Brief Announcement: A New Improved Bound for Coflow Scheduling

Mehrnoosh Shafiee and Javad Ghaderi
Department of Electrical Engineering
Columbia University, New York, NY

ABSTRACT

Many data-parallel computing frameworks in today's datacenters consist of multiple computation and communication stages. A stage often cannot start or be completed unless all the required data pieces from the preceding stages are received. *Coflow* is a recently proposed networking abstraction to capture such communication patterns. We consider the problem of efficiently scheduling coflows with release dates in a shared datacenter network so as to minimize the total weighted completion time of coflows. This problem has been shown to be NP-complete, and several polynomial-time approximation algorithms have been recently proposed with provable performance guarantees. Our main result in this paper is a new polynomial-time approximation algorithm that improves the best prior known results. Specifically, we propose a deterministic algorithm with an approximation ratio of 5, which improves the prior best known ratio of 12. For the special case when all the coflows are released at time zero, we obtain an algorithm with an approximation ratio of 4 which improves the prior best known ratio of 8.

KEYWORDS

Scheduling Algorithms, Approximation Algorithms, Coflow, Datacenter Network

1 INTRODUCTION

Many data-parallel computation frameworks, such as MapReduce [3], Dryad [5], Hadoop [13], and Spark [14], alternate between computation and communication stages. Usually, a computation stage produces many pieces of data that need to be processed in remote servers, therefore, it is followed by a communication stage that transfers the intermediate data across the datacenter network. The next computation stage often cannot start unless all the required data pieces from the previous stage are received. Hence, the collective effect of all the flows between the two server groups is more important than that of any of the individual flows.

Recently Chowdhury and Stoica [1] have introduced the *coflow* abstraction to capture these application level communication requirement. *A coflow is defined as a collection of parallel flows whose completion time is determined by the completion time of the last flow in the collection.*

This work is supported by NSF Grants CNS-1652115 and CNS-1565774.

SPAA'17, July 24–26, 2017, Washington, DC, USA
 978-1-4503-4593-4/17/07.
DOI: http://dx.doi.org/10.1145/3087556.3087598

In this paper, we study the coflow scheduling problem with release dates, namely, the algorithmic task of determining when to start serving each flow and at what rate, in order to minimize the weighted sum of completion times of coflows in the system.

1.1 Related Work

The problem of coflow scheduling without release dates was introduced in Varys [2] where the authors proposed a heuristic to minimize the average completion time of coflows. Here, we would like to highlight three papers [6, 7, 10] that are more relevant to our work. The two papers [6, 10] consider the problem of minimizing the total weighted completion time of coflows with release dates. This problem is shown to be NP-complete through its connection with the concurrent open shop problem [2, 10], and then approximation algorithms are proposed which run in polynomial time and return a solution whose value is guaranteed to be within a constant fraction of the optimal. Both papers rely on linear programming relaxation techniques.
In [10], the authors utilize an interval-indexed linear program formulation to partition the coflows into disjoint groups. All coflows that fall into one partition are then viewed as a single coflow, where a polynomial-time algorithm is used to optimize its completion time. The deterministic algorithm proposed in [10] is a 67/3 and 64/3-approximation algorithm for coflow scheduling problem with and without release dates, respectively.
Authors of [6] improved these bounds to 12 and 8, respectively, by constructing an instance of the concurrent open shop problem from the original coflow scheduling problem, and applying the well-known approximation algorithms for the concurrent open shop problem to the constructed instance to obtain an ordering of coflows which is then used in a similar fashion as in [10].

2 MODEL AND PROBLEM STATEMENT

Similar to [2, 10], we abstract out the datacenter network as one giant $N \times N$ non-blocking switch, with N input links connected to N source servers and N output links connected to N destination servers. Thus, the network can be viewed as a bipartite graph with source nodes denoted by set $\mathcal{I}$ on one side and destination nodes denoted by set $\mathcal{J}$ on the other side. Moreover, there are capacity constraints on the input and output links. We assume that all the link capacities are equal and normalized to one.

A coflow is a collection of flows whose completion time is determined by the completion time of the latest flow in the collection. The coflow k can be denoted as an $N \times N$ matrix $D^{(k)}$. Every flow is a triple (i, j, k), where $i \in \mathcal{I}$ is its source node, $j \in \mathcal{J}$ is its destination node, and k is the coflow to which it belongs. The size of

flow (i, j, k) is denoted by d_{ij}^k, which is the (i, j)-th element of the matrix $D^{(k)}$.

There is a set of K coflows denoted by $\mathcal{K}$. Coflow $k \in \mathcal{K}$ is released (arrives) at time r_k which means it can only be scheduled after time r_k. For simplicity, we assume that all flows within a coflow arrive to the system at the same time (as in [10]).

For a source node $i \in \mathcal{I}$ and a coflow $k \in \mathcal{K}$, we define

$$d_i^k = \sum_{j \in \mathcal{J}} d_{ij}^k,$$

which is the aggregate flow that node i needs to transmit for coflow k. d_j^k is defined similarly for destination node $j \in \mathcal{J}$ and coflow $k \in \mathcal{K}$.

We use f_k to denote the finishing (completion) time of coflow k, which, by definition of coflow, is the time when all its flows have finished processing. In other words, for every coflow $k \in \mathcal{K}$, $f_k = \max_{i \in \mathcal{I}, j \in \mathcal{J}} f_{ij}^k$, where f_{ij}^k is the completion time of flow (i, j, k). Then the coflow scheduling problem with release dates is defined as follows. For given positive weights w_k, $k \in \mathcal{K}$, the goal is to minimize the weighted sum of coflow completion times, i.e.,

minimize	$\sum_{k \in \mathcal{K}} w_k f_k$
subject to	Capacity and release date constraints.

The weights w_k can capture different priority for different coflows.

3 MAIN RESULT

The main contribution of this paper is that we propose a polynomial-time approximation algorithm with the following improved approximation guarantees for the offline coflow scheduling problem.

Theorem 3.1. *There exists a deterministic* 5*-approximation algorithm for coflow scheduling with release dates so as to minimize total weighted completion times.*

Corollary 3.2. *When all coflows are released at time zero, the approximation ratio of this algorithm is* 4.

The prior best known ratios for this problem is 12 for the case of release dates and 8 for the case of without release dates [6]. Furthermore, our deterministic algorithm has better performance ratios even compared with the randomized algorithms of [6, 10, 12].

4 LINEAR PROGRAMING RELAXATION

In this section, we use *linear ordering variables* (see, e.g., [4, 8, 9]) to present a relaxed linear program of coflow scheduling problem. This formulation is very similar to what has been introduced in [8] for concurrent open shop problem. In the next section, we use the optimal solution to this LP as a subroutine in our deterministic algorithm.

Ordering variables. For each pair of coflows, we define a binary variable which indicates which coflow is completed (finishes all its flows) before the other coflow is completed. Formally, for any two coflows k, k', we introduce a binary variable $\delta_{kk'} \in \{0, 1\}$ such that $\delta_{kk'} = 1$ if coflow k is finished before coflow k', and it is 0 otherwise.

Relaxed Integer Program (IP). We formulate the following Integer Program (IP):

$$\textbf{(IP)} \ \min \ \sum_{k \in \mathcal{K}} w_k f_k \tag{1a}$$

$$f_k \geq d_i^k + \sum_{k' \in \mathcal{K}} d_i^{k'} \delta_{k'k} \quad i \in \mathcal{I}, k \in \mathcal{K} \tag{1b}$$

$$f_k \geq d_j^k + \sum_{k' \in \mathcal{K}} d_j^{k'} \delta_{k'k} \quad j \in \mathcal{J}, k \in \mathcal{K} \tag{1c}$$

$$f_k \geq r_k \quad k \in \mathcal{K} \tag{1d}$$

$$\delta_{kk'} + \delta_{k'k} = 1 \quad k, k' \in \mathcal{K} \tag{1e}$$

$$\delta_{kk'} \in \{0, 1\} \quad k, k' \in \mathcal{K}. \tag{1f}$$

The constraint (1b) (similarly (1c)) follows from the definition of ordering variables and the fact that flows incident to a source node i (a destination node j) are processed by a single link of unit capacity. By constraint (1d), each coflow cannot get completed before its release date. This optimization problem is a relaxed integer program for coflow scheduling problem since the set of constraints in (IP) do not capture all the requirements which a feasible schedule should satisfy.

Relaxed Linear Program (LP). In the linear program relaxation, we allow the ordering variables to be fractional. Specifically, we replace the constraint (1f) with the constraints (2b) below. We refer to the obtained linear problem by (LP).

$$\textbf{(LP)} \ \min \ \sum_{k=1}^{K} w_k f_k \tag{2a}$$

$$\text{subject to: (1b) – (1e)},$$

$$\delta_{kk'} \in [0, 1] \quad k, k' \in \mathcal{K}. \tag{2b}$$

We denote by $\tilde{f}_k$ the optimal solution to the (LP) for completion time of coflow $k \in \mathcal{K}$. We order coflows based on values of $\tilde{f}_k$ in nondecreasing order. More precisely, we number coflows such that,

$$\tilde{f}_1 \leq \tilde{f}_2 \leq \ldots \leq \tilde{f}_K. \tag{3}$$

Ties are broken arbitrarily. Also, we define $W(k)$ to be the maximum aggregate data that a node should send or receive considering the first k coflows according to the ordering in (3). Specifically,

$$W(k) = \max\{\max_{i \in \mathcal{I}}(\sum_{l=1}^{k} d_i^l), \max_{j \in \mathcal{J}}(\sum_{l=1}^{k} d_j^l)\}. \tag{4}$$

Now we characterize the solution to the linear program (LP).

Lemma 4.1. $\tilde{f}_k \geq \frac{W(k)}{2}$.

Proof. Variant versions of this lemma were used in other scheduling problems (see e.g., [4, 8, 9]). We refer to the extended version of this paper [11] for the proof. □

Furthermore, the following lemma establishes a relationship between optimal value of (LP) solution, i.e., $\sum_{k=1}^{K} w_k \tilde{f}_k$ and optimal value of coflow scheduling problem, i.e., $\sum_{k=1}^{K} w_k f_k^\star$, where $f_k^\star$ is the completion time of coflow k in the optimal schedule.

Lemma 4.2. $\sum_{k=1}^{K} w_k \tilde{f}_k \leq \sum_{k=1}^{K} w_k f_k^\star$.

Proof. Consider an optimal solution to the coflow scheduling problem. We set the ordering variables so as $\delta_{kk'} = 1$ if coflow k precedes coflow k' in this solution, and $\delta_{kk'} = 0$, otherwise. We note that this set of ordering variables and coflow completion times satisfies constraints (1b) and (1c) since the optimal solution should respect capacity constraints on the communication links. It also satisfies constraint (1d). Therefore, the optimal solution can be converted to a feasible solution to (LP). This implies the desired inequality. □

5 APPROXIMATION ALGORITHM

The approximation algorithm is depicted in Algorithm 1 which is a simple list scheduling algorithm based on the ordering in (3). More specifically, the algorithm maintains a list of flows such that for every two flows (i, j, k) and (i', j', k') with $k < k'$, flow (i, j, k) is before flow (i', j', k') in the list. Flows of the same coflow are listed arbitrarily. The algorithm scans the list starting from the first flow and schedules a flow if both its corresponding source and destination links are idle at that time. Upon completion of a flow or arrival of a coflow, the algorithm preempts the schedule, updates the list, and starts scheduling the flows in the updated list.

Algorithm 1 Deterministic Coflow Scheduling Algorithm

Suppose Coflows $\left\{d_{ij}^k\right\}_{i,j=1}^N$ for $k \in \mathcal{K}$ with release dates r_k, $k \in \mathcal{K}$, and weights w_k, $k \in \mathcal{K}$, are given.

1: Solve the linear program (LP) and denote optimal solution by $\{\tilde{f}_k; k \in \mathcal{K}\}$.
2: Order and re-index coflows such that:

$$\tilde{f}_1 \le \tilde{f}_2 \le \ldots \le \tilde{f}_K, \tag{5}$$

where ties are broken arbitrarily.
3: Wait until the first coflow(s) is released.
4: **while** There is some incomplete flow, **do**
5: List the released and incomplete flows respecting the ordering in (5). Let L be the total number of flows in the list.
6: **for** $l = 1$ to L **do**
7: Denote the l-th flow in the list by (i_l, j_l, k_l),
8: **if** Both the links i_l and j_l are unused, **then**
9: Schedule flow (i_l, j_l, k_l).
10: **end if**
11: **end for**
12: **while** No flow is complete and no coflow is released **do**
13: Transmit the flows that get scheduled in line 9 with rate 1.
14: **end while**
15: **end while**

Now we present a sketch of the proof of Theorem 3.1 and Corollary 3.2 regarding performance of Algorithm 1.

Proof of Theorem 3.1. Denote by f_k completion time of coflow k under Algortihm 1. Suppose flow (i, j, k) is the last flow of coflow k that is completed. In general, Algorithm 1 may preempt a flow several times during its execution. For now, suppose flow (i, j, k) is not preempted and use t_k to denote the time when its transmission is started (the arguments can be easily extended to the preemption case as we show at the end of the proof). Therefore

$$f_k = f_{ij}^k = t_k + d_{ij}^k \tag{6}$$

From the algorithm description, t_k is the first time both links i and j are available and there is no flow from i to j before flow (i, j, k) in the list to be scheduled. By definition of $W(k)$ (Equation (4)), node i (similarly node j) has at most $W(k) - d_{ij}^k$ data flow to send by time t_k. Recall that capacity of all links are normalized to 1. Hence,

$$t_k \le r_k + W(k) - d_{ij}^k + W(k) - d_{ij}^k.$$

Combining this inequality with equality (6) yields that: $f_k \le r_k + 2W(k)$. Using Lemma 4.1 and constraint (1d), we can conclude that

$$f_k \le 5\tilde{f}_k,$$

which implies that

$$\sum_{k=1}^K w_k f_k \le 5 \sum_{k=1}^K w_k \tilde{f}_k.$$

This shows approximation ratio of 5 for Algorithm 1 using Lemma 4.2. Finally, if flow (i, j, k) is preempted, the above argument can still be used by letting t_k to be the starting time of its last piece and d_{ij}^k to be the remaining size of its last piece at time t_k. This completes the proof. □

Proof of Corollary 3.2. When all coflows are released at time 0, $t_k \le W(k) - d_{ij}^k + W(k) - d_{ij}^k$. The rest of the argument is similar. Therefore, the algorithm has approximation ratio of 4 when all coflows are release at time 0. □

REFERENCES

[1] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.
[2] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 443–454. ACM, 2014.
[3] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[4] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *SODA*, volume 96, pages 142–151, 1996.
[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.
[6] S. Khuller and M. Purohit. Brief announcement: Improved approximation algorithms for scheduling co-flows. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 239–240. ACM, 2016.
[7] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li. Towards practical and near-optimal coflow scheduling for data center networks. 2016.
[8] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.
[9] C. Potts. An algorithm for the single machine sequencing problem with precedence constraints. In *Combinatorial Optimization II*, pages 78–87. Springer, 1980.
[10] Z. Qiu, C. Stein, and Y. Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 294–303. ACM, 2015.
[11] M. Shafiee and J. Ghaderi. An improved bound for minimizing the total weighted completion time of coflows in datacenters. *arXiv preprint arXiv:1704.08357*, 2017.
[12] M. Shafiee and J. Ghaderi. Scheduling coflows in datacenter networks: Improved bound for total weighted completion time. *ACM SIGMETRICS, Poster Paper*, 2017.
[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. IEEE, 2010.
[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.