Near Optimal Coflow Scheduling in Networks

Mosharaf Chowdhury*
mosharaf@umich.edu
University of Michigan, Ann Arbor

Samir Khuller†
samir.khuller@northwestern.edu
Northwestern University

Manish Purohit
mpurohit@google.com
Google, Mountain View

Sheng Yang†
styang@cs.umd.edu
University of Maryland, College Park

Jie You*
jieyou@umich.edu
University of Michigan, Ann Arbor

ABSTRACT

The coflow scheduling problem has emerged as a popular abstraction in the last few years to study data communication problems within a data center [6]. In this basic framework, each coflow has a set of communication demands and the goal is to schedule many coflows in a manner that minimizes the total weighted completion time. A coflow is said to complete when all its communication needs are met. This problem has been extremely well studied for the case of complete bipartite graphs that model a data center with full bisection bandwidth and several approximation algorithms and effective heuristics have been proposed recently [1, 2, 29].

In this work, we study a slightly different model of coflow scheduling in general graphs (to capture traffic between data centers [15, 29]) and develop practical and efficient approximation algorithms for it. Our main result is a randomized 2 approximation algorithm for the single path and free path model, significantly improving prior work. In addition, we demonstrate via extensive experiments that the algorithm is practical, easy to implement and performs well in practice.

CCS CONCEPTS

• **Mathematics of computing** → **Approximation algorithms**; *Graph algorithms*; • **Networks** → **Packet scheduling**; **Network control algorithms**; • **Theory of computation** → **Scheduling algorithms**; **Routing and network design problems**; **Linear programming**; *Network flows*; *Rounding techniques*; • **Computer systems organization** → **Cloud computing**;

KEYWORDS

coflow, scheduling, LP relaxation, network flow, LP rounding, cloud computing

*Research supported by the National Science Foundation (CNS-1563095 and CNS-1617773)

†Research partially supported by an Amazon research award.

SPAA '19, June 22–24, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6184-2/19/06.
https://doi.org/10.1145/3323165.3323179

ACM Reference Format:
Mosharaf Chowdhury, Samir Khuller, Manish Purohit, Sheng Yang, and Jie You. 2019. Near Optimal Coflow Scheduling in Networks. In *31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '19), June 22–24, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3323165.3323179

1 INTRODUCTION

Modern computing applications have rather intensive computational needs. Many machine learning applications require up to tens of thousands of machines and often involve processing units across multiple data centers collaborating on the same application. This collaboration is usually handled by a large-scale distributed computing framework that ideally ensures a close-to-linear speedup compared to a single machine. A crucial part of the collaboration is that large chunks of data require both inter and intra-datacenter transmissions.

For intra-datacenter transmission, a common example would be the MapReduce framework. Map workers write all intermediate results independently to several servers to guard against failure and allow possible re-calculation. These results are shuffled and sent to Reduce workers. The volume of transmission between machines is so large that it has become a major bottleneck in the performance. In addition to this challenge, multiple applications may share the same cluster, and an un-coordinated schedule of their data transmission may cause an unacceptable delay in their completion times.

Chowdhury and Stoica [6] first introduced the abstraction of *coflow scheduling*, which assumes that each application consists of a set of flows, and is finished once all the flows are completed. In their framework the network between machines is modeled as a switch: the input ports of different machines on one side, and output ports on the other side. A machine can send (receive) data to (from) any other machine, but to (from) only one machine at a time (sending and receiving may happen concurrently). The transmission speed between all machines is uniform. This describes a "perfect" datacenter where networking between machines is handled by a high-speed central switch (modeled by a complete bipartite graph) connected directly to all the machines [6]. However, real world datacenters are far more complicated; direct (virtual) links between machines may exist to avoid latency, duplicate links may exist to tolerate failure, network speeds may vary widely for different machines and links, and complicated network structures may exist for a variety of reasons. To make things worse, some tasks may involve multiple datacenters around the globe, and the switch model

simply cannot accurately capture the graph based network that connects all the data centers.

For inter-datacenter transmission, distributed machine learning tasks can generate huge amounts of traffic. Due to legal or cost reasons, some datasets cannot be gathered into a single datacenter for processing. Instead, several geographically distributed datacenters work together to train a single model, and exchange local updates frequently to ensure accuracy and convergence. Though the size of a single transmission may be small considering the network bandwidth, the repeated exchange blows up the volume of transmission and makes network traffic its bottleneck.

In order to solve these problems, a slightly different model of coflow scheduling was proposed by Jahanjou et al. [15], which assumes that the underlying connection between machines is an arbitrary graph rather than a complete bipartite graph. Each node can be a machine, a datacenter or an exchange point (switch, router, etc.), and an edge between two nodes represents a physical link between the two Internet infrastructures. When some data needs to be transmitted from one node to another, it needs to be transmitted along edges. Unlike in the switch model where only one packet can be sent at each time slot, data for multiple jobs is allowed to transfer on the same link at the same time, or in other words, shared traffic on links is allowed. The total volume of data transmission on a link however is bounded by the link bandwidth[1]. Jahanjou et al. [15] considered the model in which data has to travel along a single specified path. In addition to this model, we also consider the *free path* model which allows data to be split or merged at nodes to utilize the whole graph when transmitting the same piece of data as long as the capacity of each link is respected. This seems much more complicated in practice than a single path transmission, but modern distributed computation frameworks [29] allow this kind of fine-grained control on network routing and transfer rate, which makes the model realistic. See Figure 1 for a brief illustration of the two models. The formal definitions come in Section 2.

1.1 Related Work

The idea of scheduling coflows was first introduced by Chowdhury and Stoica [6]. Since then, it has been a hot topic in both the systems [7, 19, 20, 30] and the theory [2, 15, 18, 24, 28] communities. Most theoretical research has focused on coflow scheduling in the switch model, where the communication graph is a complete, bipartite graph. Since this basic problem generalizes concurrent open shop scheduling and is thus NP-hard, the main results focus on the development of approximation algorithms. Over the last three years, a series of papers [2, 18, 24] have brought down the approximation factor from 67/3 to 5 for coflow scheduling with arbitrary release times and to 4 for the case without release times [2, 28] [2]. We would like to note that a very simple primal-dual framework is proposed by Ahmadi et al. [2], and this yields a very practical combinatorial algorithm for the problem without requiring the need to solve an LP (as in [28]). Furthermore, in recent work, a system called Sincronia [1] was also developed based on the primal-dual method. It improves upon state-of-the-art methods and gives practical and near-optimal solutions in real testbeds.

One natural extension is to take the graph structure into consideration. Zhao et al. [31] consider coflow scheduling over arbitrary graphs and attempt to jointly optimize routing and scheduling. They give a heuristic based on shortest job first, and use the idle slots to schedule flows from the longest job. Jahanjou et al. [15] studied two variants of coflow scheduling over general graphs, namely, when the path for a flow is given or if the path is unspecified. In both cases, the transmission rate may change over time, but each flow can only take a single path, whether given to or chosen by the fractional routing algorithm. In the first case, Jahanjou et al. [15] develop the first constant approximation algorithm (approximation ratio 17.6) and in the second case they develop an $O(\frac{\log n}{\log\log n})$ approximation algorithm (n is the number of nodes in the graph), matching the lower bound given by Chuzhoy et al. [8].

Since preemption often incurs large overheads, some recent work [30] has tackled the problem of non-preemptive coflow scheduling. Mao, Aggarwal, and Chiang [21] consider the non-preemptive coflow scheduling problem with stochastic sizes and give an algorithm with an approximation factor of $(2\log m + 1)(1 + \sqrt{m}\Delta)(1 + m\Delta)(3 + \Delta)/2$, where Δ is an upper bound of squared coefficient of variation of processing times. This simplifies to a $(3\log m + \frac{3}{2})$ approximation for non-stochastic cases.

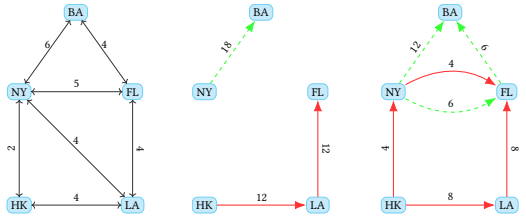

Figure 1: Example of coflow. The first graph shows the network topologies and the bandwidth of each link. We have one coflow consisting of two flows: one from NY to BA of demand 18 (denoted with dashed, green lines), the other from HK to FL of demand 12 (denoted with solid, red lines). The second graph shows the single path model, where each flow needs to be transmitted along a given path. It also implies a schedule in this model: transmit according to the path for 3 time units, and both flows are done. The third graph shows the free path model, where each flow can be split along multiple paths as long as the capacity of edges are respected. Here both flows can share the link from NY to FL and the entire coflow finishes in 2 units of time.

1.2 Our Contributions

The main result of this paper is a unified, tight 2-approximation algorithm for the coflow scheduling problem in both the single path model and the free path model when all release times and demands are polynomially sized, and a $(2+\epsilon)$-approximation when the release times and demands can be super-polynomial. This improves upon

[1] One major challenge in the switch model is the node-wise I/O speed constraint. In order to capture this in the graph model, we can replace every datacenter with a gadget of two nodes. The first node has exactly the same neighbors and edges that the original node for the datacenter has, plus links from and to the second node. The second node is only connected to the first node, and is the true source and destination for all demands involving this datacenter. By setting capacity on the links between these two nodes, we can enforce I/O limit for the whole datacenter like in the switch model.

[2] 4 is still the best known bound.

the 17.6 approximation given by Jahanjou et al. [15] for the single path model, and is the first approximation algorithm for the free path model (introduced by You and Chowdhury [29]).

We also evaluated our algorithm using two WAN topologies (Microsoft's SWAN [12] and Google's G-Scale [16]) on four different workloads (BigBench [14], TPC-DS [22], TPC-H [23], and Facebook (FB) [5, 9]) and compared with state-of-the-art for both models[15, 29]. For the single path model, we significantly improved over Jahanjou et al. [15]. For the free path model, we are close to what Terra [29] gets, but have the extra capability of dealing with weights. Across all variants and models, we have shown that taking the LP solution directly is an effective heuristic in practice.

1.3 Paper Organization

In Section 2 we give a formal definition of the two models for coflow scheduling. In Section 3 we give a general linear program that deals with both models. We give the additional flow constraints for the two models in Section 3.1. In Section 4.1 we describe the main algorithm and present the analysis in Section 4.2. We prove both models to be NP-hard in Section 5. In Section 6, we show experimental results by comparing our algorithms to some baseline algorithms. We conclude in Section 7 with some new directions to work on.

2 MODEL AND PROBLEM DEFINITION

We now formally define the models of coflow scheduling that we consider in this paper. Let $G = (V, E)$ be a directed graph that represents the data center network and $c : E \to \mathbb{R}^+$ be a function that denotes the capacity (bandwidth) available on each edge of the network. Let $\mathcal{J} = \{F_1, F_2, \ldots, F_n\}$ denote the set of n coflows. A coflow F_j has weight w_j that denotes its priority and consists of n_j individual flows, i.e., $F_j = \{f_j^1, \ldots, f_j^{n_j}\}$ where $f_j^i = (s_j^i, t_j^i, \sigma_j^i)$ denotes a flow from source node $s_j^i \in V$ to sink $t_j^i \in V$ with demand $\sigma_j^i \in \mathbb{R}^+$. We assume that time is discrete and data transfer is instantaneous, i.e., it takes negligible time for data to cover multiple hops of edges as network delay is low compared to the time to transmit large chunks of data. A coflow F_j is said to be completed at the earliest time t such that for each flow $f_j^i \in F_j$, σ_j^i units of data have been transferred from source s_j^i to sink t_j^i. Our goal is to find a schedule that routes all the requisite flows (i.e. at any time, what fraction of a certain flow is transmitted and along which path/paths) subject to the edge bandwidth constraints so that the total weighted completion time of the coflows $\sum_j w_j C_j$ is minimized. Figure 2 gives an example of an instance of the coflow scheduling problem over a simple network.

We consider *two* different transmission models, based on whether a flow f_j^i has restrictions as to how the data is transmitted. In the *single path model*, each flow f_j^i specifies a path p_j^i from source $s_j^i \in V$ to sink $t_j^i \in V$ so that the flow can only be routed along that path. This is exactly the "circuit-based coflows with paths given" model studied by Jahanjou *et al.* [15].

In the *free path model*, we can freely select the routing we desire for any flow f_j^i. In any time slot, data transmission occurs as a feasible multi-commodity flow so that both flow-conservation and edge bandwidth constraints are satisfied. Thus, we can split any flow f_j^i along multiple paths from its source to destination. This model was proposed in Terra [29]. Since the shortest paths of different flows can share edges and cause congestion, the free path model offers the flexibility of rerouting flows along less congested paths. In addition, modern internet infrastructures support using multiple paths together to get a higher overall speed (known as link aggregation), which is captured in the free path model as network flow.

In fact, both models are handled *uniformly* by the same framework, and the only difference is the set of flow constraints that describe what are considered feasible transmissions. It is also possible to handle other kinds of transmissions, like an intermediate case between single path and free path: several paths are given, and we can use them together and decide at what rate we are transmitting along each path. Figures 3 and 4 show the optimal solutions for the example coflow problem in the single path and free path models respectively.

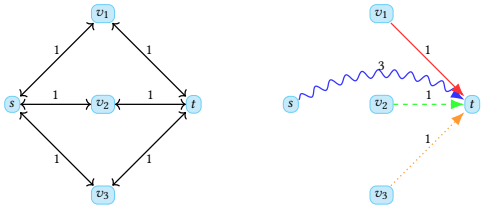

Figure 2: On the left is the graph structure: bi-directed edge of independent capacity of 1, on the right is the demanded coflow. There are four coflows each containing one single flow: red (solid) from v_1 to t, green (dashed) from v_2 to t, orange (dotted) from v_3 to t, and blue (curly) from s to t. The first three have demand 1, while the blue coflow has a demand of 3. All of them have the same weight of 1.

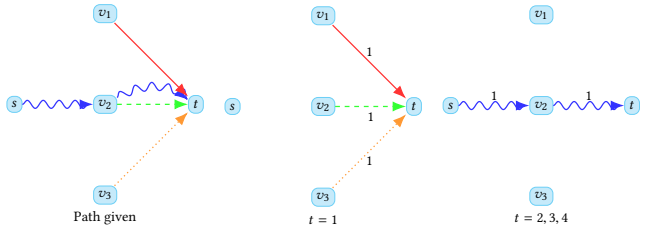

Figure 3: For the *single path* model, we have the path assignment in the left figure. Notice the path for green (dashed) flow shares an edge with that for the blue (curly) flow. Here is one optimal solution for the *single path model*. The total weighted completion time is $1 + 1 + 1 + 4 = 7$.

3 LINEAR PROGRAMMING RELAXATION

We use a time-indexed linear program to model this problem. Let T denote an upper bound on the total time required to schedule all the coflows. Note that T might be super-polynomial if the release times or coflow sizes are large. However, there is a standard technique

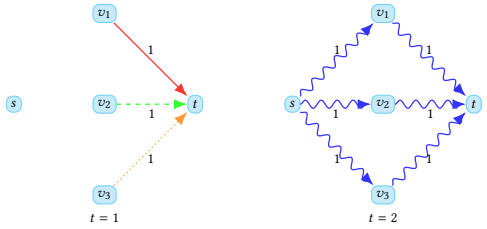

Figure 4: This is the optimal solution in the *free path model*. At time 1, send the red (solid), green (dashed), and orange (dotted) coflows. At time 2, send the blue (curly) coflow on all paths. The total weighted completion time is $1 + 1 + 1 + 2 = 5$.

that achieves polynomial size at the cost of a $(1 + \epsilon)$ factor on approximation ratio. We will assume T to be polynomial in the main paper, and present the fix for super-polynomial T in Appendix A.

Let time be slotted and time slot t cover the interval of time $[t-1, t]$. For a given flow f_j^i and a time slot t, we introduce the variable $x_j^i(t)$ to indicate the fraction of flow f_j^i that is scheduled at time t. For each coflow F_j, we introduce variables $X_j(t)$ to indicate if all the flows $f_j^i \in F_j$ have been completely scheduled by time t. Finally, we introduce a variable C_j that models the completion time of coflow F_j.

To make the linear program compatible with both single path model and free path model, we exclude the flow constraints and edge bandwidth constraints for now and delay them to Section 3.1.

$$\text{Minimize} \quad \sum_j w_j C_j, \qquad \text{subject to}$$

$$\sum_t x_j^i(t) = 1 \qquad \forall j \in [n], \forall i \in [n_j] \qquad (1)$$

$$X_j(t) \le \sum_{\ell=1}^{t} x_j^i(\ell) \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in T \qquad (2)$$

$$C_j \ge 1 + \sum_t (1 - X_j(t)) \qquad \forall j \in [n] \qquad (3)$$

$$r_j^i \ge t \Rightarrow x_j^i(t) = 0 \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in T \qquad (4)$$

$$x_j^i(t) \ge 0 \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in T \qquad (5)$$

Constraint (1) certifies that each flow is fully scheduled. Constraint (2) ensures that coflow F_j is considered completed at time t only if all flows $f_j^i \in F_j$ have been fully scheduled by time t. In Proposition 3.1, we show that Constraint (3) enforces a valid lower bound on the completion time of coflow F_j. Finally, Constraint (4) ensures that no flow is scheduled before it has been released. Note this is not a typical LP relaxation, since any fractional solution is valid. The main relaxation is around the completion time, since representing the exact completion time of job is beyond the capability of a linear program.

Proposition 3.1. *The completion time of a coflow F_j can be lower bounded by $C_j \ge 1 + \sum_t (1 - X_j(t))$ where $X_j(t) \in [0, 1]$ denotes the fraction of coflow F_j that has been completed by (the end of) time slot t.*

Proof. Conventionally, in time-indexed linear programming relaxations, the completion time of a job j is lower bounded by the fractional completion time in the schedule, or $C_j = C_j \cdot \sum_{t=1}^{T} x_j(t) \ge \sum_{t=1}^{T} t \cdot x_j(t)$. In our setting, this corresponds to the constraint $C_j \ge \sum_t t \cdot x_j(t)$ where $x_j(t) = X_j(t) - X_j(t-1)$ denotes the fraction of coflow F_j that is scheduled during time slot t. The desired constraint in Eq (3) is exactly the same constraint rearranged in a format that is more convenient for analysis.

$$\begin{aligned} C_j &\ge \sum_{t=1}^{T} t \cdot x_j(t) = \sum_{t=1}^{T} x_j(t) \sum_{\tau=1}^{t} 1 \\ &= \sum_{\tau=1}^{T} \sum_{t \ge \tau}^{T} x_j(t) = \sum_{\tau=1}^{T} \left(\sum_{t=1}^{T} x_j(t) - \sum_{t=1}^{\tau-1} x_j(t) \right) \\ &= \sum_{\tau=1}^{T} (1 - X_j(\tau - 1)) = \sum_{\tau=0}^{T-1} (1 - X_j(\tau)) = 1 + \sum_{\tau=1}^{T-1} (1 - X_j(\tau)) \end{aligned}$$

□

3.1 Model-specific Constraints

3.1.1 Single Path Model. In the single path model, a flow f_j^i can only be routed along a specified path p_j^i. Thus, we do not need to make any routing decisions in the linear program and only need to ensure that edge bandwidths are respected.

$$\sum_{p_j^i \ni e} x_j^i(t) \cdot \sigma_j^i \le c(e), \qquad \forall e \in E, \forall t \in T \qquad (6)$$

Constraint (6) enforces that the total flow scheduled through edge e at any time slot t does not exceed the edge bandwidth. Constraints (1)-(6) thus form the complete linear programming relaxation for coflow scheduling in the single path model.

3.1.2 Free Path Model. In the free path model, the path for flow f_j^i is not specified. In fact, data can split and merge at vertices to utilize all possible capacity. We use variable $x_j^i(t, e)$ to denote the fraction of flow f_j^i transmitted through edge e in time slot t. Recall that we use $x_j^i(t)$ to denote the total fraction of flow f_j^i that is transmitted in time slot t. $\delta_{in}(v)$ (δ_{out}) represents the set of edges that comes in (out of) vertex v. Here are the flow conservation constraints we need.

$$\sum_{e \in \delta_{out}(s_j^i)} x_j^i(t, e) = x_j^i(t), \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in T \qquad (7)$$

$$\sum_{e \in \delta_{in}(t_j^i)} x_j^i(t, e) = x_j^i(t), \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in T \qquad (8)$$

$$\sum_{e \in \delta_{in}(v)} x_j^i(t, e) = \sum_{e \in \delta_{out}(v)} x_j^i(t, e), \quad \forall j \in [n], \forall i \in [n_j], \forall t \in T, \forall v \in V \setminus \{s_j^i, t_j^i\} \qquad (9)$$

$$\sum_{j \in [n], i \in [n_j]} x_j^i(t, e) \cdot \sigma_j^i \le c(e), \qquad \forall t \in T, \forall e \in E \qquad (10)$$

Constraints (7) and (8) enforce that the total fraction of flow f_j^i satisfied at time t over all the paths is exactly $x_j^i(t)$. Constraints (9)

ensure flow conservation at all nodes other than source and sink. Constraints (10) guarantee that all edge bandwidths are satisfied at all time steps. Constraints (1)-(5) and (7)-(10) thus form the complete linear programming relaxation for coflow scheduling in the free path model.

Let C_j^* denote the completion time of coflow F_j in an optimal solution of the LP relaxation, and let $C_j(opt)$ denote the completion time of coflow F_j in the corresponding optimal integral solution. Thus, for both the models, we have

$$\sum_j w_j C_j^* \leq \sum_j w_j C_j(opt). \tag{11}$$

4 APPROXIMATION ALGORITHMS

Let $x_j^i(t)$ denote the fraction of flow f_j^i that is scheduled at time step t in an optimal solution to the above LP. The LP constraints guarantee that this yields a feasible schedule to the coflow scheduling problem (in both the single path as well as the free path models). However, since the completion time of a coflow F_j is defined as the earliest time t such that all flows $f_j^i \in F_j$ have been completely scheduled, the true completion time of coflow F_j obtained in this scheduled is given by

$$C_j(LP\ Sched) = \max_i \{ \max_{t: x_j^i(t) > 0} [t] \}. \tag{12}$$

Unfortunately, this completion time $C_j(LP\ Sched)$ can be much greater than the completion time variable in the optimal LP solution C_j^*, and thus the obtained schedule is not a constant-approximate coflow schedule. For instance, consider a coflow F_j with only one flow ($n_j = 1$) and let the optimal LP solution set its schedule as follows $x_j^1(1) = 0.9, x_j^1(10) = 0.1$, and $x_j^1(t) = 0, \forall t \notin \{1, 10\}$. Now, the completion time variable in the optimal LP solution is $C_j^* = \sum_t t x_j^1(t) = 1.9$. However, true completion time of the coflow F_j in such a schedule is $C_j(LP\ Sched) = 10 \gg C_j^*$.

To overcome the obstacle above, we propose the following algorithm called Stretch (see Section 4.1) that modifies the schedule obtained by the linear program so that the completion time of each coflow in the modified schedule can be compared with the completion time variable of the corresponding coflow in an optimal LP solution. The schedule "stretching" idea (also called 'slow-motion') used in our algorithm has been used before successfully in other scheduling contexts [13, 25, 27].

4.1 Stretch Algorithm

(1) Solve the linear program in Section 3 and obtain a fractional optimal solution.
(2) Let $\lambda \in (0, 1)$ be drawn randomly according to the p.d.f $f(v) = 2v$. We can verify that this is indeed a valid probability distribution.
(3) Stretch the LP schedule by $\frac{1}{\lambda}$. This means that we schedule everything exactly as per the LP solution - but whatever LP schedules in the interval $[a, b]$, we will schedule in the interval $[\frac{a}{\lambda}, \frac{b}{\lambda}]$.
(4) Once σ_j^i units of flow f_j^i have been scheduled, leave the remaining slots for f_j^i empty.

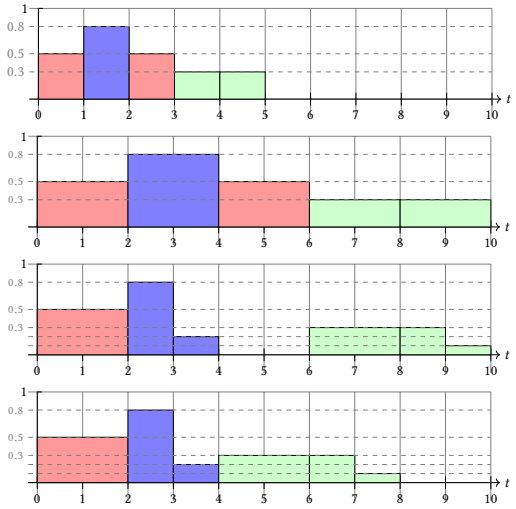

Figure 5: Here we show an example solution obtained from the LP, different color indicate different flows. In the second picture, we stretch with $\lambda = 0.5$. In the third picture, we leave the slots empty if the corresponding flow is finished. In the fourth picture, we utilize the idle slots and move some flows to earlier times. Though this does not improve the theoretically bound, it is beneficial in practice and is used in our experimental evaluation.

Figure 5 illustrates the key ideas of the algorithm. To help understand this algorithm, start with the simple case where we have a fixed $\lambda = 0.5$, in other words stretch the time axis by a factor of $1/\lambda = 2$. Intuitively, we move everything at time slot t and to both time slots $2t - 1$ and $2t$. What used to be transmitted at time t will be transmitted no later than time $2t$. Consider any flow f_j^i and let τ denote the earliest time by which the LP has scheduled at least $1/2$ fraction of the flow. Then, it is easy to verify that the flow f_j^i is completely scheduled by time 2τ.

Now we consider a general λ and prove that this algorithm does output a feasible schedule. Due to fractional λ, it might be the case that some flow f_j^i of LP variable $x_j^i(t)$ in integral interval $[t-1, t]$ becomes $[\frac{t-1}{\lambda}, \frac{t}{\lambda}]$, a fractional interval. In this case, for a time slot τ, or a interval $[\tau - 1, \tau]$ after stretching, we just add $x_j^i(t) \cdot |[\tau - 1, \tau] \cap [\frac{t-1}{\lambda}, \frac{t}{\lambda}]|$.

The only flows that might be scheduled in time slot τ are those scheduled in time slot $1 + \lfloor \lambda(\tau - 1) \rfloor$ and $1 + \lfloor \lambda\tau \rfloor$ before stretching, or flows $f_j^i(1 + \lfloor \lambda(\tau - 1) \rfloor)$ and flows $f_j^i(1 + \lfloor \lambda\tau \rfloor)$. (The two time slots might be the same. If so, feasibility is automatically met. Otherwise, we have $1 + \lfloor \lambda(\tau - 1) \rfloor + 1 = 1 + \lfloor \lambda\tau \rfloor$.) For all flows at time $1 + \lfloor \lambda(\tau - 1) \rfloor$ before stretching, the factor we multiplied with is $w_1 = \left| [\tau - 1, \tau] \cap [\frac{\lfloor \lambda(\tau-1) \rfloor}{\lambda}, \frac{1+\lfloor \lambda(\tau-1) \rfloor}{\lambda}] \right|$. For all flows at time $1 + \lfloor \lambda\tau \rfloor$ before stretching, the factor we use to multiply with is $w_2 = \left| [\tau - 1, \tau] \cap [\frac{\lfloor \lambda\tau \rfloor}{\lambda}, \frac{1+\lfloor \lambda\tau \rfloor}{\lambda}] \right|$. Note $w_1 + w_2 = 1$. In fact, the schedule at time τ can be viewed as a weighted average of the schedule at time $[\lfloor \lambda(\tau - 1) \rfloor, 1 + \lfloor \lambda(\tau - 1) \rfloor]$ and $[\lfloor \lambda\tau \rfloor, 1 + \lfloor \lambda\tau \rfloor]$ (if $\lambda(\tau - 1)$ is a integer, then the schedule will be exactly what it

used to be at time $\lambda\tau$), the first with weight w_1 and the second with weight w_2. The nature of network flow ensures that the weighted sum of two feasible flows is a feasible flow.

Another fact that needs proof is that every flow is finished. This is guaranteed since schedules are stretched, and we only leave the remaining slots empty for f_j^i if σ_j^i units of flow have been scheduled, or in other words, all the demand for this flow has been scheduled.

4.2 Analysis

Recall that C_j^* denotes the completion time of coflow F_j in the optimal LP solution. While we consider that time is slotted in the LP formulation and time slot t covers the interval of time $[t-1, t]$, at this stage it is more convenient to work with continuous time rather than discrete time. For any continuous time $\tau \in [0, T]$, define $X_j(\tau)$ to be the fraction of coflow F_j that has been scheduled in the LP solution by time τ. We define $X_j(\tau)$ by assuming that the flow is scheduled at an uniform rate in every time slot. Formally, we have

$$X_j(\tau) = X_j(\lfloor\tau\rfloor) + (\tau - \lfloor\tau\rfloor)\left(X_j(\lfloor\tau\rfloor + 1) - X_j(\lfloor\tau\rfloor)\right). \tag{13}$$

The LP constraints (3) guarantee that for any coflow F_j, we have $C_j^* \geq 1 + \sum_t (1 - X_j(t))$. We can now lower-bound the LP completion time by replacing the above summation by an integral.

Lemma 4.1. *$\int_{\tau=0}^{T}(1 - X_j(\tau))d\tau \leq C_j^* - \frac{1}{2}$ where $X_j(\tau)$ is defined as per Eq. (13).*

Proof. By definition of $X_j(\tau)$, we have the following.

$$\begin{aligned}
\int_{\tau=0}^{T}(1-X_j(\tau))d\tau &= T - \int_{\tau=0}^{T} X_j(\tau)d\tau \\
&= T - \sum_{t=0}^{T-1}\int_{\tau=t}^{t+1} X_j(\tau)d\tau \\
&= T - \sum_{t=0}^{T-1}\int_{\tau=t}^{t+1}\left[X_j(t) + (\tau - t)\left(X_j(t+1) - X_j(t)\right)\right]d\tau \\
&= T - \sum_{t=0}^{T-1}\left[X_j(t) + \left(X_j(t+1) - X_j(t)\right)\int_{\tau=t}^{t+1}(\tau - t)d\tau\right] \\
&= T - \sum_{t=0}^{T-1}\frac{1}{2}\left[X_j(t) + X_j(t+1)\right] \\
&= T - \left[\frac{1}{2}\left(X_j(0) + X_j(T)\right) + \sum_{t=1}^{T-1} X_j(t)\right]
\end{aligned}$$

Since by definition, $X_j(0) = 0$ and $X_j(T) = 1$, we get

$$= T - \left[\frac{1}{2} + \sum_{t=1}^{T-1} X_j(t)\right]$$

Rearranging the terms, we get

$$= 1 + \sum_{t=1}^{T-1}(1 - X_j(t)) - \frac{1}{2} \leq C_j^* - \frac{1}{2}$$

where the last inequality follows from Constraint (3). □

For any $\lambda \in [0, 1]$, define $C_j^*(\lambda)$ to be the earliest time τ such that λ fraction of the coflow F_j has been scheduled in the LP solution, i.e., in other words its the smallest τ such that $X_j(\tau) = \lambda$. Note that by time $C_j^*(\lambda)$, λ fraction of *every flow* $f_j^i \in F_j$ has been scheduled by the LP.

Proposition 4.2. $\int_{\lambda=0}^{1} C_j^*(\lambda)\mathbf{d}\lambda = \int_{\tau=0}^{T}(1 - X_j(\tau))\mathbf{d}\tau$

Proof.

$$\begin{aligned}
\int_{\lambda=0}^{1} C_j^*(\lambda)\mathbf{d}\lambda &= \int_{\lambda=0}^{1}\int_{\tau=0}^{T}\mathbb{1}_{[C_j^*(\lambda)>\tau]}\mathbf{d}\tau\mathbf{d}\lambda \\
&= \int_{\tau=0}^{T}\int_{\lambda=0}^{1}\mathbb{1}_{[C_j^*(\lambda)>\tau]}\mathbf{d}\lambda\mathbf{d}\tau \\
&= \int_{\tau=0}^{T}\int_{\lambda=X_j(\tau)}^{1} 1\mathbf{d}\lambda\mathbf{d}\tau = \int_{\tau=0}^{T}(1 - X_j(\tau))\mathbf{d}\tau
\end{aligned}$$

□

Finally, we are ready to bound the completion time of coflow F_j in the stretched schedule (denoted as $C_j(alg)$). For any fixed $\lambda \in (0, 1)$, since we stretch the schedule by a factor of $\frac{1}{\lambda}$, we have $C_j(alg) \leq \left\lceil \frac{C_j^*(\lambda)}{\lambda} \right\rceil$. Notice the ceiling function in the bound [3]. Since λ is drawn randomly from a distribution, the following lemma bounds the expected completion time of coflow F_j in the stretched schedule.

Lemma 4.3. *The expected completion time of any coflow F_j in the stretched schedule is bounded by $2C_j^*$.*

Proof.

$$\begin{aligned}
\mathbb{E}[C_j(alg)] &\leq \int_{\lambda=0}^{1} f(\lambda)\left\lceil\frac{C_j^*(\lambda)}{\lambda}\right\rceil d\lambda \leq \int_{\lambda=0}^{1}(2\lambda)\left(\frac{C_j^*(\lambda)}{\lambda} + 1\right)d\lambda \\
&= 2\int_{\lambda=0}^{1} C_j^*(\lambda)d\lambda + 1
\end{aligned}$$

By Lemma 4.1 and Proposition 4.2,

$$= 2\int_{\tau=0}^{T}(1 - X_j(\tau))\mathbf{d}\tau + 1 \leq 2\left(C_j^* - \frac{1}{2}\right) + 1 = 2C_j^*$$

□

Theorem 4.4 thus follows from the linearity of expectation.

Theorem 4.4. *There is a randomized 2-approximation algorithm for coflow scheduling in networks in both the single path and free path models when all release times and coflow sizes are polynomially sized.*

For the case where the total time we need to schedule all coflows is super-polynomial, we use the standard trick of geometric series time intervals, and claim the following theorem. Proof comes in Appendix A.

[3] All flows $f_j^i \in F_j$ were completed by at least λ fraction by time $C_j^*(\lambda)$. So in the stretched schedule, all those flows must be completed by time $\frac{C_j^*(\lambda)}{\lambda}$. The ceiling is necessary since $\frac{C_j^*(\lambda)}{\lambda}$ may be fractional (i.e. occur in the middle of a time slot)

Theorem 4.5. *For any $\epsilon > 0$, there is a randomized $(2+\epsilon)$-approximation algorithm for coflow scheduling in networks in both the single path and the free path models (with possibly super polynomial release times and demands).*

5 HARDNESS OF APPROXIMATION

We claim the following theorem:

Theorem 5.1. *For the coflow scheduling problem, in both the single path and the free path model, it is NP-hard to obtain a $(2-\epsilon)$ approximation, for any $\epsilon > 0$.*

Proof. We prove it by a reduction from concurrent open-shop problem (proved NP-hard to approximate within a factor better than $(2-\epsilon)$ [3, 26]). The definition of concurrent open shop problem is as follows: there are m machines and n jobs, each job j need to be processed on machine i for p_j^i time non-preemptively. We would like to minimize the total weighted completion time. Unlike the open shop problem, in the concurrent open shop problem a job can be processed on more than one machine at the same time.

Given a concurrent open-shop problem instance with M machines, we construct an instance of the coflow scheduling problem as follows. For every machine i, we have two nodes x_i and y_i, and an edge of unit bandwidth from x_i to y_i. Notice the graph has M different components, between each pair (x_i, y_i), there is only one path from x_i to y_i. Thus this construction works for both the single path model and the free path model. We will not distinguish the models in the following proof.

For a certain job j with demands σ_j^i in the concurrent open shop instance, we add a coflow j with demand of σ_j^i from x_i to y_i. Weights are directly taken from the concurrent open shop problem instance. Suppose we get a solution for this coflow scheduling instance, we can get a solution of no larger cost for the concurrent open shop instance as follows. If we have a flow f_j^i for job j on edge (x_i, y_i) of size $x_j^i(t)$ at time t, then we schedule a fraction of $x_j^i(t)$ for job j on machine i at time t. Suppose a flow f_j^i is finished at time C_j^i in the coflow scheduling problem, the corresponding concurrent open shop problem for job j and machine i is also finished at time C_j^i. Similarly, the finishing time C_j of coflow j and concurrent open shop job j are the same. However, the solution we get is fractional, and might be preemptive (we might pause a job and resume it later).

Now we prove that we can modify this solution to get a non-preemptive integral solution without raising the total weighted completion time. For each machine i, consider all completion times C_j^i. Sort them in non-decreasing order $C_{l_1}^i, C_{l_2}^i, \ldots, C_{l_J}^i$, and we can safely reschedule these demand in the order of $l_1, l_2, \ldots, l_j$, and get new completion times $\mathbb{C}_{l_1}^i, \ldots, \mathbb{C}_{l_j}^i$ while not raising any completion time. We know all demand of job l_1 on machine i has been finished by $C_{l_1}^i$, so $\mathbb{C}_{l_1}^i = d_{l_1,i} \le C_{l_1}^i$, similarly all demands of job l_1 and l_2 have been finished by $C_{l_2}^i$, and $\mathbb{C}_{l_2}^i = d_{l_1,i} + d_{l_2,i} \le C_{l_2}^i$. We can continue and get $\mathbb{C}_{l_j}^i \le C_{l_j}^i, \forall j \in \{J\}$. Thus the total weighted completion time for this integral solution would be upper bounded by the cost for the coflow scheduling instance.

$$\sum_{j\in\{J\}} w_j \cdot \mathbb{C}_j = \sum_{j\in\{J\}} w_j \cdot \max_{i\in\{M\}} \mathbb{C}_j^i \le \sum_{j\in\{J\}} w_j \cdot \max_{i\in\{M\}} C_j^i = \sum_{j\in\{J\}} w_j \cdot C_j$$

For the other direction, for a certain solution of a concurrent open-shop problem, if task i of job j is scheduled from time t_1 to time t_2, we make the flow f_j^i take up all bandwidth of edge (x_i, y_i) from time t_1 to time t_2. Then flow f_j^i is finished the same time when task i of job j is finished. Since every task i is finished the same time before and after reduction, completion times and the objective weighted completion time stays the same for the coflow scheduling problem.

In conclusion, for a solution SOL of concurrent open-shop problem with weighted completion time W, we can construct a solution SOL_{coflow} for coflow scheduling problem of the same weighted completion time W. For a solution SOL'_{coflow} of coflow scheduling problem with weighted completion time W', we can construct a solution SOL' for the original concurrent open-shop problem, with cost at most W'. Since concurrent open-shop problem is NP-hard to get a $(2-\epsilon)$ approximation, we know it is also NP-hard to approximate coflow scheduling problem to a factor of $(2-\epsilon)$, for both single path model and free path model. □

6 EXPERIMENTS

We evaluated the Stretch Algorithm on 2 topologies and 4 benchmarks/industrial workloads. Experiments were run on a machine with dual Intel(R) Xeon(R) CPU E5-2430, and 64GB of RAM, and using Gurobi [11] as the LP solver. We first discuss the experimental set up and then in Section 6.2 discuss what evaluation we performed.

WAN topology: We consider the following graph topologies.

(1) Swan [12]: Microsoft's inter-datacenter WAN with 5 datacenters and 7 inter-datacenter links. We calculate link bandwidth using the setup described by Hong et al.[12].
(2) G-Scale [16]: Google's inter-datacenter WAN with 12 datacenters and 19 inter-datacenter links.

Workloads: We use the following mix of jobs from public benchmarks - TPC-DS [22], TPC-H [23], and BigBench [14] - and from Facebook (FB) production traces [5, 9]. We follow [29] to set up the benchmarks: for a certain workload, jobs are randomly chosen and since they do not have a release time, we assign a release time similar to that in production traces. Each job lasts from a few minutes to dozens of minutes. Each benchmark experiment has 200 jobs. We randomly assign these jobs to nodes in the datacenter, and the demand will be between the corresponding nodes. Since weights are not available, we assign weights that are uniformly chosen from the interval between 1.0 and 100.0.

6.1 Implementation Details

In this subsection we discuss some details related to the implementation.

Time Index: There is a trade-off in selecting the size of a time slot. If the length of a time slot is shorter, we get more accurate answers, but need to solve a larger LP. On the contrary, if we make each time slot longer, the amount of computational resources need is greatly reduced, but the quality of the solution suffers. In all our experiments, we considered time slots of length 50 seconds as this led to tractable LP relaxations.

Rounding: Algorithm Stretch is meant for easy theoretical analysis, and is not a sophisticated rounding method; we are not trying to schedule later flows in the slots that are idle. This can cause huge overhead in experiments. See Figure 5 for an illustration. In our implementation, we deal with this issue by moving the schedule of every time slot t to an earlier idle slot t' if for all flows scheduled at t, its release time is before t'.

To address the random sampling of λ, we sample 20 times from the distribution mentioned in Section 4.1 to get the expected weighted completion time for Algorithm Stretch, and denote it with "Average λ". We also measure the best solution obtained over these random choices (denoted by "Best λ").

6.2 Baselines

LP-based Heuristic: In addition to algorithms with theoretically worst case guarantee, we also propose a heuristic that works well in practice. Recall in Section 4.1, we mentioned that the LP solution itself is a valid schedule. We can use this solution as a heuristic, for both the single path and free path models. Note the weighted completion time for this LP solution is *NOT* the same as the LP objective function, as explained in Section 4.1. This implies that the solution from the heuristic can be arbitrarily bad in the worst case. In practice, however, this proves to be a very effective algorithm that can be quite close to the lower bound we get from LP.

Jahanjou et al. (*Single path model*): Since path information is not available in the datasets, we randomly generate one for each flow. For a source sink pair (s_j^i, t_j^i), we randomly select one of the shortest paths as the path for flow f_j^i. For this model, we compare our algorithm with the algorithm presented by Jahanjou et al. [15]. Here is a brief description of their approach. First write an LP using geometric time intervals, then schedule each job according to the interval its α point (the time when α fraction of this job is finished) belongs to. A common reason for geometric time intervals is to avoid having a super-polynomial time horizon (a practical reason is to make the LP smaller), and a time series of $\{(1+\epsilon)^i\}$ is chosen where ϵ is close to 0. The closer ϵ is to 0, the better the approximation ratio can be. However, in Jahanjou et al.'s algorithm, the rounding step has a dependency on ϵ. To optimize the approximation ratio, ϵ is set to 0.5436. Our algorithm, on the contrary, is time slot based, and can be turned into a geometric series of time intervals by losing a factor of $(1+\epsilon)$. In experiments, we include both the case of $\epsilon = 0.2$ and the case of $\epsilon = 0.5436$ for completeness.

Terra (*Free path model*): For the flow-based model, we are comparing to the offline algorithm in Terra [29]. This algorithm only works for the unweighted case. It calculates the time for each single coflow to finish individually, and then schedule with SRTF (shortest remaining time first). Instead of one large LP like all other algorithms compared here, this algorithm solves a large number of LPs, twice the number of coflow jobs. Terra can work with very fine grained time, to the order of milliseconds (and does not need time to be slotted). Since there is no previous work on weighted case, we compare the weighted case with the LP solution and our heuristic directly based on time indexed LP.

6.3 Experimental Results

Impact of λ: See Figure 6 and Figure 7. When λ is 1.0, we take the LP solution directly (this is exactly the LP-based heuristic). Across all experiments, this seems the best choice of λ. The best sampled λ and the average case λ are pretty close, indicating the performance does not change much across different λ.

Impact of ϵ: To study the effect of the size of the time interval, we measure the LP objective and the schedule obtained by the LP-based heuristic as we vary ϵ in Figure 8. As ϵ increases, the size of the linear program will drop, making it faster to solve. On the other hand, the quality of solution drops, as we will not start a job until the whole current interval is after its release time, and will not consider a job finished until the interval its completion time belongs to ends. Thus a proper selection of ϵ may depend on the available computational resources for solving the LP.

Single Path Model: Figures 9 and 10 compare the performance of our algorithms with that of Jahanjou et al. [15] on all the benchmarks and topologies. Across all the experiments, we observe that our algorithms perform significantly better.

Free Path Model: See Figure 11 and Figure 12 for comparisons with the algorithm in Terra[29]. Since Terra only handles uniform coflow weights, we set all weights to be unit for these experiments. Surprisingly, we observe that Terra performs slightly better than even the LP objective itself. This disparity arises as the LP relies on time slots of 50 seconds while Terra deals with time slots of much finer granularity. For the weighted case, we are not aware of previous work, and only compare to LP solution in Figures 6 and 7.

7 CONCLUSION

In this paper we developed an efficient approximation algorithm for the coflow scheduling problem in general graph topologies. This algorithm is shown to be practical and one that delivers extremely high quality solutions. The new insight was to write a time indexed LP formulation and to convert it using the idea of stretching the schedule.

The next major challenge is developing *online* methods for coflow shceduling to minimize weighted flow time. Prior work [17] deals with the problem of minimizing weighted completion time by making use of offline approximation algorithms. However, the problem of minimizing weighted flow time is considerably more challenging. The technical difference is that flow time is defined as $C_j - r_j$ where C_j is the completion time of a job, and r_j the release time. Optimizing flow time non-preemptively even on a single machine (a different model) is a notoriously difficult problem with some recent progress [4, 10].

REFERENCES

[1] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. 2018. Sincronia: near-optimal network design for coflows. In *SIGCOMM*. ACM, 16–29.

[2] Saba Ahmadi, Samir Khuller, Manish Purohit, and Sheng Yang. 2017. On scheduling coflows. In *IPCO*. Springer, 13–24.

[3] Nikhil Bansal and Subhash Khot. 2010. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In *ICALP*. Springer, 250–261.

[4] Jatin Batra, Amit Kumar, and Naveen Garg. 2018. Constant Factor Approximation Algorithm for Weighted Flow Time on a Single Machine in Pseudo-polynomial time. In *FOCS*. IEEE.

[5] Mosharaf Chowdhury. 2015. Coflow Benchmark Based on Facebook Traces. Retrieved April 22, 2019 from https://github.com/coflow/coflow-benchmark

Figure 6: Free path model on SWAN, showing the performance bound of time indexed LP value, the performance of heuristic ($\lambda = 1$), best λ among samples, and the expected value when λ is chosen from the distribution mentioned in Section 4.1.

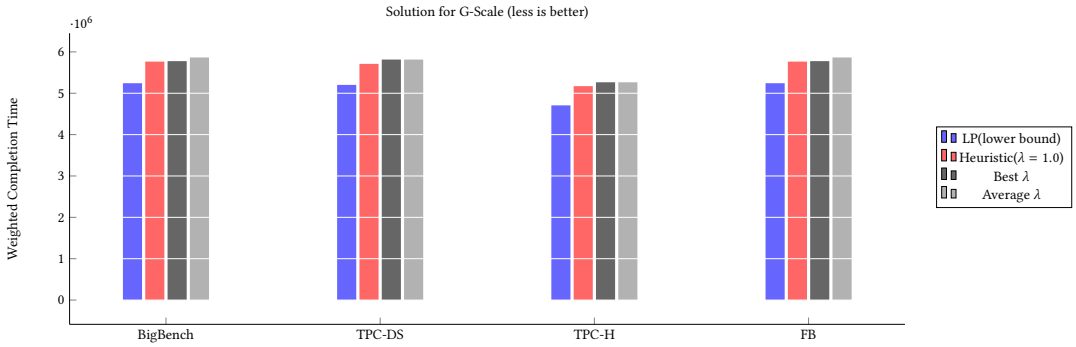

Figure 7: Free path model on G-Scale, showing the performance bound of time indexed LP value, the performance of heuristic ($\lambda = 1$), best λ among samples, and the expected value when λ is chosen from the distribution mentioned in Section 4.1.

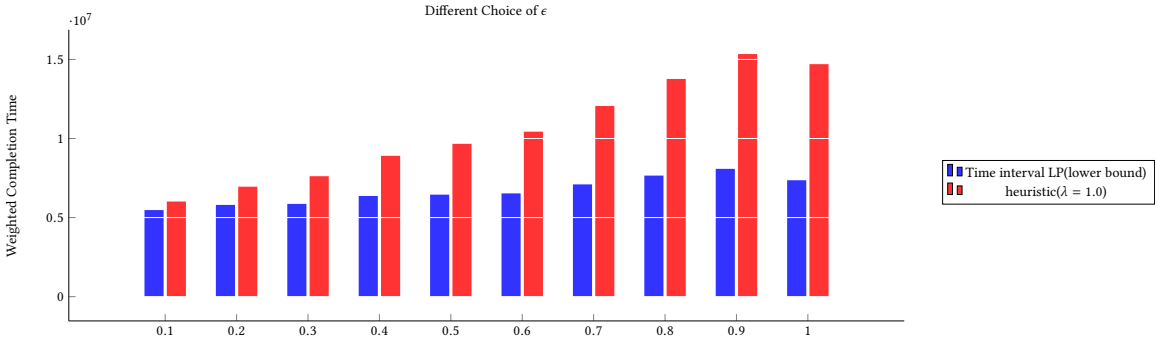

Figure 8: Free path model on SWAN for workload FB, the different choice of time interval ϵ may affect the performance bound of time interval LP value and the performance of heuristic ($\lambda = 1$).

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

[7] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient Coflow Scheduling with Varys. In *SIGCOMM*. ACM, New York, NY, USA, 443–454. https://doi.org/10.1145/2619239.2626315

[8] Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, and Kunal Talwar. 2007. Hardness of routing with congestion in directed graphs. In *STOC*. ACM, 165–178.

[9] Facebook. 2014. Statistical Workload Injector for MapReduce (SWIM). Retrieved April 22, 2019 from https://github.com/SWIMProjectUCB/SWIM

[10] Uriel Feige, Janardhan Kulkarni, and Shi Li. 2019. A Polynomial Time Constant Approximation For Minimizing Total Weighted Flow-time. (2019).

[11] Gurobi Optimization, LLC. 2018. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[12] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *SIGCOMM*, Vol. 43. ACM, 15–26.

[13] Sungjin Im, Maxim Sviridenko, and Ruben Van Der Zwaan. 2014. Preemptive and non-preemptive generalized min sum set cover. *Mathematical Programming* 145, 1-2 (2014), 377–401.

[14] Intel Hadoop. 2016. Big Data Benchmark for Big Bench. Retrieved April 22, 2019 from https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench

[15] Hamidreza Jahanjou, Erez Kantor, and Rajmohan Rajaraman. 2017. Asymptotically Optimal Approximation Algorithms for Coflow Scheduling. In *SPAA*. ACM, 45–54.

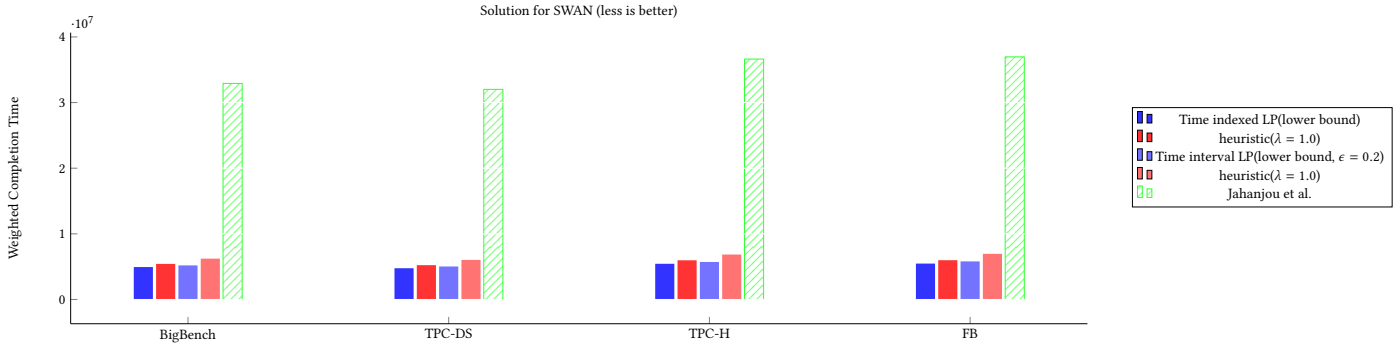

Figure 9: Single path model on SWAN, showing the performance bound of time indexed and time interval LP value, the performance of heuristic ($\lambda = 1$), best λ among samples, and the expected value when λ is chosen from the distribution mentioned in Section 4.1. Here we compare against algorithm by Jahanjou et al.[15].

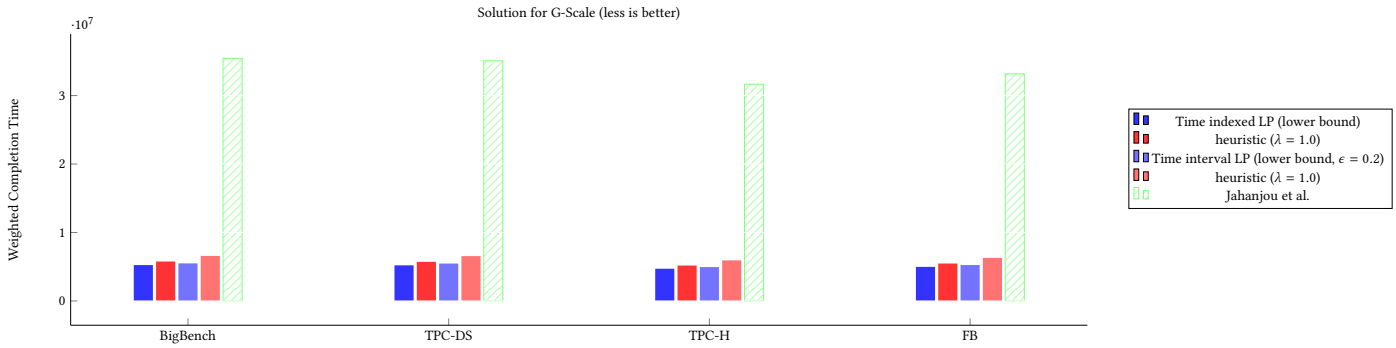

Figure 10: Single path model on G-Scale, showing the performance bound of time indexed and time interval LP value, the performance of heuristic ($\lambda = 1$), best λ among samples, and the expected value when λ is chosen from the distribution mentioned in Section 4.1. Here we compare against algorithm by Jahanjou et al.[15].

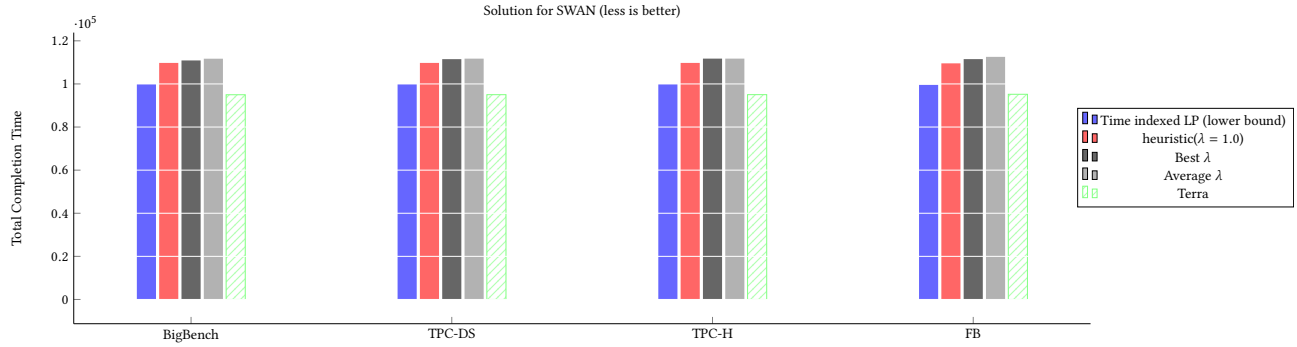

Figure 11: Free path model with no weight on graph SWAN, showing the performance bound of time indexed LP value, the performance of heuristic ($\lambda = 1$), best λ among samples, and the expected value when λ is chosen from the distribution mentioned in Section 4.1. Here we compare against Terra[29]

[16] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, Vol. 43. ACM, 3–14.

[17] Samir Khuller, Jingling Li, Pascal Sturmfels, Kevin Sun, and Prayaag Venkat. 2018. Select and permute: An improved online framework for scheduling to minimize weighted completion time. In *LATIN*. Springer, 669–682.

[18] Samir Khuller and Manish Purohit. 2016. Brief Announcement: Improved Approximation Algorithms for Scheduling Co-Flows. In *SPAA*. ACM, New York, NY, USA, 239–240. https://doi.org/10.1145/2935764.2935809

[19] Yupeng Li, Shaofeng H-C Jiang, Haisheng Tan, Chenzi Zhang, Guihai Chen, Jipeng Zhou, and Francis CM Lau. 2016. Efficient online coflow routing and scheduling. In *MobiHoc*. ACM, 161–170.

[20] Shouxi Luo, Hongfang Yu, Yangming Zhao, Sheng Wang, Shui Yu, and Lemin Li. 2016. Towards practical and near-optimal coflow scheduling for data center

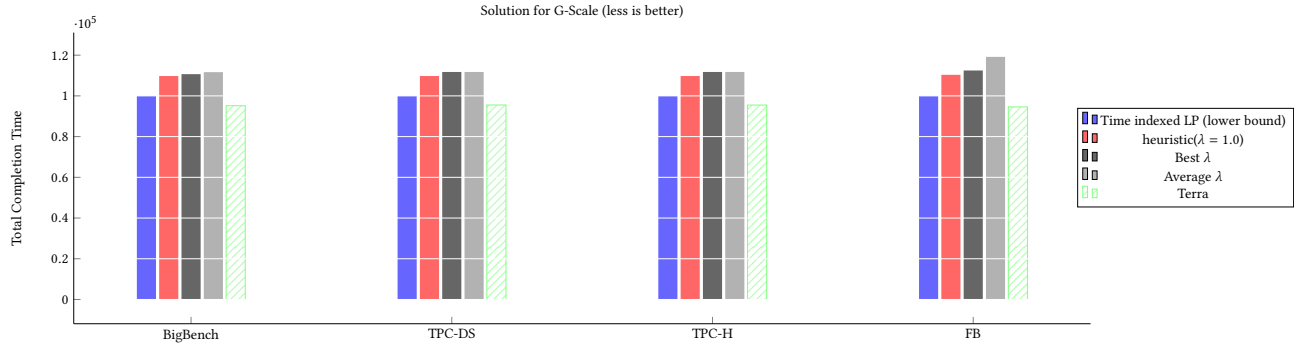

Figure 12: Free path model with no weight on graph G-Scale, showing the performance bound of time indexed LP value, the performance of heuristic ($\lambda = 1$), best λ among samples, and the expected value when λ is chosen from the distribution mentioned in Section 4.1. Here we compare against Terra[29].

networks. *IEEE Transactions on Parallel and Distributed Systems* 27, 11 (2016), 3366–3380.

[21] Ruijiu Mao, Vaneet Aggarwal, and Mung Chiang. 2018. Stochastic non-preemptive co-flow scheduling with time-indexed relaxation. *INFOCOM* (2018).

[22] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The making of TPC-DS. In *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 1049–1058.

[23] Meikel Poess and Chris Floyd. 2000. New TPC benchmarks for decision support and web commerce. *ACM Sigmod Record* 29, 4 (2000), 64–71.

[24] Zhen Qiu, Cliff Stein, and Yuan Zhong. 2015. Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks. In *SPAA*. ACM, New York, NY, USA, 294–303. https://doi.org/10.1145/2755573.2755592

[25] Maurice Queyranne and Maxim Sviridenko. 2002. A (2+ ε)-approximation algorithm for the generalized preemptive open shop problem with minsum objective. *Journal of Algorithms* 45, 2 (2002), 202–212.

[26] Sushant Sachdeva and Rishi Saket. 2013. Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. In *CCC*. IEEE, 219–229.

[27] Andreas S Schulz and Martin Skutella. 1997. Random-based scheduling new approximations and LP lower bounds. In *RANDOM*. Springer, 119–133.

[28] Mehrnoosh Shafiee and Javad Ghaderi. 2018. An improved bound for minimizing the total weighted completion time of coflows in datacenters. *IEEE/ACM Transactions on Networking (TON)* 26, 4 (2018), 1674–1687.

[29] Jie You and Mosharaf Chowdhury. 2019. Terra: Scalable Cross-Layer GDA Optimizations. https://arxiv.org/abs/1904.08480. arXiv:arXiv:1904.08480

[30] Ruozhou Yu, Guoliang Xue, Xiang Zhang, and Jian Tang. 2016. Non-preemptive Coflow Scheduling and Routing. In *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 1–6.

[31] Yangming Zhao, Kai Chen, Wei Bai, Minlan Yu, Chen Tian, Yanhui Geng, Yiming Zhang, Dan Li, and Sheng Wang. 2015. RAPIER: Integrating routing and scheduling for coflow-aware data center networks. In *INFOCOM*. IEEE, 424–432.

A SKETCH OF GENERALIZATION TO SUPER-POLYNOMIAL TIME SPAN

Geometric series time interval is defined as follows. For an $\epsilon > 0$, let $\tau_0 = 0, \tau_1 = 1, \cdots, \tau_k = (1+\epsilon)^{k-1}, \cdots$. We define the k-th interval as $l_k = [\tau_{k-1}, \tau_k]$. Since T is at most the sum of all processing time and all release time, we know the number of intervals $\mathbb{T} = 1 + \lceil \log_{1+\epsilon} T \rceil$ is polynomial.

We change the LP as follows. We abuse notation a bit and allow $\mathbb{T}$ to represent the set $\{1, 2, \cdots, \mathbb{T}\}$ when there is no confusion. We replace all accurance of T with $\mathbb{T}$ in Section 3, modify Equation (4) and Equation (3) to accommodate for release time, and get the following linear program.

$$\text{Minimize} \quad \sum_j w_j C_j, \qquad \text{subject to}$$

$$\sum_t x_j^i(t) = 1 \qquad \forall j \in [n], \forall i \in [n_j] \tag{14}$$

$$X_j(t) \le \sum_{\ell=1}^{t} x_j^i(\ell) \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in \mathbb{T} \tag{15}$$

$$C_j \ge 1 + \sum_t (\tau_t - \tau_{t-1})(1 - X_j(t)), \forall j \in [n] \tag{16}$$

$$r_j^i \ge \tau_t \Rightarrow x_j^i(t) = 0 \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in \mathbb{T} \tag{17}$$

$$x_j^i(t) \ge 0 \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in \mathbb{T} \tag{18}$$

For the model specific part of linear program, we only need to change the capacity constraints: replace Equation (6) for single path model to get

$$\sum_{p_j^i \ni e} x_j^i(t) \cdot \sigma_j^i \le (\tau_t - \tau_{t-1}) c(e), \qquad \forall e \in E, \forall t \in \mathbb{T} \tag{19}$$

and Equation (10) for free path model to get

$$\sum_{e \in \delta_{out}(s_j^i)} x_j^i(t, e) = x_j^i(t), \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in \mathbb{T} \tag{20}$$

$$\sum_{e \in \delta_{in}(t_j^i)} x_j^i(t, e) = x_j^i(t), \qquad \forall j \in [n], \forall i \in [n_j], \forall t \in \mathbb{T} \tag{21}$$

$$\sum_{e \in \delta_{in}(v)} x_j^i(t, e) = \sum_{e = \delta_{out}(v)} x_j^i(t, e), \quad \forall j \in [n], \forall i \in [n_j], \forall t \in \mathbb{T}, \forall v \in V \backslash \{s_i, t_i\} \tag{22}$$

$$\sum_{j \in [n], i \in [n_j]} x_j^i(t, e) \cdot \sigma_j^i \le (\tau_t - \tau_{t-1}) c(e), \forall t \in \mathbb{T}, \forall e \in E \tag{23}$$

Similar to Proposition 3.1, we prove Constraint (16) is a good lower bound.

Proposition A.1. *The completion time of a coflow F_j can be lower bounded by $C_j \ge 1 + \sum_t (\tau_t - \tau_{t-1})(1 - X_j(t))$ where $X_j(t) \in [0, 1]$ denotes the fraction of coflow F_j that has been completed by (the end of) time interval $[\tau_{t-1}, \tau_t]$.*

Proof. If a job completes in the interval $(\tau_{t-1}, \tau_t]$, then its finishing time is at least $\tau_{t-1} + 1$.

$$\begin{aligned}
C_j &\geq \sum_{t=1}^{\mathbb{T}} (1 + \tau_{t-1}) \cdot x_j(t) = \sum_{t=1}^{\mathbb{T}} x_j(t) + \sum_{t=1}^{\mathbb{T}} x_j(t) \sum_{\rho=1}^{t-1} (\tau_\rho - \tau_{\rho-1}) \\
&= 1 + \sum_{\rho=1}^{\mathbb{T}-1} (\tau_\rho - \tau_{\rho-1}) \sum_{t=\rho+1}^{\mathbb{T}} x_j(t) \\
&= 1 + \sum_{\rho=1}^{\mathbb{T}-1} (\tau_\rho - \tau_{\rho-1}) \left(\sum_{t=1}^{\mathbb{T}} x_j(t) - \sum_{t=1}^{\rho} x_j(t) \right) \\
&= 1 + \sum_{\rho=1}^{\mathbb{T}-1} (\tau_\rho - \tau_{\rho-1})(1 - X_j(\rho))
\end{aligned}$$

□

After getting a solution, we would schedule coflows into intervals instead of into time slots. Inside each time interval, we just schedule each flow at uniform speed, and break into actual time slots. Similar to Section 4.1, we can prove that this solution is feasible.

A.1 Analysis

Recall that C_j^* denotes the completion time of the coflow F_j in the optimal LP solution. For any continuous time $t \in [0, T]$, define $\widehat{X}_j(t)$ to be the fraction of coflow F_j that has been scheduled in the LP solution by time t. Note $X_j(t)$ is for time interval $[\tau_{t-1}, \tau_t]$, but $\widehat{X}_j(t)$ is for original time slots. Flows are scheduled at an uniform rate in every time interval. Use $\rho(t)$ to denote the smallest ρ such that $t \in (\tau_{\rho-1}, \tau_\rho]$, we have

$$\widehat{X}_j(t) = X_j(\rho(t)) + \frac{t - \tau_{\rho(t)-1}}{\tau_{\rho(t)} - \tau_{\rho(t)-1}} \left(X_j(\rho(t) + 1) - X_j(\rho(t)) \right) \quad (24)$$

Similar to Lemma 4.1, we state and prove the following lemma.

Lemma A.2. *$\int_{t=0}^{T} (1 - \widehat{X}_j(t)) \mathbf{d}t \leq (1 + \epsilon) C_j^* - \frac{1}{2}$ where $X_j(\rho)$ is defined as per Eq. (24).*

Proof. From constraints (16), we have that

$$\int_{t=0}^{T} (1 - \widehat{X}_j(t)) \mathbf{d}t = \sum_{\rho=1}^{\mathbb{T}} \int_{t=\tau_{\rho-1}}^{\tau_\rho} (1 - \widehat{X}_j(t)) \mathbf{d}t$$

Since $1 - \widehat{X}_j(t)$ is linear for $t \in (\tau_{\rho-1}, \tau_\rho]$,

$$\begin{aligned}
&= \sum_{\rho=1}^{\mathbb{T}} \frac{\tau_\rho - \tau_{\rho-1}}{2} (1 - X_j(\rho) + 1 - X_j(\rho - 1)) \\
&= \sum_{\rho=1}^{\mathbb{T}} \frac{\tau_\rho - \tau_{\rho-1}}{2} (1 - X_j(\rho)) + \sum_{\rho=1}^{\mathbb{T}} \frac{\tau_\rho - \tau_{\rho-1}}{2} (1 - X_j(\rho - 1)) \\
&= \sum_{\rho=1}^{\mathbb{T}} \frac{\tau_\rho - \tau_{\rho-1}}{2} (1 - X_j(\rho)) \\
&\quad + (1 + \epsilon) \sum_{\rho=2}^{\mathbb{T}} \frac{\tau_{\rho-1} - \tau_{\rho-2}}{2} (1 - X_j(\rho - 1)) + \frac{\tau_1 - \tau_0}{2} (1 - X_j(0)) \\
&= \sum_{\rho=1}^{\mathbb{T}-1} \frac{\tau_\rho - \tau_{\rho-1}}{2} (1 - X_j(\rho)) + \frac{\tau_{\mathbb{T}} - \tau_{\mathbb{T}-1}}{2} (1 - X_j(\mathbb{T})) \\
&\quad + (1 + \epsilon) \sum_{\rho=1}^{\mathbb{T}-1} \frac{\tau_\rho - \tau_{\rho-1}}{2} (1 - X_j(\rho)) + \frac{\tau_1 - \tau_0}{2} (1 - X_j(0)) \\
&= \frac{2 + \epsilon}{2} \sum_{\rho=1}^{\mathbb{T}-1} (\tau_\rho - \tau_{\rho-1})(1 - X_j(\rho)) + \frac{1}{2}
\end{aligned}$$

Plugging in Proposition A.1,

$$\leq \frac{2 + \epsilon}{2} (C_j^* - 1) + \frac{1}{2} \leq (1 + \epsilon) C_j^* - \frac{1}{2}$$

□

For any $\lambda \in [0, 1]$, define $C_j^*(\lambda)$ to be the earliest time ρ such that λ fraction of the coflow F_j has been scheduled in the LP solution, i.e., in other words its the smallest t such that $\widehat{X}_j(t) = \lambda$. Note that by time $C_j^*(\lambda)$, λ fraction of *every flow* $f_j^i \in F_j$ has been scheduled by the LP.

Proposition A.3. $\int_{\lambda=0}^{1} C_j(\lambda) \mathbf{d}\lambda \leq \int_{t=0}^{T} (1 - \widehat{X}_j(t)) \mathbf{d}t$.

Proof.

$$\begin{aligned}
\int_{\lambda=0}^{1} C_j(\lambda) \mathbf{d}\lambda &= \int_{\lambda=0}^{1} \int_{t=0}^{T} \mathbb{1}_{[C_j(\lambda) > t]} \mathbf{d}t \mathbf{d}\lambda = \int_{t=0}^{T} \int_{\lambda=0}^{1} \mathbb{1}_{[C_j(\lambda) > t]} \mathbf{d}\lambda \mathbf{d}t \\
&= \int_{t=0}^{T} \int_{\lambda=X_j(t)}^{1} 1 \mathbf{d}\lambda \mathbf{d}t = \int_{t=0}^{T} (1 - \widehat{X}_j(t)) \mathbf{d}t
\end{aligned}$$

□

Finally, we are ready to bound the completion time $C_j(alg)$ of coflow F_j in the stretched schedule. For any fixed $\lambda \in (0, 1)$, since we stretch the schedule by a factor of $\frac{1}{\lambda}$, it is easy to verify[4] that $C_j(alg) \leq \left\lceil \frac{C_j(\lambda)}{\lambda} \right\rceil$. Since λ is drawn randomly from a distribution, the following lemma bounds the expected completion time of coflow F_j in the stretched schedule.

Lemma A.4. *The expected completion time of any coflow F_j in the stretched schedule is bounded by $2(1 + \epsilon) C_j^*$.*

Proof.

$$\begin{aligned}
\mathbb{E}[C_j(alg)] &\leq \int_{\lambda=0}^{1} f(\lambda) \left\lceil \frac{C_j(\lambda)}{\lambda} \right\rceil d\lambda \leq \int_{\lambda=0}^{1} 2\lambda \left(1 + \frac{C_j(\lambda)}{\lambda} \right) d\lambda \\
&= \int_{\lambda=0}^{1} 2\lambda d\lambda + 2 \int_{\lambda=0}^{1} C_j(\lambda) d\lambda \\
&= 1 + 2 \int_{\lambda=0}^{1} C_j(\lambda) d\lambda
\end{aligned}$$

By Lemma A.2 and Proposition A.3,

$$\leq 2(1 + \epsilon) C_j^*.$$

□

Theorem 4.5 thus follows from the linearity of expectation.

[4] All flows $f_j^i \in F_j$ were completed by at least λ fraction by time $C_j(\lambda)$. So in the stretched schedule, all those flows must be completed by time $\left\lceil \frac{C_j(\lambda)}{\lambda} \right\rceil$.