# Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks

Zhen Qiu*, Cliff Stein* and Yuan Zhong*

## ABSTRACT

Communications in datacenter jobs (such as the shuffle operations in MapReduce applications) often involve many parallel flows, which may be processed simultaneously. This highly parallel structure presents new scheduling challenges in optimizing job-level performance objectives in data centers.

Chowdhury and Stoica [11] introduced the coflow abstraction to capture these communication patterns, and recently Chowdhury et al. [13] developed effective heuristics to schedule coflows. In this paper, we consider the problem of efficiently scheduling coflows with release dates so as to minimize the total weighted completion time, which has been shown to be strongly NP-hard [13]. Our main result is the first polynomial-time deterministic approximation algorithm for this problem, with an approximation ratio of 67/3, and a randomized version of the algorithm, with a ratio of $9 + 16\sqrt{2}/3$. Our results use techniques from both combinatorial scheduling and matching theory, and rely on a clever grouping of coflows. We also run experiments on a Facebook trace to test the practical performance of several algorithms, including our deterministic algorithm. Our experiments suggest that simple algorithms provide effective approximations of the optimal, and that our deterministic algorithm has near-optimal performance.

## 1. INTRODUCTION

With the explosive growth of data-parallel computation frameworks such as MapReduce [14], Hadoop [1, 8, 32], Spark [36], Google Dataflow [2], etc., modern data centers are able to process large-scale data sets at an unprecedented speed. A key factor in materializing this efficiency is parallelism: many applications written in these frameworks

*The authors are with the Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA. {zq2110, cs2035, yz2561}@columbia.edu. This research is supported in part by NSF grants CCF-1349602 and CCF-1421161.

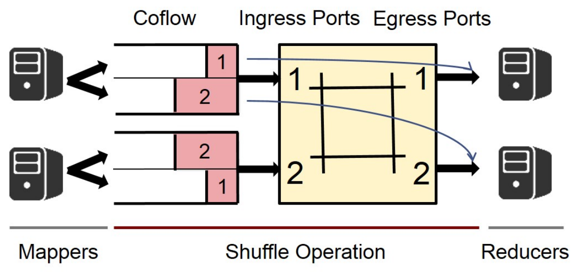


Figure 1: A MapReduce application in a $2 \times 2$ network

alternate between computation and communication stages, where a typical computation stage produces many pieces of intermediate data for further processing, which are transferred between groups of servers across the network. Data transfer within a communication stage involves a large collection of parallel flows, and a computation stage often cannot start until all flows within a preceding communication stage have finished [12, 15].

While the aforementioned parallelism creates opportunities for faster data processing, it also presents challenges for network scheduling. In particular, traditional networking techniques focus on optimizing flow-level performance such as minimizing flow completion times, and ignore application-level performance metrics. For example, the time that a communication stage completes is the time that the last flow within that stage finishes, so it does not matter if other flows of the same stage complete much earlier than the last.

To faithfully capture application-level communication requirements, Chowdhury and Stoica [11] introduced the *coflow* abstraction, defined to be a collection of parallel flows with a common performance goal. Effective scheduling heuristics were proposed in [13] to optimize coflow completion times. In this paper, we are interested in developing scheduling algorithms with provable performance guarantees, and our main contribution is a deterministic polynomial-time 67/3 approximation-algorithm and a randomized polynomial-time $(9 + 16\sqrt{2}/3)$-approximation algorithm, for the problem of minimizing the total weighted completion time of coflows with release dates. These are the first $O(1)$-approximation algorithms for this problem. We also conduct experiments on real data gathered from Facebook, in which we compare our deterministic algorithm and its modifications to several other algorithms, evaluate their relative performances, and compare our solutions to an LP-based lower bound. Our algorithm performs well, and is much closer to the lower bound than the worst-case analysis predicts.

## 1.1 System Model

## *The network.*

In order to describe the coflow scheduling problem, we first need to specify the conceptual model of the datacenter network. Similar to [13], we abstract out the network as one giant *non-blocking* switch [4, 6, 19, 29] with $m$ *ingress ports* and $m$ *egress ports* – which we call an $m \times m$ network switch – where $m$ specifies the network size. Ingress ports represent physical or virtual links (e.g., Network Interface Cards) where data is transferred from servers to the network, and egress ports represent links through which servers receive data. We assume that the transfer of data within the network switch is *instantaneous*, so that any data that is transferred out of an ingress port is immediately available at the corresponding egress port. There are capacity constraints on the ingress and egress ports. For simplicity, we assume that all ports have unit capacity, i.e., one unit of data can be transferred through an ingress/egress port per unit time. See Figure 1 for an example of a $2 \times 2$ datacenter network switch. In the sequel, we sometimes use the terms *inputs* and *outputs* to mean ingress and egress ports respectively.

## *Coflows.*

A *coflow* is defined as a collection of parallel flows with a common performance goal. We assume that all flows within a coflow arrives to the system at the same time, the *release date* of the coflow. To illustrate, consider the shuffling stage of a MapReduce application with 2 mappers and 2 reducers that arrives to a $2 \times 2$ network at time 0, as shown in Figure 1. Both mappers need to transfer intermediate data to both reducers. Therefore, the shuffling stage consists of $2 \times 2 = 4$ parallel flows, each one corresponding to a pair of ingress and egress ports. For example, the size of the flow that needs to be transferred from input 1 to output 2 is 2. In general, we use an $m \times m$ matrix $D = (d_{ij})_{i,j=1}^m$ to represent a coflow, in a $m \times m$ network. $d_{ij}$ denotes the size of the flow to be transferred from input $i$ to output $j$. We also assume that flows consist of discrete data units, so their sizes are integers.

## *Scheduling constraints.*

Since each input can transmit at most one data unit and each output can receive at most one data unit per time slot, a *feasible* schedule for a single time slot can be described by a *matching* between the inputs and outputs. When an input is matched to an output, a corresponding data unit (if available) is transferred across the network. To illustrate, consider again the coflow in Figure 1, given by the matrix $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$. When this coflow is the only one present, it can be completed in 3 time slots, using matching schedules described by $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Here entry 1 indicates a connection between the corresponding input and output, so for example, the first matching connects input 1 with output 1, and input 2 with output 2. When multiple coflows are present, it is also possible for a matching to involve data units from different coflows.

An alternative approach toward modeling the scheduling constraints is to allow the feasible schedules to consist of *rate allocations*, so that *fractional* data units can be processed in each time slot. This approach corresponds to finding fractional matchings, and is used in most of the networking literature. We do *not* adopt this approach. When rate allocations can vary continuously over time, there is a much larger (infinite) set of allowable schedules. Furthermore, unless the time horizon is exceptionally short, restricting decision making to integral time units results in a provably negligible degradation of performance. Integral matchings will give rise to a much cleaner problem formulation, as we see below, without sacrificing the richness of the problem.

## *Problem statement.*

We consider the following offline coflow scheduling problem with release dates. There are $n$ coflows, indexed by $k = 1, 2, \ldots, n$. Coflow $k$ is released to the system at time $r_k$, $k = 1, 2, \ldots, n$. Let the matrix of flow sizes of coflow $k$ be denoted by $D^{(k)} = \left(d_{ij}^{(k)}\right)_{i,j=1}^m$, where $d_{ij}^{(k)}$ is the size of the flow to be transferred from input $i$ to output $j$, of coflow $k$. The completion time of coflow $k$, denoted by $C_k$, is the time when all flows from coflow $k$ have finished processing. Data units can be transferred across the network subject to the scheduling constraints described earlier. Let $y(i, j, k, t)$ be the number of data units being served in time slot $t$, which belong to coflow $k$ and which require transfer from input $i$ to output $j$. Then, in each time slot $t$, the following $2m$ *matching* constraints must be satisfied. For $i = 1, 2, \ldots, m$, $\sum_{k=1}^n \sum_{j'=1}^m y(i, j', k, t) \leq 1$, ensuring that input $i$ processes at most one data unit at a time, and similarly, $\sum_{k=1}^n \sum_{i'=1}^m y(i', j, k, t) \leq 1$, for each output $j = 1, 2, \ldots, m$.

For given positive weight parameters $w_k$, $k = 1, 2, \ldots, n$, we are interested in minimizing $\sum_{k=1}^m w_k C_k$, the total weighted completion time of coflows with release dates. In a data center, coflows often come from different applications, so the total weighted completion time of coflows is a reasonable user/application oriented performance objective. A larger weight indicates higher priority, i.e., the corresponding coflow needs to be completed more quickly.

To summarize, we can formulate our problem as the following mathematical program.

$$\text{(O)} \qquad \text{Minimize} \sum_{k=1}^n w_k C_k \qquad \text{subject to}$$

$$\sum_{t=1}^{C_k} y(i, j, k, t) \geq d_{ij}^{(k)}, \text{ for } i, j = 1, \ldots, m, k = 1, \ldots, n; \tag{1}$$

$$\sum_{k=1}^n \sum_{j'=1}^m y(i, j', k, t) \leq 1, \text{ for } i = 1, \ldots, m, \ \forall t; \tag{2}$$

$$\sum_{k=1}^n \sum_{i'=1}^m y(i', j, k, t) \leq 1, \text{ for } j = 1, \ldots, m, \ \forall t; \tag{3}$$

$$y(i, j, k, t) = 0 \text{ if } t < r_k, \text{ for } i, j = 1, \ldots, m, \ \forall t; \tag{4}$$

$$y(i, j, k, t) \text{ binary}, \ \forall i, j, t, k. \tag{5}$$

The load constraints (1) state that all processing requirements of flows need to be met upon the completion of each coflow. (2) and (3) are the matching constraints. The release date constraints (4) guarantee that coflows are being served only after they are released in the system. Note that this mathematical program is not an integer linear programming formulation because variables $C_k$ are in the limit of the summation.

The coflow scheduling problem (O) generalizes some well-known scheduling problems. First, when $m = 1$, it is easy to see that coflow scheduling is equivalent to single-machine scheduling with release dates with the objective of minimizing the total weighted completion time, where preemption is allowed. The latter problem is strongly NP-hard [22], which immediately implies the NP-hardness of problem (O) in general. When all coflow matrices are diagonal, coflow scheduling is equivalent to a concurrent open shop scheduling problem [3, 30]. This connection has been observed in [13], and is described in more detail in Appendix A for completeness. Utilizing this connection, it can be shown that the problem (O) is strongly NP-hard, even in the special case where $r_k = 0$ and $w_k = 1$ for all $k$. A major difference between concurrent open shop scheduling and coflow scheduling is that for concurrent open shop, there exists an optimal *permutation* schedule in which jobs can be processed in the same order on all machines [3], whereas permutation schedules need not be optimal for coflow scheduling [13].

## 1.2 Main Results

### *Theoretical results.*

Since the coflow scheduling problem (O) is NP-hard, we focus on finding approximation algorithms, that is, algorithms which run in polynomial time and returns a solution whose value is guaranteed to be close to optimal. Let $C_k(OPT)$ and $C_k(A)$ be the completion times of coflow $k$ under an optimal and an approximation scheduling algorithm respectively. Our main results are:

THEOREM 1. *There exists a deterministic polynomial time* 67/3*-approximation algorithm, i.e.*

$$\frac{\sum_{k=1}^{n} w_k C_k(A)}{\sum_{k=1}^{n} w_k C_k(OPT)} \le \frac{67}{3}.$$

THEOREM 2. *There exists a randomized polynomial time* $(9 + 16\sqrt{2}/3)$*-approximation algorithm, i.e.*

$$\frac{\mathbb{E}\left[\sum_{k=1}^{n} w_k C_k(A)\right]}{\sum_{k=1}^{n} w_k C_k(OPT)} \le 9 + \frac{16\sqrt{2}}{3}.$$

When all coflows have release dates 0, i.e., $r_k = 0$ for all $k$, we can improve the approximation ratios.

COROLLARY 1. *If all coflows are released into the system at time* 0*, then there exists a deterministic polynomial time* 64/3*-approximation algorithm.*

COROLLARY 2. *If all coflows are released into the system at time* 0*, then there exists a randomized polynomial time* $(8 + 16\sqrt{2}/3)$*-approximation algorithm.*

Our deterministic (described in Algorithm 2) and randomized algorithms combine ideas from combinatorial scheduling and matching theory along with some new insights. First, as with many other scheduling problems (see e.g. [17]), particularly for average completion time, we relax the problem formulation (O) to a polynomial-sized interval-indexed linear program (LP). The relaxation involves both dropping the matching constraints (2) and (3), and using intervals to make the LP polynomial sized. We then solve this LP, and use an optimal solution to the LP to obtain an ordered list of coflows. Then, we use this list to derive an actual schedule. To do so, we partition coflows into a polynomial number of groups, based on the minimum required completion times of the ordered coflows, and schedule the coflows in the same group as a single coflow using matchings obtained from an integer version of the Birkhoff-von Neumann decomposition theorem (Lemma 4 and Algorithm 1).

The analysis of the algorithm couples techniques from the two areas in interesting ways. We analyze the interval indexed linear program using tools similar to those used for other average completion time problems, especially concurrent open shop. The interval-indexed rather than time-indexed formulation is necessary to obtain a polynomial time algorithm, and we use a lower bound based on a priority-based load calculation (see Lemma 3). We also show how each coflow can be completed by a time bounded by a constant times the optimal completion time, via a clever grouping and decomposition of the coflow matrices. Here a challenge is to decompose the not necessarily polynomial length schedule into a polynomial number of matchings.

### *Experimental findings.*

We evaluate our algorithm as well as the impact of several additional algorithmic and heuristic decisions, including coflow ordering, coflow grouping and backfilling. Our evaluation uses a Hive/MapReduce trace collected from a large production cluster at Facebook [12, 13]. The main findings are as follows, with a more detailed discussion in Section 4 and Appendix D.

- Algorithms with coflow grouping consistently outperform those without grouping. Similarly, algorithms that use backfilling consistently outperform those that do not use backfilling.
- The performance of algorithms that use the LP-based ordering (15) is similar to those that order coflows according to their loads (see (18)). When combined with grouping and backfilling, these algorithms are nearly optimal. Note that the ordering of coflows according to load is used in [13].
- Our LP-based deterministic algorithm has near-optimal performance.

## 1.3 Related work

The coflow abstraction was first proposed in [11], although the idea was present in a previous paper [12]. Chowdhury et al. [12] observed that the optimal processing time of a coflow is exactly equal to its *load*, when the network is scheduling only a single coflow, and built upon this observation to schedule data transfer in a datacenter network. Chowdhury et al. [13] introduced the coflow scheduling problem without release dates, and provided effective scheduling heuristics. They also observed the connection of coflow scheduling with concurrent open shop, established the NP-hardness of the coflow scheduling problem, and showed via a simple counter-example how permutation schedules need not be optimal for coflow scheduling.

There is a great deal of success over the past 20 years on combinatorial scheduling to minimize average completion time, see e.g. [17, 27, 28, 33]. This line of works typically uses a linear programming relaxation to obtain an ordering of jobs and then uses that ordering in some other polynomial-time algorithm. There has also been much work on shop scheduling, which we do not survey here, but note that traditional shop scheduling is not "concurrent". In the language of our problem, that would mean that traditionally, two flows in the same coflow could *not* be processed simultaneously. The recently studied concurrent open shop problem removes this restriction and models flows that can be processed in parallel. There have been several results showing that even restrictive special cases are NP-hard [3, 10, 16, 34]. There were several algorithms with super-constant (actually at least $m$, the number of machines) approximation ratios, e.g., [3, 23, 34, 35]. Recently there have been several constant factor approximation algorithms using LP-relaxations. Wang and Cheng [35] used an interval-indexed formulation. Several authors have observed that a relaxation in completion time variables is possible [10, 16, 23], and Mastrolilli et al. [25] gave a primal-dual 2-approximation algorithm and showed stronger hardness results. Relaxations in completion time variables presume the optimality of permutation schedules, which does not hold for the coflow scheduling problem. Thus, our work builds on the formulation in Wang and Cheng [35], even though their approach does not yield the strongest approximation ratio.

The design of our scheduling algorithms relies crucially on a fundamental result (Lemma 4 in this paper) concerning the decomposition of nonnegative integer-valued matrices into permutation matrices, which states that such a matrix can be written as a sum of $\rho$ permutation matrices, where $\rho$ is the maximum column and row sum. This result is closely related to the classical Birkhoff-von Neumann theorem [7], and has been stated in different forms and applied in different contexts. For an application in scheduling theory, see e.g., [21]. For applications in communication networks, see e.g., [9, 24, 31, 26].

## 2. LINEAR PROGRAM (LP) RELAXATION

In this section, we present an interval-indexed linear program relaxation of the scheduling problem (O), which produces a lower bound on $\sum_{k=1}^{n} w_k C_k(OPT)$, the optimal value of the total weighted completion time, as well as an ordering of coflows for our approximation algorithms (§2.1). We then define and analyze the concepts of *maximum total input/output loads*, which respect the ordering produced by the LP, and relate these concepts to $C_k(OPT)$ (§2.2). These relations will be used in the proofs of our main results in §3.3.

### 2.1 Two Linear Program Relaxations

From the discussion in §1.1, we know that problem (O) is NP-hard. Furthermore, the formulation in (1) - (5) is not immediately of use, since it is at least as hard as an integer linear program. We can, however, formulate an *interval-indexed linear program* (LP) by relaxing the following components from the original formulation. (i) First, we develop new load constraints (see (8) and (9)), by relaxing the matching constraints (2) and (3) and the load constraints (1), and formulate a *time-indexed linear program.* (The matching constraints will be enforced in the actual scheduling algorithm.) The time-indexed LP has been used many times (e.g. [5, 17]) but typically for non-shop scheduling problems. Note that in order to use it for our problem, we drop the explicit matching constraints. We call this (LP-EXP) below. (ii) Second, in order to get a polynomial sized formulation, we divide time (which may not be polynomially bounded) into a set of geometrically increasing intervals. We call this an *interval-indexed integer program*, and it is also commonly used in combinatorial scheduling. In doing so, we have a weaker relaxation than the time-indexed one, but one that can be solved in polynomial time. We then relax the interval-indexed integer program to a linear program and solve the linear program.

To implement relaxation (i), let us examine the load and matching constraints (1) – (3). Constraints (2) and (3) imply that in each time slot $t$, each input/output can process at most one data unit. Thus, the total amount of work that can be processed by an input/output by time $t$ is at most $t$. For each time slot $t$ and each $k = 1, 2, \dots, n$, let $z_t^{(k)} \in \{0, 1\}$ be an indicator variable of the event that coflow $k$ completes in time slot $t$. Then

$$\sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{j'=1}^{m} d_{ij'}^{(k)} z_s^{(k)} \quad \text{and} \quad \sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{i'=1}^{m} d_{i'j}^{(k)} z_s^{(k)}$$

are, respectively, the total amount of work on input $i$ and output $j$, from all coflows that complete before time $t$. Therefore, for each $t$, we must have

$$\sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{j'=1}^{m} d_{ij'}^{(k)} z_s^{(k)} \leq t, \quad \text{for all } i = 1, 2, \dots, m, \quad (6)$$

$$\sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{i'=1}^{m} d_{i'j}^{(k)} z_s^{(k)} \leq t, \quad \text{for all } j = 1, 2, \dots, m, \quad (7)$$

which are the load constraints on the inputs and outputs.

To complete relaxation (i), we require an upper bound on the time needed to complete all coflows in an optimal scheduling algorithm. To this end, note that the naive algorithm which schedules one data unit in each time slot can complete processing all the coflows in $T = \max_k\{r_k\} + \sum_{k=1}^{n}\sum_{i,j=1}^{m} d_{ij}^{(k)}$ units of time, and it is clear that an optimal scheduling algorithm can finish processing all coflows by time $T$. Taking into account constraints (6) and (7), and relaxing the integer constraints $z_t^{(k)} \in \{0, 1\}$ into the corresponding linear constraints, we can formulate the following linear programming relaxation (LP-EXP) of the coflow scheduling problem (O).

$$\text{(LP-EXP)} \quad \text{Minimize } \sum_{k=1}^{n} w_k \sum_{t=1}^{T} t z_t^{(k)} \quad \text{subject to}$$

$$\sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{j'=1}^{m} d_{ij'}^{(k)} z_s^{(k)} \leq t, \text{ for } i = 1, \dots, m, \ t = 1, \dots, T; \quad (8)$$

$$\sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{i'=1}^{m} d_{i'j}^{(k)} z_s^{(k)} \leq t, \text{ for } j = 1, \dots, m, \ t = 1, \dots, T; \quad (9)$$

$$z_t^{(k)} = 0 \text{ if } r_k + \sum_{j'=1}^{m} d_{ij'}^{(k)} > t \text{ or } r_k + \sum_{i'=1}^{m} d_{i'j}^{(k)} > t; \quad (10)$$

$$\sum_{t=1}^{T} z_t^{(k)} = 1, \text{ for } k = 1, \dots, n;$$

$$z_t^{(k)} \geq 0, \text{ for } k = 1, \dots, n, \ t = 1, 2, \dots, T.$$

Since the time $T$ can be exponentially large in the sizes of the problem inputs, it is not *a priori* clear that the relaxation LP-EXP can be solved in polynomial time. In order to reduce the running time and find a polynomial time algorithm, we divide the time horizon into increasing time intervals: $[0, 1], (1, 2], (2, 4], \dots, (2^{L-2}, 2^{L-1}]$, where $L$ is chosen to be the smallest integer such that $2^{L-1} \geq T$. The inequality guarantees that $2^{L-1}$ is a sufficiently large time horizon to complete all the coflows, even under a naive schedule. We also define the following notation for time points: $\tau_0 = 0$, and $\tau_l = 2^{l-1}$, for $l = 1, ..., L$. Thus, the $l$th time interval runs from time $\tau_{l-1}$ to $\tau_l$.

For $k = 1, \dots, n$ and $l = 1, \dots, L$, let $x_l^{(k)}$ be the binary decision variable which indicates whether coflow $k$ is scheduled to complete within the interval $(\tau_{l-1}, \tau_l]$. We approximate the completion time variable $C_k$ by $\sum_{l=1}^{L} \tau_{l-1} x_l^{(k)}$, the left end point of the time interval in which coflow $k$ finishes, and consider the following linear program relaxation (LP).

$$(LP) \qquad \text{Minimize } \sum_{k=1}^{n} w_k \sum_{l=1}^{L} \tau_{l-1} x_l^{(k)} \qquad \text{subject to}$$

$$\sum_{u=1}^{l} \sum_{k=1}^{n} \sum_{j'=1}^{m} d_{ij'}^{(k)} x_u^{(k)} \le \tau_l, \text{ for } i = 1, \dots, m, \ l = 1, \dots, L; \quad (11)$$

$$\sum_{u=1}^{l} \sum_{k=1}^{n} \sum_{i'=1}^{m} d_{i'j}^{(k)} x_u^{(k)} \le \tau_l, \text{ for } j = 1, \dots, m, \ l = 1, \dots, L; \quad (12)$$

$$x_l^{(k)} = 0 \text{ if } r_k + \sum_{j'=1}^{m} d_{ij'}^{(k)} > \tau_l \text{ or } r_k + \sum_{i'=1}^{m} d_{i'j}^{(k)} > \tau_l; \quad (13)$$

$$\sum_{l=1}^{L} x_l^{(k)} = 1, \text{ for } k = 1, \dots, n;$$

$$x_l^{(k)} \ge 0, \text{ for } k = 1, \dots, n, \ l = 1, \dots, L.$$

The relaxations (LP-EXP) and (LP) are similar, except that the time-indexed variables $z_t^{(k)}$ are replaced by interval-index variables $x_l^{(k)}$. The following lemma is immediate.

Lemma 1. *The optimal value of the linear program (LP) is a lower bound on the optimal total weighted completion time $\sum_{k=1}^{n} w_k C_k(OPT)$ of coflow scheduling problem (O).*

Proof. Consider an optimal schedule of problem (O) and set $x_u^{(k)} = 1$ if coflow $k$ completes within the $u$th time interval. This is a feasible solution to problem (LP) with the load and capacity constraints (11) and (12) and the feasibility constraint (13) all satisfied. Moreover, since coflow $k$ completes within the $u$th interval, the coflow completion time is at least $\tau_{l-1}$. Hence, the objective value of the feasible solution constructed is no more than the optimal total weighted completion time $\sum_{k=1}^{n} w_k C_k(OPT)$. $\square$

Since the constraint matrix in problem (LP) is of size $O((n+m)\log T)$ by $O(n \log T)$ and the maximum size of the coefficients is $O(\log T)$, the number of bits of input to the problem is $O(n(m+n)(\log T)^3)$. The interior point method can solve problem (LP) in polynomial time [20].

From an optimal solution to (LP), we can obtain an ordering of coflows, and use this order in our scheduling algorithms (see e.g., Algorithm 2). To do so, let an optimal solution to problem (LP) be $\bar{x}_l^{(k)}$ for $k = 1, \dots, n$ and $l = 1, \dots, L$. The relaxed problem (LP) computes an approximated completion time

$$\bar{C}_k = \sum_{l=1}^{L} \tau_{l-1} \bar{x}_l^{(k)} \quad (14)$$

for coflow $k$, $k = 1, 2, \dots, n$, based on which we reorder coflows. More specifically, we re-order and index the coflows in a nondecreasing order of the approximated completion times $\bar{C}_k$, i.e.,

$$\bar{C}_1 \le \bar{C}_2 \le \dots \le \bar{C}_n. \quad (15)$$

For the rest of the paper, we will stick to this ordering and indexing of the coflows.

Wang and Cheng [35] gave a 16/3-approximation algorithm for the concurrent open shop problem using a similar interval-indexed linear program. Our algorithms are more involved because we also have to address the matching constraints, which do not appear in the concurrent open shop problem.

## 2.2 Maximum Total Input / Output Loads

Here we define the *maximum total input/output loads*, respecting the ordering and indexing of (15). For each $k$, $k = 1, 2, \dots n$, define the *maximum total input load* $I_k$, the *maximum total output load* $J_k$ and the *maximum total load* $V_k$ by

$$I_k = \max_{i=1,\dots,m} \left\{ \sum_{j'=1}^{m} \sum_{g=1}^{k} d_{ij'}^{(g)} \right\}, \ J_k = \max_{j=1,\dots,m} \left\{ \sum_{i'=1}^{m} \sum_{g=1}^{k} d_{i'j}^{(g)} \right\}$$

$$\text{and} \qquad V_k = \max\{I_k, J_k\} \quad (16)$$

respectively. For each $i$ (each $j$, respectively), $\sum_{j'=1}^{m} \sum_{g=1}^{k} d_{ij'}^{(g)}$ $\left(\sum_{i'=1}^{m} \sum_{g=1}^{k} d_{i'j}^{(g)}, \text{ respectively}\right)$ is the total processing requirement on input $i$ (output $j$) from coflows $1, 2, \dots, k$. That is, the *total load* is the sum of the loads of the lower numbered coflows. By the load constraints, $V_k$ is a universal lower bound on the time required to finish processing coflows $1, 2, \dots, k$, under any scheduling algorithm. We state this fact formally as a lemma.

Lemma 2. *For $k = 1, 2, \dots, n$, let $\tilde{C}^{(k)}$ be the time that all coflows $1, \dots, k$ complete, where the indexing respects the order* (15). *Then, under any scheduling algorithm,*

$$V_k \le \tilde{C}^{(k)} \quad (17)$$

*for all $k$ simultaneously.*

The following lemma, which states that with a proper ordering of the coflows, $V_k$ is a 16/3-approximation of the optimal $C_k(OPT)$ for all $k$ simultaneously, is crucial for the proof of our main results in the next section. The proof of the lemma is similar to that of Theorem 1 in [35]. We defer this proof to Appendix C.

Lemma 3. *Let $\bar{C}_k$ be computed from problem (LP) by Eq. (14) and be indexed such that (15) is satisfied. Then $V_k \le (16/3)C_k(OPT)$, $k = 1, \dots, n$.*

## 3. APPROXIMATION ALGORITHMS

In this section, we describe a deterministic and a randomized polynomial time scheduling algorithm, with approximation ratios of 67/3 and $9 + 16\sqrt{2}/3$ respectively, in the presence of arbitrary release dates. The same algorithms have approximation ratios 64/3 and $8+16\sqrt{2}/3$, respectively, when all coflows are released at time 0. Both algorithms are based on the idea of efficiently scheduling coflows according to the ordering (15) produced by (LP). To fully describe the scheduling algorithms, we first present some preliminaries on the minimal amount of time to finish processing an arbitrary coflow using only matching schedules (Algorithm 1 and §3.1). Algorithm 1 will be used crucially in the design of the approximation algorithms, and, as we will see, it is effectively an integer version of the famous Birkhoff-von Neumann theorem [7], and hence the name *Birkhoff-von Neumann decomposition.* Details of the main algorithms are provided in §3.2, and we supply proofs of the complexities and approximation ratios in §3.3.

## 3.1 Birkhoff-von Neumann Decomposition

For an arbitrary coflow matrix $D = (d_{ij})_{i,j=1}^m$, where $d_{ij} \in \mathbb{Z}_+$ for all $i$ and $j$, we define $\rho(D)$, the *load* of coflow $D$, as follows.

$$\rho(D) = \max\left\{ \max_{i=1,\ldots,m} \left\{ \sum_{j'=1}^m d_{ij'} \right\}, \max_{j=1,\ldots,m} \left\{ \sum_{i'=1}^m d_{i'j} \right\} \right\}. \tag{18}$$

Note that for each $i$, $\sum_{j'=1}^m d_{ij'}$ is the total processing requirement of coflow $D$ on input $i$, and for each $j$, $\sum_{i'=1}^m d_{i'j}$ is that on output $j$. By the matching constraints, $\rho(D)$ is a universal lower bound on the completion time of coflow $D$, were it to be scheduled alone.

Lemma 4. *There exists a polynomial time algorithm which finishes processing coflow $D$ in $\rho(D)$ time slots (using the matching schedules), were it to be scheduled alone.*

Algorithm 1 describes a polynomial-time scheduling algorithm that can be used to prove Lemma 4. The idea of the algorithm is as follows. For a given coflow matrix $D = (d_{ij})_{i,j=1}^m$, we first augment it to a "larger" matrix $\tilde{D}$, whose row and column sums are all equal to $\rho(D)$ (Step 1). In each iteration of Step 1, we increase one entry of $D$ such that at least one more row or column sums to $\rho$. Therefore, at most $2m-1$ iterations are required to get $\tilde{D}$. We then decompose $\tilde{D}$ into *permutation matrices* that correspond to the matching schedules (Step 2). More specifically, at the end of Step 2, we can write $\tilde{D} = \sum_{u=1}^U q_u \Pi_u$, so that $\Pi_u$ are permutation matrices, $q_u \in \mathbb{N}$ are such that $\sum_{u=1}^U q_u = \rho(D)$, and $U \leq m^2$. The key to this decomposition is Step 2 (ii), where the existence of a perfect matching $M$ can be proved by a simple application of Hall's matching theorem [18]. We now consider the complexity of Step 2. In each iteration of Step 2, we can set at least one entry of $\tilde{D}$ to zero, so that the algorithm ends in $m^2$ iterations. The complexity of finding a perfect matching in Step 2 (ii) is $O(m^3)$, e.g., by using the maximum bipartite matching algorithm. Since both Steps 1 and 2 of Algorithm 1 has polynomial-time complexity, the algorithm itself has polynomial-time complexity.

If we divide both sides of the identity $\tilde{D} = \sum_{u=1}^U q_u \Pi_u$ by $\rho(D)$, then $\tilde{D}/\rho(D)$ is a doubly stochastic matrix, and the coefficients $q_u/\rho(D)$ sum up to 1. Therefore, $\tilde{D}/\rho(D)$ is a convex combination of the permutation matrices $\Pi_u$. Because of this natural connection of Algorithm 1 with the Birkhoff-von Neumann theorem [7], we call the algorithm the Birkhoff-von Neumann decomposition. Lemma 4 has been stated in slightly different forms, see e.g., Theorem 1 in [21], Theorem 4.3 in [31], and Fact 2 in [26].

**Algorithm 1**: Birkhoff-von Neumann Decomposition

```
Data: A single coflow $D = (d_{ij})_{i,j=1}^m$.
Result: A scheduling algorithm that uses at most a
        polynomial number of different matchings.
• Step 1: Augment $D$ to a matrix $\tilde{D} = \left(\tilde{d}_{ij}\right)_{i,j=1}^m$, where
  $\tilde{d}_{ij} \geq d_{ij}$ for all $i$ and $j$, and all row and column sums of
  $\tilde{D}$ are equal to $\rho(D)$.
  Let $\eta = \min\left\{\min_i \left\{\sum_{j'=1}^m d_{ij'}\right\}, \min_j \left\{\sum_{i'=1}^m d_{i'j}\right\}\right\}$ be
  the minimum of row sums and column sums, and let
  $\rho(D)$ be defined according to Equation (18).
  $\tilde{D} \leftarrow D$.
  while ($\eta < \rho$) do
      $i^* \leftarrow \arg\min_i \sum_{j'=1}^m \tilde{D}_{ij'}$; $j^* \leftarrow \arg\min_j \sum_{i'=1}^m \tilde{D}_{i'j}$.
                      $\tilde{D} \leftarrow \tilde{D} + pE,$
      where $p = \min\{\rho - \sum_{j'=1}^m \tilde{D}_{i^*j'}, \rho - \sum_{i'=1}^m \tilde{D}_{i'j^*}\}$,
      $E_{ij} = 1$ if $i = i^*$ and $j = j^*$, and $E_{ij} = 0$ otherwise.
      $\eta \leftarrow \min\left\{\min_i \left\{\sum_{j'=1}^m \tilde{D}_{ij'}\right\}, \min_j \left\{\sum_{i'=1}^m \tilde{D}_{i'j}\right\}\right\}$
  end
• Step 2: Decompose $\tilde{D}$ into permutation matrices $\Pi$.
  while ($\tilde{D} \neq 0$) do
  (i) Define an $m \times m$ binary matrix $G$ where $G_{ij} = 1$ if
      $\tilde{D}_{ij} > 0$, and $G_{ij} = 0$ otherwise, for $i, j = 1, \ldots, m$.
  (ii) Interpret $G$ as a bipartite graph, where an
      (undirected) edge $(i, j)$ is present if and only if
      $G_{ij} = 1$. Find a perfect matching $M$ on G and
      define an $m \times m$ binary matrix $\Pi$ for the matching
      by $\Pi_{ij} = 1$ if $(i, j) \in M$, and $\Pi_{ij} = 0$ otherwise, for
      $i, j = 1, \ldots, m$.
  (iii) $\tilde{D} \leftarrow \tilde{D} - q\Pi$, where $q = \min\{\tilde{D}_{ij} : \Pi_{ij} > 0\}$.
      Process coflow $D$ using the matching $M$ for $q$ time
      slots. More specifically, process dataflow from input
      $i$ to output $j$ for $q$ time slots, if $(i, j) \in M$ and there
      is processing requirement remaining, for
      $i, j = 1, \ldots, m$.
  end
```

## 3.2 Approximation Algorithms

Here we present our deterministic and randomized scheduling algorithms. The deterministic algorithm is summarized in Algorithm 2, which consists of 2 steps. In Step 1, we solve (LP) to get the approximated completion time $\bar{C}_k$ for coflow ordering. Then, in Step 2, for each $k = 1, 2, \ldots, n$, we compute the maximum total load $V_k$ of coflow $k$, and identify the time interval $(\tau_{r(k)-1}, \tau_{r(k)}]$ that it belongs to, where $\tau_l$ are defined in §2.1. All the coflows that fall into the same time interval are combined into and treated as a single coflow, and processed using Algorithm 1 in §3.1.

The randomized scheduling algorithm follows the same steps as the deterministic one, except for the choice of the time intervals in Step 2 of Algorithm 2. Define the *random* time points $\tau'_l$ by $\tau'_0 = 0$, and $\tau'_l = T_0 a^{l-1}$, where $a = 1+\sqrt{2}$, and $T_0 \sim Unif[1, a]$ is uniformly distributed between 1 and $a$. Having picked the random time points $\tau'_l$, we then proceed to process the coflows in a similar fashion to the deterministic algorithm. Namely, for each $k = 1, 2, \ldots, n$, we identify the (random) time interval $(\tau'_{r'(k)-1}, \tau'_{r'(k)}]$ that coflow $k$ belongs to, and for all coflows that belong to the same time interval, they are combined into a single coflow and processed using Algorithm 1.

## 3.3 Proofs of Main Results

We now establish the complexity and performance properties of our algorithms. We first provide the proof of Theorem 1 in detail. By a slight modification of the proof of Theorem 1, we can establish Corollary 1. We then provide the proofs of Theorem 2 and Corollary 2.

```
Algorithm 2: Deterministic LP-based Approximation
Data: Coflows $\left(d_{ij}^{(k)}\right)_{i,j=1}^{m}$, for $k = 1, \ldots, n$.
Result: A scheduling algorithm that uses at most a
        polynomial number of different matchings.
  • Step 1: Given $n$ coflows, solve the linear program (LP).
    Let an optimal solution be given by $\bar{x}_l^{(k)}$, for
    $l = 1, 2, \ldots, L$ and $k = 1, 2, \ldots, n$. Compute the
    approximated completion time $\bar{C}_k$ by Eq. (14). Order
    and index the coflows according to (15).
  • Step 2: Compute the maximum total load $V_k$ for each $k$
    by (16). Suppose that $V_k \in (\tau_{r(k)-1}, \tau_{r(k)}]$ for some
    function $r(\cdot)$ of $k$. Let the range of function $r(\cdot)$ consist
    of values $s_1 < s_2 < \ldots < s_P$, and define the sets
    $S_u = \{k : \tau_{s_u - 1} < V_k \leq \tau_{s_u}\}$, $u = 1, 2, \ldots, P$.
    $u \leftarrow 1$.
    while $u \leq P$ do
      After all the coflows in set $S_u$ are released, schedule
      them as a single coflow with transfer requirement
      $\sum_{k \in S_u} d_{ij}^{(k)}$ from input $i$ to output $j$ and finish
      processing the coflow using Algorithm 1.
      $u \leftarrow u + 1$;
    end
```

## *Proofs of Theorem 1 and Corollary 1.*

Let $C_k(A)$ be the completion time of coflow $k$ under the deterministic scheduling algorithm (Algorithm 2). The following proposition will be used to prove Theorem 1.

PROPOSITION 1. *For all $k = 1, 2, \ldots, n$, the coflow completion time $C_k(A)$ satisfies*

$$C_k(A) \leq \max_{1 \leq g \leq k} \{r_g\} + 4V_k, \tag{19}$$

*where we recall that $r_g$ is the release time of coflow $g$, and the total loads $V_k$ are defined in Eq. (16).*

PROOF. Recall the notation used in Algorithm 2. For any coflow $k \in S_u$, $V_k \leq \tau_{s_u}$. By Lemma 4, we know that all coflows in the set $S_u$ can be finished processing within $\tau_{s_u}$ units of time. Define $\bar{\tau}_0 = 0$ and $\bar{\tau}_u = \bar{\tau}_{u-1} + \tau_{s_u}$, $u = 1, 2, \ldots, P$. A simple induction argument shows that under Algorithm 2, $C_k(A)$, the completion time of coflow $k$, satisfies $C_k(A) \leq \max_{1 \leq g \leq k}\{r_g\} + \bar{\tau}_u$, if $k \in S_u$.

We now prove by induction on $u$ that $\bar{\tau}_u \leq 2\tau_{r(k)}$, if $k \in S_u$, $u = 1, 2, \ldots, P$. Suppose that this holds for $k \in S_u$. Let $k^* = \max\{k : k \in S_u\}$ such that $k^* + 1 \in S_{u+1}$. Then,

$$\bar{\tau}_{u+1} = \bar{\tau}_u + \tau_{s_{u+1}} \leq 2\tau_{r(k^*)} + \tau_{r(k^*+1)}.$$

Since the time interval increases geometrically and satisfies $\tau_{l+1} = 2\tau_l$ for $l = 1, 2, \ldots, L$, $\bar{\tau}_{u+1} \leq 2\tau_{r(k^*+1)} = 2\tau_{r(k)}$, if $k \in S_{u+1}$. This completes the induction. Furthermore, if $\tau_{r(k)-1} < V_k \leq \tau_{r(k)}$, $\tau_{r(k)} = 2\tau_{r(k-1)} < 2V_k$. Thus,

$$C_k(A) \leq \max_{1 \leq g \leq k}\{r_g\} + 2\tau_{r(k)} \leq \max_{1 \leq g \leq k}\{r_g\} + 4V_k.$$

□

The proof of Theorem 1 is now a simple consequence of Lemmas 1 and 3 and Proposition 1.

*Proof of Theorem 1.* For all $k$, $\max_{1 \leq g \leq k}\{r_g\} \leq \bar{C}_k$, where $\bar{C}_k$ (cf. (15)) are the approximated completion times computed from (LP), follows immediately from the feasibility constraints (13), and the ordering of $\bar{C}_k$ in (15). By Proposition 1 and Lemma 3, we have

$$C_k(A) \leq \max_{1 \leq g \leq k}\{r_g\} + 4V_k \leq \bar{C}_k + \frac{64}{3} C_k(OPT).$$

It follows that

$$\begin{aligned}
\sum_{k=1}^{n} w_k C_k(A) &\leq \sum_{k=1}^{n} w_k \left(\bar{C}_k + \frac{64}{3} C_k(OPT)\right) \\
&\leq \sum_{k=1}^{n} w_k C_k(OPT) + \frac{64}{3} \sum_{k=1}^{n} w_k C_k(OPT) \\
&= \frac{67}{3} \sum_{k=1}^{n} w_k C_k(OPT),
\end{aligned}$$

where the second inequality follows from Lemma 1.

We now consider the running time of Algorithm 2. The program (LP) in Step 1 can be solved in polynomial time, as discussed in §2.1. Thus, it suffices to show that Step 2 runs in polynomial time. Since there are only $O(\log T)$, a polynomial number of intervals of the form $(\tau_{l-1}, \tau_l]$, it now suffices to show that for each $u = 1, 2, \ldots, P$, Algorithm 1 completes processing all coflows in the set $S_u$ in polynomial time, where we recall the definition of $S_u$ in Step 2 of Algorithm 2. But this follows from Lemma 4. Thus, Algorithm 2 runs in polynomial time. □

*Proof of Corollary 1.* Consider Algorithm 2 and suppose that all coflows are released at time 0, i.e., $r_k = 0$ for all $k$. By Proposition 1, $C_k(A) \leq 4V_k$ for all $k = 1, 2, \ldots, n$. By inspection of the proof of Theorem 1, we have,

$$\sum_{k=1}^{n} w_k C_k(A) \leq \frac{64}{3} \sum_{k=1}^{n} w_k C_k(OPT).$$

The fact that Algorithm 2 has a polynomial running time was established in the proof of Theorem 1. □

Before proceeding to the proofs of Theorem 2 and Corollary 2, let us provide some remarks on the upper bounds (19) in Proposition 1. Recall that Ineq. (19) hold simultaneously for all $k$. Then, a natural question is whether the upper bounds in (19) are tight. We leave this question as future work, but provide the following observation for now. Suppose that all coflows are released at time 0, then the upper bounds in (19) become $C_k(A) \leq 4V_k$ for all $k$. By inspecting the proof of Proposition 1, it is easy to see that in fact, $\tilde{C}^{(k)}(A) \leq 4V_k$ for all $k$, where $\tilde{C}^{(k)}(A)$ is the completion time of coflows $1, 2, \ldots, k$ under our algorithm. Compared this with the lower bounds (17) in Lemma 2, we see that the upper bounds are off by a factor of at most 4. The lower bounds (17) cannot be achieved simultaneously for all $k$; this fact can be demonstrated through a simple counter-example. See Appendix B for details.

## *Proofs of Theorem 2 and Corollary 2.*

Similar to the proof of Theorem 1, the proof of Theorem 2 relies on the following proposition, the randomized counterpart of Proposition 1.

PROPOSITION 2. *Let $C_k(A')$ be the* random *completion time of coflow $k$ under the randomized scheduling algorithm* *described in §3.2. Then, for all $k = 1, 2, \ldots, n$,*

$$\mathbb{E}[C_k(A')] \le \max_{1\le g\le k}\{r_g\} + \left(\frac{3}{2} + \sqrt{2}\right) V_k. \qquad (20)$$

PROOF. Recall the random time points $\tau'_l$ defined by $\tau'_0 = 0$, and $\tau'_l = T_0 a^{l-1}$, where $a = 1 + \sqrt{2}$, and $T_0 \sim Unif[1, a]$. Suppose that $V_k \in (\tau'_{r(k)-1}, \tau'_{r(k)}]$ and let $T_k = \tau'_{r(k)} - \tau'_{r(k)-1}$ for all $k$. Then,

$$\frac{\tau'_{r(k)}}{T_k} = \frac{T_0 a^{\tau'_{r(k)-1}}}{T_0 a^{\tau'_{r(k)-1}} - T_0 a^{\tau'_{r(k)-2}}} = \frac{a}{a-1}.$$

Since $T_0 \sim Unif[1, a]$, $T_k$ is uniformly distributed on the interval $((a-1)V_k/a, (a-1)\,V_k)$. Thus,

$$\begin{aligned}\mathbb{E}\left[\tau'_{r(k)}\right] &= \frac{a}{a-1}\mathbb{E}[T_k] \\ &= \frac{a}{a-1} \times \frac{1}{2}\left(\frac{a-1}{a} + a - 1\right) V_k = \frac{1+a}{2} V_k.\end{aligned}$$

Similar to the $\bar{\tau}_u$ used in the proof of Proposition 1, define $\bar{\tau}'_u$ inductively by $\bar{\tau}'_0 = 0$ and $\bar{\tau}'_u = \bar{\tau}'_{u-1} + \tau'_{s_u}$, $u = 1, 2, \ldots P$. If $k \in S_u$, then

$$\bar{\tau}'_u \le \sum_{l=1}^{r(k)} \tau'_l = \tau'_{r(k)} + \frac{\tau'_{r(k)}}{a} + \frac{\tau'_{r(k)}}{a^2} + \ldots + T_0 \le \frac{a}{a-1}\tau'_{r(k)}.$$

Similar to the proof of Proposition 1, we can establish that if $k \in S_u$, then $C_k(A') \le \max_{1\le g\le k}\{r_g\} + \bar{\tau}'_u$. Thus, with $a = 1 + \sqrt{2}$,

$$\begin{aligned}\mathbb{E}[C_k(A')] &\le \mathbb{E}\left[\max_{1\le g\le k}\{r_g\} + \bar{\tau}'_u\right] \\ &\le \max_{1\le g\le k}\{r_g\} + \frac{a}{a-1}\mathbb{E}[\tau'_{r(k)}] \\ &\le \max_{1\le g\le k}\{r_g\} + \frac{a^2+a}{2(a-1)} V_k \\ &= \max_{1\le g\le k}\{r_g\} + \left(\frac{3}{2} + \sqrt{2}\right) V_k.\end{aligned}$$

□

*Proof of Theorem 2.* By Proposition 2 and Lemma 3, we have $\mathbb{E}[C_k(A')] \le \max_{1\le g\le k}\{r_g\} + \left(3/2 + \sqrt{2}\right) V_k < \bar{C}_k + \left(8 + 16\sqrt{2}/3\right) C_k(OPT)$. It follows that

$$\begin{aligned}\mathbb{E}\left[\sum_{k=1}^{n} w_k C_k(A')\right] &\le \sum_{k=1}^{n} w_k \left(\bar{C}_k + \left(8 + \frac{16\sqrt{2}}{3}\right) C_k(OPT)\right) \\ &\le \sum_{k=1}^{n} w_k C_k(OPT) \\ &\quad + \left(8 + \frac{16\sqrt{2}}{3}\right) \sum_{k=1}^{n} w_k C_k(OPT) \\ &= \left(9 + \frac{16\sqrt{2}}{3}\right) \sum_{k=1}^{n} w_k C_k(OPT),\end{aligned}$$

where the second inequality follows from Lemma 1. □

*Proof of Corollary 2.* Consider the randomized algorithm and suppose that all coflows are released at time 0, i.e., $r_k = 0$ for all $k$. By Proposition 2, for all $k = 1, 2, \ldots, n$, $\mathbb{E}[C_k(A')] \le \left(3/2 + \sqrt{2}\right) V_k$. By inspection of the proof of Theorem 2, we have,

$$\mathbb{E}\left[\sum_{k=1}^{n} w_k C_k(A')\right] \le \left(8 + \frac{16\sqrt{2}}{3}\right) \sum_{k=1}^{n} w_k C_k(OPT).$$

□

## 4. EXPERIMENTS

In previous sections, we presented deterministic and randomized approximation algorithms with provable performance guarantees. In this section, we conduct some preliminary experiments to evaluate the practical performance of several algorithms, including our deterministic algorithm described in §3.2.

At a high level, both of our algorithms consist of two related stages. The *ordering stage* computes an ordering of coflows, and the *scheduling stage* produces a sequence of feasible schedules that respects this ordering. It is intuitively clear that an intelligent ordering of coflows in the ordering stage can substantially reduce coflow completion times. As a result, we consider three different coflow orderings, including the LP-based ordering (15), and study how they affect algorithm performance. See §4.1 for more details.

The derivation of the actual sequence of schedules in the scheduling stage relies on two key ideas: scheduling according to an optimal (Birkhoff-von Neumann) decomposition, and a suitable grouping of the coflows. We note that to some extent, grouping can be thought of as a *dovetailing* procedure, where skewed coflow matrices are consolidated to form more uniform ones, which can be efficiently cleared by matching schedules. It is then reasonable to expect that grouping can improve performance, so we compare algorithms with grouping and those without grouping to understand its effect. The particular grouping procedure that we consider here is the one described in Algorithm 2.

Backfilling is a common strategy used in scheduling for computer systems to increase utilization of system resources (see, e.g. [13]). Therefore, we will also investigate the performance gain of using backfilling in the scheduling stage. We focus on one natural backfilling technique, described in detailed in §4.1.

In summary, we will evaluate the performance impact of coflow ordering, coflow grouping and backfilling. Our evaluation uses a Hive/MapReduce trace collected from a large production cluster at Facebook [12, 13]. The main findings are as follows.

- Algorithms with coflow grouping consistently outperform those without grouping. Similarly, algorithms that use backfilling consistently outperform those that do not use backfilling.
- The performance of algorithms that use the LP-based ordering (15) is similar to those that order coflows according to their loads (see (18)). When combined with grouping and backfilling, these algorithms are nearly optimal. Furthermore, our LP-based deterministic algorithm has near-optimal performance.

### 4.1 Methodology

*Workload.*

We use the same workload as described in [13] The workload is based on a Hive/MapReduce trace at Facebook that was collected on a 3000-machine cluster with 150 racks, so the datacenter in the experiments can be modeled as $150 \times 150$ network switch (and each coflow represented by a $150 \times 150$ matrix). The cluster has a 10:1 core-to-rack oversubscription ratio with a total bisection bandwidth of 300Gbps. Therefore, each ingress/egress port has a capacity of 1Gbps, or equivalently 128MBps. We select the time unit to be 1/128 second accordingly so that each port has the capacity of 1MB per time unit. We filter the coflows based on the number of non-zero flows, which we denote by $M'$, and we consider three collections of coflows, filtered by the conditions $M' \geq 50$, $M' \geq 40$ and $M' \geq 30$, respectively. As pointed out in [13], coflow scheduling algorithms may be ineffective for very sparse coflows in real datacenters, due to communication overhead, so we investigate the performance of our algorithms for these three collections. We also assume that all coflows arrive at time 0, so we do not consider the effect of release dates.

*Algorithms and Metrics.*

We consider 12 different scheduling algorithms, which are specified by the ordering used in the ordering stage, and the actual sequence of schedules used in the scheduling stage. We consider three different orderings, which we describe in detail below, and the following 4 cases in the scheduling stage: (a) without grouping or backfilling, which we refer to as the base case, (b) without grouping but with backfilling, (c) with grouping and without backfilling, and (d) with both grouping and backfilling. Algorithm 2 corresponds to the combination of LP-based ordering and case (c).

For ordering, three different possibilities are considered. We use $H_A$ to denote the naive ordering of coflows by coflow IDs from the production trace, $H_\rho$ to denote the ordering of coflows by the ratios between the maximum load $\rho$ (defined in (18)) and weight $w$, which is also considered in [13], and $H_{LP}$ to denote the LP-based coflow ordering given in (15).

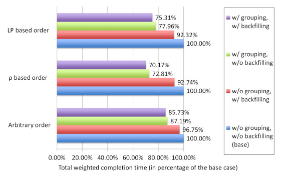
Given an ordering of the coflows, it is possible to partition them into groups using the procedure described in Step 2 of Algorithm 2. As discussed, we will consider algorithms with and without such grouping. When we do group the coflows, we treat all coflows within a group as a whole and say that they are *consolidated* into an *aggregated* coflow. Scheduling of coflows within a group makes use of the Birkhoff-von Neumann decomposition described in Algorithm 1, respecting the coflow order. Namely, if two data units from coflows $k$ and $k'$ within the same group use the same pair of input and output, and $k$ is ordered before $k'$, then we always process the data unit from coflow $k$ first.

For backfilling, given a sequence of coflows with a given order, for coflow $k$ with a coflow matrix $D^{(k)}$, $k = 1, \dots, n$, we compute the augmented coflow matrix $\tilde{D}^{(k)}$ and schedule the coflow by the Birkhoff-von Neumann decomposition in Algorithm 1. This decomposition may introduce unforced idle time, whenever $D^{(k)} \neq \tilde{D}^{(k)}$. When we use a schedule that matches input $i$ to output $j$ to serve coflow $k$ with $D_{ij}^{(k)} < \tilde{D}_{ij}^{(k)}$, and if there is no more service requirement on the pair of input $i$ and output $j$ for coflow $k$, we backfill in order from the flows on the same pair of ports in the subsequent coflows. When grouping is used, backfilling is applied to the aggregated coflows.

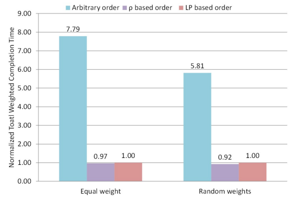
We compare the performances of different algorithms, by considering ratios of the total weighted completion times. For the choice of coflow weights, we consider both the case of uniform weights, as well as the case where weights are given by a random permutation of the set $\{1, 2, \dots, n\}$.

## 4.2 Performance

We compute the total weighted completion times for all 3 orders in the 4 different cases (a) – (d) described in §4.1, through a set of experiments on filtered coflow data. The complete results can be found in Appendix D, and we present representative comparisons of the algorithms here.



(a) Comparison of total weighted completion times with respect to the base case for each order. Data are filtered by $M' \geq 50$. Weights are random.



(b) Comparison of total weighted completion times evaluated on data filtered by $M' \geq 50$ in case (d) with both grouping and backfilling.

Figure 2: Comparisons of total weighted completion times

Figure 2a, plots the total weighted completion times as percentages of the base case (a), for the case of random weights. Grouping and backfilling both improve the total weighted completion time with respect to the base case for all 3 orders. The reduction in the total weighted completion time from grouping is up to 27.19%, and is consistently higher than the reduction from backfilling, which is up to 8.68%. For all 3 orders, scheduling with both grouping and backfilling (i.e., case (d)) gives the smallest total weighted completion time, but the improvement from case (c) (only grouping) to case (d) is marginal.

We then compare the performances of different coflow orderings. Figure 2b shows the comparison of total weighted completion times evaluated on filtered coflow data for both equal weight and random weights in case (d) where the scheduling stage uses both grouping and backfilling. Compared with $H_A$, both $H_\rho$ and $H_{LP}$ reduce the total weighted completion times of coflows by a ratio up to 8.05 and 7.79, respectively, with $H_\rho$ performing consistently better than $H_{LP}$. A natural question to ask is how close $H_\rho$ and $H_{LP}$ are to the optimal. In order to get a tight lower bound of the scheduling problem (O), we solve (LP-EXP) for the case of random weights and when the number of non-zero flows $M' \geq 50$. The ratio of the lower bound over the weighted completion time under $H_{LP}$ is 0.9447, which implies that both $H_\rho$ and $H_{LP}$ in the ordering stage provide a good approximation of the optimal in practice. Due to time constraint, we did not compute the lower bounds for all cases, because (LP-EXP) is exponential in the size of the input and is extremely time consuming to solve.

Our experiments are only preliminary and leave open many questions. For example, we should systematically measure the benefit of the time-indexed versus the interval-indexed linear program. We should also compare the performance of the randomized algorithm, include varying release dates, and think of other heuristics to improve the solutions obtained. We leave this work, and testing on other data sets, to the full paper.

## 5. CONCLUSION AND OPEN PROBLEMS

We have given the first $O(1)$-approximation algorithms for minimizing the total weighted completion time of coflows in a datacenter network, in the presence of release dates, and have performed preliminary experiments to evaluate the algorithm and several additional heuristics. Beyond the obvious question of improving the approximation ratio, this work opens up several additional interesting directions in coflow scheduling, such as the consideration of other metrics and the addition of other realistic constraints, such as precedence constraints. We are particularly interested in minimizing weighted coflow *processing* time (usually called *flow time* in the literature), which is harder to approximate (the various hardness results from single machine schedule will clearly carry over), but may lead to the development and understanding of better algorithms, possibly by considering resource augmentation.

Perhaps the most interesting questions involve making the algorithms and models more practical for them to work in real-time in a real system. While we consider release dates, our algorithms are not *on-line*, as they require the solution of an LP to compute a global ordering. We would also like to remove the centralized control and develop distributed algorithms, more suitable for implementation in a data center. To do so would require the development of a much simpler algorithm than the one here, possibly a primal-dual based algorithm. Finally, we observe that in applications the $D$ matrices may have uncertainty, and it would be interesting to design algorithms to deal with this uncertainty, via robust or stochastic optimization.

*Acknowledgement.*

Yuan Zhong would like to thank Mosharaf Chowdhury and Ion Stoica for numerous discussions on the coflow scheduling problem, and for sharing the Facebook data.

## 6. REFERENCES

[1] Apache hadoop. `http://hadoop.apache.org`.

[2] Google dataflow. `https://www.google.com/events/io`.

[3] Reza Ahmadi, Uttarayan Bagchi, and Thomas Roemer. Coordinated scheduling of customer orders for quick response. *Naval Research Logistics*, 52(6):493–512, 2005.

[4] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: Minimal near-optimal datacenter transport. *SIGCOMM Computer Communication Review*, 43(4):435–446, 2013.

[5] E. Balas. On the facial structure of scheduling polyhedra. *Mathematical Programming Studies*, 24:179–218, 1985.

[6] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. *SIGCOMM Computer Communication Review*, 41(4):242–253, 2011.

[7] Garrett Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tkcumán. Rev. A*, 5:147–151, 1946.

[8] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. Hadoop Project Website, 2007.

[9] Cheng-Shang Chang, Wen-Jyh Chen, and Hsiang-Yi Huang. Birkhoff-von neumann input buffered crossbar switches. In *INFOCOM*, volume 3, pages 1614–1623, 2000.

[10] Zhi-Long Chen and Nicholas G. Hall. Supply chain scheduling: Conflict and cooperation in assembly systems. *Operations Research*, 55(6):1072–1089, 2007.

[11] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *HotNets-XI*, pages 31–36, 2012.

[12] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *SIGCOMM Computer Communication Review*, 41(4):98–109, 2011.

[13] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with Varys. In *SIGCOMM*, 2014.

[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 10–10, 2004.

[15] Fahad Dogar, Thomas Karagiannis, Hitesh Ballani, and Ant Rowstron. Decentralized task-aware scheduling for data center networks. Technical Report MSR-TR-2013-96, 2013.

[16] Naveen Garg, Amit Kumar, and Vinayaka Pandit. Order scheduling models: Hardness and algorithms. In *FSTTCS*, pages 96–107, 2007.

[17] Leslie A Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[18] Marshall Hall. *Combinatorial Theory*. Addison-Wesley, 2nd edition, 1998.

[19] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the "one big switch" abstraction in software-defined networks. In *CoNEXT*, pages 13–24, 2013.

[20] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[21] E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM*, 25(4):612–619, 1978.

[22] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[23] Joseph Y. T. Leung, Haibing Li, and Michael Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics*, 155(8):945–970, 2007.

[24] J. Li and N. Ansari. Enhanced birkhoff-von neumann decomposition algorithm for input queued switches. *IEE Proceedings - Communications*, 148(6):339–342, 2001.

[25] Monaldo Mastrolilli, Maurice Queyranne, Andreas S. Schulz, Ola Svensson, and Nelson A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.

[26] Michael J. Neely, Eytan Modiano, and Yuan-Sheng Cheng. Logarithmic delay for $N \times N$ packet switches under the crossbar constraint. *IEEE/ACM Transactions on Networking*, 15(3):657–668, 2007.

[27] Cynthia A. Phillips, Cliff Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82(1-2):199–223, 1998.

[28] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, NY, USA, 3rd edition, 2008.

[29] Lucian Popa, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. Faircloud: Sharing the network in cloud computing. In *HotNets-X*, pages 22:1–22:6, 2011.

[30] Thomas A. Roemer. A note on the complexity of the concurrent open shop problem. In *Integer Programming and Combinatorial Optimization*, pages 301–315, 2006.

[31] Devavrat Shah, John. N. Tsitsiklis, and Yuan Zhong. On queue-size scaling for input-queued switches. preprint, 2014.

[32] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *MSST*, pages 1–10, 2010.

[33] Martin Skutella. *List Scheduling in Order of $\alpha$-Points on a Single Machine*, volume 3484 of *Lecture Notes in Computer Science*, pages 250–291. Springer, Berlin, Heidelberg, 2006.

[34] Chang Sup Sung and Sang Hum Yoon. Minimizing total weighted completion time at a pre-assembly stage composed of two feeding machines. *International Journal of Production Economics*, 54(3):247–255, 1998.

[35] Guoqing Wang and T.C. Edwin Cheng. Customer order scheduling to minimize total weighted completion time. *Omega*, 35(5):623–626, 2007.

[36] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, pages 2–2, 2012.

# Appendices

## A. CONNECTION WITH CONCURRENT OPEN SHOP SCHEDULING

The coflow scheduling problem (O) is closely related with the concurrent open shop problem [3, 30], as observed in [13]. Consider the formulation (O) of coflow scheduling. When all the coflows are given by *diagonal* matrices, coflow scheduling is equivalent to a concurrent open shop scheduling problem [13]. To see the equivalence, recall the problem setup of concurrent open shop. A set of $n$ jobs are released into a system with $m$ parallel machines at different times. For $k = 1, 2, \ldots, n$, job $k$ is released at time $r_k$ and requires $p_i^{(k)}$ units of processing time on machine $i$, for $i = 1, 2, \ldots, m$. The completion time of job $k$, which we denote by $C_k$, is the time that its processing requirements on all machines are completed. Let $z(i, k, t)$ be the indicator variable which equals 1 if job $k$ is served on machine $k$ at time $t$, and 0 otherwise. Let $w_k$ be positive weight parameters, $k = 1, 2, \ldots, n$. Then, we can formulate a program (CO) for concurrent open shop scheduling.

$$\text{(CO)} \quad \text{Minimize } \sum_{k=1}^{n} w_k C_k \qquad \text{subject to}$$

$$\sum_{t=1}^{C_k} z(i,k,t) \geq p_{ik}, \quad \text{for } i = 1,\ldots,m, k = 1,\ldots,n; \tag{21}$$

$$z(i,k,t) = 0 \text{ if } t < r_k, \quad \forall i,k,t; \tag{22}$$

$$z(i,k,t) \text{ binary}, \quad \forall i,k,t. \tag{23}$$

Compare program (CO) with (O). Suppose for each $k = 1, 2, \ldots, n$, $D^{(k)}$ is a diagonal matrix with diagonal entries given by $p_i^{(k)}$, $i = 1, 2, \ldots, m$. Then, when $i \neq j$, we always have $y(i, j, k, t) = 0$. If we rewrite $y(i, i, k, t)$ as $z(i, k, t)$, then in this case, program (O) is equivalent to (CO).

The concurrent open shop scheduling problem has been shown to be strongly NP-hard [3]. Due to the connection discussed above, we see immediately that the coflow scheduling problem is also strongly NP-hard. We summarize this result in the following lemma.

Lemma 5. *Problem (O) is NP-hard for $m \geq 2$.*

Although there are similarities between the concurrent open shop and the coflow scheduling problem, there are also key differences which make the coflow scheduling problem a more challenging one. First, coflow scheduling involves *coupled resources* – the inputs and outputs are coupled by the matching constraints. As such, the coflow scheduling problem has also been called *concurrent open shop with coupled resources* in [13]. Second, for concurrent open shop, there exists an optimal schedule in which jobs can be processed in the same order on all machines, i.e., the schedule is a *permutation* schedule [3]. In contrast, permutation schedules need not be optimal for coflow scheduling [13]. LP-relaxations based on completion time variables, which have been used for concurrent open shop (see e.g., [25]) use permutation schedules in a crucial way and do not immediately extend to the coflow scheduling problem.

## B. LOWER BOUNDS (17) CANNOT BE ACHIEVED SIMULTANEOUSLY

We construct a simple counter-example to show that we cannot achieve the lower bounds $V_k$ in (17) simultaneously for all the $k$. The counter-example involves 2 coflows in a datacenter with 3 inputs/outputs, which we can represent as $3 \times 3$ matrices. Let us use $D^{(1)}$ and $D^{(2)}$ to denote these two matrices for coflow 1 and coflow 2 respectively. Suppose that $D^{(1)}$ and $D^{(2)}$ are given by

$$\begin{pmatrix} 9 & 0 & 9 \\ 0 & 9 & 0 \\ 9 & 0 & 9 \end{pmatrix}, \begin{pmatrix} 1 & 10 & 1 \\ 10 & 1 & 10 \\ 1 & 10 & 1 \end{pmatrix}.$$

Define two time points $t_1 = \max\{I_1, J_1\} = 18$, and $t_2 = \max\{I_2, J_2\} = 30$. For coflow 1 to complete before time $t_1$, all the flows of coflow 1 must finish processing at time $t_1$. By the structure of $D^{(1)}$, inputs 1 & 3 and outputs 1 & 3 must work at full capacity on coflow 1 throughout the duration of time period 0 to $t_1$. Furthermore, for both coflows to complete before time $t_2$, all ports must work at full capacity throughout the time period 0 to $t_2$, due to the structure of $D^{(1)} + D^{(2)}$. Therefore, at time $t_1$, the remaining flows to be transferred across the network switch are all from coflow 2, the amount of which must be exactly $t_2 - t_1 = 12$ for each pair of input and output. Let matrix $\tilde{D}^{(2)} = \left(\tilde{d}_{ij}^{(2)}\right)$ represent the collection of remaining flows to be transferred from time $t_1$ to time $t_2$. $\tilde{D}^{(2)}$ is a $3\times3$ matrix with all the row sum and column sum equal to 12 and satisfies $\tilde{D}^{(2)} \leq D^{(2)}$. However, since coflow 1 uses up all the capacity on inputs 1 & 3 and outputs 1 & 3 from time 0 up to time $t_1$, the remaining flows $\tilde{d}_{ij}^{(2)} = d_{ij}^{(2)}$ for $(i, j) \neq (2, 2)$. We conclude that such matrix does not exist because $\tilde{d}_{21}^{(2)} + \tilde{d}_{23}^{(2)} = 20 > 12$.

## C. PROOF OF LEMMA 3

The idea of the proof follows Wang and Cheng [35]. Suppose that $\tau_{u-1} < \bar{C}_k \leq \tau_u$ for some u. We consider the following three cases.
Case (1) $\bar{C}_k < \frac{5\tau_{u-1}}{4}$.
For any $g = 1, \cdots, k$, we have

$$\frac{5\tau_{u-1}}{4} > \bar{C}_k \geq \bar{C}_g = \sum_{l=1}^{L} \tau_{l-1} \bar{x}_l^{(g)} \geq \tau_u \sum_{l=u+1}^{L} \bar{x}_l^{(g)}$$
$$= \tau_u \left(1 - \sum_{l=1}^{u} \bar{x}_l^{(g)}\right) = 2\tau_{u-1} \left(1 - \sum_{l=1}^{u} \bar{x}_l^{(g)}\right).$$

Therefore,

$$\sum_{l=1}^{u} \bar{x}_l^{(g)} > \frac{3}{8}.$$

Let $g^* = \arg\min_{1 \leq g \leq k} \sum_{l=1}^{u} \bar{x}_l^{(g)}$. We know from Eq. (16) that

$$V_k = \max\left\{\max_i \left\{\sum_{j'=1}^{m} \sum_{g=1}^{k} d_{ij'}^{(g)}\right\}, \max_j \left\{\sum_{i'=1}^{m} \sum_{g=1}^{k} d_{i'j}^{(g)}\right\}\right\}$$
$$= \max\left\{\max_i \left\{\sum_{j'=1}^{m} \sum_{g=1}^{k} d_{ij'}^{(g)}\right\}, \max_j \left\{\sum_{i'=1}^{m} \sum_{g=1}^{k} d_{i'j}^{(g)}\right\}\right\}$$
$$\times \sum_{l=1}^{u} \bar{x}_l^{(g^*)} \Bigg/ \sum_{l=1}^{u} \bar{x}_l^{(g^*)}.$$

We then have

$$V_k = \max\left\{ \max_i \left\{ \left( \sum_{j'=1}^{m}\sum_{g=1}^{k} d_{ij'}^{(g)} \right) \left( \sum_{l=1}^{u} \bar{x}_l^{(g^*)} \right) \right\}, \max_j \left\{ \left( \sum_{i'=1}^{m}\sum_{g=1}^{k} d_{i'j}^{(g)} \right) \left( \sum_{l=1}^{u} \bar{x}_l^{(g^*)} \right) \right\} \right\} \Big/ \sum_{l=1}^{u} \bar{x}_l^{(g^*)}$$

$$\leq \max\left\{ \max_i \left\{ \sum_{j'=1}^{m}\sum_{g=1}^{k} \left( d_{ij'}^{(g)} \sum_{l=1}^{u} \bar{x}_l^{(g)} \right) \right\}, \max_j \left\{ \sum_{i'=1}^{m}\sum_{g=1}^{k} \left( d_{i'j}^{(g)} \sum_{l=1}^{u} \bar{x}_l^{(g)} \right) \right\} \right\} \Big/ \sum_{l=1}^{u} \bar{x}_l^{(g^*)}$$

$$\leq \max\left\{ \max_i \left\{ \sum_{l=1}^{u}\sum_{j'=1}^{m}\sum_{g=1}^{n} d_{ij'}^{(g)} \bar{x}_l^{(g)} \right\}, \max_j \left\{ \sum_{l=1}^{u}\sum_{i'=1}^{m}\sum_{g=1}^{n} d_{i'j}^{(g)} \bar{x}_l^{(g)} \right\} \right\} \Big/ \sum_{l=1}^{u} \bar{x}_l^{(g^*)}.$$

Using the constraints (11) and (12), we have

$$V_k \leq \frac{\tau_u}{\sum_{l=1}^{u} \bar{x}_l^{(g^*)}} < \frac{8}{3}\tau_u = \frac{16}{3}\tau_{u-1} < \frac{16}{3}\bar{C}_k.$$

Case (2) $\frac{5\tau_{u-1}}{4} \leq \bar{C}_k < \frac{3\tau_{u-1}}{2}$.
Define a sequence $\alpha_h, h = 1, 2, \ldots$, by

$$\alpha_1 = \frac{1}{4},$$
$$\alpha_h = 1 - \sum_{q=1}^{h-1} \alpha_q - \frac{1-\alpha_1}{1+\sum_{q=1}^{h-1}\alpha_q}, h = 2, \cdots, \infty.$$

Note that a simple induction shows that for all $h$, $\alpha_h > 0$, and $\sum_{q=1}^{h} \alpha_q < 1/2$. Furthermore, $\sum_{q=1}^{h} \alpha_q \to 1/2$ as $h \to \infty$. Thus, there exists some $h$ such that

$$\left(1 + \sum_{q=1}^{h-1}\alpha_q\right)\tau_{u-1} < \bar{C}_k \leq \left(1 + \sum_{q=1}^{h}\alpha_q\right)\tau_{u-1}.$$

For any $g = 1, \cdots, k$, we have

$$\left(1 + \sum_{q=1}^{h}\alpha_q\right)\tau_{u-1} \geq \bar{C}_k \geq \bar{C}_g = \sum_{l=1}^{L} \tau_{l-1}\bar{x}_l^{(g)} > \tau_u \sum_{l=u+1}^{L} \bar{x}_l^{(g)} = \tau_u\left(1 - \sum_{l=1}^{u} \bar{x}_l^{(g)}\right),$$

and

$$\sum_{l=1}^{u} \bar{x}_l^{(g)} > \frac{1 - \sum_{q=1}^{h}\alpha_q}{2}.$$

It follows that,

$$V_k < \frac{\tau_u}{\sum_{l=1}^{u} \bar{x}_l^{(g^*)}} < \frac{2\tau_u}{1 - \sum_{q=1}^{h}\alpha_q} = \frac{4\tau_{u-1}}{1 - \sum_{q=1}^{h}\alpha_q} < \frac{4\bar{C}_k}{(1 - \sum_{q=1}^{h}\alpha_q)(1 + \sum_{q=1}^{h-1}\alpha_q)}.$$

By the definition of $\alpha_q$, we have

$$\left(1 - \sum_{q=1}^{h}\alpha_q\right)\left(1 + \sum_{q=1}^{h-1}\alpha_q\right) = \left(\frac{1-\alpha_1}{1+\sum_{q=1}^{h-1}\alpha_q}\right)\left(1 + \sum_{q=1}^{h-1}\alpha_q\right) = 1 - \alpha_1,$$

Therefore,

$$V_k < \frac{4\bar{C}_k}{1-\alpha_1} < \frac{16}{3}\bar{C}_k.$$

Case (3) $\bar{C}_k \geq \frac{3\tau_{u-1}}{2}$.
For any $g = 1, \cdots, k$, we have

$$\tau_u > \bar{C}_k \geq \bar{C}_g = \sum_{l=1}^{L} \tau_{l-1}\bar{x}_l^{(g)} > \tau_{u+1} \sum_{l=u+2}^{L} \bar{x}_l^{(g)} = \tau_{u+1}\left(1 - \sum_{l=1}^{u+1} \bar{x}_l^{(g)}\right) = 2\tau_u\left(1 - \sum_{l=1}^{u+1} \bar{x}_l^{(g)}\right),$$

and hence

$$\sum_{l=1}^{u+1} \bar{x}_l^{(g)} > \frac{1}{2}.$$

Using the same argument as in case (1), we have

$$V_k < \frac{\tau_{u+1}}{\sum_{l=1}^{u+1} \bar{x}_l^{(g^*)}} < 2\tau_{u+1} = 8\tau_{u-1} < \frac{16}{3}\bar{C}_k.$$

We know from Lemma 1 that $\bar{C}_k \leq C_k(OPT)$ and the result follows.

## D. TABLE OF EXPERIMENTAL RESULTS

We present all the results from the set of experiments on data under various filtering rules using different weights $w$, as we describe in the experimental section. We compute the *normalized* total weighted completion times in Table 1 to evaluate the performance of the combination of 3 orders, $H_A$, $H_\rho$ and $H_{LP}$, in the ordering stage and 4 cases in the scheduling stage, (a) without grouping and without backfilling, (b) without grouping and with backfilling, (c) with grouping and without backfilling, (d) with grouping and with backfilling. The normalization is with respect to the completion times in case (d) with the ordering $H_{LP}$.

| Filtering rules | Case | Equal weight | | | Random weights | | |
|---|---|---|---|---|---|---|---|
| | | $H_A$ | $H_\rho$ | $H_{LP}$ | $H_A$ | $H_\rho$ | $H_{LP}$ |
| $M' \geq 50$ | (a) | 9.19 | 1.41 | 1.44 | 6.78 | 1.31 | 1.33 |
| | (b) | 8.95 | 1.30 | 1.34 | 6.56 | 1.22 | 1.23 |
| | (c) | 7.99 | 1.01 | 1.04 | 5.91 | 0.96 | 1.04 |
| | (d) | 7.79 | 0.97 | 1.00 | 5.81 | 0.92 | 1.00 |
| $M' \geq 40$ | (a) | 10.14 | 1.46 | 1.49 | 7.44 | 1.36 | 1.40 |
| | (b) | 9.86 | 1.34 | 1.37 | 7.24 | 1.27 | 1.27 |
| | (c) | 8.80 | 1.01 | 1.04 | 6.40 | 0.96 | 1.04 |
| | (d) | 8.61 | 0.97 | 1.00 | 6.30 | 0.93 | 1.00 |
| $M' \geq 30$ | (a) | 10.25 | 1.49 | 1.51 | 8.18 | 1.40 | 1.44 |
| | (b) | 9.98 | 1.37 | 1.40 | 7.77 | 1.30 | 1.30 |
| | (c) | 8.86 | 1.01 | 1.04 | 7.04 | 0.97 | 1.04 |
| | (d) | 8.68 | 0.97 | 1.00 | 6.89 | 0.93 | 1.00 |

Table 1: Normalized total weighted completion times for the combination of 3 ordering heuristics and 4 scheduling methods. Data are filtered based on $M'$.