Optimizing Coflow Completion Times with Utility Max-Min Fairness

Li Chen[1] Wei Cui[1] Baochun Li[1] Bo Li[2]
[1]University of Toronto
[2]The Hong Kong University of Science and Technology

***Abstract*—In data parallel frameworks such as MapReduce and Spark, a *coflow* represents a set of network flows used to transfer intermediate data between successive computation stages for a job. The completion time of a job is then determined by the collective behavior of such a coflow, rather than any individual flow within, and influenced by the amount of network bandwidth allocated to it. Different jobs in a shared cluster have different degrees of sensitivity to their completion times, modeled by their respective utility functions. In this paper, we focus on the design and implementation of a new utility optimal scheduler across competing coflows, in order to provide differential treatment to coflows with different degrees of sensitivity, yet still satisfying max-min fairness across these coflows. Though this objective can be formulated as a lexicographical maximization problem, it is challenging to solve in practice due to its inherent multi-objective and discrete nature. To address this challenge, we first divide the problem into iterative steps of single-objective subproblems; and in each of these steps, we then perform a series of transformations to obtain an equivalent linear programming (LP) problem, which can be efficiently solved in practice. To demonstrate that our solutions are practically feasible, we have implemented it as a real-world coflow scheduler based on the Varys open-source framework to evaluate its effectiveness.**

I. Introduction

Thanks to the exponential growth of data that needs to be processed in cloud datacenters, data parallel frameworks, such as MapReduce [1] and Spark [2], have emerged as foundations of cloud computing. In data parallel frameworks, a *job* typically proceeds in consecutive computation *stages*; and each of these stages consists of a number of computation *tasks* that are processed in parallel. Before the next computation stage may begin, multiple network flows need to be initiated in parallel to transfer intermediate data from the preceding stage.

Due to the volume of intermediate data to be transferred, such network transfers across stages may have a significant impact on the performance of typical jobs in big data processing. It has been shown that they usually account for more than 50% of entire job completion times [3]. Since a network transfer is not considered complete till all of its constituent flows have finished, it is the collective behavior of all of these flows that matters, rather the individual behavior of each flow. These flows are hence referred to as a *coflow* [4]. In a MapReduce job, for example, a coflow consists of all the flows in the *shuffle* phase, which transfers intermediate data from *mapper* tasks to *reducer* tasks.

As the datacenter network is shared by active coflows from multiple competing jobs, it is critical to schedule these coflows efficiently and fairly. Network resources should be allocated at the level of *coflows* — rather than individual flows — in order to achieve the best possible performance with respect to job completion times. However, due to their inherent nature, different jobs have widely diverging requirements with respect to their completion times: an interactive query in a web application should not be similarly treated as a background job for data analytics. Intuitively, we may use different *utility functions* [5] to model such a diverging range of sensitivity to job completion times.

Existing research efforts on coflow scheduling focused on minimizing coflow completion times [6]–[8] and meeting coflow deadlines [6]. All coflows were treated identically as equal citizens in a datacenter, and the *average* coflow completion time was to be minimized. In this paper, we depart from such conventional wisdom, and argue that more time-sensitive coflows should be allocated more network resources, allowing them to complete earlier, achieving a higher utility based on their utility functions. Of course, such differential treatment may negatively affect the performance of background jobs that tolerate longer completion times. Therefore, from a global perspective and in the general case, we will need to allocate resources optimally across all concurrent coflows, such that they will achieve their best posible utilities, and max-min fairness across coflows can be achieved.

Though the problem of maximizing the utilities achieved by each of the coflows can be formally formulated as a *lexicographical maximization* problem, there are unique challenges that make it difficult to solve this problem with multiple objectives. Scheduling among coflows over a discretized time domain is essentially an integer optimization problem, which in general is NP-hard [9]. In addition, to accommodate a diverse set of practical utility functions (such as a sigmoid function), we do not assume that these utility functions are convex or concave; standard convex optimization techniques are therefore not directly applicable.

To address these challenges, we first consider the subproblem of maximizing the worst utility among all the concurrent coflows, which turns out to have a *totally unimodular* coefficient matrix for linear constraints, based on an in-depth

Email: {lchen,wcui,bli}@ece.utoronto.ca; bli@cse.ust.hk. The co-authors would like to acknowledge the gracious research support from the NSERC Discovery Research Program and the SAVI NSERC Strategic Networks Grant at the University of Toronto, the grants from RGC under the contracts 615613 and 16211715, and the grant from NSFC and Guangdong joint project under the contract U1301253.

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

investigation of the problem structure. Such a nice property guarantees that the extreme points in a feasible solution polyhedron are integers. Moreover, with several steps of non-trivial transformations, we show that the optimal solution to the original problem can be obtained by solving an equivalent problem with a *separable convex objective*. With these structures identified, we can then apply the λ-technique and linear relaxation to obtain a linear programming (LP) problem, which is guaranteed to have the same solution to the original problem. As a result, any LP solver can be used for maximizing the utility in each coflow, and to efficiently compute the overall scheduling decisions that achieve the optimal coflow utilities with max-min fairness.

We proceed to design and implement a real-world coflow scheduler that enforces our utility optimal scheduling policy in the Varys [6] coflow scheduling framework. Experimental results demonstrate the effectiveness of our scheduler in optimizing coflow utilities. Compared with the state-of-the-art coflow scheduler which is utility agnostic, our scheduler achieves significant utility improvement, and thus better satisfy job requirements.

The remainder of this paper is organized as follows. We motivate our utility optimal scheduling problem with convincing examples in Sec. II, and formulate it as a lexicographical maximization problem in Sec. III. Starting from the subproblem of maximizing the worst coflow utility, we transfer the problem to an equivalent LP problem, with the main techniques presented in Sec. IV. We then design a new algorithm to obtain the optimal solution for the original problem in Sec. V. Our implementation details and experimental results are presented in Sec. VI to demonstrate the practicality and effectiveness of our solution. Finally we differentiate our work from related research efforts in Sec. VII and conclude the paper in Sec. VIII.

II. Background and Motivation

A. Completion-Time-Dependent Utility

In a shared datacenter cluster, different jobs may have different sensitivity to their completion times. Such differences can be reflected in their evaluation of completion times as utility values. For example, a user-interactive query job is critical to its completion time, or probably associated with a hard deadline. Failing to complete the job by its target time would result in a small or zero value of utility. In contrast, a background job for data analytics may be completion-time-insensitive, which means that as long as it successfully generates the final result, it would obtain a fixed value of utility, regardless of the time when it completes.

Such a diverging range of sensitivity can be captured by different utility functions that are dependent upon completion times [5]. As shown in Fig. 1, the two solid lines represent the utility functions for completion-time-critical jobs, which decrease sharply to zero after the target completion time. On the contrary, the utility value of a completion-time-insensitive job keeps the same along with time, represented by the horizontal line with star markers. In between, the two dashed lines decrease gradually with time, which characterize utilities of jobs that are completion-time-sensitive.

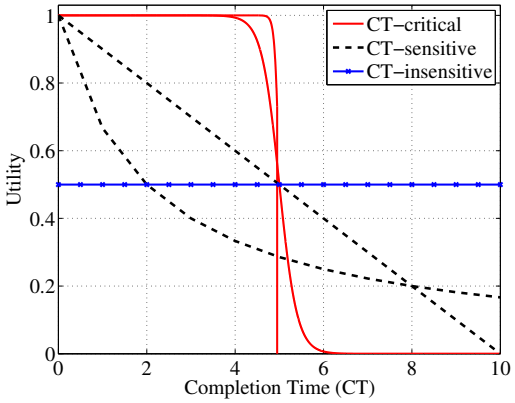

Fig. 1: Completion-time-dependent utility functions.

The completion time of a job depends on both the computation time and the coflow completion time, *i.e.*, the completion time of the slowest flow. As the compution time is predictable due to mature techniques in allocation and isolation of computing resources, the job utility function can be easily translated to the coflow-completion-time dependent utility function. For convenience, the value of this function is referred to as the *coflow utility*, which is to be optimized in our network resource allocation.

B. Network Shared by Coflows

As a common practice, we consider a congestion-free datacenter network, which can be abstracted as a giant non-blocking switch (*e.g.*, [6], [7], [10], [11]) that interconnects all the machines. This is reasonable and practical, thanks to the recent efforts in full bisection bandwidth topologies (*e.g.*, [12], [13]). In such a switch model, as illustrated in Fig. 2, each ingress port corresponds to the outgoing link of the connected machine, where the data is transferred from the machine to the network, and each egress port represents the incoming link of the connected machine which receives data from the network.

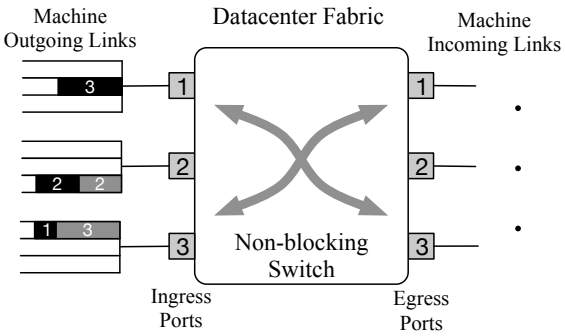

Fig. 2: Coflows to be scheduled through a 3×3 datacenter fabric.

In the example shown in Fig. 2, 3 machines are interconnected by a non-blocking switch. Coflow $C1$ and $C2$ arrive at time 0, of which the flows are represented by the black and grey blocks in the figure, respectively. $C1$ has 3 flows transferring 3, 2 and 1 units of data; $C2$ has 2 flows with sizes of 2 and 3. The virtual input queues at the ingress ports are used for convenience to illustrate the source and destination of these flows. For example, at machine 1, a flow of $C1$ transfers

3 units of data to machine 2; at machine 2, flows of $C1$ and $C2$ each transfer 2 units of data to machine 3.

C. Utility Optimal Scheduling

The typical objective of existing coflow scheduling mechanisms was to minimize the average coflow completion time (CCT) or to meet coflow deadlines, and they did not account for different sensitivity levels to completion times. To better satisfy job requirements, we wish to optimize utilities achieved across all the competing coflows to achieve max-min fairness.

Fig. 4 illustrates two schedules of coflows $C1$ and $C2$ aforementioned. Assume that $C1$ is completion-time-insensitive, which achieves a constant utility as long as it completes, while $C2$ is completion-time-sensitive, with its utility decreasing gradually with increasing completion times. A scheduler that tries to minimize the average CCT would choose either of these schedules, as both of them are optimal, making two coflows complete in 3 and 4 time units. However, when coflow utilities are to be optimized, Schedule 2 is the only optimal one, because it outperforms Schedule 1 with the same utility for $C1$ and better utility for $C2$. In this way, the awareness of coflow utility results in the optimal schedule that best satisfies coflow requirements.

Further, we consider a more specific example shown in Fig. 3 for utility optimal schedule among coflows $C1$, $C2$ and $C3$. As shown in Fig. 3(a), the 2 flows of $C1$, corresponding to the black blocks, transfer 3 and 1 units of data respectively; $C2$ has 3 flows (grey) transferring 1, 2 and 2 units of data; $C3$ has 3 flows (white) transferring 1 unit of data each. The utilities of coflows achieved at different completion times are presented in Fig. 3(b), where we can see that $C2$ is the most sensitive to its completion times, while $C3$ is the least sensitive.

Fig. 3(c) and Fig. 3(d), respectively, present the schedule that minimizes the average CCT, and the schedule that optimizes coflow utilities with max-min fairness. Fig. 3(c) achieves 5, 3 and 1 units of coflow completion times, better than Fig. 3(d) with 5, 2 and 3 when the average CCT is considered. However, with respect to coflow utilities, Fig. 3(c) obtains the utility values of 5, 4 and 9, while Fig. 3(d) achieves 5, 7 and 8. Clearly, the schedule in Fig. 3(d) is utility optimal, with max-min fairness achieved among coflow utilities.

Note that for simplicity of illustration, there is no contention on machine incoming links in these examples. In general, however, both outgoing and incoming links may experience contention, and the utility optimal schedule can be more effective.

III. Model and Formulation

We consider a cloud cluster with a set of servers denoted by $\mathcal{N} = \{1, 2, \cdots, N\}$, which are shared by multiple data parallel jobs. A set of coflows $\mathcal{K} = \{1, 2, \cdots, K\}$ are submitted by these concurrently running jobs, to transfer the intermediate data through the network in their communication stages.

As mentioned in the previous section, the network is considered as a non-blocking switch, thus our coflow schedule takes place at its ingress and egress ports, corresponding to the incoming and outgoing links at each server. For simplicity, we assume that at each time slot $t \in \mathcal{T} = \{1, 2, \cdots, T\}$, each server can transmit one unit of data through its outgoing (egress) link, and receive one unit of data through its incoming (ingress) link.

A flow that belongs to coflow $k \in \mathcal{K}$ is represented by $D^k_{i,j},\ i, j \in \mathcal{N}$, which specifies that it transfers $D^k_{i,j}$ units of data from server i to j. Coflow k completes when its last flow finishes at time $t \in \mathcal{T}$, and achieves a utility of u^t_k which is determined by its non-increasing utility function $\mathcal{U}_k(t)$.

As server links are shared by flows from multiple coflows, we would like to obtain a utility optimal coflow schedule without exceeding link capacities. To be particular, our objective is to maximize the worst utility achieved among all the coflows, then maximize the next worst utility without impacting the previous one, which is executed repeatedly until utilities have been optimized for all the coflows. Such an objective can be rigorously formulated as a *lexicographical maximization* problem, with the following definitions as the basis.

Definition 1: Let $\langle \boldsymbol{u} \rangle_k$ denote the k-th ($1 \le k \le K$) smallest element of $\boldsymbol{u} \in \mathbb{Z}^K$, implying $\langle \boldsymbol{u} \rangle_1 \le \langle \boldsymbol{u} \rangle_2 \le \cdots \le \langle \boldsymbol{u} \rangle_K$. Intuitively, $\langle \boldsymbol{u} \rangle = (\langle \boldsymbol{u} \rangle_1, \langle \boldsymbol{u} \rangle_2, \cdots, \langle \boldsymbol{u} \rangle_K)$ represents the non-decreasingly sorted version of $\boldsymbol{u}$.

Definition 2: For any $\boldsymbol{\alpha} \in \mathbb{Z}^K$ and $\boldsymbol{\beta} \in \mathbb{Z}^K$, if $\langle \boldsymbol{\alpha} \rangle_1 > \langle \boldsymbol{\beta} \rangle_1$ or $\exists k \in \{2, 3, \cdots, K\}$ such that $\langle \boldsymbol{\alpha} \rangle_k > \langle \boldsymbol{\beta} \rangle_k$ and $\langle \boldsymbol{\alpha} \rangle_i = \langle \boldsymbol{\beta} \rangle_i, \forall i \in \{1, \cdots, k-1\}$, then $\boldsymbol{\alpha}$ is *lexicographically greater* than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$. Similarly, if $\langle \boldsymbol{\alpha} \rangle_k = \langle \boldsymbol{\beta} \rangle_k, \forall k \in \{1, 2, \cdots, K\}$ or $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$, then $\boldsymbol{\alpha}$ is *lexicographically no smaller* than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \succeq \boldsymbol{\beta}$.

Definition 3: $\underset{\boldsymbol{x}}{\text{lexmax}}\ \boldsymbol{f}(\boldsymbol{x})$ represents the *lexicographical maximization* of the vector $\boldsymbol{f} \in \mathbb{R}^M$, which consists of M objective functions of $\boldsymbol{x}$. To be particular, the optimal solution $\boldsymbol{x}^* \in \mathbb{R}^K$ achieves the optimal $\boldsymbol{f}^*$, in the sense that $\boldsymbol{f}^* = \boldsymbol{f}(\boldsymbol{x}^*) \succeq \boldsymbol{f}(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^K$.

Let $x^{k,t}_{i,j}$ denote the number of data units of coflow k that is transferred from server i to j at time slot t. We are now ready to formulate our utility optimal coflow scheduling as follows:

$$\underset{\boldsymbol{x}}{\text{lexmax}} \quad \boldsymbol{f} = (\mathcal{U}_1(\tau_1), \mathcal{U}_2(\tau_2), \cdots, \mathcal{U}_K(\tau_K)) \tag{1}$$

$$\text{s.t.} \quad \tau_k = \max\{t \in \mathcal{T} | x^{k,t}_{i,j} > 0, \forall i, j\}, \forall k \in \mathcal{K} \tag{2}$$

$$\sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{N}} x^{k,t}_{i,j} \le 1, \quad \forall i \in \mathcal{N},\ \forall t \in \mathcal{T} \tag{3}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}} x^{k,t}_{i,j} \le 1, \quad \forall j \in \mathcal{N},\ \forall t \in \mathcal{T} \tag{4}$$

$$\sum_{t \in \mathcal{T}} x^{k,t}_{i,j} = D^k_{i,j}, \quad \forall i, j \in \mathcal{N},\ \forall k \in \mathcal{K} \tag{5}$$

$$x^{k,t}_{i,j} \in \mathbb{Z}^+, \quad \forall i, j \in \mathcal{N},\ \forall t \in \mathcal{T},\ \forall k \in \mathcal{K} \tag{6}$$

where τ_k represents the completion time of coflow k, which is the time slot when the last unit of data that belongs to k is transmitted, according to constraint (2). Constraint (3) indicates that the total number of data units transmitted through the outgoing link of server i at time slot t does not exceed the link capacity. Similarly, constraint (4) specifies that each incoming link transmits at most 1 unit of data per time slot. Constraint (5) implies the total amount of data units to be transmitted for each flow.

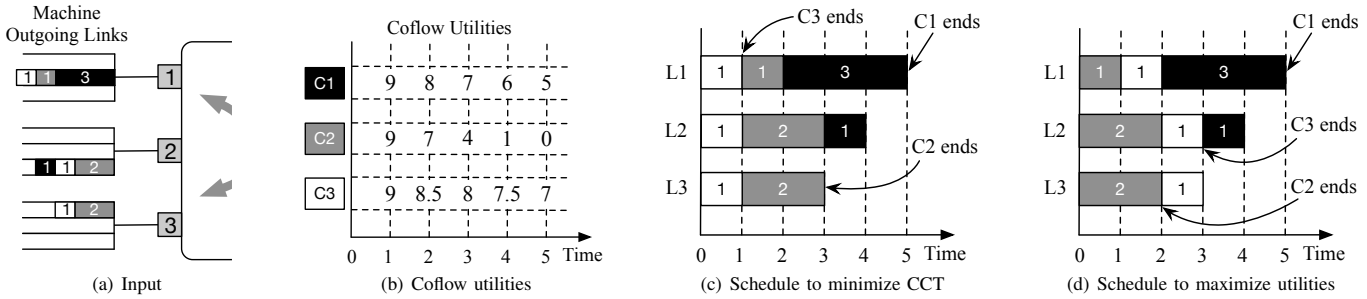

Fig. 3: Utility optimal schedule among three coflows.

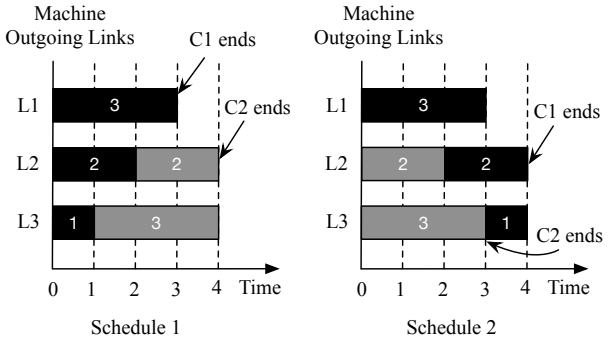

Fig. 4: Two possible schedules for coflows $C1$ and $C2$. They are the same with respect to the average coflow completion time, but they achieve different coflow utilities.

The objective is a vector $\boldsymbol{f} \in \mathbb{R}^K$ with K elements, each standing for the utility of a particular coflow $k \in \mathcal{K}$. According to the previous definitions, the optimal $\boldsymbol{f}^*$ is lexicographically no smaller than any $\boldsymbol{f}$ obtained with a feasible schedule, which means that when sorting them in a non-descending order, if their k-th smallest element satisfies $\langle \boldsymbol{f}^* \rangle_{k'} = \langle \boldsymbol{f} \rangle_{k'}, \forall k' < k$ and $\langle \boldsymbol{f}^* \rangle_k \neq \langle \boldsymbol{f} \rangle_k$, then we have $\langle \boldsymbol{f}^* \rangle_k > \langle \boldsymbol{f} \rangle_k$. This implies that the first smallest element of $\boldsymbol{f}^*$, *i.e.*, the worst coflow utility, is the maximum among all $\boldsymbol{f}$. Then among all $\boldsymbol{f}$ with the same worst utility, the second worst utility in $\boldsymbol{f}^*$ is the maximum, and so on. In this way, solving this problem would result in an optimal schedule vector $\boldsymbol{x}^*$, with which all the coflow utilities are maximized.

IV. Optimizing the Worst Utility among Concurrent Coflows

As interpreted in the previous section, Problem (1) is a vector optimization with multiple objectives. In this section, we consider the single-objective subproblem of optimizing the worst coflow utility as follows:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \min_{k \in \mathcal{K}} \left(\mathcal{U}_k(\tau_k) \right) \\ \text{s.t.} \quad & \text{Constraints } (2), (3), (4), (5) \text{ and } (6). \end{aligned} \tag{7}$$

which is the primary step for solving the original problem, to be elaborated in the next section.

This problem is an integer programming problem with a nonlinear constraint (2). With an in-depth investigation of the problem structure, we transform Problem (7) into an equivalent linear programming (LP) problem that can be solved efficiently to obtain the optimal schedule vector $\boldsymbol{x}$. To be more specific, the features of *separable convex objective* and *totally unimodular linear constraints* are utilized in our transformation, which involves three major steps to be elaborated in the following subsections.

A. Totally Unimodular Linear Constraints

We first eliminate the nonlinear constraint (2) by expressing the utility of coflow $k \in \mathcal{K}$ directly with its scheduling variables $x_{i,j}^{k,t}, \forall i, j \in \mathcal{N}, \forall t \in \mathcal{T}$, rather than its completion time τ_k.

As previously defined, u_k^t is the utility value achieved by coflow k, if it completes at time slot t, *i.e.*, $u_k^t = \mathcal{U}_k(t)$. Since the utility function is non-increasing, we have $u_k^t \geq u_k^{t'}$ for $t < t'$. Let $\phi(x_{i,j}^{k,t})$ denote an indicator function, so that $\phi(x_{i,j}^{k,t}) = 1$ if $x_{i,j}^{k,t} > 0$, and $\phi(x_{i,j}^{k,t}) = 0$, otherwise. Hence, the utility of coflow k can be represented as $\min_{i,j,t,x_{i,j}^{k,t} \neq 0} u_k^t \phi(x_{i,j}^{k,t})$, which correctly indicates that the coflow utility is the smallest utility value achieved at the maximum t that satisfies $x_{i,j}^{k,t} \neq 0$, *i.e.*, the time slot when the last flow of k completes. Therefore, Problem (7) can be transformed as follows with the nonlinear constraint (2) eliminated:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \min_{k \in \mathcal{K}} \Big(\min_{i,j,t,x_{i,j}^{k,t} \neq 0} u_k^t \phi(x_{i,j}^{k,t}) \Big) \\ \text{s.t.} \quad & \text{Constraints } (3), (4), (5) \text{ and } (6). \end{aligned} \tag{8}$$

Now we investigate the coefficient matrix of linear constraints (3) (4) and (5). An m-by-n matrix is *totally unimodular* [9], if it satisfies two conditions: 1) any of its elements belongs to $\{-1, 0, 1\}$; 2) any row subset $R \subset \{1, 2, \cdots, m\}$ can be divided into two disjoint sets, R_1 and R_2, such that $|\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij}| \leq 1, \forall j \in \{1, 2, \cdots, n\}$.

Lemma 1: The coefficients of constraints (3), (4) and (5) form a totally unimodular matrix.

Proof: Let $\mathbf{A}_{m \times n}$ denote the coefficient matrix of all the linear constraints (3), (4) and (5), where $m = 2NT + KN^2$, representing the total number of the constraints, and $n = KTN^2$, denoting the dimension of the variable $\boldsymbol{x}$.

It is obvious that any element of $\mathbf{A}_{m \times n}$ is either 0 or 1, satisfying the first condition. For any row subset $R \subset \{1, 2, \cdots, m\}$, to check the second condition, we consider the following cases:

Case I: If $Q_1 = \{1, 2, \cdots, NT\} \subseteq R$, then we can group Q_1 into R_1 and $R - Q_1$ into R_2. In this way, $\forall j \in \{1, 2, \cdots, n\}$, we have $\sum_{i \in R_1} a_{ij} = 1$, and $0 \leq \sum_{i \in R_2} a_{ij} \leq 2$, hence, $|\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij}| \leq 1$, satisfying the second

condition. In a similar vein, if $Q_2 = \{NT+1, NT+2, \cdots, 2NT\} \subseteq R$, then we can group Q_2 into R_1 and $R-Q_2$ into R_2; if $Q_3 = \{2NT+1, 2NT+2, \cdots, 2NT+KN^2\} \subseteq R$, then we can group Q_3 into R_1 and $R-Q_3$ into R_2.

Case II: If $Q_1 \nsubseteq R$ and $Q_2 \nsubseteq R$ and $Q_3 \nsubseteq R$, we can divide R with the following steps. Initialize $R' = R$ as the set of ungrouped rows. First, for any column j from 1 to n, if $\sum_{i\in R} a_{ij} = 2$, $a_{i_1 j} = a_{i_2 j} = 1$, *i.e.*, there are two elements with value 1, we separate rows i_1 and i_2 into two sets and remove them from R'. To be specific, i_1 is grouped into R_1 if $\sum_{i\in(R_1+\{i_1\})} a_{il} \neq 3, \forall l \in \{1, 2, \cdots, j-1\}$; otherwise, i_1 is added to R_2. Second, for any column j from 1 to n, if $\sum_{i\in R} a_{ij} = 3$, $a_{i_1 j} = a_{i_2 j} = a_{i_3 j} = 1$ and if $i_1 \in R'$ or $i_2 \in R'$ or $i_3 \in R'$, add the ungrouped row(s) so that i_1, i_2 and i_3 are not in the same subset, and remove them from R'. Finally, add the remaining rows in R' to any of the subset. In this way, we can guarantee that for any column $j \in \{1, 2, \cdots, n\}$, if $\sum_{i\in R} a_{ij} = 2$, then $\sum_{i\in R_1} a_{ij} = \sum_{i\in R_2} a_{ij} = 1$; if $\sum_{i\in R} a_{ij} = 3$, then $1 \leq \sum_{i\in R_1} a_{ij} \leq 2$ and $1 \leq \sum_{i\in R_2} a_{ij} \leq 2$. Therefore, the second condition is satisfied.

In summary, we have shown that both conditions for total unimodularity are satisfied, thus the coefficient matrix $\mathbf{A}_{m\times n}$ is totally unimodular. ∎

B. Separable Convex Objective

In this subsection, we will show that the optimal solution for Problem (8) can be obtained by solving a problem with a separable convex objective function, which is represented as a summation of convex functions with respect to each single variable $x_{i,j}^{k,t}$.

We first show that the optimal solution of Problem (8) can be obtained by solving the following problem:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{lexmax}} \quad & \boldsymbol{g} = (u_1^1\phi(x_{1,1}^{1,1}), \cdots, u_k^t\phi(x_{i,j}^{k,t}), \cdots, u_K^T\phi(x_{N,N}^{K,T})) \\ \text{s.t.} \quad & \text{Constraints } (3), (4), (5) \text{ and } (6). \end{aligned}$$

where $\boldsymbol{g}$ is a vector with the dimension of $M = |\boldsymbol{g}| = KTN^2$. For any feasible $\boldsymbol{x}$, as the total number of data units to be transferred is fixed, it is intuitive that the total number of $x_{i,j}^{k,t}$ that equals to 0 is also fixed, denoted by p. Thus, any $\boldsymbol{g}$ achieved by a feasible $\boldsymbol{x}$ has at least p elements of value 0. Consider the optimal $\boldsymbol{g}^*$ and any feasible $\boldsymbol{g}$, we have $\langle \boldsymbol{g}^* \rangle_k = \langle \boldsymbol{g} \rangle_k = 0, \forall k \leq p$, and $\langle \boldsymbol{g}^* \rangle_{p+1} \geq \langle \boldsymbol{g} \rangle_{p+1}$, where $\langle \boldsymbol{g}^* \rangle_{p+1} = \min_{i,j,t,x_{i,j}^{k,t}\neq 0} u_k^t$, representing the worst coflow utility. This indicates that $\boldsymbol{g}^*$ achieves the maximal worst coflow utility. Therefore, the optimal schedule $\boldsymbol{x}^*$ which gives $\boldsymbol{g}^*$ is also the optimal solution for Problem (8).

For ease of expression, we use $\mathcal{X}$ to represent the set of feasible $\boldsymbol{x}$ that satisfies constraints (3), (4), (5) and (6). We then transform the problem $\text{lexmax}_{\boldsymbol{x}\in\mathcal{X}}\ \boldsymbol{g}$ to an equivalent problem with a separable convex objective, based on the following lemmas.

Lemma 2: $\text{lexmax}_{\boldsymbol{x}\in\mathcal{X}}\ \boldsymbol{g} \iff \text{lexmin}_{\boldsymbol{x}\in\mathcal{X}}\ -\boldsymbol{g}$.

Proof: We first define the lexicographical order that is opposite to the direction in Definition 2 and 3. For a vector $\boldsymbol{\alpha} \in \mathbb{Z}^K$, $\langle \boldsymbol{\alpha} \rangle' = (\langle \boldsymbol{\alpha} \rangle_K, \langle \boldsymbol{\alpha} \rangle_{K-1}, \cdots, \langle \boldsymbol{\alpha} \rangle_1)$ represents the sorted version (in non-ascending order) of $\boldsymbol{\alpha}$. For any $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^K$, if the first non-zero element of $\langle \boldsymbol{\alpha} \rangle' - \langle \boldsymbol{\beta} \rangle'$ is negative, then $\boldsymbol{\alpha}$ is *lexicographically smaller* than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. If $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$ or $\langle \boldsymbol{\alpha} \rangle' = \langle \boldsymbol{\beta} \rangle'$, then $\boldsymbol{\alpha}$ is *lexicographically no greater* than $\boldsymbol{\beta}$, represented as $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$. Note that $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ does not imply $\boldsymbol{\beta} \succeq \boldsymbol{\alpha}$, or vice versa. The *lexicographical minimization* of $\boldsymbol{f}(\boldsymbol{x})$ over $\boldsymbol{x} \in \mathbb{Z}^K$, represented as $\underset{\boldsymbol{x}}{\text{lexmin}}\ \boldsymbol{f}(\boldsymbol{x})$, is to find the optimal $\boldsymbol{x}^*$ that satisfies $\boldsymbol{f}^* = \boldsymbol{f}(\boldsymbol{x}^*) \preceq \boldsymbol{f}(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathbb{R}^K$.

Now we consider the optimal $\boldsymbol{x}^*$ to $\text{lexmax}_{\boldsymbol{x}\in\mathcal{X}}\ \boldsymbol{g}$. According to Definition 3, $\boldsymbol{g}(\boldsymbol{x}^*) \succeq \boldsymbol{g}(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{X}$, which implies that the first non-zero element of $\langle \boldsymbol{g}(\boldsymbol{x}^*) \rangle - \langle \boldsymbol{g}(\boldsymbol{x}) \rangle$ is positive, recalling that $\langle \boldsymbol{g}(\boldsymbol{x}) \rangle = (\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1, \langle \boldsymbol{g}(\boldsymbol{x}) \rangle_2, \cdots, \langle \boldsymbol{g}(\boldsymbol{x}) \rangle_K)$. Since $\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1$ is the minimum element in $\boldsymbol{g}(\boldsymbol{x})$, it is obvious that $-\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1$ is the maximum element in $-\boldsymbol{g}(\boldsymbol{x})$, *i.e.*, $\langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_K = -\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1$. The same applies to all the elements in $\boldsymbol{g}(\boldsymbol{x})$, which gives $\langle -\boldsymbol{g}(\boldsymbol{x}) \rangle' = (\langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_K, \langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_{K-1}, \cdots, \langle -\boldsymbol{g}(\boldsymbol{x}) \rangle_1) = (-\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_1, -\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_2, \cdots, -\langle \boldsymbol{g}(\boldsymbol{x}) \rangle_K) = -\langle \boldsymbol{g}(\boldsymbol{x}) \rangle$. Hence, we have $\langle -\boldsymbol{g}(\boldsymbol{x}^*) \rangle' - \langle -\boldsymbol{g}(\boldsymbol{x}) \rangle' = -(\langle \boldsymbol{g}(\boldsymbol{x}^*) \rangle - \langle \boldsymbol{g}(\boldsymbol{x}) \rangle)$, of which the first non-zero element is negative, indicating that $-\boldsymbol{g}(\boldsymbol{x}^*) \preceq -\boldsymbol{g}(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{X}$.

As such, we have proved that

$$\boldsymbol{g}(\boldsymbol{x}^*) \succeq \boldsymbol{g}(\boldsymbol{x}) \iff -\boldsymbol{g}(\boldsymbol{x}^*) \preceq -\boldsymbol{g}(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathcal{X},$$

and thus the lemma holds obviously with the definitions of $\text{lexmax}_{\boldsymbol{x}\in\mathcal{X}}\ \boldsymbol{g}$ and $\text{lexmin}_{\boldsymbol{x}\in\mathcal{X}}\ -\boldsymbol{g}$. ∎

Let $\varphi(\boldsymbol{g})$ define a function of $\boldsymbol{g}$:

$$\varphi(\boldsymbol{g}) = \sum_{m=1}^{|\boldsymbol{g}|} |\boldsymbol{g}|^{g_m} = \sum_{m=1}^{M} M^{g_m}$$

where g_m is the m-th element of $\boldsymbol{g}$.

Lemma 3: $\varphi(\cdot)$ preserves the order of *lexicographically no greater* ($\preceq$), *i.e.*, $-\boldsymbol{g}(\boldsymbol{x}^*) \preceq -\boldsymbol{g}(\boldsymbol{x}) \iff \varphi(-\boldsymbol{g}(\boldsymbol{x}^*)) \leq \varphi(-\boldsymbol{g}(\boldsymbol{x}))$.

Proof: We first consider $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{Z}^K$ that satisfies $\boldsymbol{\alpha} \prec \boldsymbol{\beta}$. If we use the integer $\tilde{k} (1 \leq \tilde{k} \leq K)$ to represent the first non-zero element of $\langle \boldsymbol{\alpha} \rangle' - \langle \boldsymbol{\beta} \rangle'$, we have $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} < \langle \boldsymbol{\beta} \rangle_{\tilde{k}}$.

If $\tilde{k} = K$, assume $\langle \boldsymbol{\alpha} \rangle_K = n$, then $\langle \boldsymbol{\beta} \rangle_K \geq n+1$.

$$\begin{aligned} \varphi(\boldsymbol{\alpha}) &= \sum_{k=1}^{K} K^{\langle \boldsymbol{\alpha} \rangle_k} = K^{\langle \boldsymbol{\alpha} \rangle_K} + \sum_{k=1}^{K-1} K^{\langle \boldsymbol{\alpha} \rangle_k} \\ &\leq K^{\langle \boldsymbol{\alpha} \rangle_K} + (K-1)K^{\langle \boldsymbol{\alpha} \rangle_K} = K \cdot K^{\langle \boldsymbol{\alpha} \rangle_K} = K^{n+1}, \end{aligned}$$

where the inequality holds because $\langle \boldsymbol{\alpha} \rangle_K \geq \langle \boldsymbol{\alpha} \rangle_k, \forall 1 \leq k \leq K-1$.

$$\begin{aligned} \varphi(\boldsymbol{\beta}) &= \sum_{k=1}^{K} K^{\langle \boldsymbol{\beta} \rangle_k} = K^{\langle \boldsymbol{\beta} \rangle_K} + \sum_{k=1}^{K-1} K^{\langle \boldsymbol{\beta} \rangle_k} \\ &> K^{\langle \boldsymbol{\beta} \rangle_K} + (K-1) \cdot 0 \geq K^{n+1}. \end{aligned}$$

Hence, we have $\varphi(\boldsymbol{\alpha}) < \varphi(\boldsymbol{\beta})$ for the case of $\tilde{k} = K$.

If $\tilde{k} < K$, we have $\langle \boldsymbol{\alpha} \rangle_k = \langle \boldsymbol{\beta} \rangle_k, \forall \tilde{k} \leq k \leq K$ and $\langle \boldsymbol{\alpha} \rangle_{\tilde{k}} <$

$\langle\boldsymbol{\beta}\rangle_{\tilde{k}}$. Assume $\langle\boldsymbol{\alpha}\rangle_{\tilde{k}} = m$, then $\langle\boldsymbol{\beta}\rangle_{\tilde{k}} \geq m+1$.

$$\begin{aligned}\varphi(\boldsymbol{\alpha}) &= \sum_{k=1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} + K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} + \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\alpha}\rangle_k} \\ &\leq \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} + K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} + (\tilde{k}-1)K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} \\ &= \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} + \tilde{k}\cdot K^{\langle\boldsymbol{\alpha}\rangle_{\tilde{k}}} = \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} + \tilde{k}K^m,\end{aligned}$$

where the inequality holds as $\langle\boldsymbol{\alpha}\rangle_{\tilde{k}} \geq \langle\boldsymbol{\alpha}\rangle_k, \forall 1 \leq k \leq \tilde{k}-1$.

$$\begin{aligned}\varphi(\boldsymbol{\beta}) &= \sum_{k=1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} = \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{\langle\boldsymbol{\beta}\rangle_{\tilde{k}}} + \sum_{k=1}^{\tilde{k}-1} K^{\langle\boldsymbol{\beta}\rangle_k} \\ &> \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{\langle\boldsymbol{\beta}\rangle_{\tilde{k}}} + (\tilde{k}-1)\cdot 0 \\ &\geq \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + K^{m+1} > \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} + \tilde{k}K^m.\end{aligned}$$

Given that $\sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=\tilde{k}+1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k}$, we have $\varphi(\boldsymbol{\alpha}) < \varphi(\boldsymbol{\beta})$ for the case of $\tilde{k} < K$.

If $\boldsymbol{\alpha} = \boldsymbol{\beta}$, which means that $\langle\boldsymbol{\alpha}\rangle_k = \langle\boldsymbol{\beta}\rangle_k, \forall 1 \leq k \leq K$, it is trivially true that $\varphi(\boldsymbol{\alpha}) = \sum_{k=1}^{K} K^{\langle\boldsymbol{\alpha}\rangle_k} = \sum_{k=1}^{K} K^{\langle\boldsymbol{\beta}\rangle_k} = \varphi(\boldsymbol{\beta})$. Thus, we have proved $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \Longrightarrow \varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta})$.

We further prove $\varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta}) \Longrightarrow \boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ by proving its contrapositive: $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta}) \Longrightarrow \varphi(\boldsymbol{\alpha}) > \varphi(\boldsymbol{\beta})$. $\neg(\boldsymbol{\alpha} \preceq \boldsymbol{\beta})$ implies $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$ and the first non-zero element of $\langle\boldsymbol{\alpha}\rangle' - \langle\boldsymbol{\beta}\rangle'$ is positive, which further indicates the first non-zero element of $\langle\boldsymbol{\beta}\rangle' - \langle\boldsymbol{\alpha}\rangle'$ is negative, *i.e.*, $\boldsymbol{\beta} \prec \boldsymbol{\alpha}$. Thus, the contrapositive is equivalent to $\boldsymbol{\beta} \prec \boldsymbol{\alpha} \Longrightarrow \varphi(\boldsymbol{\beta}) < \varphi(\boldsymbol{\alpha})$, which has already been proved previously using the exchanged notations of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

With $\boldsymbol{\alpha} \preceq \boldsymbol{\beta} \iff \varphi(\boldsymbol{\alpha}) \leq \varphi(\boldsymbol{\beta})$ holding for any $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ of the same dimension, we are done with the proof by letting $\boldsymbol{\alpha} = -\boldsymbol{g}(\boldsymbol{x}^*)$ and $\boldsymbol{\beta} = -\boldsymbol{g}(\boldsymbol{x})$. ∎

Based on Lemma 2 and Lemma 3, we have

$$\operatorname*{lexmax}_{\boldsymbol{x}\in\mathcal{X}} \ \boldsymbol{g} \iff \min_{\boldsymbol{x}\in\mathcal{X}} \ \varphi(-\boldsymbol{g}) = \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} M^{-u_k^t\phi(x_{i,j}^{k,t})}$$

where the objective function $\varphi(-\boldsymbol{g})$ is a summation of the term $M^{-u_k^t\phi(x_{i,j}^{k,t})}$, which is a convex function of the single variable $x_{i,j}^{k,t}$.

Therefore, solving Problem (8) is equivalent to solving the following problem with a separable convex objective:

$$\begin{aligned}\min_{\boldsymbol{x}} \quad & \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} M^{-u_k^t\phi(x_{i,j}^{k,t})} \qquad (9)\\ \text{s.t.} \quad & \text{Constraints } (3), (4), (5) \text{ and } (6).\end{aligned}$$

C. Structure-Inspired Equivalent LP Transformation

Exploiting the problem structure of totally unimodular constraints and separable convex objective, we can use the λ-representation technique [14] to transform the Problem (9) to a linear programming problem that has the same optimal solution.

For each single integer variable $x_{i,j}^{k,t} \in \{0,1\}$, the convex function $h_{i,j}^{k,t}(x_{i,j}^{k,t}) = M^{-u_k^t\phi(x_{i,j}^{k,t})}$ can be linearized with the λ-representation as follows:

$$h_{i,j}^{k,t}(x_{i,j}^{k,t}) = \sum_{s\in\{0,1\}} M^{-u_k^t s}\lambda_{i,j}^{k,t,s} = \lambda_{i,j}^{k,t,0} + M^{-u_k^t}\lambda_{i,j}^{k,t,1}$$

which removes the variable $x_{i,j}^{k,t}$ by sampling at each of its possible value $s \in \{0,1\}$, weighted by the newly introduced variables $\lambda_{i,j}^{k,t,s} \in \mathbb{R}^+, \forall s \in \{0,1\}$ that satisfy

$$\begin{aligned} x_{i,j}^{k,t} &= \sum_{s\in\{0,1\}} s\lambda_{i,j}^{k,t,s} = \lambda_{i,j}^{k,t,1} \\ \sum_{s\in\{0,1\}} \lambda_{i,j}^{k,t,s} &= \lambda_{i,j}^{k,t,0} + \lambda_{i,j}^{k,t,1} = 1\end{aligned}$$

Further, with linear relaxation on the integer constraints (6), we obtain the following linear programming problem:

$$\begin{aligned}\min_{\boldsymbol{x},\boldsymbol{\lambda}} \quad & \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} (\lambda_{i,j}^{k,t,0} + M^{-u_k^t}\lambda_{i,j}^{k,t,1}) \qquad (10)\\ \text{s.t.} \quad & x_{i,j}^{k,t} = \lambda_{i,j}^{k,t,1}, \ \ \forall i,j \in \mathcal{N}, k \in \mathcal{K}, t \in \mathcal{T} \\ & \lambda_{i,j}^{k,t,0} + \lambda_{i,j}^{k,t,1} = 1, \ \ \forall i,j \in \mathcal{N}, k \in \mathcal{K}, t \in \mathcal{T} \\ & \lambda_{i,j}^{k,t,0}, \lambda_{i,j}^{k,t,1}, x_{i,j}^{k,t} \in \mathbb{R}^+, \ \forall i,j \in \mathcal{N}, k \in \mathcal{K}, t \in \mathcal{T} \\ & \text{Constraints } (3), (4) \text{ and } (5).\end{aligned}$$

***Theorem** 1:* An optimal solution to Problem (10) is an optimal solution to Problem (7).

Proof: The property of total unimodularity ensures that an optimal solution to the relaxed LP problem (10) has integer values of $x_{i,j}^{k,t}$, which is an optimal solution to Problem (9), and thus an optimal solution to Problem (8) as demonstrated in the previous subsection. Moreover, Problem (7) and (8) are equivalent forms, completing the proof. ∎

Therefore, an optimal schedule that maximizes the worst utility among all the coflows can be obtained by solving Problem (10) with efficient LP solvers, such as MOSEK [15].

V. Iteratively Optimizing Worst Utilities to Achieve Max-Min Fairness

With the subproblem of maximizing the worst utility efficiently solved as an LP problem (10), we continue to solve our original multi-objective problem (1) by maximizing the next worst utility repeatedly.

After solving the subproblem, it is known that the optimal worst utility is achieved by coflow k^* at time slot t^*, when its slowest flow from server i^* to j^* completes. We then fix the computed schedule of all the i^*-to-j^* flows at time t^*, which means that the corresponding schedule variables $x_{i^*,j^*}^{k,t^*}, \forall k \in \mathcal{K}$ are removed from the variable set $\boldsymbol{x}$ for the next round. Also, since coflow k^* completes at time t^*, it is intuitive that all the scheduling variables associated with it after time t^* should be fixed as zero and removed: $x_{i,j}^{k^*,t} = 0, \forall i,j \in \mathcal{N}, t \in \{t^*+1, t^*+2, \cdots, T\}$.

As we have fixed a part of the schedule, link capacities and remaining flow sizes should be updated in problem constraints

in the next round. For example, if $x^{1,1}_{i^*,j^*} = 1$, which means that at time slot 1, the i^*-to-j^* flow that belongs to coflow 1 would be scheduled, then for the problem in the next round, $x^{1,1}_{i^*,j^*}$ is no longer the variable. The link capacity constraints should be updated as $\sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{N},(j,k)\neq(j^*,1)} x^{k,1}_{i^*,j} \leq 0$ and $\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{N},(i,k)\neq(i^*,1)} x^{k,1}_{i,j^*} \leq 0$, which means that there is no capacity at the outgoing link of server i^* and the incoming link of server j^* at time slot 1 to schedule other flows; the flow size capacity constraint should be updated as $\sum_{t\in\{2,3,\cdots,T\}} x^{1,t}_{i^*,j^*} = D^1_{i^*,j^*} - 1$.

Moreover, the utility of coflow k^* is obtained as $u^{t^*}_{k^*}$, yet the schedules of all its flows except the slowest have not been fixed, which would be the variables $(x^{k^*,t}_{i,j}, \forall (i,j) \neq (i^*,j^*), \forall t \in \{1,2,\cdots,t^*\})$ of the problem in the next round. We set the associated utilities of these variables as $u^{t^*}_{k^*} x^{k^*,t}_{i,j}$. The rationale is that no matter how fast other flows of k^* complete, the utility is determined by the slowest flow that finishes at t^*. This ensures that the utility optimized in the next round is achieved by another coflow, rather than a flow of coflow k^* (other than its slowest). This is better illustrated with the example in Fig. 5.

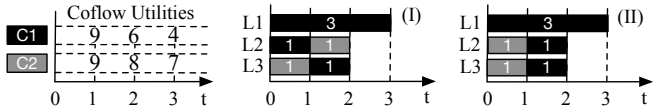

Fig. 5: Utilities and two possible schedules of coflows $C1$ and $C2$.

It is obvious that coflow $C1$ achieves the worst utility of 4. In the second round, if we do not change the associated utility for the flows of $C1$ except the slowest, schedule (I) would result in $(4,6,6,8,9,9,9)$ while schedule (II) gives $(4,6,6,6,9,9,9)$. Thus, the optimization would result in schedule (I) to achieve the optimal next worst utility as 8 for another flow of $C1$, which does not match our original objective to optimize the second worst utility for coflow $C2$. In contrast, if we set the associated utilities of all $C1$'s flows as 4, then schedule (II) would give $(4,4,4,4,4,9,9)$, better than $(4,4,4,4,4,8,9)$ given by schedule (I). In this way, the optimization can correctly choose schedule (II) to achieve the optimal utility of $C2$.

As a result, the subproblem in the next round is solved over a decreased set of variables with updated constraints and objectives, so that the next worst coflow utility would be optimized, without impacting the worst coflow utility in this round. Such procedure is repeatedly executed until the last worst utility of coflow has been optimized, and the max-min fairness has been achieved, as summarized in Algorithm 1.

VI. Real-Wrold Implementation and Performance Evaluation

Having proved the theoretical optimality of our solution, we proceed to implement a practical coflow scheduler in the real world and demonstrate its effectiveness in optimizing coflow utilities.

Design and Implementation. Varys [6] is an open-source framework that provides a simple API to data parallel jobs for coflow submission, and coordinates competing coflows

Algorithm 1: Utitity Optimal Schedule among Coflows with Max-Min Fairness.

Input:
Coflow traffic matrix $\mathbf{D}^k = \left(D^k_{i,j}\right)^N_{i,j=1}$; Time slots $\mathcal{T} = \{1,2,\cdots,T\}$; Coflow utility $u^t_k,\ \forall k \in \mathcal{K}, \forall t \in \mathcal{T}$;
Output:
Flow schedule at each time slot: $x^{k,t}_{i,j}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T}$;

```
Initialize K' = K;
while K' ≠ ∅ do
    Solve the LP Problem (10) to obtain the solution x;
    Obtain x^{j*,t*}_{k*,i*} = argmin_{x^{k,t}_{i,j} ∈ x} u^t_k x^{k,t}_{i,j};
    Fix x^{k*,t}_{i,j}, ∀i,j ∈ N, t ∈ {t*+1, t*+2, ..., T} and
    x^{k,t*}_{i*,j*}, ∀k ∈ K; remove them from variable set x;
    Update corresponding link capacities in Constraints
    (3), (4) and flow sizes in Constraints (5);
    Set u^t_{k*} = u^{t*}_{k*}, ∀t ∈ {1,2,...,t*};
    Remove k* from K';
end while
```

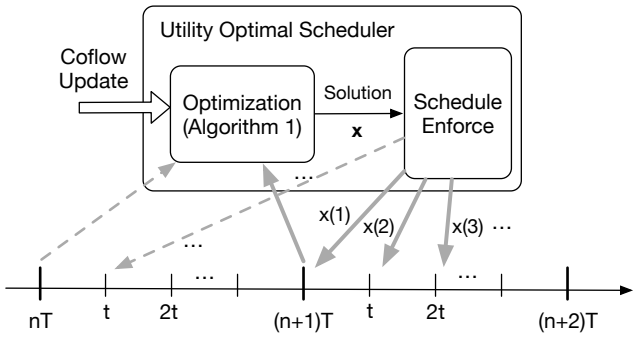

Fig. 6: An illustration of the signaling cycle of utility optimal scheduler.

through bandwidth allocation at the application layer. Our scheduler is implemented in the context of Varys to leverage the global view of the network and coflow information provided by the framework. As a natural fit for the time-slotted theoretical model, we design our scheduler to calculate and enforce scheduling decisions based on two-level time-triggered events, as shown in Fig. 6. The large time interval T consists of a certain number of small intervals t ($T = mt, m \in \mathbb{Z}^+$), responsible for decision making and enforcement, respectively. The small interval t is the duration of time required to transfer a data block of a specific size at the full line rate. This corresponds to the length of time slot in our theoretical model. At each large interval, the scheduler invokes Algorithm 1 to calculate the scheduling decisions for all the competing coflows, based on the updated information on flow sizes and coflow utilities. With the input readily available, the linear programming problem is formulated and solved using the simplex method implemented in the `Breeze` optimization library [16]. The calculated decisions will then be enforced by application-layer bandwidth allocation slot by slot, updating the rate limit to the `ThrottledInputStream` (java I/O object) for each flow at each small interval. These parameters are tuneable, with the tradeoff that a smaller T would be more responsive to flow dynamics, yet incurring more computation overhead to the scheduler.

Meaningful Utility Functions. The utility function of each coflow can be flexibly specified by a utility type and several parameters, with specific practical implications. In our experiment, we consider four types: 1) *constant* utility for background coflows, 2) *linear* utility for time-sensitive coflows, 3) *sigmoid* utility for time-sensitive coflows, and 4) *all-or-nothing* utility for time-critical coflows. Each type of utility function is defined by a few parameters, which can be set according to the desired completion time and the priority. For example, the sigmoid function has the following form:

$$\mathcal{U}(t) = \frac{p_1}{1 + e^{p_2 * (t - p_3)}}$$

where p_1 specifies the priority of the coflow, p_2 is a decay factor of the utility function, quantifying the degree of sensitivity to the coflow completion time (CCT), and p_3 denotes the desired completion time, which is the time to complete the coflow without any network contention.

For each coflow, the starting time of its utility function is the time when it is registered in the scheduler. As the utility decreases along the timeline, it accurately characterizes critical information, such as the arrival time and the sensitivity degree, for the scheduler to make the optimal decisions. As an example, for two identical coflows arriving at the same time, which have sigmoid utility functions with different decay factors, the one that decreases faster would be considered with a higher priority in the scheduler; for two identical coflows with the same utility function, which arrive at different times, the one that arrives earlier would be scheduled ahead of the other one, which naturally avoids starvation.

Experiment Setup. We deploy our utility optimal scheduler on a cluster of 6 Virtual Machine instances on Google's Cloud Compute Engine, with a total of 24 CPU cores and 156 GB memory. To allow an in-depth evaluation and analysis of our scheduler, we examine the scheduling process of 6 coflows illustrated in Table I, in comparison with the Varys Shortest-Effective-Bottleneck-First (SEBF) scheduler [6], which is a state-of-the-art coflow scheduler to minimize CCTs. For simplicity, we choose 1 GB as the size of the block to be sent with full bandwidth at each time slot. Based on the measured bandwidth, the length of the slot, *i.e.*, the small time interval, is 3 seconds, and each large interval has 10 slots.

To be more representative, the utility functions of the coflows in our case study cover all the four types. Particularly, coflow C6 is a CCT insensitive background coflow, while C3 is time critical with a deadline. The other four are time sensitive coflows, with linear or sigmoid utility functions. C1 is more sensitive than C2, and the same applies to C5 versus C4. The parameters are set with the consideration of target completion times and sensitivity degrees. For example, C1 has a total of 3 blocks to be transferred. It would achieve a utility of 1 if it completes exactly with 3 time slots, which is the target completion time. Similarly, with 2 blocks of data, C4 would obtain a utility of 1 if it finishes at the end of the second time slot. Failing to complete by the target time would result in a decrease of utility at different rates, in the linear or sigmoid form, as shown in Table I.

The comparison of coflow utilities obtained by our utility optimal scheduler and the Varys SEBF scheduler is presented in Fig. 7, which is the average of 5 runs. It is clearly shown that our scheduler improves the worst utility among all the coflows by nearly 70%, from 0.4887 achieved by C5 to 0.8377 of C4. Moreover, with our scheduler, 2 coflows have achieved a significant increase of their utilities, 2 coflows keep nearly the same utilities, while only 1 coflow experienced a slight utility decrease.

TABLE I: 6 coflows with 4 types of utility functions for case study.

Coflow Id	Flow Id	Src	Dst	Volume (GB)	Utility Function (t: slots)
C1	1 2	S1 S1	S0 S5	2 1	$\mathcal{U}(t) = 1.3 - 0.1 * t$
C2	1 2	S1 S1	S4 S5	1 2	$\mathcal{U}(t) = 1.02 - 0.01 * t$
C3	1 2 3	S3 S3 S3	S0 S1 S2	1 1 2	$\mathcal{U}(t) = 1$, if $t < 4$; $\mathcal{U}(t) = 0$, otherwise.
C4	1	S4	S2	2	$\mathcal{U}(t) = \frac{2}{1+e^{0.1*(t-2)}}$
C5	1 2	S5 S5	S1 S2	2 2	$\mathcal{U}(t) = \frac{2}{1+e^{1.1*(t-4)}}$
C6	1 2	S1 S1	S3 S4	1 1	$\mathcal{U}(t) = 1$

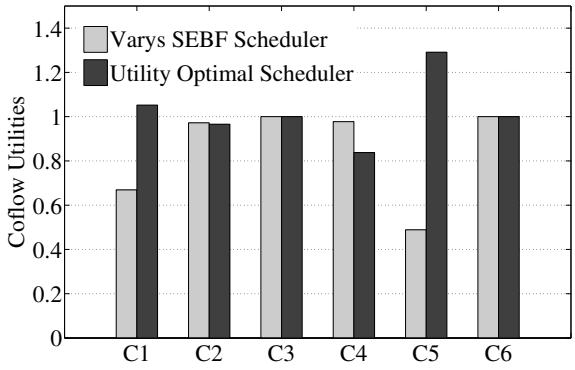

Fig. 7: Comparison of utilities achived by coflows with two schedulers.

To clearly illustrate the underlying reason for such an improvement, we next provide a detailed analysis on how our scheduler provides the differential treatment to these coflows appropriately, according to their degrees of sensitivity to CCTs. At server S1, the outgoing link is bottlenecked by flows of C1, C2 and C6. According to our utility optimal schedule presented in Fig. 8, C1 with a higher degree of CCT sensitivity is scheduled first at S1, followed by C2 whose utility decreases much more slowly. C6 always achieves the constant utility so that it is scheduled at last when the link of S1 is idle. At server S2, C3 and C5 occupies the incoming link during the first 4 time slots in an interleaved manner, so that they both complete by their target time. As a result, C4 is delayed to complete at the 6-th slot. Such a schedule is utility optimal, since C4 is much more tolerant to the delay, with a small decay factor in its sigmoid utility function. If C4 is scheduled before C3 or C5, it results in either a zero utility for the time critical C3, or a much smaller utility value of C5, whose utility decreases sharply with a large decay factor.

In comparison to our utility optimal schedule, the SEBF scheduler in Varys for minimizing CCTs treats all the coflows equally, as if they all have the same linear utility function. Without the proper differentiation among the diverse degrees of CCT sensitivity, it schedules the background coflow C6

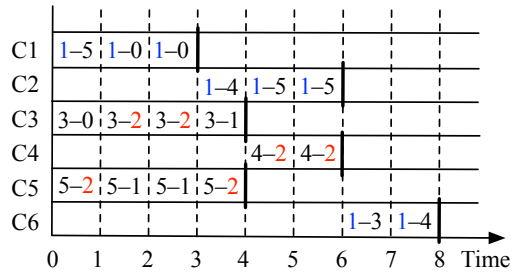

Fig. 8: Utility optimal schedule. The 1–5 block of C1 represents a block of data sent from S1 to S5 in the first time slot. The bottleneck links are highlighted with blue and red colors, respectively.

ahead of the time sensitive C1, which results in worse utilities, despite the improved CCT. In a similar vein, the less CCT sensitive C4 completes faster than C5 with the Varys scheduler, which is not utility optimal either.

Although we consider an offline case for optimality analysis, our scheduler is readily adaptive to the online scenario, where Algorithm 1 can be invoked to recalculate the optimal schedule based on the latest information as flow arrives and leaves.

VII. Related Work

With data parallel frameworks extensively deployed for big data processing, it has received an increasing amount of research attention to optimize the network allocation among coflows for better job-level performance. Varys [6] took the initiative to propose effective heuristics for coflow scheduling, in order to minimize the average coflow completion time (CCT) and meeting coflow deadlines. As a followup, Aalo [17] studied the same problem without any prior knowledge of coflows. Qiu *et al.* [7] considered the release dates of coflows and proposed the first polynomial-time approximation algorithm to minimize the weighted CCT. Extending coflow awareness into routing, RAPIER [8] designed a heuristic for joint coflow scheduling and routing to minimize CCTs.

However, all the existing efforts treat competing coflows equally in minimizing CCTs, or at most differentiate them based on whether they have deadlines or not. Such a lack of fine grained differentiation can not satisfy the diversing requirements of coflows with respect to their CCTs. In this paper, we capture these requirements with utility functions, which can quantify the degrees of sensitivity to CCTs, and our scheduling objective is to optimize all the coflow utilities with max-min fairness.

The heterogeneous sensitivity of job completion times were studied in CORA [5], which designed a utility optimal scheduler to allocate computing resource to jobs. Despite sharing a similar philosophy with CORA, our scheduling problem is quite different and more challenging, due to the inherent complexity in network scheduling that involves coupled resources.

VIII. Concluding Remarks

In this paper, we use utility functions to model different levels of sensitivity to the completion times of different coflows, in the context of data parallel jobs. With coflows competing for the network bandwidth in a shared cluster, we have designed and implemented a new utility optimal coflow scheduler to better satisfy their requirements with max-min fairness. To achieve this objective, we first formulated a lexicographical maximization problem to optimize all the coflow utilities, which is challenging due to the inherent complexity of both multi-objective and discrete optimizations. Starting from the single-objective subproblem and based on an in-depth investigation of the problem structure, we performed a series of transformations to finally obtain an equivalent linear programming (LP) problem, which can be efficiently solved in practice with a standard LP solver. With our algorithm repeatedly solving updated LP subproblems, we can optimize all the coflow utilities, with max-min fairness achieved. Last but not the least, we have implemented our utility optimal scheduler and demonstrated convincing evidence on the effectiveness of our new algorithm using real-world experiments.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for in-Memory Cluster Computing," in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2012.

[3] M. Chowdhury, M. Zaharia, J. Ma, M. Jordan, and I. Stoica, "Managing Data Transfers in Computer Clusters with Orchestra," in *Proc. ACM SIGCOMM*, 2011.

[4] M. Chowdhury and I. Stoica, "Coflow: A Networking Abstraction for Cluster Applications," in *Proc. ACM SIGCOMM HotNet Workshop*, 2012.

[5] Z. Huang, B. Balasubramanian, M. Wang, T. Lan, M. Chiang, and D. H. Tsang, "Need for Speed: CORA Scheduler for Optimizing Completion-Times in the Cloud," in *Proc. IEEE INFOCOM*, 2015.

[6] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient Coflow Scheduling with Varys," in *Proc. ACM SIGCOMM*, 2014.

[7] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2015.

[8] Y. Zhao, K. Chen, W. Bai, Y. Minlan, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "RAPIER: Integrating Routing and Scheduling for Coflow-Aware Data Center Networks," in *Proc. IEEE INFOCOM*, 2015.

[9] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 3rd ed., ser. Algorithms and Combinatorics. Springer, 2006, vol. 21, ch. 5, p. 104.

[10] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "Deconstructing Datacenter Packet Transport," in *Proc. of ACM SIGCOMM HotNet Workshop*, 2012.

[11] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the Network in Cloud Computing," in *Proc. ACM SIGCOMM*, 2012.

[12] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proc. ACM SIGCOMM*, 2009.

[13] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *Proc. ACM SIGCOMM*, 2009.

[14] R. Meyer, "A Class of Nonlinear Integer Programs Solvable by A Single Linear Program," *SIAM Journal on Control and Optimization*, vol. 15, no. 6, pp. 935–946, 1977.

[15] E. Andersen and K. Andersen, "The MOSEK Interior Point Optimizer for Linear Programming: an Implementation of the Homogeneous Algorithm," in *High performance optimization*. Springer, 2000, pp. 197–232.

[16] Breeze: A Numerical Processing Library for Scala. [Online]. Available: http://www.scalanlp.org

[17] M. Chowdhury and I. Stoica, "Efficient Coflow Scheduling Without Prior Knowledge," in *Proc. ACM SIGCOMM*, 2015.