Scheduling Coflows with Incomplete Information

Tong Zhang, Fengyuan Ren, Ran Shu, Bo Wang
Beijing National Research Center for Information Science and Technology (BNRist), Beijing, China
Dept. of Computer Science and Technology, Tsinghua Univeresity, Beijing, China
{zhangt, renfy, shuran, wangbo}@csnet1.cs.tsinghua.edu.cn

***Abstract*—In recent years, the coflow abstraction has received significant attentions, for its prominent ability to capture application semantics. On this basis, multiple coflow scheduling mechanisms have been proposed to minimize the coflow completion time (CCT). Currently, existing coflow scheduling mechanisms mainly belong to two categories: information-omniscient and information-agnostic. However, in data center applications, there are still quite a few cases in between where incomplete coflow information is known, and such incomplete information makes great contributions to improving the CCT performance. To address such cases, we propose IICS, a coflow scheduling algorithm based on incomplete coflow information. IICS leverages information of a coflow's arrived parts to deduce the coflow's remaining transmission time, and uses it to approximate the Minimum Remaining Time First (MRTF) heuristic. Besides, IICS allocates bandwidth by monopolization and in a maximal manner, which achieves high bandwidth utilization. Extensive simulations under realistic settings show that IICS achieves the average CCT comparable to that of the information-omniscient algorithm and the 99th percentile CCT much smaller than both information-omniscient and information-agnostic algorithms. Furthermore, IICS holds observably higher throughput and is robust to algorithm parameters.**

***Index Terms*—Coflow Scheduling, Incomplete Information, Coflow Completion Time, Data Center**

I. Introduction

Current data centers widely adopt cluster computing frameworks (e.g., MapReduce [1], Dryad [2], Spark [3]) to handle application demands. In spite of the differences among these frameworks, their network traffic is similarly shaped by application-level requirements. That is, data transfers are organized in communication stages between successive computation stages. In addition, there is a barrier at the end of each communication stage: only when all containing transfers finish can the next stage start. In this case, the coflow abstraction [4] has gathered a lot of interest, for its capability of capturing application-level semantics.

A coflow refers to a collection of concurrent flows with associated semantics and a collective performance goal [4]. A coflow does not complete until all its inner flows finish. In cluster computing frameworks, a communication stage's all containing flows compose a typical coflow. According to the measurement in [5], communication could account for more than 50% of job completion time. Therefore, minimizing the coflow completion time (CCT) truly accelerates the application proceeding, which is closely correlated with user experience and commercial profit. And there have been plenty of coflow scheduling mechanisms devoting to optimizing CCT [5–16].

Existing coflow scheduling mechanisms can be roughly divided into two categories: information-omniscient [5–7, 9, 10, 12, 14–16] and information-agnostic [8, 11, 13]. The former assumes that the complete coflow characteristics are known upon coflow arrival and utilizes such information to scheduling coflows, while the latter does not assume any prior knowledge about coflows' characteristics thus can only use flows' transfered sizes in scheduling. However, in effect there still exist a case between these two, where partial but not complete coflow information is available during scheduling. In all non-pipelining processing systems (Hadoop [17], Apache Hive [18], Apache Spark [19], etc.), the application could see the complete flow data before they are transfered, then can notify the flow's all characteristics to the scheduler [20] at its arrival to the network. However, an application may not carry out all its computations in one stage at the same time [4, 21], hence a coflow's inner flows may not arrive together. In short, a coflow's all currently arrived flows' characteristics constitute its incomplete information. To our knowledge, there has been no mechanism special for such cases so far.

In this paper, we exactly address the incomplete-information-available cases. Through a simple example, we illustrate how leveraging incomplete coflow information improves the CCT performance. To understand coflow scheduling more explicitly, we abstract out the whole data center network into a non-blocking giant switch. On this basis, we formalize the incomplete-information-based coflow scheduling problem into a stochastic online machine scheduling problem.

Based on the network model and problem formalization, we propose IICS, an **I**ncomplete-**I**nformation-based **C**oflow **S**cheduling algorithm in data center networks. IICS leverages each coflow's arrived parts to speculate its remaining transmission time. Specifically, IICS continuously refines each coflow's unique transmission time distribution as its arrived flows gradually increases. With the constructed distributions, IICS adopts the Generalized Gittins Index Priority Policy (GEN-GIPP) to approximate the Minimum Remaining Time First (MRTF) heuristic. Gittins index here represents the probability of coflow completing within a given time period normalized by the remaining completion time expectation. At any moment, the larger a coflow's Gittins index is, the less remaining transmission time this coflow may have. Therefore IICS assigns a higher priority to the coflow with a larger Gittins index. Furthermore, IICS lets flows monopolize the link bandwidth as long as they are scheduled to transfer data, which ensures work-conservation and achieves high bandwidth

978-1-5386-2542-2/18/$31.00 2018 IEEE

utilization. We theoretically analyze IICS's CCT performance.

We conduct a series of trace-driven simulations to roundly evaluate IICS. Results show that IICS achieves the average CCT comparable to that of Varys and 30% smaller than that of Aalo, and achieves 33% of reduction in 99th percentile CCT relative to both Varys and Aalo. Benefiting from link bandwidth monopolization, IICS obtains remarkably higher throughput than the other two ($1.35\times$ Varys and $1.72\times$ Aalo). We also investigate the influence of algorithm parameters on IICS performance, and IICS is robust to parameter variations.

In brief, our contributions are three-fold:

- First addressing the cases where incomplete coflow information is available.
- Designing a coflow scheduling algorithm based on incomplete coflow information.
- A systematic evaluation of the proposed algorithm over practical workloads and configurations.

The rest of the paper is organized as follows. In Section II, we introduce coflow scheduling in data center networks. We also discuss the cases where incomplete coflow information is available upon scheduling. Through a simple example, we illustrate how incomplete information helps to improve CCT performance. Section III establishes the non-blocking giant switch model and formalizes the coflow scheduling problem. On this basis, an incomplete-information-based coflow scheduling algorithm called IICS is proposed in Section IV. Then Section V theoretically analyzes IICS's CCT performance and Section VI evaluates IICS by simulations. Finally, this paper is concluded in Section VII.

II. Background and Motivation

In this section, we introduce coflow abstraction, coflow scheduling, as well as existing coflow scheduling mechanisms. On this basis, we investigate the extensive cases where incomplete coflow information is available during scheduling. Using a simple example, we illustrate how such incomplete information helps to reduce CCT, which motivates our design.

A. Background

Communication is crucial in data center applications. According to the measurement in [5], communication can account for over 50% of the job completion time. Therefore, optimizing data transfers can truly speed up the whole application proceeding. In current data centers, data transfers are generally organized in communication stages with one barrier (synchronization constraint) at the end of each: only when all transfers in a communication stage complete can the subsequent operation begin.

In this case, the coflow abstraction rises in response to the proper time and conditions and perfectly captures application-level semantics. The coflow abstraction is first proposed in [4], referring to a collection of concurrent flows with associated semantics and a collective performance goal. For example, all the data fetching flows in a MapReduce shuffle phase compose a typical coflow. Only when all inner flows finish transferring can the coflow completes, which exactly corresponds to the barrier stated above. Therefore, optimizing coflow completion time (CCT) is a natural choice, and coflow scheduling is the most widely-adopted means of minimizing CCT.

There have been various coflow scheduling mechanisms [5–13, 16]. Existing coflow scheduling mechanisms can be roughly classified into two categories: information-omniscient and information-agnostic, according to the coflow information they use during scheduling. Information-omniscient scheduling mechanisms assume the complete coflow characteristics (the number of inner flows, flow sizes, sources and destinations) are known upon arrival. Such kind of mechanisms include Orchestra [5], Varys [7], Baraat [6], RAPIER [10], Zhen Qiu et al. [9], D-CAS [12], Sunflow [14], SkipL [15], and Adia [16]. On the contrary, information-agnostic mechanisms do not assume anything about coflow characteristics. Upon each flow arrival, only its belonging coflow id and endpoints are learned. And as the flow proceeds, only its transfered data size can be known. However, all coflow-level characteristics like the number of inner flows, entire flow sizes, and all incoming inner flows (arrival time, size, endpoints, etc.) remain unknown until the coflow completion. Therefore, this kind of mechanisms employ the LAS principle and base coflow scheduling decisions only on currently transfered sizes. Such mechanisms mainly include Aalo [8], CODA [11], and Stream [13].

Actually, there can still be a third case between the above two, and we name it incomplete-information-available case. Just as the name implies, in this case partial but not complete information about coflow characteristics can be obtained and leveraged during the scheduling process. Specifically, a coflow's incomplete information refers to its currently arrived flows' all flow-level characteristics. Compared with information-agnostic cases, the additional information here is the full flow size rather than transfered size upon flow arrival. In practice, there are a large number of incomplete-information-available scenarios in data centers. In all non-pipelining processing systems (e.g., HDFS [17], Apache Hive [18], Apache Spark [19], HTTP-based web search), intermediate results are fully calculated before submitted to the network. That is, applications could see the complete flow data then notify the flow's all characteristics to the scheduler [20]. On the other hand, an application may not perform all its computation within one stage concurrently [4, 21], and there do not exist barriers for computation stages (each server directly transfers its intermediate result after its own computation completes). Therefore, a coflow's inner flows may not arrive together. And those unarrived flows indicate their data have not been worked out yet, thus there is no way to know their characteristics (arrival time and size). To summarize, within each coflow, information of all arrived flows is known upon their arrivals, while flows yet to come remains unknown. And as flows come, the known information gradually increases. Our designed algorithm exactly applies to these situations, and is right based on such incomplete information. As far as we know, there has been no mechanism addressing such cases so far. Next we will illustrate how incomplete information helps with optimizing CCT.

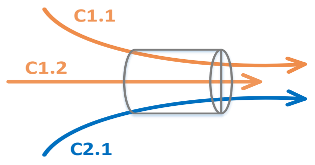
B. Motivation

Here we demonstrate the possible improvement in CCT performance with leveraging incomplete information, using a simple example. Consider a simple scenario with two coflows contending for one bottleneck link, as shown in Fig.1(a). Two orange (light) flows ($C1.1$ and $C1.2$) belong to coflow1 ($C1$), while the blue (dark) one ($C2.1$) belongs to the single-flow coflow2 ($C2$). The information about flow size and arrival time is listed in Fig.1(b). Both $C1.1$ and $C1.2$ have 1 unit of data, and $C2.1$ has two. And $C1.1$ and $C2.1$ arrive at time 0, while $C1.2$ at time 2. This scenario is a simplest but very common case that can occur on a data center network link, and this toy example is enough to illuminate the power of incomplete information. Assuming the link in Fig.1(a) has the capacity 1 (can transfer at most 1 data unit per time unit), then the scheduling results of an information-agnostic algorithm as well as an incomplete-information-aware one are respectively depicted in Fig.1(c) and Fig.1(d).

Both two algorithms try to approximate the well known MRTF heuristic, whose basic principle is to prioritize coflows with less remaining completion time when they exclusively occupy the network. It should be stressed that when a coflow's inner flows do not arrive together, the completion time should be replaced by the transmission time, and does not involve the inter-flow-arrival time. That is because the coflow only occupies bandwidth when it is really transferring data, and the interval between flows can still be used to transfer other coflow data. MRTF is really intuitive: letting faster coflows to complete first could reduce the waiting time of others, thus could decrease the average CCT value. In addition, MRTF has been demonstrated as CCT-effective [7, 10, 12, 16].

In Fig.1(c), the scheduler only knows a flow's belonging coflow id, source and destination upon its arrival, as well as its transfered data size as it proceeds. Except these, others are utterly ignorant. In this case, the scheduler adopts LAS to approximate MRTF. As the name suggests, LAS means the coflow that has transferred least bytes gets the highest priority. At time 0, both $C1$ and $C2$ have transferred 0 bytes, therefore the two arrived flows $C1.1$ and $C2.1$ will equally share the link bandwidth on the macro scale until $C1.1$ completes at time 2. Also at time 2, $C1.2$ arrives. Because both $C1$ and $C2$ have transferred 1 data unit, they again own equal priorities, thus $C1.2$ and $C2.1$ continue to equally share the link bandwidth until completion. In this information-agnostic case, both $C1$ and $C2$ complete at time 4, and the average CCT is 4.

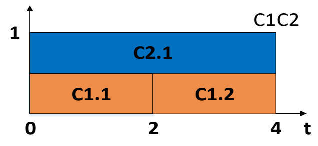
Relatively, in Fig.1(d), there is only one more thing that the scheduler knows relative to Fig.1(c), that is the total flow size upon flow arrival. At time 0, the scheduler learns the sizes of both $C1.1$ and $C2.1$, and then uses them to infer the transmission time of $C1$ and $C2$. Because $C1.1$ is smaller than $C2.1$, the scheduler believes $C1$ could complete faster than $C2$ and let $C1.1$ preferentially monopolize the link bandwidth. At time 1, $C1.1$ completes and $C2.1$ takes over the full link bandwidth. At time 2, $C1.2$ comes and $C2.1$ still has one data unit to transfer. The scheduler again infers the



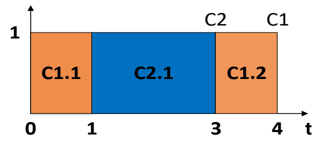
(a) Scenario

Flow	C1.1	C1.2	C2.1
Arrival Time	0	2	0
Size	1	1	2

(b) Coflow information



(c) Information-agnostic



(d) Incomplete-information-aware

Fig. 1. A motivating example

remaining transmission time for the two coflows according to remaining sizes of $C1.2$ and $C2.1$. Here the scheduler believes $C2$ could complete faster, thus $C2.1$ takes up the full link bandwidth until completion at time 3. Finally $C1.2$ finishes at time 4. With utilizing incomplete information, two coflows separately complete at time 3 and 4, and the average CCT is 3.5, decreasing by 12.5%. Note that even if the scheduler prioritizes $C1.2$ at time 2, the final average CCT is still 3.5, and this is actually the optimal average CCT that could be achieved by any scheduling algorithm in this case.

The 12.5% of average CCT decrease is because the known flow sizes provide the basis of deducing coflows' remaining transmission time. In this way, flows do not simply share bandwidth, but follow the conjectural priorities, overcoming the poor CCT performance of fair sharing. Furthermore, a coflow's priority can be continuously rectified as its inner flows arrive. In summary, incomplete information contributes in the following way: a coflow's total transmission time must be no less than that of its already arrived portion. Thus we can get the lower bound of the total transmission time in real time, then derive the lower bound of the remaining transmission time. If we can further learn the statistical distribution of all coflow transmission time, we can use this lower bound to derive the distribution of each coflow's own remaining transmission time. Our design is right based on this forthright idea.

III. Model and Formalization

In this section, we establish a model to abstract out the data center fabric. Besides, we describe coflow characteristics from the perspective of statistical properties. On this basis, we formalize our coflow scheduling problem into a preemptive stochastic online scheduling problem in the scheduling theory. Both the network model and problem formalization serve as the foundation of our designed algorithm in the next section.

A. Network and Coflow Model

Like in many previous data center transport designs [7–9, 12, 14, 16, 20], we abstract out the whole data center fabric into a non-blocking input-queued giant switch, as shown in Fig.2. Therefore, a flow can only be throttled by its edge link, and there is no congestion inside the network. This abstraction

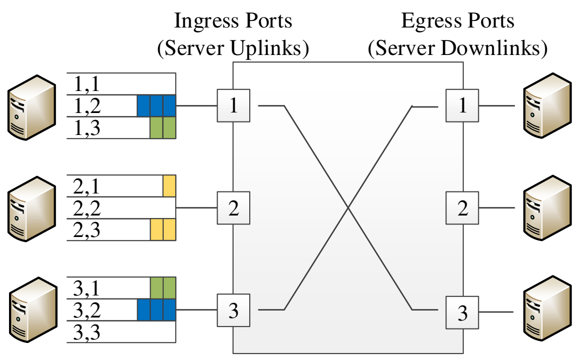


Fig. 2. Non-blocking giant switch abstraction

is practical due to the advances in full-bisection bandwidth networks [22–24].

Fig.2 depicts a 3×3 giant switch fabric. Each ingress port represents a server's uplink, while each egress port denotes a downlink. Each ingress port organizes its outgoing flows in virtual output queues by their destinations. In Fig.2, flows in the virtual queues belong to three coflows, which are coded by different colors. The blue (dark) coflow is an aggregation (many to one) coflow, because two senders (server 1 and server 3) send data to one receiver (server 2). By parity of reasoning, the yellow (light) coflow is a broadcast (one to many) coflow, and the green (medium) one is a shuffle (many to many) coflow. According to the coflow definition [4], these three patterns can cover all coflows.

Under the giant switch abstraction, a flow could be described by a triple $\langle source, destination, size\rangle$, then a coflow is naturally a set of such triples. A coflow's transmission time should be its longest transmission time on all ports (including ingress and egress) when the coflow monopolizes the fabric bandwidth. For example, if all ports in Fig.2 own the same capacity C, the transmission time of the blue coflow should be its transmission time on egress 2. Since the blue coflow has 6 packets to transfer on egress 2, assuming each packet has the size L, the transmission time of the blue coflow is $\frac{6L}{C}$. In general, a coflow's transmission time T should be:

$$T = \max\{\max_i \frac{\sum_j d_{ij}}{C_i^{in}}, \max_j \frac{\sum_i d_{ij}}{C_j^{out}}\} \tag{1}$$

where d_{ij} denotes the total data volume that the coflow needs to transfer from ingress port i to egress port j, C_i^{in} is the capacity of ingress port i, and C_j^{out} the capacity of egress port j. Assuming there are respectively N ingress ports and N egress ports, we define the vector $\boldsymbol{C}^{in}$ to be $[C_1^{in}, C_2^{in}, \cdots, C_N^{in}]^T$, and $\boldsymbol{C}^{out}$ to be $[C_1^{out}, C_2^{out}, \cdots, C_N^{out}]^T$. Intuitively, $\sum_j d_{ij}$ is the total data size through ingress i, then $\frac{\sum_j d_{ij}}{C_i^{in}}$ expresses the transmission time on ingress i. Similarly, $\frac{\sum_i d_{ij}}{C_j^{out}}$ means the transmission time on egress j. Finally, taking the maximum value through all ports (ingress and egress), we can get the transmission time of the coflow, which is also called "effective bottleneck" in [7]. Within one coflow, we refer to those flows through the port that achieves T in Equation (1) as bottleneck flows, others as non-bottleneck flows.

In addition to the network model, we also need to describe coflow characteristics from a statistical point of view, which helps with inferring coflows' remaining transmission time when only incomplete information is known. According to the measurement in [7], within a certain time period, coflow characteristics (including length, width, size, skew, sender-to-receiver ratio) can all be described by probability distributions. Therefore we can say, within a certain time period and assuming all edge link bandwidths are homogeneous, the transmission time of coflow also follows a probability distribution. We define the cumulative distribution function (CDF) of coflow transmission time to be $F_T(t)$. However, as illustrated in [25], the flow size distribution is not fixed but time-varying. Correspondingly, $F_T(t)$ is also time-varying.

B. Problem Formalization

Our design goal is a centralized online CCT-optimizing coflow scheduling algorithm based on incomplete information. Regarding the entire giant switch fabric as an integral resource, we formulate the coflow scheduling problem into the preemptive stochastic online machine scheduling problem.

The general machine scheduling problem can be described as follows: there is a set of jobs $\mathcal{J} = \{j_1, j_2, \cdots, j_n\}$ to be scheduled on a single or multiple machines (one job can occupy at most one machine at the same time). Each job j_i has an associated release time r_j, a positive processing time p_j, as well as a non-negative weight ω_j. r_j defines the earliest time point when job j is available for processing, p_j denotes the time period that job j needs to spend on the machine, and ω_j indicates the importance of job j. Scheduling means assigning these jobs to time slots and machines, and the goal is to minimize an objective function. Let c_j be the completion time of job j, the objective function is $\sum_j \omega_j c_j$ or $\mathrm{E}[\sum_j \omega_j c_j]$, the weighted sum of completion time or in expectation.

Here we refer to the whole giant switch fabric as the single machine. The release time r_j should be the arrival time of the first-coming flow(s) in coflow j, the processing time p_j be the transmission time of coflow j. And because we simply concern the average CCT, all ω_j are set to 1. The scheduling objective is to minimize $\mathrm{E}[\sum_j c_j]$, where c_j is the completion time of coflow j, corresponding to our goal of minimizing average CCT. Because the number of coflows n is a predefined value (could be infinity), minimizing $\mathrm{E}[\sum_j c_j]$ is equivalent to minimizing $\frac{\mathrm{E}[\sum_j c_j]}{n}$, the average completion time expectation. In our scenario, a coflow's r_j can only be known upon its first flow arrival, and p_j after all inner flows arrive. Besides, the number of total coflows keeps unknown during the scheduling process. That is to say, any coflow's transmission time is not known a priori, future coflow information cannot be predicted in advance, and we allow preemption during scheduling, thus our coflow scheduling problem corresponds to the *preemptive stochastic online* version of the machine scheduling problem.

Based on the above formalization, we propose our IICS in the next section.

IV. Incomplete-Information-Based Coflow Scheduling Algorithm

In this section, we propose our **I**ncompletion-**I**nformation-based **C**oflow **S**cheduling algorithm (IICS). On the whole, IIC-S is a centralized online algorithm that has two main parts, one for constructing the expected transmission time distribution (Distribution Construction), the other for assigning coflows to time slots (Coflow Scheduling). More specifically, IICS utilizes information of completed coflows to iteratively update the transmission time distribution $F_T(t)$, and continuously leverages the current $F_T(t)$ to schedule coflows. For coflow scheduling, IICS further has two components: one is priority assignment, the other is bandwidth allocation. The former is responsible for ranking coflows and individual flows, while the latter regulates how flows occupy the link bandwidth.

In what follows, we will describe the IICS design in two steps: first respectively presenting these two components in each static decision making moment, then interpreting IICS's dynamic coflow scheduling behavior in online situations.

A. Distribution Construction

For $F_T(t)$ construction, IICS employs the classic Exponent Weighted Moving Average (EMWA) to balance the historical distribution and the newly obtained coflow information.

In the initial period, $F_T(t)$ is assigned the following uniform distribution function,

$$F_T(t) = \begin{cases} 0 & t \leq Min \\ \frac{t-Min}{Max-Min} & Min < t \leq Max \\ 1 & t > Max \end{cases} \tag{2}$$

where Min and Max respectively represent the possible minimum and maximum coflow transmission time, which can be predicted according to applications' transmission requirements. Since at the beginning we know nothing about the transmission time distribution, we first suppose all possible values are with equal probability density.

During the algorithm proceeding, IICS keeps constructing another distribution function $F_{\widetilde{T}}(t)$ to keep track of the time-varying transmission time distribution. $F_{\widetilde{T}}(t)$ is defined as the statistical CDF of all coflows that has fully arrived within the latest H period. H acts as both the reset cycle of $F_{\widetilde{T}}(t)$ and the time threshold of full coflow arrival. On one hand, we recount $F_{\widetilde{T}}(t)$ from scratch every H period. On the other hand, we judge a coflow has fully arrived if none of its inner flows arrive in the whole H period from last arrival. To guarantee accuracy, we take the H value orders of magnitude larger than general inter-flow arrival intervals. Every H interval, the scheduler checks flow arrivals, judges out those fully arrived coflows, calculates their transmission time using Equation (1), and reconstructs $F_{\widetilde{T}}(t)$ using the following formula:

$$F_{\widetilde{T}}(t) = \frac{\sum_{c \in \mathbb{C}^H} I_{\{T_c < t\}}}{|\mathbb{C}^H|} \tag{3}$$

where $\mathbb{C}^{(H)}$ denotes the set of coflows that has fully arrived within the latest H interval, T_c represents the transmission time of coflow c, and $I_{\{.\}}$ is the indicator variable taking values from $\{0, 1\}$.

After getting the new $F_{\widetilde{T}}(t)$, we utilize it to update $F_T(t)$:

$$F_T(t) = (1-\alpha)F_T(t) + \alpha F_{\widetilde{T}}(t) \tag{4}$$

where α is the weight assigned to the newly constructed transmission time distribution. When α takes a constant value, the weighting for each historical distribution decreases exponentially. Relatively, the role of the new coflow is more outstanding. In this way, IICS manages to catch up with $F_T(t)$ variation and guarantee timeliness.

It is easy to deduce that the updated $F_T(t)$ has the form shown in Fig.3(a), which is a singular function consisting of interleaved linear and step functions. It must be stressed that all linear parts hold the same slope, and each step point value is taken the higher one. We define that each step point holds the probability of the jump value. Fig.3(a) is only a sketch map to manifest the function form, and the actual $F_T(t)$ is much more elaborate. Here we describe $F_T(t)$ construction at each renewal point (H timeout), and the online behavior will be explained in the Subsection IV-C.

B. Coflow Scheduling

With the help of $F_T(t)$, IICS schedules coflows to the data center fabric. Subsequently we will introduce how IICS prioritizes coflows and inner flows as well as allocates fabric bandwidth to them.

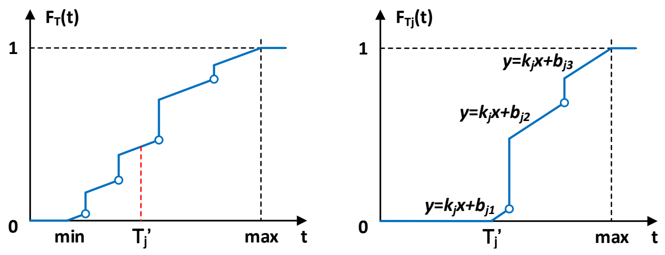
1) Priority Assignment: IICS assigns two priorities to each flow in the network: an inter-coflow priority and an intra-coflow priority. On inter-coflow priorities, IICS not only adopts but also enhances Generalized Gittins Index Priority Policy (GEN-GIPP) to rank coflows. As stated in Section III, our coflow scheduling problem is formalized into the preemptive stochastic online machine scheduling problem, and GEN-GIPP is right to solve this kind of problems. After the inter-coflow priorities are determined, IICS then directly leverages the MRTF heuristic to set intra-coflow priorities for each individual flow.

At every moment of scheduling decision making, IICS first calculates all present coflows' Gittins indices according to their currently obtained information, and uses it as the inter-coflow priorities. That is, a coflow's all inner flows inherit their belonging coflow's Gittins index as their inter-coflow priorities. For an uncompleted coflow j with the stochastic total transmission time T_j and deterministic attained transmission time y_j, its Gittins index $R_j(y_j)$ is calculated with the following equations.

$$I_j(q, y_j) = \mathrm{E}[\min\{T_j - y_j, q\} | T_j > y_j] \tag{5}$$

$$R_j(q, y_j) = \frac{Pr[T_j - y_j \leq q | T_j > y_j]}{I_j(q, y_j)} \tag{6}$$

$$R_j(y_j) = \max_{q \in R^+} R_j(q, y_j) \tag{7}$$



(a) $F_T(t)$ (b) $F_{T_j}(t)$

Fig. 3. CCT performance under varying loads

In Equation (5), q is a given time period, and $I_j(q, y_j)$ indicates the expected time investment of transmitting coflow j for q time units or up to completion, whichever comes first. And in Equation (6), $R_j(q, y_j)$ is deemed as the rank of the q-length part of coflow j after it has attained y_j units of transmission time. And this rank is calculated as the ratio of the probability that coflow j will be completed within the next q time units over the expected time investment. In Equation (7), the final Gittins index $R_j(y_j)$ is calculated by taking the maximum of $R_j(q, y_j)$ over all positive q values. Such definition indicates that the we are interested in the maximal rank that a coflow could achieve. In the physical sense, Gittins index describes the maximum probability of coflow completion within a given time period, normalized by the remaining transmission time expectation. After computing the Gittins index for every coflow, the inter-coflow priorities are determined, and a larger index value means a higher priority. We use Gittins index as the inter-coflow priority for its near-optimal property in the stochastic online machine scheduling problem, which we will prove in Section V.

The constructed distribution $F_T(t)$ specifically plays a part in calculating $R_j(y_j)$, because in Equation (6) computation of both the numerator and denominator needs the probability distribution of T_j. However, $F_T(t)$ is only a general CDF of the coflow transmission time, and for each specific coflow j with incomplete information, its $F_{T_j}(t)$ can be further refined. This is also the place where incomplete coflow information takes effect and IICS enhances GEN-GIPP. In any time, if the arrived part of coflow j has the transmission time T_j', then T_j must be larger than or equal to T_j'. Under this additional condition, we can define $F_{T_j}(t)$ on the basis of $F_T(t)$:

$$F_{T_j}(t) = \begin{cases} 0 & t \le T_j' \\ \frac{F_T(t) - F_T(T_j')}{1 - F_T(T_j')} & t > T_j' \end{cases} \tag{8}$$

The generation of $F_{T_j}(t)$ is simply demonstrated in Fig.3. After learning $T_j \ge T_j'$, in Fig.3(b), the $F_T(T)$ part on the left of T_j' directly becomes 0, while the part on the right of T_j' is reserved and scaled up. Because the scale factor varies with different coflows, each $F_{T_j}(t)$ has its own slope k_j for its linear parts. And we define b_{ji} to be the intercept of the i^{th} linear part in $F_{T_j}(t)$ on vertical axis. In this way, $F_{T_j}(t)$ becomes more accurate than $F_T(t)$ for each coflow, and it is

Algorithm 1 Static Bandwidth Allocation

```
procedure BANDWIDTHALLOCATION(Flows $\mathbb{F}$)
 1: $\mathbb{D} \leftarrow \emptyset$
 2: for each $f \in \mathbb{F}$, in non-increasing order of overall priority do
 3:     $C \leftarrow f$.belongingCoflow
 4:     if $f$.isNonBottleneckFlow and $C$.bottleneckFlows $\bigcap \mathbb{D} = \emptyset$ then
 5:         $\mathbb{F}$.REMOVE($C$.flows)
 6:         continue
 7:     end if
 8:     $f$.band = $f$.ingress.band
 9:     $\mathbb{D}$.ADD($f$)
10:     $\mathbb{F}$.REMOVE($f$.ingress.flows)
11:     $\mathbb{F}$.REMOVE($f$.egress.flows)
12: end for
13: return $\mathbb{D}$
```

incomplete coflow information that enables this enhancement. IICS updates $F_{T_j}(t)$ before calculating $R_j(y_j)$ each time, so $R_j(y_j)$ is calculated more precisely.

On the other hand, we prove that $R_j(y_j)$ is always obtained at a step point or boundary point of $F_{T_j}(t)$ in our case. Therefore, each time IICS only needs to calculate $R_j(q, y_j)$ when $q + y_j$ is a step point or a boundary and not traverse all q values, which greatly decreases the computing complexity. The detailed proof is presented in the appendix.

After solving inter-coflow priorities, IICS still needs to deal with intra-coflow priorities, because there may also exist bandwidth contentions among flows within a single coflow. Because a coflow's total transmission time hinges on its bottleneck flows, only when its bottleneck flows make progress does its remaining transmission time really decrease. To this end, IICS prioritizes a coflow's all bottleneck flows over its non-bottleneck ones. And further within the group of a coflow's bottleneck (non-bottleneck) flows, the priority rule is simply MRTF. In detail, for each bottleneck flow IICS uses its remaining size as its intra-coflow priority, while for a non-bottleneck flow this priority is defined as its remaining size plus the maximum priority value of its fellow bottleneck flows. By definition, a smaller value indicates a higher intra-coflow priority, which is contrary to the inter-coflow case.

With the above definition, each flow in the network holds both inter-coflow and intra-coflow priorities, of which the inter-coflow one is in preference. A flow is prior to another one if and only if its inter-coflow priority is higher another's, or their inter-coflow priorities are equal but the former has a higher intra-coflow priority. In this way, we can define an order between any two flows in the network.

2) Bandwidth Allocation: When it comes to bandwidth allocation, IICS lets flows fully take up port bandwidths in the non-increasing order of their priorities. Concretely, the whole bandwidth allocation process on each static decision making moment is described in Algorithm 1. The scheduler each time selects the highest-priority flow from all available flows (line

Algorithm 2 IICS Online

```
procedure IICSOnline(Coflows $\mathbb{C}$, flow $f$, event)
 1: if event == H_Timeout then
 2:     $\mathbb{C}' \leftarrow \emptyset$
 3:     for each $C \in \mathbb{C}$ do
 4:         if $C$.noFlowArrivalDuringLastH then
 5:             $\mathbb{C}'$.ADD($C$)
 6:             if $C$.numOfFlows == 0 then
 7:                 $\mathbb{C}$.REMOVE($C$)
 8:             end if
 9:         else
10:             $C$.noFlowArrivalDuringLastH $\leftarrow TRUE$
11:         end if
12:     end for
13:     Calculate $F_{\widetilde{T}}(t)$ with $\mathbb{C}'$ using Equation (3)
14:     Update $F_T(t)$ using Equation (4)
15: end if
16: if event == flowCompletion then
17:     $C \leftarrow f$.belongingCoflow
18:     $C$.REMOVE($f$)
19:     $C$.numOfFlows $\leftarrow$ $C$.numOfFlows - 1
20:     Recompute priorities for changed coflows in $\mathbb{C}$
21:     $\mathbb{F} \leftarrow \mathbb{C}$.flows
22:     $\mathbb{F} \leftarrow$SORT_DSC($\mathbb{F}$) using overall priority
23:     BANDWIDTHALLOCATION($\mathbb{F}$)
24: end if
25: if event == flowArrival then
26:     $C \leftarrow f$.belongingCoflow
27:     $C$.ADD($f$)
28:     if $C \notin \mathbb{C}$ then
29:         $\mathbb{C}$.ADD($C$)
30:     end if
31:     $C$.noFlowArrivalDuringLastH $\leftarrow FALSE$
32:     Recompute priorities of changed coflows in $\mathbb{C}$
33:     $\mathbb{F} \leftarrow$SORT_DSC($\mathbb{C}$.flows) using overall priority
34:     BANDWIDTHALLOCATION($\mathbb{F}$)
35: end if
```

2) to monopolize both its ingress and egress port bandwidth (line 8). Then the scheduler excludes all flows sharing the occupied ingress or egress ports from the available flow set (line 10-11). It should be stressed that if none of a coflow's bottleneck flows is allocated bandwidth, all its non-bottleneck flows will be excluded from the available set directly (line 4-7). Such practice indicates that when a coflow gets the bandwidth share, it must really make progress, or the bandwidth will be reserved for other coflows. Repeat the above process until no flow could be added, then all selected flows as well as their bandwidth share constitute a scheduling decision. Note that IICS's bandwidth allocation follows strict priorities, which is totally different from the prevailing bandwidth sharing rules. This approach is to guarantee the work-conservation property and improve bandwidth utilization, and we will detailedly explain this point in Subsection VI-B.

C. IICS Online

The above two subsections separately describe the IICS behaviors in distribution construction and coflow scheduling at every static scheduling moment. Furthermore, we still need to characterize IICS in the online situations because it is an online algorithm.

On the whole, IICS is event-driven, and IICS operations are triggered by three events: H timeout, flow completion, and flow arrival. The detailed operations are shown in Algorithm 2. H timeout corresponds to the H interval in $F_T(t)$ construction. Every H period, H timeout will happen once, driving IICS to screen out those coflows without new flow arrival (line 3-12) and to leverage their information to update $F_T(t)$ (line 13-14). On the other hand, flow completion and arrival will both lead to available flow set changes, therefore should trigger the scheduling decision update. Upon each flow completion, IICS first updates the available flow set (line 17-19), then recomputes priorities for changed coflows (line 20), and finally regenerates the scheduling decision (line 21-23). On each flow arrival, the IICS operation is similar to that upon flow completion: updates the available flow set (line 26-31), recomputes priorities (line 32), and regenerates the scheduling decision (line 33-35). However, there is difference between the priority updates upon flow completion and arrival. With flow completion, a changed coflow may have varied bottleneck flow set, remaining flow sizes, as well as attained transmission time. However, with flow arrival, the belonging coflow may also need to update the distribution $F_{T_j}(t)$.

V. Algorithm Analysis

In this section, we conduct theoretical analysis on the CCT performance of IICS. Through the property of GEN-GIPP, we have the following theorem:

Theorem 1. *IICS achieves the average CCT expectation that is twice the optimal value plus a constant, if the distribution $F_{T_j}(t)$ is accurate.*

Proof. Consider an ideal scenario where a coflow's all inner flows arrive together and complete coflow information is known. We define the optimal schedule in this ideal scenario to be S^* and the average CCT expectation achieved by S^* to be $E^*[\frac{\sum_{j=1}^{n} c_j}{n}]$. Because all inner flows become available at the coflow arrival, S^* could traverse the entire solution space under the given coflows and coflow arrival time. Thus S^* is also the optimal schedule when coflows and coflow arrival time are specified.

Consider another scenario where all coflows and arrivals are the same as the above ideal one. The difference is, the accurate coflow information is not known until completion, but the transmission time distribution $F_{T_j}(t)$ for each coflow is available. In this scenario, according to Theorem 3.11 in [26], if $F_{T_j}(t)$ is accurate, GEN-GIPP is a 2-approximation for minimizing $E[\frac{\sum_{j=1}^{n} c_j}{n}]$. Therefore we have

$$E^{GEN-GIPP}[\frac{\sum_{j=1}^{n} c_j}{n}] \leq 2E^*[\frac{\sum_{j=1}^{n} c_j}{n}]$$

Further consider a scenario where all coflows are same as in the ideal scenario, but arrivals are different. Inner-coflow flows do not come together, but with intervals, and only $F_{T_j}(t)$ is available. Our IICS is right in this scenario.

IICS applies GEN-GIPP to the non-simultaneous arriving scenario, and obtains the same coflow priorities as those of GEN-GIPP in the second scenario, because the all $F_{T_j}(t)$ are the same. Since IICS is preemptive, any coflow could only be delayed by its own inter-flow intervals or by other higher-priority coflow parts existing during its lifetime, whichever is larger. Therefore, the exceeding part in $E^{IICS}[\frac{\sum_{j=1}^{n} c_j}{n}]$ relative to $E^{GEN-GIPP}[\frac{\sum_{j=1}^{n} c_j}{n}]$ can be bounded by a constant $E[\max\{I,T\}]$ with I being the sum of a coflow's all inter-arrival intervals and T denoting the transmission time of all higher-priority coflow parts that exist during a coflow's life time. In effect, T is no larger than the maximum inter-arrival time between any two adjacent fellow flows. As a whole, $\max\{I,T\}$ must be less than or equal to the maximum value of total inter-arrival time within a coflow. Above all, we have

$$E^{IICS}[\frac{\sum_{j=1}^{n} c_j}{n}] \leq 2E^{*}[\frac{\sum_{j=1}^{n} c_j}{n}] + c \qquad (9)$$

where c is the maximum total inter-arrival time in all coflows. $\square$

Actually, the constant c representing a coflow's total inter-arrival time is inherent in the non-simultaneous arrival property and seems inevitable. In hence, Theorem 1 provides a high-quality bound for IICS's CCT performance.

VI. Evaluation

In this section, we conduct a series of flow-level simulations to evaluate IICS performance, including CCT (average and 99th percentile CCT), throughput, as well as algorithm sensitivity to changes of parameters. We take both Varys [7] and Aalo [8] as performance references.

A. Simulation Setup

Platform and topology: We develop our own trace-driven simulator in java. The simulator reproduces the whole flow transferring process and can count various flow-level or coflow-level metrics simultaneously. The simulation topology is the leaf-spine topology depicted in Fig.4, which is widely adopted in data centers. The whole fabric consists of 144 servers, 9 top of rack (ToR) switches, and 4 aggregation switches. All 144 servers are evenly distributed in 9 racks. Each server connects to one ToR switch through a 10Gbps link, while each ToR switch simultaneously connects to all aggregation switches by 4 40Gbps links. That is, there is a full connection between ToR and Aggregation switches. In this topology, the core-network provides enough bisection bandwidth physically. Note that we do not enforce the giant-switch assumption in our simulations.

Workload: We replay the traffic traces in [7] and scale down them to match our 144Gbps-bisection-bandwidth environment. However, we preserve the coflow characteristics of the original

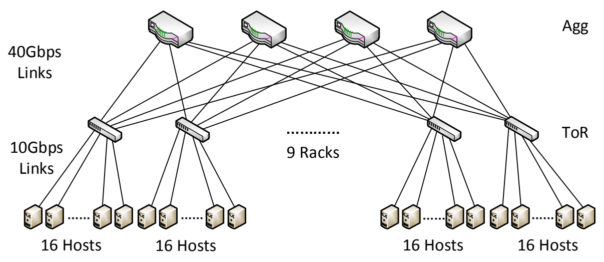


Fig. 4. Simulation topology

workload, including coflow length, width, size, and skew. Each flow's source and destination are both randomly selected from all 144 servers according to the uniform distribution. In our simulation, we vary the workload from 10% to 100% to observe the IICS performance under different levels of traffic loads. More specifically, we separately generate 1Gbps, 2Gbps, $\cdots$, 10Gbps workloads on every server.

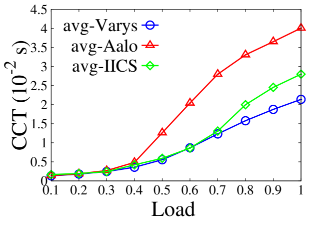
Performance metrics: We mainly care for the following two metrics, which are also the main performance indices concerned by data center transport designs.

- **CCT**. CCT is the most CCT is the most important performance metric that coflow scheduling mechanisms concern. In whether compute-intensive or data-intensive applications, CCT all greatly affect the application proceeding speed. We count both the average and 99th percentile CCT respectively.
- **Throughput**. Throughput immediately reflects bandwidth utilization, which is another important indicator that data center operators care for. Besides, there are also very large coflows ($>$ 1GB) in data centers accounting for over 99.6% of total bytes [7]. These large coflows are generally more concerned about throughput. Here the throughput is calculated by dividing the data volume through the whole fabric by the corresponding time period, thus is a global measure.

Reference algorithms: We take Varys [7] and Aalo [8] as the performance references, for their respective representativeness and superior CCT performance in information-omniscient and information-agnostic coflow scheduling mechanisms.

B. Algorithm Performance

Firstly, the average as well as 99th percentile CCTs of Varys, Aalo, and IICS under varying workloads are respectively shown in Fig.5(a) and Fig.5(b). In average CCT, IICS significantly outperforms Aalo (0.7× at 100% workload) and achieves very close performance to Varys, especially when the traffic load is below 70%. On one hand, IICS leverages incomplete information hence achieves smaller average CCT than Aalo. On the other hand, IICS's different bandwidth allocation manner from Varys makes up for the lack of coflow information to some extent, explaining the very close average CCT to that of Varys. Varys and Aalo separately adopt Minimum-Allocation-for-Desired-Duration (MADD) algorithm and max-min fairness to allocate bandwidth among flows


(a) Average CCT

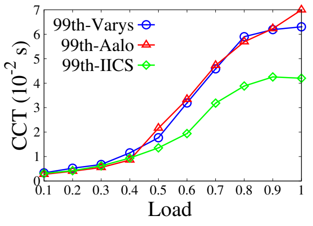
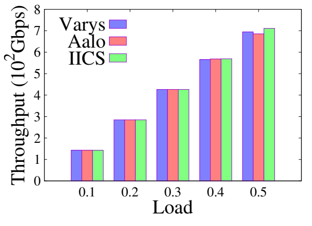

(b) 99th percentile CCT

Fig. 5. CCT under different workloads


(a) Throughput under lower loads

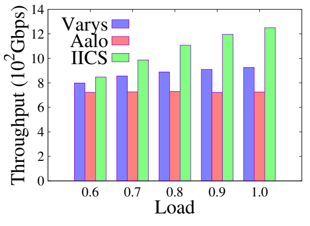
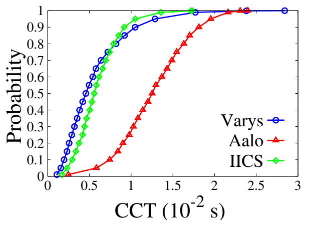

(b) Throughput under higher loads

Fig. 7. Throughput under different workloads


(a) CDF under 50% traffic load

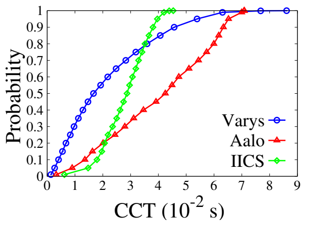
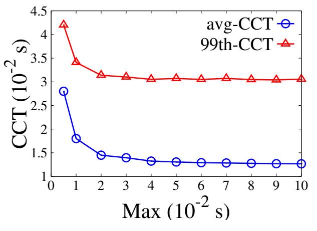

(b) CDF under 100% traffic load

Fig. 6. CCT distributions in low and high loads


(a) Influence of $F_T(t)$ upper bound

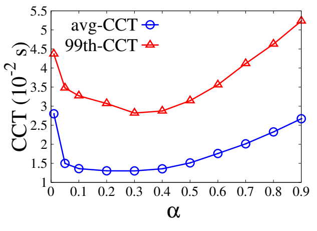

(b) Influence of EWMA weight

Fig. 8. Influence of IICS parameters

within each coflow. Such practices may violate the work-conservation property: letting the bandwidth that could have been utilized idle [7, 8], which hurts bandwidth utilization. Although Varys and Aalo both take actions to backfill such idle bandwidth, their proportional backfilling manner could still lead to a residual (the only flows capable of occupying the remaining bandwidth may have tiny weights). Relatively, IICS lets each flow monopolize the edge link bandwidth at every opportunity and fill the fabric with flows as much as possible, which directly guarantees work-conservation. Therefore, IICS can achieve a higher bandwidth utilization. This point could be seen more clearly in the subsequent throughput results. When it comes to 99th percentile CCT, we can see that Varys and Aalo hold almost identical performance, while IICS is obviously superior to them (0.66× at 100% workload). This is also because IICS improves the bandwidth utilization and then reduces the total makespan of all coflows.

Fig.6 depicts comparative CDFs of all coflows' CCTs with Varys, Aalo, and IICS separately. Fig.6(a) is under 50% traffic load, while Fig.6(b) is under 100%, in order to observe the CCT performance in different load conditions. It is intuitive that in either light or heavy traffic load, Varys generally performs better than Aalo in any percentile CCT. Under 50% load, IICS performance is roughly close to Varys but a little superior in tail CCTs. Under 100% workload, IICS is between Varys and Aalo in most percentiles, but obviously surpasses them in tail CCTs as well. Here the tail results are consistent with the 99th percentile CCTs in Fig.5(b), which can also be explained by IICS's work-conservation property.

Fig.7 presents the throughput of the three mechanisms under varying loads. In Fig.7(a), the traffic load is not exceeding 50%, and the three mechanisms achieve almost the same throughput. However, in Fig.7(b) we can see that IICS achieves observably higher throughput than Varys and Aalo (1.35× Varys and 1.72× Aalo at 100% workload). Such improvement confirms that IICS utilizes the bandwidth more efficiently, and also partly interprets IICS's prominent performance in average and tail CCTs above.

C. Influence of Parameters

After evaluating the basic metrics, we investigate the influence of IICS parameters on CCT performance. There are two main tunable parameters in the IICS algorithm, one is the pre-estimated upper bound of the expected coflow transmission time Max (the lower bound Min is always 0), the other is the EWMA weight α.

In this series of simulations, we use 100% workload to study algorithm sensitivity in stressed network situations. We run IICS under different values of these two parameters, and observe the CCT performance. The results are exhibited in Fig.8. Fig.8(a) shows the average and 99th percentile CCT under different Max values. During our simulation, the maximum coflow transmission time is about 0.016s, therefore the optimal Max should also be 0.016s. We vary Max from 0 to 0.1s and it can be seen that the both average and 99th percentile CCTs keep at the optimal value as long as Max is larger than 0.016s. Therefore we can simply set a large value for Max without influencing IICS accuracy. On the other hand, Fig.8(b) depicts the CCT performance under different α values. The larger α value, the more emphasis we lay on new data. We can see that, when α varies from 0 to 0.9 (almost the entire α range), both average and 99th percentile CCTs only change in a small range (the top is no more than twice the bottom). Furthermore, both CCTs keep at a low level when α is in the range $[0.05, 0.6]$. Therefore we can say, IICS is also not sensitive to the EWMA weight α. In summary, IICS is robust to parameters.

VII. Conclusions

In this paper, we address the problem of coflow scheduling with incomplete information. Correspondingly, we propose an incomplete-information-based coflow scheduling algorithm IICS. IICS leverages the information of coflow parts that have arrived to infer the remaining transmission time, and uses the GEN-GIPP scheduling policy to approximate the MRTF heuristic. It refines each coflow's specific transmission time distribution as arrived part increases. Besides, IICS allocates link bandwidths by monopolization, achieving high bandwidth utilization. A variety of simulations under practical configurations show that IICS achieves the average CCT much smaller than that of Aalo (0.7×) and comparable to that of Varys, and achieves reduction in the 99th percentile CCT relative to both Aalo and Varys (0.66×). Besides, IICS holds the highest throughput among the three (1.35× Varys and 1.72× Aalo) and is very robust to parameter variations.

VIII. Acknowledgement

The authors gratefully acknowledge the reviewers for their constructive comments. This work is supported in part by Suzhou-Tsinghua Special Project for Leading Innovation.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS operating systems review*, 2007.

[3] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *NSDI*, 2012.

[4] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *HotNets*, 2012.

[5] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *SIGCOMM*, 2011.

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

[7] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *SIGCOMM*, 2014.

[8] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *SIGCOMM*, 2015.

[9] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *SPAA*, 2015.

[10] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *INFOCOM*, 2015.

[11] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *SIGCOMM*, 2016.

[12] 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 TPDS*, 2016.

[13] H. Susanto, H. Jin, and K. Chen, "Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks," in *ICNP*, 2016.

[14] X. S. Huang, X. S. Sun, and T. Ng, "Sunflow: Efficient optical circuit scheduling for coflows," in *CoNEXT*, 2016.

[15] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, Y. Liu, J. Li, and F. Li, "Skipping congestion-links for coflow scheduling," in *IWQoS*, 2017.

[16] J. Jiang, S. Ma, B. Li, and B. Li, "Adia: Achieving high link utilization with coflow-aware scheduling in data center networks," *IEEE Transactions on Cloud Computing*, 2017.

[17] "Hadoop," http://hadoop.apache.org.

[18] "Apache hive," http://hive.apache.org.

[19] "Apache spark," https://spark.apache.org.

[20] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *SIGCOMM*, 2013.

[21] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I. Stoica, "Pacman: Coordinated memory caching for parallel jobs," in *NSDI*, 2012.

[22] 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," in *SIGCOMM*, 2009.

[23] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, 2010.

[24] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *SIGCOMM*, 2014.

[25] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *NSDI*, 2015.

[26] N. Megow, "Coping with incomplete information in schedulingstochastic and online models," *Operations Research Proceedings 2007*, 2008.

Appendix

A. Calculation of Gittins index $R_j(y_j)$

Theorem 2. *A coflow's Gittins index $R_j(y_j)$ is always obtained at a step point of $F_{T_j}(t)$ or T'_j, i.e., $R_j(q, y_j)$ always achieves the maximum value when $y_j + q$ is a step point's location, or $q = T'_j - y_j$.*

Proof. For any coflow j with the attained transmission time y_j, random expected total transmission time T_j, and the distribution $F_{T_j}(t)$, let $x = y_i + q$. Then we can get the expression of $R(q, y_j)$ in x as follows:

$$R(x - y_j, y_j) = \frac{F_{T_j}(x)}{x(1 - k_j x - b_{ji}) + \frac{k}{2}(x^2 - T_j'^2) + \Sigma} \tag{10}$$

where the numerator represents $Pr[T_j \le x | T_j \ge y_j]$ ($Pr[T_j - y_j \le q | T_j \ge y_j]$), and the denominator calculates $I_j(x - y_j, y_j)(I_j(q, y_j))$. $x(1 - k_j x - b_{ji})$ denotes the partial expectation when T_j exceeds x but is capped by x, $\frac{k}{2}(x^2 - T_j'^2)$ counts the partial expectation within linear parts of $F_{T_j}(t)$ before x, while Σ denotes the partial expectation at all step points before x.

Firstly, each step point x_0, the $R(x_0 - y_j, y_j)$ value is larger than its left limit $\lim_{x \to x_0^-} R(x - y_j, y_j)$. That is because relative to the left limit, when x_0 is really taken, the numerator increases by the jump value at x_0, while the denominator keeps unchanged.

Then within an arbitrary linear interval i of $F_{T_j}(x)$,

$$R(x - y_j, y_j) = \frac{k_j x + b_{ji}}{x(1 - k_j x - b_{ji}) + \frac{k}{2}(x^2 - T_j'^2) + \Sigma}$$

which is a function that decreases at first and then increases. Therefore, reflecting on each linear interval, the function must be monotone increasing or monotone decreasing or decreasing at first then increasing. Therefore, the maximum value of $R(x - y_j, y_j)$ will certainly not fall into a linear part, and can only be taken at step points or boundaries (T'_j and Max). □