10]),R3([10,100]) and R4([100,200]) are size range (MB) improvement factor of IAOA is 3.1, while Varys is 2.1. IAOA improves 30 percent better than Varys. The 95th result shows IAOA performs 31 percent better than Varys.

In reality, scheduler should be fast enough to compute out the scheduler order and the rate of coflows. Fig. 14b shows the overhead comparison between FlyTransfer and Varys scheduler. We can see for peak load, when active coflows is 1000, computation time of the scheduler is about 29 ms. We also test the scheduler time for Varys, we can see that overhead of Varys is about 20 percent larger than FlyTransfer. We think the average computation time is fast enough to get the schedule result.

Fig. 14c shows the result comparison for the simulator and real testbed. In this experiment, we use the same parameters just as the file distribution. We can see that the total performance gap between the two methods are about 20 percent. This result shows the credibility of our simulation.

7 Conclusion

In this paper, we use weight to quantify the emergence of applications. We design IAOA the online coflow scheduling algorithm. We design and implement FlyTransfer, a coflow scheduling system that aims to minimize the average weighted coflow completion time. We test its performance by trace-driven simulations and experiments in real testbed. Evaluation results show that, FlyTransfer can effectively reduce the average WCCT. IAOA is non-clairvoyant (or information agnostic), in the sense that it does not need to know flow size beforehand. This makes IAOA usable in practice. In our experiments, we use 5 levels of importance, however, in practice, a thiner granularity of importance may be needed. In the future, we plan to develop formal theories to investigate how weights can be set most efficiently.

Acknowledgments

The work is partly supported by Natural Science Foundation of China through projects 61702315 and by the National Key R&D Program of China through project 2017YFB0802500, by the National Cryptography Development Fund through project MMJJ20170106, by the foundation of Science and Technology on Information Assurance Laboratory through project 61421120305162112006, the Natural Science Foundation of China through projects 61772538, 61672083, 61532021, 61472429 and 61402029.

References

[1] Actor model. [Online]. Available: https://en.wikipedia.org/wiki/Actor_model/

[2] Akka. [Online]. Available: http://akka.io/

[3] The coflow generator. [Online]. Available: https://github.com/zhanghan1990/IAOA/blob/master/weightcoflowgenerator.py

[4] coflow generator. [Online]. Available: https://github.com/zhanghan1990/IAOA/blob/master/weightcoflowgenerator.py/

[5] Flytransfer. [Online]. Available: https://github.com/zhanghan1990/FlyTransfer

[6] Iaoa. [Online]. Available: https://github.com/zhanghan1990/IAOA

[7] kryo. [Online]. Available: https://code.google.com/p/kryo/

[8] openstack. [Online]. Available: https://www.openstack.org/

[9] Outputstream. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/org/omg/CORBA/portable/OutputStream.html/

[10] Traffic control. [Online]. Available: http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html/

[11] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 16–29.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 63–74.

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

[14] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. Usenix Conf. Networked Syst. Des. Implementation*, 2015, pp. 455–468.

[15] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[16] Z.-L. Chen and N. G. Hall, "Supply chain scheduling: Assembly systems," Dept. Syst. Eng., Univ. Pennsylvania, Philadelphia, PA, USA, Tech. Rep., Working Paper, 2000.

[17] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Top. Netw.*, 2012, pp. 31–36.

[18] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, pp. 393–406, 2015.

[19] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 443–454, 2014.

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

[21] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," *IEEE/ACM Trans. Netw.*, vol. 14, no. SI, pp. 2809–2816, June 2006.

[22] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 431–442, 2014.

[23] 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, pp. 51–62, 2009.

[24] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 127–138, 2012.

[25] X. S. Huang, X. S. Sun, and T. E. Ng, "Sunflow: Efficient optical circuit scheduling for coflows," in *Proc. 12th Int. Conf. Emerging Netw. EXperiments Technol.*, 2016, pp. 297–311.

[26] A. Iselt, A. Kirstadter, A. Pardigon, and T. Schwabe, "Resilient routing using mpls and ecmp," in *Proc. Workshop High PERFORMANCE Switching Routing*, 2004, pp. 345–349.

[27] H. Jahanjou, E. Kantor, and R. Rajaraman, "Asymptotically optimal approximation algorithms for coflow scheduling," in *Proc. 29th ACM Symp. Parallelism Algorithms Archit.*, 2017, pp. 45–54.

[28] A. Kumar, R. Manokaran, M. Tulsiani, and N. K. Vishnoi, "On lp-based approximability for strict csps," in *Proc. 22nd Annu. ACM-SIAM Symp. Discrete Algorithms*, 2011, pp. 1560–1573.

[29] Latency, "Latency is everywhere and it costs you sales - how to crush it," 2009. [Online]. Available: http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it

[30] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 467–478.

[31] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau, "Efficient online coflow routing and scheduling," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2016, pp. 161–170.

[32] H.-Y. Lin and W.-G. Tzeng, "A secure decentralized erasure code for distributed networked storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 11, pp. 1586–1594, 2010.

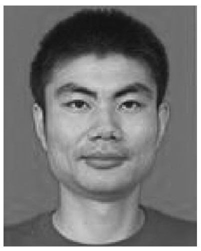
[33] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3366–3380, Nov. 2016.

[34] F. Ma, F. Liu, and Z. Liu, "Multi-objective optimization for initial virtual machine placement in cloud data center," *J. Inf. Comput. Sci.*, vol. 9, no. 16, pp. 5029–5038, 2012.

[35] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan, "Minimizing the sum of weighted completion times in a concurrent open shop," *Operations Res. Lett.*, vol. 38, no. 5, pp. 390–395, 2010.
[36] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 491–502, 2014.
[37] A. Munir, I. Qazi, Z. Uzmi, A. Mushtaq, S. Ismail, M. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2157–2165.
[38] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proc. 27th ACM Symp. Parallelism Algorithms Archit.*, 2015, pp. 294–303.
[39] Z. Qiu, C. Stein, and Y. Zhong, "Experimental analysis of algorithms for coflow schedulingâŃĘ," *arXiv preprint arXiv:1603.07981*, 2016.
[40] T. A. Roemer, "A note on the complexity of the concurrent open shop problem," *J. Scheduling*, vol. 9, no. 4, pp. 389–396, 2006.
[41] A. S. Schulz, "Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds," in *Proc. Int. Conf. Integer Program. Combinatorial Optimization*, 1996, pp. 301–315.
[42] M. Shafiee and J. Ghaderi, "An improved bound for minimizing the total weighted completion time of coflows in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 1–14, Aug. 2018.
[43] H. Susanto, H. Jin, and K. Chen, "Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2016, pp. 1–10.
[44] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Computer Commun.*, 2012, pp. 115–126.
[45] S. Wang, J. Zhang, T. Huang, J. Liu, T. Pan, and Y. Liu, "A survey of coflow scheduling schemes for data center networks," *IEEE Commun. Mag.*, vol. 56, no. 6, pp. 2–8, Jun. 2018.
[46] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, and Y. Liu, "Multi-attributes-based coflow scheduling without prior knowledge," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1962–1975, Aug. 2018.
[47] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 50–61.
[48] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proc. Conf. ACM SIGCOMM Conf.*, 2016, pp. 160–173.
[49] H. Zhang, X. Shi, X. Yin, F. Ren, and Z. Wang, "More load, more differentiation-a design principle for deadline-aware congestion control," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 127–135.
[50] H. Zhang, X. Shi, X. Yin, Z. Wang, and Y. Guo, "Fdrc-flow duration time based rate control in data center networks," in *Proc. IEEE/ACM 24th Int. Symp. Quality Serv.*, 2016, pp. 1–10.
[51] Y. Zhao, K. Chen, W. Bai, and M. Yu, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 424–432.

Zhiliang Wang received the BE, ME, and PhD degrees in computer science from Tsinghua University, China, in 2001, 2003, and 2006, respectively. Currently he is an associate professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include formal methods and protocol testing, next generation Internet, network measurement.

Han Zhang received the BS degree in computer science and technology from JiLin University and the PhD degree from Tsinghua University. He is now working with the School of Cyber Science and Technology, Beihang University. His research concerns computer networks, network security and AI. He is a member of the IEEE.


Xingang Shi received the BS degree from Tsinghua University and the PhD degree from The Chinese University of Hong Kong. He is now working with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network measurement and routing protocols.

Xia Yin received the BE, ME, and PhD degrees in computer science from Tsinghua University, in 1995, 1997, and 2000, respectively. She is a full professor with the Department of Computer Science and Technology, Tsinghua University. Her research interests include future internet architecture, formal method, protocol testing, and large-scale internet routing.

Yahui Li received the BS degree in computer science and technology from JiLin University. She is working toward the PhD degree at Tsinghua University. Her research concerns computer networks, network security and AI. She is a member of the IEEE.

Haijun Geng received the BE degree from Yantai University, in 2008, the ME degree from Capital Normal University, in 2011, and the PhD degree from Tsinghua University, in 2015. He is now working in the School of Software Engineering, Shanxi University. His research interests include future Internet architecture and largescale internet routing.

Qianhong Wu received the PhD degree in cryptography from Xidian University, in 2004. Since then, he has been with Wollongong University (Australia) as an associate research fellow, with Wuhan University (China) as an associate professor, and with Universitat Rovira i Virgili (Spain) as a research director. He is currently a professor with Beihang University in China. His research interests include cryptography, information security and privacy, VANET security and cloud computing security. He has been a holder/co-holder of 10 China/Australia/Spain funded projects. He has authored 30 patents and more than 130 publications. He has served as associate/guest editor in several international ISI journals and in the program committee of dozens of international conferences. He is a member of IACR IEEE and ACM.

Jianwei Liu received the BS and MS degrees in electronic engineering from Shandong University, China, in 1985 and 1988, and the PhD degree in communication engineering from Xidian University, China, in 1998. He is currently a professor and dean of School of Cyber Science and Technology, Beihang University. His current research interests include cryptographic protocol design, security on wireless and mobile network, computer network security, and cryptography. He has published six books and nearly 200 papers in his research fields. He is a senior member of the Chinese Institute of Electronics and director of the Chinese Association for Cryptologic Research. He has been awarded the first prize of technological invention of China.