Experimental Analysis of Algorithms for Coflow Scheduling

Zhen Qiu, Clifford Stein, and Yuan Zhong(✉)

Department of IEOR, Columbia University, New York, NY 10027, USA
yz2561@columbia.edu

Abstract. Modern data centers face new scheduling challenges in optimizing job-level performance objectives, where a significant challenge is the scheduling of highly parallel data flows with a common performance goal (e.g., the shuffle operations in MapReduce applications). Chowdhury and Stoica [6] introduced the coflow abstraction to capture these parallel communication patterns, and Chowdhury et al. [8] proposed effective heuristics to schedule coflows efficiently. In our previous paper [18], we considered the strongly NP-hard problem of minimizing the total weighted completion time of coflows with release dates, and developed the first polynomial-time scheduling algorithms with $O(1)$-approximation ratios.

In this paper, we carry out a comprehensive experimental analysis on a Facebook trace and extensive simulated instances to evaluate the practical performance of several algorithms for coflow scheduling, including our approximation algorithms developed in [18]. Our experiments suggest that simple algorithms provide effective approximations of the optimal, and that the performance of the approximation algorithm of [18] is relatively robust, near optimal, and always among the best compared with the other algorithms, in both the offline and online settings.

1 Introduction

Data-parallel computation frameworks such as MapReduce [9], Hadoop [1,5,19], Spark [21], Google Dataflow [2], etc., are gaining tremendous popularity as they become ever more efficient in storing and processing large-scale data sets in modern data centers. This efficiency is realized largely through massive parallelism. Typically, a datacenter job is broken down into smaller tasks, which are processed in parallel in a *computation stage*. After being processed, these tasks produce intermediate data, which may need to be processed further, and which are transferred between groups of servers across the datacenter network, in a *communication stage*. As a result, datacenter jobs often alternate between computation and communication stages, with parallelism enabling the fast completion of these large-scale jobs. While this massive parallelism contributes to efficient data processing, it presents many new challenges for network scheduling.

Research partially supported by NSF grant CCF-1421161.

A.V. Goldberg and A.S. Kulikov (Eds.): SEA 2016, LNCS 9685, pp. 262–277, 2016.
DOI: 10.1007/978-3-319-38851-9_18

In particular, traditional networking techniques focus on optimizing flow-level performance such as minimizing flow completion times[1], and ignore job-level performance metrics. However, since a computation stage often can only start after all parallel dataflows within a preceding communication stage have finished [7,10], *all* these flows need to finish early to reduce the processing time of the communication stage, and of the entire job.

To faithfully capture application-level communication requirements, Chowdhury and Stoica [6] introduced the *coflow* abstraction, defined to be a collection of parallel flows with a common performance goal. Effective scheduling heuristics were proposed in [8] to optimize coflow completion times. In our previous paper [18], we developed scheduling algorithms with constant approximation ratios for the strongly NP-hard problem of minimizing the total weighted completion time of coflows with release dates, and conducted preliminary experiments to examine the practical performance of our approximation algorithms. These are the first $O(1)$-approximation algorithms for this problem. In this paper, we carry out a systematic experimental study on the practical performance of several coflow scheduling algorithms, including our approximation algorithms developed in [18]. Similar to [18], the performance metric that we focus on in this paper is the total weighted coflow completion time. As argued in [18], it is a reasonable user-oriented performance objective. It is also natural to consider other performance objectives, such as the total weighted *flow time*[2], which we leave as future work. Our experiments are conducted on real-world data gathered from Facebook and extensive simulated data, where we compare our approximation algorithm and its modifications to several other scheduling algorithms in an offline setting, and evaluate their relative performances, and compare them to an LP-based lower bound. The algorithms that we consider in this paper are characterized by several main components, such as the coflow order in which the algorithms follow, the grouping of the coflows, and the backfilling rules. We study the impact of each such component on the algorithm performance, and demonstrate the robust and near-optimal performance of our approximation algorithm [18] and its modifications in the offline setting, under the case of zero release times as well as general release times. We also consider online variants of the offline algorithms, and show that the online version of our approximation algorithm has near-optimal performance on real-world data and simulated instances.

The rest of this section is organized as follows. In Sect. 1.1, we quickly recall the problem formulation of coflow scheduling, the approximation algorithm of [18] as well as its approximation ratio. Section 1.2 gives an overview of the experimental setup and the main findings from our experiments. A brief review of related works is presented in Sect. 1.3.

[1] In this paper, the term "flow" refers to data flows in computer networking, and is not to be confused with the notion of "flow time," commonly used in the scheduling literature.

[2] Here "flow time" refers to the length of time from the release time of a coflow to its completion time, as in scheduling theory.

1.1 Coflow Model and Approximation Algorithm

We consider a discrete-time system where n *coflows* need to be scheduled in an $m \times m$ datacenter network with m *inputs* and m *outputs*. For each $k \in \{1, 2, \cdots, n\}$, coflow k is released at time r_k, and can be represented by an $m \times m$ matrix $D^{(k)} = \left(d_{ij}^{(k)}\right)_{i,j=1}^{m}$, where $d_{ij}^{(k)}$ is the number of data units (a.k.a. *flow size*) that need to be transferred from input i to output j. The network has the so-called non-blocking switch architecture [3,4,12,16], so that a data unit that is transferred out of an input is immediately available at the corresponding output. We also assume that all inputs and outputs have unit capacity. Thus, in a time slot, each input/output can process at most one data unit; similar to [18], these restrictions are called *matching constraints*. Let C_k denote the completion time of coflow k, which is the time when all data units from coflow k have finished being transferred. We are interested in developing efficient (offline) scheduling algorithms that minimize $\sum_{k=1}^{n} w_k C_k$, the total weighted completion time of coflows, where w_k is a weight parameter associated with coflow k.

A main result of [18] is the following theorem.

Theorem 1 [18]. *There exists a deterministic polynomial time* 67/3-*approximation algorithm for the coflow scheduling problem, with the objective of minimizing the total weighted completion time.*

The approximation algorithm of [18] consists of two related stages. First, a *coflow order* is computed by solving a polynomial-sized interval-indexed linear program (LP) relaxation, similar to many other scheduling algorithms (see e.g., [11]). Then, we use this order to derive an actual schedule. To do so, we define a *grouping* rule, under which 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 optimally, according to an integer version of the Birkhoff-von Neumann decomposition theorem. The detailed description of the algorithm can be found in Algorithm 4 of the Appendix in [17]. Also see [18] for more details. From now on, the approximation algorithm of [18] will be referred to as the LP-based algorithm.

1.2 Overview of Experiments

Since our LP-based algorithm consists of an ordering and a scheduling stage, we are interested in algorithmic variations for each stage and the performance impact of these variations. More specifically, we examine the impact of different ordering rules, coflow grouping and backfilling rules, in both the offline and online settings. Compared with the very preliminary experiments we did in [18], in this paper we conduct a substantially more comprehensive study by considering many more ordering and backfilling rules, and examining the performance of algorithms on general instances in addition to real-world data. We also consider the offline setting with general release times, and online extensions of algorithms, which are not discussed in [18].

Workload. Our evaluation uses real-world data, which is a Hive/MapReduce trace collected from a large production cluster at Facebook [7,8,18], as well as extensive simulated instances.

For real-world data, we use the same workload as described in [8,18]. collected on a 3000-machine cluster with 150 racks, so the datacenter in the experiments can be modeled as a 150×150 network switch (and coflows be represented by 150×150 matrices). We select the time unit to be $1/128$ s (see [18] for details) 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 $M' \geq 25$, $M' \geq 50$ and $M' \geq 100$, respectively.

We also consider synthetic instances in addition to the real-world data. For problem size with $k = 160$ coflows and $m = 16$ inputs and outputs, we randomly generate 30 instances with different numbers of non-zero flows involved in each coflow. For instances 1–5, each coflow consists of m flows, which represent sparse coflows. For instances 5–10, each coflow consists of m^2 flows, which represent dense coflows. For instances 11–30, each coflow consists of u flows, where u is uniformly distributed on $\{m, \cdots, m^2\}$. Given the number k of flows in each coflow, k pairs of input and output ports are chosen randomly. For each pair of (i, j) that is selected, an integer processing requirement $d_{i,j}$ is randomly selected from the uniform distribution on $\{1, 2, \cdots, 100\}$.

Our main experimental 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 benefit of backfilling can be further improved by using a balanced backfilling rule (see Sect. 3.2 for details).
- The performance of the LP-based algorithm and its extensions is relatively robust, and among the best compared with those that use other simpler ordering rules, in the offline setting.
- In the offline setting with general release times, the magnitude of inter-arrival times relative to the processing times can have complicated effects on the performance of various algorithms (see Sect. 4.1 for details).
- The LP-based algorithm can be extended to an online algorithm and has near-optimal performance.

1.3 Related Work

There has been a great deal of success over the past 20 years on combinatorial scheduling to minimize average completion time, see e.g., [11,14,15,20], typically using a linear programming relaxation to obtain an ordering of jobs and then using that ordering in some other polynomial-time algorithm. There has also been success in shop scheduling. We do not survey that work here, but note that traditional shop scheduling is not "concurrent". In the language of our problem, that would mean that 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 is a close connection between concurrent open shop problem and coflow scheduling problem. When all coflow matrices are diagonal, coflow scheduling is equivalent to a concurrent open shop scheduling problem [8,18]. Leung et al. [13] presented heuristics for the total completion time objective and conducted an empirical analysis to compare the performance of different heuristics for concurrent open shop problem. In this paper, we consider a number of heuristics that include natural extensions of heuristics in [13] to coflow scheduling.

2 Preliminary Background

In [18], we formulated the following interval-indexed linear program (LP) relaxation of the coflow scheduling problem, where τ_l's are the end points of a set of geometrically increasing intervals, with $\tau_0 = 0$, and $\tau_l = 2^{l-1}$ for $l \in \{1, 2, \ldots, L\}$. Here L is such that $\tau_L = 2^{L-1}$ is an upper bound on the time that all coflows are finished processing under any optimal algorithm.

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

$$\sum_{u=1}^{l} \sum_{k=1}^{n} \sum_{j'=1}^{m} d_{ij'}^{(k)} x_u^{(k)} \leq \tau_l, \text{ for } i = 1, \ldots, m, \ l = 1, \ldots, L; \tag{1}$$

$$\sum_{u=1}^{l} \sum_{k=1}^{n} \sum_{i'=1}^{m} d_{i'j}^{(k)} x_u^{(k)} \leq \tau_l, \text{ for } j = 1, \ldots, m, \ l = 1, \ldots, L; \tag{2}$$

$$x_l^{(k)} = 0 \text{ if } r_k + \sum_{j'=1}^{m} d_{ij'}^{(k)} > \tau_l \text{ or } r_k + \sum_{i'=1}^{m} d_{i'j}^{(k)} > \tau_l; \tag{3}$$

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

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

For each k and l, $x_l^{(k)}$ can be interpreted as the LP-relaxation of the binary decision variable which indicates whether coflow k is scheduled to complete within the interval $(\tau_{l-1}, \tau_l]$. Constraints (1) and (2) are the *load constraints* on the inputs and outputs, respectively, which state that the total amount of work completed on each input/output by time τ_l cannot exceed τ_l, due to matching constraints. Contraint (3) takes into account of the release times.

(LP) provides a lower bound on the optimal total weighted completion time of the coflow scheduling problem. If, instead of being end points of geometrically increasing time intervals, τ_l are end points of the discrete time units, then (LP) becomes exponentially sized (which we refer to as (LP-EXP)), and gives a tighter lower bound, at the cost of longer running time. (LP) computes an approximated completion time $\bar{C}_k = \sum_{l=1}^{L} \tau_{l-1} \bar{x}_l^{(k)}$, for each k, based on which we re-order and index the coflows in a nondecreasing order of $\bar{C}_k$, i.e.,

$$\bar{C}_1 \le \bar{C}_2 \le \ldots \le \bar{C}_n. \tag{4}$$

3 Offline Algorithms with Zero Release Time

In this section, we assume that all the coflows are released at time 0. We compare our LP-based algorithm with others that are based on different ordering, grouping, and backfilling rules.

3.1 Ordering Heuristics

An intelligent ordering of coflows in the ordering stage can substantially reduce coflow completion times. We consider the following five greedy ordering rules, in addition to the LP-based order (4), and study how they affect algorithm performance.

Definition 1. *The First in first (FIFO) heuristic orders the coflows arbitrarily (since all coflows are released at time* 0*).*

Definition 2. *The Shortest Total Processing Time first (STPT) heuristic orders the coflows based on the total amount of processing requirements over all the ports, i.e.,* $\sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij}$.

Definition 3. *The Shortest Maximum Processing Time first (SMPT) heuristic orders the coflows based on the load* ρ *of the coflows, where* $\rho = \max\{\max_{i=1,\ldots,m} \eta_i, \max_{j=1,\ldots,m} \theta_j\}$, $\eta_i = \{\sum_{j'=1}^{m} d_{ij'}\}$ *is the load on input* i, *and* $\theta_j = \{\sum_{i'=1}^{m} d_{i'j}\}$ *is the load on output* j.

Definition 4. *To compute a coflow order, the Smallest Maximum Completion Time first (SMCT) heuristic treats all inputs and outputs as* $2m$ independent *machines. For each input* i*, it solves a single-machine scheduling problem where* n *jobs are released at time* 0*, with processing times* $\eta_i^{(k)}$, $k = 1, 2, \cdots, n$*, where* $\eta_i^{(k)}$ *is the* i*th input load of coflow* k*. The jobs are sequenced in the order of increasing* $\eta_i^{(k)}$*, and the completion times* $C^{(i)}(k)$ *are computed. A similar problem is solved for each output* j*, where jobs have processing times* $\theta_j^{(k)}$*, and the completion times* $C_{(j)}(k)$ *are computed. Finally, the SMCT heuristic computes a coflow order according to non-decreasing values of* $C'(k) = \max_{i,j}\{C^{(i)}(k), C_{(j)}(k)\}$.

Definition 5. *The Earliest Completion Time first (ECT) heuristic generates a sequence of coflow one at a time; each time it selects as the next coflow the one that would be completed the earliest*[3].

[3] These completion times depend on the scheduling rule used. Thus, ECT depends on the underlying scheduling algorithm. In Sect. 3.2, the scheduling algorithms are described in more detail.

3.2 Scheduling via Birkhoff-Von Neumann Decomposition, Backfilling and Grouping

The derivation of the actual sequence of schedules in the scheduling stage of our LP-based algorithm relies on two key ideas: scheduling according to an optimal (Birkhoff-von Neumann) decomposition, and a suitable grouping of the coflows. It is reasonable to expect grouping to improve algorithm performance, because it may consolidate skewed coflow matrices to form more balanced ones that can be scheduled more efficiently. Thus, we compare algorithms with grouping and those without grouping to understand its effect. The particular grouping procedure that we consider here is the same as that in [18] (also see Step 2 of Algorithm 4 of the Appendix in [17]), and basically groups coflows into geometrically increasing groups, based on aggregate demand. Coflows of the same group are treated as a single, *aggregated* coflow, and this consolidated coflow is scheduled according to the Birkhoff-von Neumann decomposition (see [18] or Algorithm 5 of the Appendix in [17]).

Backfilling is a common strategy used in scheduling for computer systems to increase resource utilization (see, e.g. [8]). While it is difficult to analytically characterize the performance gain from backfilling in general, we evaluate its performance impact experimentally. We consider two backfilling rules, described as follows. Suppose that we are currently scheduling coflow D. The schedules are computed using the Birkhoff-von Neumann decomposition, which in turn makes use of a related, *augmented* matrix $\tilde{D}$, that is component-wise no smaller than D. The decomposition may introduce unforced idle time, whenever $D \neq \tilde{D}$. When we use a schedule that matches input i to output j to serve the coflow with $D_{ij} < \tilde{D}_{ij}$, and if there is no more service requirement on the pair of input i and output j for the coflow, 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. The two backfilling rules that we consider – which we call *backfilling* and *balanced backfilling* – are only distinguished by the *augmentation* procedures used, which are, respectively, the augmentation used in [18] (Step 1 of Algorithm 5 in [17]) and the balanced augmentation described in Algorithm 1.

The balanced augmentation (Algorithm 1) results in less skewed matrices than the augmentation step in [18], since it first "spreads out" the unevenness among the components of a coflow. To illustrate, let

$$D = \begin{pmatrix} 10 & 0 & 0 \\ 10 & 0 & 0 \\ 10 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{pmatrix}, \text{ and } C = \begin{pmatrix} 10 & 20 & 0 \\ 10 & 0 & 20 \\ 10 & 10 & 10 \end{pmatrix}.$$

Under the balanced augmentation, D is augmented to B and under the augmentation of [18], D is augmented to C.

3.3 Scheduling Algorithms and Metrics

We consider 30 different scheduling algorithms, which are specified by the ordering used in the ordering stage, and the actual sequence of schedules used in the

Algorithm 1. Balanced Coflow Augmentation

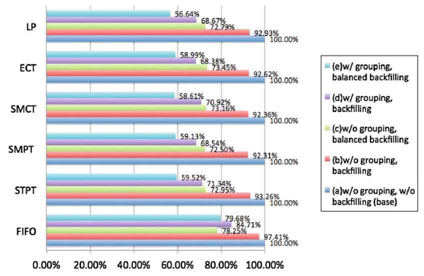
```
**Data**: A single coflow $D = (d_{ij})_{i,j=1}^m$.
**Result**: A matrix $\tilde{D} = \left(\tilde{d}_{ij}\right)_{i,j=1}^m$ with equal row and column sums, and $D \le \tilde{D}$.
Let $\rho$ be the load of $D$.
$p_i \leftarrow \rho - \sum_{j'=1}^m d_{ij'}$, for $i = 1, 2, \ldots, m$.
$q_i \leftarrow \rho - \sum_{i'=1}^m d_{i'j}$, for $j = 1, 2, \ldots, m$.
$\Delta \leftarrow m\rho - \sum_{i=1}^m \sum_{j=1}^m d_{ij}$.
$d'_{ij} = \lfloor d_{ij} + p_i q_i / \Delta \rfloor$.
Augment $D' = (d'_{ij})$ to a matrix $\tilde{D}$ with equal row and column sums (see Step 1 of Algorithm 5 of the Appendix in [17]; also see [18]).
```

scheduling stage. We consider 6 different orderings described in Sect. 3.1, and the following 5 cases in the scheduling stage:

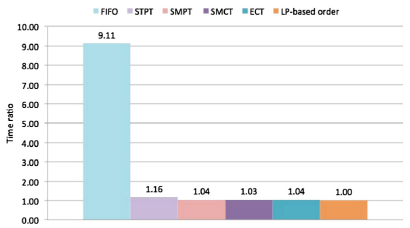
– (a) without grouping or backfilling, which we refer to as the base case;
– (b) without grouping but with backfilling;
– (c) without grouping but with balanced backfilling;
– (d) with grouping and with backfilling;
– (e) with grouping and with balanced backfilling.

We will refer to these cases often in the rest of the paper. Our LP-based algorithm corresponds to the combination of LP-based ordering and case (d).

For ordering, six different possibilities are considered. We use H_A to denote the ordering of coflows by heuristic A, where A is in the set {FIFO, STPT, SMPT, SMCT, ECT}, and H_{LP} to denote the LP-based coflow ordering. Note that in [18], we only considered orderings H_{FIFO}, H_{SMPT} and H_{LP}, and cases (a), (b) and (d) for scheduling, and their performance on the Facebook trace.



(a) Comparison of total weighted completion times normalized using the base case (e) for each order



(b) Comparison of 6 orderings with zero release times on Facebook data.

Fig. 1. Facebook data are filtered by $M' \ge 50$. Weights are equal.

3.4 Performance of Algorithms on Real-World Data

We compute the total weighted completion times for all 6 orders in the 5 different cases (a)–(e) described in Sect. 3.3, through a set of experiments on filtered coflow data. We present representative comparisons of the algorithms here.

Figure 1a plots the total weighted completion times as percentages of the base case (a), for the case of equal weights. Grouping and backfilling both improve the total weighted completion time with respect to the base case for all 6 orders. In addition to the reduction in the total weighted completion time from backfilling, which is up to 7.69 %, the further reduction from grouping is up to 24.27 %, while the improvement from adopting the balanced backfilling rule is up to 20.31 %. For 5 non-arbitrary orders (excluding FIFO), scheduling with both grouping and balanced backfilling (i.e., case (e)) gives the smallest total weighted completion time.

We then compare the performances of different coflow orderings. Figure 1b shows the comparison of total weighted completion times evaluated on filtered coflow data in case (e) where the scheduling stage uses both grouping and balanced backfilling. Compared with H_{FIFO}, all other ordering heuristics reduce the total weighted completion times of coflows by a ratio between 7.88 and 9.11, with H_{LP} performing consistently better than other heuristics.

3.5 Cost of Matching

The main difference between our coflow scheduling problem and the well-studied concurrent open shop problem we discussed in Sect. 1.3 is the presence of matching constraints on paired resources, i.e. inputs and outputs, which is the most challenging part in the design of approximation algorithms [18]. Since our approximation algorithm handles matching constraints, it is more complicated than scheduling algorithms for concurrent open shop problem. We are interested in how much we lose by imposing these matching constraints.

To do so, we generate two sets of coflow data from the Facebook trace. For each coflow k, let the coflow matrix $D^{(k)}$ be a diagonal matrix, which indicates that coflow k only has processing requirement from input i to output i, for $i = 1, \dots, m$. The processing requirement $D_{i,i}^{(k)}$ is set to be equal to the sum of all dataflows of coflow k in the Facebook trace that require processing from input i. We then construct coflow matrix $\tilde{D}^{(k)}$ such that $\tilde{D}^{(k)}$ is not diagonal and has the same row sum and column sum as $D^{(k)}$. The details of the generation is described as in Algorithm 2.

The diagonal structured coflow matrices can reduce the total completion time of by a ratio up to 2.09, which indicates the extra processing time introduced by the matching constraints.

3.6 Performance of Algorithms on General Instances

In previous sections, we present the experimental results of several algorithms on the Facebook trace. In order to examine the consistency of the performance of these algorithms, we consider more instances, including examples where certain algorithms behave badly.

Bad Instances for Greedy Heuristics. We consider the following examples which illustrate instances on which the ordering heuristics do not perform well.

Example 1. *Consider a* 2×2 *network and* n *coflows with* $D = \begin{pmatrix} 10 & 0 \\ 0 & 0 \end{pmatrix}$, n *coflows with* $D = \begin{pmatrix} 0 & 0 \\ 0 & 10 \end{pmatrix}$, *and* $a \cdot n$ *coflows with* $D = \begin{pmatrix} 9 & 0 \\ 0 & 9 \end{pmatrix}$. *The optimal schedule in this case is to schedule the orders with the smallest total processing time first, i.e., the schedule is generated according to the STPT rule. The limit of the ratio* $\frac{\sum_{k=1}^{n} C_k(ECT\&SMCT\&SMPT)}{\sum_{k=1}^{n} C_k(STPT)}$ *is increasing in* n *and when* $n \to \infty$ *it becomes* $\frac{a^2+4a+2}{a^2+2a+2}$. *This ratio reaches its maximum of* $\sqrt{2}$ *when* $a = \sqrt{2}$.

We can generalize this counterexample to an arbitrary number of inputs and outputs m. To be more specific, in an $m \times m$ network, for $j = 1, 2, \cdots, m$, we have n coflows only including flows to be transferred to output j, i.e., $d_{ij} = 10$. We also have $a \cdot n$ coflows with equal transfer requirement on all pairs of inputs and outputs, i.e., $d_{ij} = 9$ for $i, j = 1, 2, \cdots, m$. The ratio

$$\lim_{n\to\infty} \frac{\sum_{k=1}^{n} C_k(ECT\&SMCT\&SMPT)}{\sum_{k=1}^{n} C_k(STPT)} = \frac{a^2 + 2ma + m}{a^2 + 2a + m}$$

has a maximum value of $\sqrt{m}$ when $a = \sqrt{m}$. Note that in the generalized example, we need to consider the matching constraints when we actually schedule the coflows.

Example 2. *Consider a* 2×2 *network and* n *coflows with* $D = \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}$, *and* $a \cdot n$ *coflows with* $D = \begin{pmatrix} 10 & 0 \\ 0 & 0 \end{pmatrix}$. *The optimal schedule in this case is to schedule the orders with the Smallest Maximum Completion Time first, i.e., the schedule is generated according to the SMCT rule. The ratio* $\frac{\sum_{k=1}^{n} C_k(STPT)}{\sum_{k=1}^{n} C_k(SMCT)}$ *is increasing in* n *and when* $n \to \infty$ *it becomes* $\frac{a^2+2a}{a^2+1}$ *This ratio reaches its maximum of* $\frac{\sqrt{5}+1}{2}$ *when* $a = \frac{\sqrt{5}+1}{2}$.

This counterexample can be generalized to an arbitrary number of inputs and outputs m. In an $m \times m$ network, for each $i = 2, 3, \cdots, m$, we have n coflows with two nonzero entries, $d_{11} = 1$ and $d_{ii} = 10$. We also have $a \cdot n$ coflows with only one zero entry $d_{11} = 10$. The limit of the ratio

$$\lim_{n\to\infty} \frac{\sum_{k=1}^{n} C_k(STPT)}{\sum_{k=1}^{n} C_k(SMCT)} = \frac{a^2 + 2(m-1)a}{a^2 + m - 1}$$

has a maximum value of $1/2 + \sqrt{m - 3/4}$ when $a = 1/2 + \sqrt{m - 3/4}$.

General Instances. We compare total weighted completion time for 6 orderings and 5 cases on general simulated instances as described in Sect. 1.2 (details in Tables 1 to 5 of [17]), normalized with respect to the LP-based ordering in case (c), which performs best on all of the instances. We have the similar observation from the general instances that both grouping and backfilling reduce the completion time. However, under balanced backfilling, grouping does not improve performance much. Both grouping and balanced backfilling form less skewed matrices that can be scheduled more efficiently, so when balanced backfilling is used, the effect of grouping is less pronounced. It is not clear whether case (c) with balanced backfilling only is in general better than case (e) with both balanced backfilling and grouping, as we have seen Facebook data on which case (e) gives the best result. As for the performance of the orderings, on the one hand, we see very close time ratios among all the non-arbitrary orderings on instances 6–30, and a better performance of H_{ECT} on sparse instances 1–5 over other orderings (Table 3, Appendix [17]); on the other hand, there are also instances where ECT performs poorly (e.g., see Sect. 3.6).

Besides their performance, the running times of the algorithms that we consider are also important. The running time of an algorithm consists of two main parts; computing the ordering and computing the schedule. On a Macbook Pro with 2.53 GHz two processor cores and 6 G memory, the five ordering rules, FIFO, STPT, SMPT, SMCT and ECT, take less than 1 s to compute, whereas the LP-based order can take up to 90 s. Scheduling with backfilling can be computed in around 1 min, whereas balanced backfilling computes the schedules with twice the amount of time, because the balanced augmented matrices have more non-zero entries. Besides improving performance, grouping can also reduce the running time by up to 90 %.

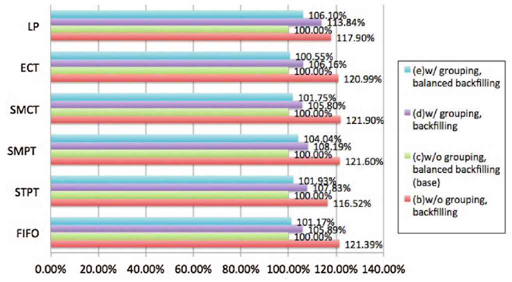
Algorithm 2. Construction of coflow data

```
**Data**: A single diagonal coflow $D = (d_{ij})_{i,j=1}^m$.
**Result**: Another coflow $\tilde{D} = \left(\tilde{d}_{ij}\right)_{i,j=1}^m$, such that row and column sums
        of the two matrices are all equal.
Let $\eta(\tilde{D}) = \sum_{i,j=1}^m \tilde{d}_{ij}$ be the sum of all entries in $\tilde{D}$. Similarly,
$\eta(D) = \sum_{i=1}^m d_{ii}$.
$\tilde{D} \leftarrow 0$.
**while** $(\eta(\tilde{D}) < \eta(D))$ **do**
    $S_i \leftarrow \{i : \sum_{j'=1}^m \tilde{D}_{ij'} < d_{ii}\}$; $S_j \leftarrow \{j : \sum_{i'=1}^m \tilde{D}_{i'j} < d_{jj}\}$. Randomly
    pick $i^*$ from set $S_i$ and $j^*$ from set $S_j$. $\tilde{D} \leftarrow \tilde{D} + pE$, where
    $p = \min\{d_{i^*i^*} - \sum_{j'=1}^m \tilde{D}_{i^*j'}, d_{j^*j^*} - \sum_{i'=1}^m \tilde{D}_{i'j^*}\}$, $E_{ij} = 1$ if $i = i^*$
    and $j = j^*$, and $E_{ij} = 0$ otherwise.
    $\eta \leftarrow \sum_{i,j=1}^m \tilde{d}_{ij}$
**end**
```

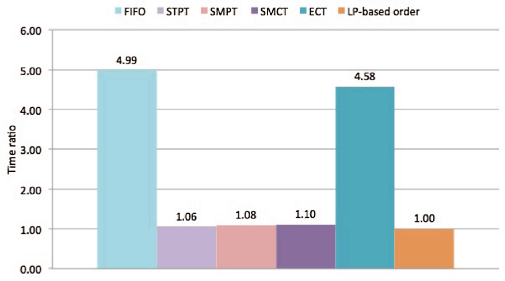
4 Offline Algorithms with General Release Times

In this section, we examine the performances of the same class of algorithms and heuristics as that studied in Sect. 3, when release times can be general. We first extend descriptions of various heuristics to account for release times.

The FIFO heuristic computes a coflow order according to non-decreasing release time r. (Note that when all release times are distinct, FIFO specifies a unique ordering on coflows, instead of any arbitrary order in the case of zero release times.) The STPT heuristic computes a coflow order according to non-decreasing values of $\sum_{i=1}^{m}\sum_{j=1}^{m} d_{ij} + r$, the total amount of processing requirements over all the ports plus the release time. The SMPT heuristic computes a coflow order according to non-decreasing values of $\rho + r$, the sum of the coflow load and release time. Similar to the case of zero release times, the SMCT heuristic first sequences the coflows in non-decreasing order of $\sum_{j'} d_{ij'} + r$ on each input i and $\sum_{i'} d_{i'j} + r$ on each output j, respectively, and then computes the completion times $C^{(i)}$ and $C_{(j)}$, treating each input and output as independent machines. Finally, the coflow order is computed according to non-decreasing values of $C' = \max_{i,j}\{C^{(i)}, C_{(j)}\}$. The ECT heuristic generates a sequence of coflows one at a time; each time it selects as the next coflow the one that has been released and is after the preceding coflow finishes processing and would be completed the earliest.



(a) Comparison of total weighted completion times normalized using the base case (c) for each order.



(b) Comparison of 6 orderings with general release times on Facebook data.

Fig. 2. Facebook data are filtered by $M' \geq 50$. Weights are equal.

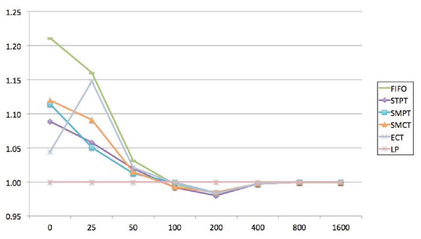
We compute the total weighted completion time for 6 orderings (namely, the LP-based ordering (4) and the orderings from definitions with release times and cases (b)–(e) (recall the description of these cases at the beginning of Sect. 3.3), normalized with respect to the LP-based ordering in case (c). The results for Facebook data are illustrated in Fig. 2a and b. For general instances, we generate the coflow inter-arrival times from uniform distribution [1, 100]. Performance ratios can be found in Tables 6 to 9 in the Appendix of [17]. As we can see from e.g., Fig. 2a, the effects of backfilling and grouping on algorithm performance are similar to those noted in Sect. 3.3, where release times are all zero. The STPT and LP-based orderings STPT appear to perform the best among all the ordering rules (see Fig. 2b), because the magnitudes of release times have a greater effect on FIFO, SMPT, SMCT and ECT than they do on STPT.

By comparing Figs. 1b and 2b, we see that ECT performs much worse than it does with common release times. This occurs because with general release times, ECT only schedules a coflow after a preceding coflow completes, so it does not backfill. While we have kept the ECT ordering heuristic simple and reasonable to compute, no backfilling implies larger completion times, hence the worse performance.

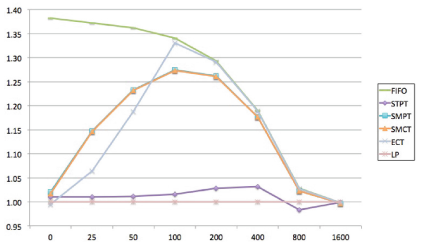
4.1 Convergence of Heuristics with Respect to Release Times

In order to have a better understanding of release times, we scale the release times of the coflows and observe the impact of release time distribution on the performance of different heuristics. For general instances, recall that we generated the inter-arrival times with an upper bound of 100. Here we also consider inter-arrival time distributions that are uniform over [0, 0], [0, 25], [0, 50], [0, 200], [0, 400], [0, 800] and [0, 1600], respectively. We compute the total weighted completion time with the adjusted release times in each case for 250 samples and take the average ratio with respect to the LP-based order.

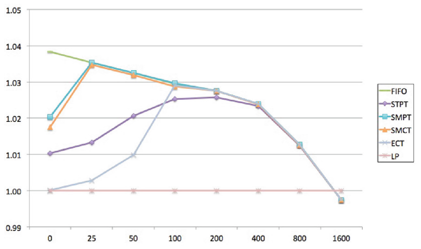
As we can see from Fig. 3a to c, all the heuristics converge to FIFO as the inter-arrival time increases. This is reasonable as the release times dominate the ordering when they are large. The speed of convergence is higher in Fig. 3a where the coflow matrices in the instance are sparse and release times are more influential in all heuristics. On the contrary, when the coflow matrices are dense, release times weigh less in heuristics, which converge slower to FIFO as shown in Fig. 3c. We also note that for heuristics other than FIFO, the relative performance of an ordering heurstic with respect to the LP-based order may deteriorate and then



(a) Number of flows is 16



(b) Number of flows is uniform in [16, 256]



(c) Number of flows is 256

Fig. 3. Comparison of total weighted completion times with respect to the upper bound of inter-arrival time for each order on general instances. Network size is 16. Number of Coflow is 160.

improve, as we increase the inter-arrival times. This indicates that when inter-arrival times are comparable to the coflow sizes, they can have a significant impact on algorithm performance and the order obtained.

5 Online Algorithms

We have discussed the experimental results of our LP-based algorithm and several heuristics in the offline setting, where the complete information of coflows is revealed at time 0. In reality, information on coflows (i.e., flow sizes) is often only revealed at their release times, i.e., in an online fashion. It is then natural to consider online modifications of the offline algorithms considered in earlier sections. We proceed as follows. For the ordering stage, upon each coflow arrival, we re-order the coflows according to their remaining processing requirements. We consider all six ordering rules described in Sect. 3. For example, the LP-based order is modified upon each coflow arrival, by re-solving the (LP) using the remaining coflow sizes (and the newly arrived coflow) at the time. We describe the online algorithm with LP-based ordering in Algorithm 3. For the scheduling stage, we use case (c) the balanced backfilling rule without grouping, because of its good performance in the offline setting.

Algorithm 3. Online LP-based Approximation

Data: Coflows $\left(d_{ij}^{(k)}\right)_{i,j=1}^{m}$ with different release times, for $k = 1, \ldots, n$.

Result: A scheduling algorithm that uses at most a polynomial number of different matchings.

- Step 1: Given n_a coflows in the system, $n_a \leq n$, solve the linear program (LP). Let an optimal solution be given by $\bar{x}_l^{(k)}$, for $l = 1, 2, \ldots, L$ and $k = 1, 2, \ldots, n_a$. Compute the approximated completion time $\bar{C}_k$ by

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

  Order and index the coflows according to

$$\bar{C}_1 \leq \bar{C}_2 \leq \ldots \leq \bar{C}_{n_a}.$$

- Step 2: Schedule the coflows in order using the Birkhoff-von Neumann decomposition (see [18] or Algorithm 5 of the Appendix in [17])) until an release of a new coflow. Update the job requirement with the remaining job for each coflow in the system and go back to Step 1.

We compare the performance of the online algorithms and we compare the online algorithms to the offline algorithms. We improve the time ratio for all the orderings except FIFO by allowing re-ordering and preemption in the online algorithm compared with the static offline version. Note that we do not preempt with FIFO order. While several ordering heuristics perform as well as LP-based ordering in the online algorithms, a natural question to ask is how close H_A's are to the optimal, where $A \in \{STPT, SMPT, SMCT, ECT, LP\}$. In order to get a tight lower bound of the coflow scheduling problem, we solve (LP-EXP) for sparse instances. Since it is extremely time consuming to solve (LP-EXP) for dense instances, we consider a looser lower bound, which is computed as follows. We first aggregate the job requirement on each input and output and solve a single machine scheduling problem for the total weighted completion time, on each input/output. The lower bound is obtained by taking the maximum of the results (see the last column of Table 11, [17]). The ratio of the lower bound over the weighted completion time under H_{LP} is in the range of 0.91 to 0.97, which indicates that it provides a good approximation of the optimal.

6 Conclusion

We have performed comprehensive experiments to evaluate different scheduling algorithms for the problem of minimizing the total weighted completion time of coflows in a datacenter network. We also generalize our algorithms to an *online* version for them to work in real-time. For additional interesting directions in experimental analysis of coflow scheduling algorithms, we would like to come up with structured approximation algorithms that take into consideration other metrics and the addition of other realistic constraints, such as precedence constraints, and distributed algorithms that are more suitable for implementation in a data center. These new algorithms can be used to design other implementable, practical algorithms.

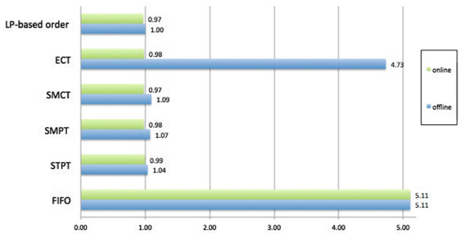


Fig. 4. Comparison of total weighted completion times with respect to the base case for each order under the offline and online algorithms. Data are filtered by $M' \geq 50$. Weights are equal.

Acknowledgment. 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.

References

1. Apache hadoop. http://hadoop.apache.org
2. Google dataflow. https://www.google.com/events/io
3. Alizadeh, M., Yang, S., Sharif, M., Katti, S., McKeown, N., Prabhakar, B., Shenker, S.: pfabric: Minimal near-optimal datacenter transport. SIGCOMM Comput. Commun. Rev. **43**(4), 435–446 (2013)

4. Ballani, H., Costa, P., Karagiannis, T., Rowstron, A.: Towards predictable datacenter networks. SIGCOMM Comput. Commun. Rev. **41**(4), 242–253 (2011)
5. Borthakur, D.: The hadoop distributed file system: Architecture and design. Hadoop Project Website (2007)
6. Chowdhury, M., Stoica, I.: Coflow: A networking abstraction for cluster applications. In: HotNets-XI, pp. 31–36 (2012)
7. Chowdhury, M., Zaharia, M., Ma, J., Jordan, M.I., Stoica, I.: Managing data transfers in computer clusters with orchestra. SIGCOMM Comput. Commun. Rev. **41**(4), 98–109 (2011)
8. Chowdhury, M., Zhong, Y., Stoica, I.: Efficient coflow scheduling with Varys. In: SIGCOMM (2014)
9. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: OSDI, p. 10 (2004)
10. Dogar, F., Karagiannis, T., Ballani, H., Rowstron, A.: Decentralized task-aware scheduling for data center networks. Technical Report MSR-TR–96 2013
11. Hall, L.A., Schulz, A.S., Shmoys, D.B., Wein, J.: Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. Math. Oper. Res. **22**(3), 513–544 (1997)
12. Kang, N., Liu, Z., Rexford, J., Walker, D.: Optimizing the "one big switch" abstraction in software-defined networks. In: CoNEXT, pp. 13–24 (2013)
13. Leung, J.Y., Li, H., Pinedo, M.: Order scheduling in an environment with dedicated resources in parallel. J. Sched. **8**(5), 355–386 (2005)
14. Phillips, C.A., Stein, C., Wein, J.: Minimizing average completion time in the presence of release dates. Math. Program. **82**(1–2), 199–223 (1998)
15. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer, New York (2008)
16. Popa, L., Krishnamurthy, A., Ratnasamy, S., Stoica, I.: Faircloud: Sharing the network in cloud computing. In: HotNets-X, pp. 22:1–22:6 (2011)
17. Qiu, Z., Stein, C., Zhong, Y.: Experimental analysis of algorithms for coflow scheduling. arXiv (2016). http://arxiv.org/abs/1603.07981
18. Qiu, Z., Stein, C., Zhong, Y.: Minimizing the total weighted completion time of coflows in datacenter networks. In: ACM Symposium on Parallelism in Algorithms and Architectures, pp. 294–303 (2015)
19. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: MSST, pp. 1–10 (2010)
20. Skutella, M.: List scheduling in order of α-points on a single machine. In: Bampis, E., Jansen, K., Kenyon, C. (eds.) Efficient Approximation and Online Algorithms. LNCS, vol. 3484, pp. 250–291. Springer, Heidelberg (2006)
21. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: NSDI, p. 2 (2012)