Approximation Algorithms for Demand-Response Contract Execution and Coflow Scheduling

Zhen Qiu

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2016

ABSTRACT

Approximation Algorithms for

Demand-Response Contract Execution and Coflow Scheduling

Zhen Qiu

Solving operations research problems with approximation algorithms has been an important topic since approximation algorithm can provide near-optimal solutions to NP-hard problems while achieving computational efficiency. In this thesis, we consider two different problems in the field of optimal control and scheduling theory respectively and develop efficient approximation algorithms for those problems with performance guarantee.

Chapter 2 presents approximation algorithms for solving the optimal execution problem for demand-response contract in electricity markets. Demand side participation is essential for achieving real-time energy balance in today's electricity grid. Demand-response contracts, where an electric utility company buys options from consumers to reduce their load in the future, are an important tool to increase demand-side participation. In this chapter, we consider the operational problem of optimally exercising the available contracts over the planning horizon such that the total cost to satisfy the demand is minimized. In particular, we consider the objective of minimizing the sum of the expected ℓ_β-norm of the load deviations from given thresholds and the contract execution costs over the planning horizon. For $\beta = \infty$, this reduces to minimizing the expected peak load. The peak load provides a good proxy to the total cost of the utility as spikes in electricity prices are observed only in peak load periods. We present a data driven near-optimal algorithm for the contract execution problem. Our algorithm is a sample average approximation (SAA) based

dynamic program over a multi-period planning horizon. We provide a sample complexity bound on the number of demand samples required to compute a $(1+\varepsilon)$-approximate policy for any $\varepsilon > 0$. Our SAA algorithm is quite general and we show that it can be adapted to quite general demand models including Markovian demands and objective functions. For the special case where the demand in each period is i.i.d., we show that a static solution is optimal for the dynamic problem. We also conduct a numerical study to compare the performance of our SAA based DP algorithm. Our numerical experiments show that we can achieve a $(1+\varepsilon)$-approximation in significantly smaller numbers of samples than what is implied by the theoretical bounds. Moreover, the structure of the approximate policy also shows that it can be well approximated by a simple affine function of the state.

In Chapter 3, we study the NP-hard coflow scheduling problem and develop a polynomial-time approximation algorithm for the problem with constant approximation ratio. Communications in datacenter jobs (such as the shuffle operations in MapReduce applications) often involve many parallel flows, which may be processed simultaneously. This highly parallel structure presents new scheduling challenges in optimizing job-level performance objectives in data centers. Chowdhury and Stoica [13] introduced the coflow abstraction to capture these communication patterns, and recently Chowdhury et al. [15] developed effective heuristics to schedule coflows. In this chapter, we consider the problem of efficiently scheduling coflows so as to minimize the total weighted completion time, which has been shown to be strongly NP-hard [15]. Our main result is the first polynomial-time deterministic approximation algorithm for this problem, with an approximation ratio of $64/3$, and a randomized version of the algorithm, with a ratio of $8+16\sqrt{2}/3$. Our results

use techniques from both combinatorial scheduling and matching theory, and rely on a clever grouping of coflows.

In Chapter 4, we carry out a comprehensive experimental analysis on a Facebook trace and extensive simulated instances to evaluate the practical performance of several algorithms for coflow scheduling, including our approximation algorithms developed in Chapter 3. Our experiments suggest that simple algorithms provide effective approximations of the optimal, and that the performance of the approximation algorithm of Chapter 3 is relatively robust, near optimal, and always among the best compared with the other algorithms, in both the offline and online settings.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisors Prof. Cliff Stein, Prof. Yuan Zhong, Prof. Garud Iyengar and Prof. Vineet Goyal for their continuous support of my Ph.D study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

I would also like to thank the rest of my committee members: Prof. Javad Ghaderi for his insightful comments and encouragement, but also for the hard questions which motivated me to widen my research from various perspectives.

Last but not the least, I would like to thank my family: my parents Heming Qiu and Chunmei Shi, and my husband Zhongxia Zhou for supporting me spiritually throughout my life.

Zhen Qiu

September 5, 2016

To my parents and my husband

Chapter 1

Demand-Response Contracts Execution

1.1 Introduction

1.1.1 Background and Motivation

Due to an increasing integration of renewable sources such as wind and solar power on the grid, the supply uncertainty in the electricity market has increased significantly. Demand-side participation has become extremely important to maintain a real-time energy balance in the grid. There are several ways to increase the demand-side participation for the real-time energy balance including time of use pricing, real-time pricing for smart appliances and interruptible demand-response contracts. In this chapter, we focus on the interruptible demand-response contracts as a tool for increased demand-side participation. A demand-response contract is a contract or an option that an electric utility company can buy from the customers to interrupt or reduce their load by a specified amount, a specified number of times until the expiration of the contract.

Typically an electric utility forecasts the day-ahead load and buys the forecast load in the day-ahead market. If the actual load turns out to be higher than the forecast, the utility can buy the difference in the real-time market by paying the real-time spot price that can be significantly higher than the day-ahead price, especially when the supply is scarce. Alternatively, the utility can exercise the available demand-response contracts (if any) to offset the imbalance instead of paying the real-time spot price. Therefore, these contracts help to achieve the real-time supply-demand balance by increasing the demand-side participation through the actions of the utility company.

In this chapter, we consider the operational problem of optimally exercising the demand-response contracts over the planning horizon from the perspective of the utility. At each time period in the planning horizon, the goal is to decide on the number of contracts to exercise, if any, such that the total cost of satisfying demand is minimized. As is the case with all option exercising problems, there is a tradeoff between exercising the options now or saving them for future periods. In order to minimize the total cost one needs to model the dynamics for the real-time electricity price in addition to the uncertainty in demand. The real-time price of electricity at any location, also referred to as the *locational marginal price*, is computed from the optimal dual prices of the energy balance constraint corresponding to that location in the linearized power flow problem. This makes modeling the real-time price dynamics quite challenging. In order to avoid this, we use a different objective function that provides a good proxy to the total cost. The real-time electricity price is high typically when the demand is high. Therefore, we consider the objective of minimizing the expected ℓ_β norm of demand deviations above a threshold. When $\beta = \infty$, the objective reduces to minimizing the expected peak load. This provides a good approximation to minimiz-

ing the total cost. Moreover, this allows a data-driven approach with minimal model assumptions where historical demand data can be used to model the uncertain future demand.

Interruptible load contracts have been considered in the literature for improving the reliability of power systems. The literature is divided in two broad problems: i) designing and pricing of interruptible load contracts, and ii) optimal dispatch or exercising of these contracts to minimize costs or improve reliability. In the regulated markets in the past, these contracts were used mainly to improve system reliability in situations of supply-demand imbalances (see [35] and [10]). With deregulation, these contracts are used as an ancillary service or as an equivalent price-based generating resource (see [47], [49], [50]).

The problem of designing and pricing such contracts has also been studied extensively (see [18], [25], and [34]). Strauss and Oren [45] proposed a methodology for designing priority pricing of interruptible load programs with an early notification option. Oren [34] proposed a double-call option which captures the effects of early notification of interruptions. Kamat and Oren [25] discussed the valuation of interruptible contracts with multiple notification times using forward contracts and option derivatives, for the supply and procurement of interruptible load programs. Baldick et al. [6] discussed both the design and execution problem and provide a good overview of the literature. Most of this work made assumptions about the demand model and the real-time price dynamics. In this chapter, we consider a data-driven approach where we make minimal assumptions about the model of demand uncertainty.

1.1.2 Basic Notation

We consider the demand-response contract execution problem to minimizing the sum of the expected ℓ_β-norm of the observed load deviations from given thresholds and the contract execution costs over the planning horizon. We formulate the contract execution problem in terms of the following quantities.

$$\begin{aligned} S_t &= \text{number of contracts available in period } t. \\ X_t &= \text{random load in period } t. \\ \Gamma_t &= \text{threshold base load in period } t. \\ g_t &= \text{execution cost in period } t. \\ n_t(X_{[t-1]}) &= \text{number of contracts to exercise at time } t. \end{aligned}$$

Here, $X_{[t]} = (X_1, X_2, \cdots, X_t)$ denotes the historical demands up to period t. We assume that the demand is realized at the end of each period and the contract execution decision is made at the beginning of the period. Therefore, the number of contracts n_t to exercise at time t must only be a function of $X_{[t-1]}$. Each contract reduces load by δ. The threshold base load Γ_t denotes the power the utility has procured in the day-ahead market for period t. To satisfy demand, X_t in period t, the utility can either buy $(X_t - \Gamma_t)_+$ from the real-time market or exercise a demand-response contract to reduce the load. Let S_1 denote the total number of demand-response contracts available in period

1. Therefore, we can formulate the optimal execution problem as follows.

$$\begin{aligned} &\min \mathbb{E}_{X_{[T]}}\left[\left(\sum_{t=1}^{T}\alpha_t\left(X_t-\Gamma_t-n_t(X_{[t-1]})\delta\right)_+^{\beta}\right)^{1/\beta}+\sum_{t=1}^{T}g_t\left(n_t(X_{[t-1]})\right)\right] \\ &\text{s.t. } \sum_{t=1}^{T}n_t(X_{[t-1]})\leq S_1 \ a.s. \end{aligned} \tag{1.1.1}$$

The ℓ_β-norm objective captures the fact that the price grows faster as the power demand increases. We also include the function $g_t\left(n_t(X_{[t-1]})\right)$ that models the cost for each execution of the contract. Note that when $g_t = 0$, we can consider the objective as sum of the load deviations raised to the power of β and ignore the $1/\beta$ exponent on the sum. For $\beta = \infty$, the ℓ_β-norm objective reduces to the peak load and we obtain the following special case of minimizing the sum of expected peak load and the execution cost.

$$\begin{aligned} &\min \mathbb{E}_{X_{[T]}}\left[\max_{t=1,\cdots,T}\alpha_t\left(X_t-\Gamma_t-n_t(X_{[t-1]})\delta\right)_+ +\sum_{t=1}^{T}g_t\left(n_t(X_{[t-1]})\right)\right] \\ &\text{s.t. } \sum_{t=1}^{T}n_t(X_{[t-1]})\leq S_1 \ a.s. \end{aligned} \tag{1.1.2}$$

1.1.3 Our contribution

Our main contributions in this chapter are the following.

Optimal policy for exchangeable demand

We consider the special case where the demand is i.i.d. or exchangeable, and the contract execution costs are zero. We show that a static solution is optimal for the dynamic problem of minimizing the ℓ_p norm objective of the load deviations from given thresholds. In particular, we show that executing an equal number of contracts evenly across the planning horizon is optimal irrespective of the

realized demand. When either the demands are not exchangeable or there is a non-zero contract execution costs, we present examples to illustrate that a static solution is no longer optimal.

Approximation for General Demand and Extension

We present a data driven near-optimal algorithm that is based on a sample average approximation (SAA) dynamic program assuming that the demand is Markovian. We provide a sample complexity bound of $O(T^2/\varepsilon^2)$ on the number of demand samples required to compute a $(1+\varepsilon)$-approximate policy for a planning horizon of T periods. The dynamic program is based on an appropriate polynomial discretization of the state space and approximating the expectation using a sample average. The main challenge is to show that the sampling and discretization error remains bounded in the multi-period problem. Our SAA algorithm is quite general and we show that it can be adapted to quite general demand models including Markovian demand and objective function.

Computational Study

We also conduct a computational study to compare the performance of our SAA based DP algorithm. In particular, we compare the performance of our algorithm with respect to the number of samples. In our numerical experiments, we observe that we can obtain a $(1+\varepsilon)$-approximation for significantly smaller number of samples that the bound of $O(T^2/\varepsilon^2)$ given by the theoretical analysis. This is indicated by a fast convergence of the value function and the cost of policy as the sample size increases. We also analyze the near-optimal policy computed by our algorithm and observe that the policy can be well approximated by a piecewise affine function of the states with appropriate rounding to obtain a integer decision on the number of contracts. Since piecewise affine approximation adds complexity to the problem, we make use of a simple affine function

of the loads to approximate the policy and show that the true cost under the approximated affine policy is close to the optimal.

Outline. The rest of this chapter is organized as follows. We first introduce the model notations and formulate the problem under certain assumptions. In Section 2, we show the equivalence of static and the dynamic solution for exchangable demand and peak load objective function. In Section 3, we present the SAA approximate DP based algorithm and the analysis of the sample complexity bound used in our algorithm. We present extensions to more general demand models and objectives in Section 4 and present numerical results in Section 5.

1.2 Optimal Policies for Exchangeable Demand

In this section, we study the structural property of optimal execution policies for a special setting when the demand sequence $\{X_t : t \geq 1\}$ is exchangeable and the cost of executing the contracts $g_t \equiv 0$ for all $t = 1, \ldots, T$. We prove that a static solution is optimal in this case. This implies that the optimal dynamic solution does not depend on the history of demand.

Without loss of generality, we assume that $\Gamma_t = 0$ and $\alpha_t = 1$ for all $t = 1, \ldots, T$ (otherwise, we can appropriately change the demand distribution). Therefore, at each period, we can formulate the optimization problem as follows.

$$\begin{aligned} V_t(S_t) = \min \mathbb{E}_{X_u : t \leq u \leq T} & \left[\sum_{u=t}^{T} \left(X_u - n_u(X_{[u-1]})\delta \right)_+^{\beta} \right] \\ s.t. & \sum_{u=t}^{T} n_u(X_{[u-1]}) \leq S_t, \end{aligned} \tag{1.2.1}$$

where S_t is the number of contracts available at time t. We can reformulate (1.2.1) as follows.

$$V_t(S_t) = \min_{n_t} \mathbb{E}_{X_t}\left[(X_t - n_t\delta)_+^\beta + V_{t+1}(S_t - n_t)\right]$$
$$s.t.\ n_t \leq S_t \tag{1.2.2}$$

Theorem 1.2.1. *Suppose demand X_t is exchangeable and the execution cost $g_t = 0$ for all $t = 1,\ldots,T$. Then a static solution is optimal for the dynamic problem* (1.2.1)*. Furthermore, for each period, the optimal solution is*

$$n_t^*(X_{[t-1]}) = \frac{S_t}{T-t+1},\ \forall X_{[t-1]}.$$

Proof. We denote $n_t(X_{[t-1]})$ as n_t for simplicity. We prove the claim by a backward induction.

Base Case. For period $j = T$, since there is only one period left, we execute all the options available. For time $j = T-1$, the problem is equivalent to solving

$$\min_{n_{T-1},n_T} \mathbb{E}_{X_{T-1},X_T}\left[(X_{T-1} - n_{T-1}\delta)_+^\beta + (X_T - (S_{T-1} - n_T)\delta)_+^\beta\right]$$
$$s.t.\ n_{T-1} \leq S_{T-1}$$

i.e., we need to balance between today and tomorrow. Note that

$$(X_{T-1} - n_{T-1}\delta)_+^\beta + (X_T - (S_{T-1} - n_T)\delta)_+^\beta$$

is a convex function in n_{T-1}. The expectation preserves the convexity, and exchangeability ensures that the function $f(s) = \mathbb{E}[(X_t - s\delta)_+^\beta]$ is independent of t. Therefore, $n_{T-1} = S_{T-1}/2$ is an optimal solution.

Induction Step. Suppose the induction hypothesis holds for $j \geq t+1$. For period t,

$$\begin{aligned} V_t(S_t) &= \min_{0\leq n_t\leq S_t} \mathbb{E}_{X_t}\left[(X_t - n_{T-1}\delta)_+^\beta + V_{t+1}(S_t - n_t)\right] \\ &= \min_{0\leq n_t\leq S_t} \mathbb{E}_{X_t}\left[(X_t - n_{T-1}\delta)_+^\beta + \mathbb{E}_{X_{t+1},\cdots,X_T}\left[\sum_{j=t+1}^{T}\left(X_j - \frac{S_t - n_t}{T-t}\delta\right)_+^\beta\right]\right] \\ &= \min_{0\leq n_t\leq S_t} \mathbb{E}_{X_t,\cdots,X_T}\left[(X_t - n_t\delta)_+^\beta + \sum_{j=t+1}^{T}\left(X_j - \frac{S_t - n_t}{T-t}\delta\right)_+^\beta\right], \end{aligned}$$

where the second equality follows from the induction hypothesis. We can show that the above function is convex in n_t. Therefore, $n_t = \frac{S_t}{T-t+1}$ is an optimal solution. □

The peak load objective is a special case of ℓ_β norm objective with $\beta = \infty$. The above arguments can be used to show static policy is optimal if we want to minimize the expected peak realized load. Theorem 1.2.1 indicates that if the demands are identically distributed and there is no cost of execution, we shall exercise the available contract uniformly over the remain time periods. The optimal decision is static and independent of the history of demand.

1.2.1 Examples for Non-optimality of Static Policy

A static solution does not remain optimal if the assumptions of demand being exchangeable or the execution cost being zero is not satisfied. We present an instance of a 3-period problem where demands are independent and there is no execution cost, but a static solution is not optimal. In

particular, we consider the following instance. Demands in periods 1 and 3 are i.i.d. as follows.

$$X_1, X_3 = \begin{cases} 60 & w.p.\ 0.55 \\ 100 & w.p.\ 0.45 \end{cases}$$

and demand in period 2, $X_2 = 78$. Let the total number of contracts $S_1 = 40$ and $\delta = 1$. Note that X_1 and X_3 are exchangeable. Suppose the objective is to minimize the expected peak load. An optimal static solution is of the form: $n_1 = n_3 = a, n_2 = S_1 - 2a$. Define

$$f(a) \quad = \quad \mathbb{E}_{X_1,X_2,X_3}\left[\max\left\{(X_1-a)_+, (X_2-(S_1-2a))_+, (X_3-a)_+\right\}\right].$$

It is easy to verify that the optimal static solution is $n_1 = n_3 = 20, n_2 = 0$.

Suppose the static solution is also an optimal dynamic solution. Therefore, we have $n_1 = 20$ and $S_2 = 20$. Suppose the demand in period 1, $X_1 = 60$ and the peak load after period 1 is 40. Therefore, in period 2, the goal is to minimize

$$g(b) = \mathbb{E}_{X_2,X_3}\left[\max\left\{40, (X_2-b)_+, (X_3-(S_2-b))_+\right\}\right].$$

It can be easily verified that $b = n_2 = 0$ is *not* optimal in this scenario. Therefore, the cost of the dynamic solution is strictly better than the static solution. Therefore, the static solution is no longer optimal for the dynamic optimization problem when the demands are not exchangeable. We

can also construct instances with exchangeable demand but non-zero execution cost where a static solution does not remain optimal.

1.3 Approximate Dynamic Program for General Demand

In this section, we consider the general demand-response contract execution problem. From the previous section, we know that a static policy is not optimal and computing an optimal dynamic policy is often intractable. We present an efficient data-driven near-optimal algorithm for the execution policy problem under mild assumptions. For exposition purposes, we consider the objective of minimizing the sum of expected peak load and execution cost, for the case of i.i.d. demand. We later relax these assumptions and show that our algorithm can be adapted for more general demand models and objective functions.

We give an efficient sample average approximation based dynamic program (DP) for the problem that computes a near-optimal execution policy. Our algorithm is based on an appropriate discretization of the state space, similar in spirit to the knapsack problem. We also approximate the expectation by the empirical mean in a data-driven approach. Therefore, we have two sources of error in discretization and sampling, and the main challenge is to show that the error propagation remains bounded in the multi-period dynamic program. We give a bound on the number of samples such that the approximation error remains bounded over T periods.

To formulate the problem, let us introduce a few notations. Let Y_t denote the peak realized load up to time t. i.e.,

$$Y_t = \max_{j=1,\ldots,t-1} \left(X_j - n_j(X_{[j-1]})\delta\right)_+.$$

Let $n_t(Y_t, S_t)$ denote the number of contracts to exercise in period t when the current peak load is Y_t and the number of contract available is S_t. We can assume without loss of generality that the optimal decision in period t is a function of Y_t and S_t. The optimization problem in period t can be formulated as

$$\begin{aligned} &\min \mathbb{E}_{X_u: t \le u \le T} \left[\max_{u=t,\cdots,T} (X_u - n_u(S_u, Y_u)\delta)_+ \right] \\ &s.t. \sum_{u=t}^{T} n_u(S_u, Y_u) \le S_t \end{aligned} \tag{1.3.1}$$

We define the value function $V_t(S_t, Y_t)$ as the minimum sum of increase in peak realized demand above Y_t and the execution cost. For simplicity, let n_t denote $n_t(X_{[t-1]})$. Therefore, we can reformulate (1.3.1) as follows.

$$V_t(S_t, Y_t) = \min_{0 \le n_t \le S_t} \mathbb{E}_{X_t} \left[(X_t - n_t\delta - Y_t)_+ + g_t(n_t) + V_{t+1}\left(S_t - n_t, Y_t + (X_t - n_t\delta - Y_t)_+\right) \right]. \tag{1.3.2}$$

for $t = 1, 2, \cdots, T$. We make the following three mild assumptions.

Assumption 1. Demand X_t is bounded for any t.

This assumption is without loss of generality if we allow a small failure probability η. Fix M such that $\mathbb{P}(X_{max} > M) < \eta$, where $X_{max} = \max_{1 \le t \le T} X_t$, is the peak demand.

Assumption 2. $\mathbb{P}(X_t > M/2) \ge 1/T$, $t = 1, \ldots, T$.

We require a lower bound on the probability that the demand is at least half the maximum demand. This assumption is not too restrictive as the lower bound is only $1/T$ and is easily satisfied

for large T. It allows us to get a lower bound on expected peak load,

$$\begin{aligned}\mathbb{P}\left(X_{max} > M/2\right) &\geq 1-\left(1-\frac{1}{T}\right)^{T} \geq 1-\frac{1}{e}\\ \mathbb{E}(X_{max}) &\geq \left(1-\frac{1}{e}\right)\frac{M}{2}.\end{aligned}$$

Assumption 3. $S_1\delta/T \leq a\mathbb{E}(X_{max})$, for some $a < 1$.

We assume that the average amount of reduction from executing contracts is a small constant fraction of the expected maximum demand. This assumption is reasonable because demand-response contracts are used to manage only peak loads and random variability which is only a small fraction of the total demand.

1.3.1 Sample Average Algorithm (SAA)

Now we are ready to present our data-driven SAA based algorithm for the contract execution problem.

We first discretize the state space. Let $K = \varepsilon M/T$. We consider discrete values of Y_t and X_t on $[0,M]$ in multiples of K. For all $t = 1,\ldots,T$, let

$$\tilde{Y}_t \triangleq \left\lceil \frac{Y_t}{K} \right\rceil, \tilde{X}_t \triangleq \left\lfloor \frac{X_t}{K} \right\rfloor, \tilde{\delta} \triangleq \left\lceil \frac{\delta}{K} \right\rceil.$$

Note that there are only $O(T/\varepsilon)$ possible values of Y_t and X_t. We define our approximate value function, $\bar{V}_t(S_t,\ \tilde{Y}_t)$, on the discrete state space where we approximate the expectation with the sample average. The detailed description appears in Algorithm 2.2.

Algorithm 1.1 FPTAS for a ε - optimal solution

1: Given $\varepsilon > 0$, let $K = \varepsilon M/T$.
2: Define the terminal value $\bar{V}_{T+1}(S_{T+1}, \tilde{Y}_{T+1}) = V_{T+1}(S_{T+1}, \tilde{Y}_{T+1}) = 0$.
3: **for** $t = T \to 1$ **do**
3: solve the following DP for the optimal strategy $\bar{n}_t^*$ by SAA:

$$\bar{V}_t\left(S_t, \tilde{Y}_t\right) = \min_{0 \le n_t \le S_t} \left\{ \frac{1}{N_t} \sum_{i=1}^{N_t} \left[\left(X_t^{(i)} - n_t\delta - \tilde{Y}_t \right)_+ + g_t(n_t) \right.\right. \\ \left.\left. + \bar{V}_{t+1}\left(S_t - n_t, \tilde{Y}_t + \left(\tilde{X}_t^{(i)} - n_t\tilde{\delta} - \tilde{Y}_t \right)_+ \right) \right] \right\}, \tag{1.3.3}$$

where

$$N_t : \text{number of samples of } X_t.$$
$$X_t^{(i)} : i^{th} \text{ sample of } X_t.$$

4: **end for**
5: **Return:** $\{\bar{n}_t^* : t = 1, \cdots, T\}$.

1.3.2 Analysis of algorithm

We make two approximations for an efficient computation of the dynamic solution. First, we discretize the state space such that the total number of states become polynomial. Secondly, we approximate the expectation in each period by a sample average. These two sources of error can propagate in the multi-period computation and possibly lead to a highly suboptimal decision. We show that if the number of samples is sufficiently large, then the solution computed in Algorithm 2.2 gives a $(1+\varepsilon)$-approximation of the optimal execution policy with high probability. In order to analyze the performance of Algorithm 2.2, we introduce two more value functions. For all $t = 1, \ldots, T$, let $\hat{V}_t$ denote the optimal dynamic solution on the discrete states. We can compute $\hat{V}_t$ as follows.

$$\hat{V}_{T+1}(S_{T+1}, \tilde{Y}_{T+1}) = V_{T+1}(S_{T+1}, \tilde{Y}_{T+1}),$$

and for all $t \leq T$,

$$\hat{V}_t\left(S_t, \tilde{Y}_t\right) = \min_{0 \leq n_t \leq S_t} \left\{ \mathbb{E}_{X_t} \left[\left(X_t - n_t\delta - \tilde{Y}_t\right)_+ + g_t(n_t) + \hat{V}_{t+1}\left(S_t - n_t, \tilde{Y}_t + \left(\tilde{X}_t - n_t\tilde{\delta} - \tilde{Y}_t\right)_+\right)\right]\right\} \quad (1.3.4)$$

Also, let $\bar{U}_t$ denote the true cost of approximate solution computed by Algorithm 2.2. Therefore,

$$\bar{U}_{T+1}(S_{T+1}, Y_{T+1}) = V_{T+1}(S_{T+1}, Y_{T+1}),$$

and for all $t \leq T$

$$\bar{U}_t(S_t, Y_t) = \mathbb{E}_{X_t}\left[(X_t - \bar{n}_t^*\delta - Y_t)_+ + g_t(n_t) + \bar{U}_{t+1}\left(S_t - \bar{n}_t^*,\, Y_t + (X_t - \bar{n}_t^*\delta - Y_t)_+\right)\right].$$

where $\bar{n}_t^*$ is the optimal solution computed by Algorithm 2.2. We prove the following sample complexity bound for Algorithm 2.2.

Theorem 1.3.1. *Suppose the number of samples $N = \tilde{O}(T^2/\varepsilon^2)$. Then the cost of the execution policy computed by Algorithm 2.2 is a $(1 + O(\varepsilon))$-approximation of the dynamic optimal solution, i.e.,*

$$\bar{U}_1(S_1, 0) \leq (1 + \tilde{O}(\varepsilon))V_1(S_1, 0).$$

with probability at least $1 - \tilde{O}\left(1/TS_1\right) - \eta$.

We first show that the discretization error is small. In particular, we can prove the following lemma.

Lemma 1.3.2. *For all S_t and Y_t, $t = 1, 2, \cdots, T$,*

$$V_t(S_t, Y_t) - K(T - t + 1) \leq \hat{V}_t(S_t,\ \tilde{Y}_t) \leq V_t(S_t, Y_t)$$

Proof. First, we show that $\hat{V}_t(S_t,\ \tilde{Y}_t) \leq V_t(S_t,\ Y_t)$ for all S_t and Y_t. At time $T+1$, $\hat{V}_{T+1}(S_{T+1},\ \lceil Y_{T+1} \rceil_K) = V_{T+1}(S_{T+1},\ Y_{T+1}) = 0$. Suppose the statement holds for t+1. Let n_t^* be the optimal solution to DP (1.3.2), then

$$\begin{aligned}
& \hat{V}_t\left(S_t,\ \tilde{Y}_t\right) \\
\leq\ & \mathbb{E}_{X_t}\left[\left(X_t - n_t^*\delta - \tilde{Y}_t\right)_+ + g_t(n_t^*) + \hat{V}_{t+1}\left(S_t - n_t^*,\ \tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+\right)\right] \\
\leq\ & \mathbb{E}_{X_t}\left[\left(X_t - n_t^*\delta - Y_t\right)_+ + g_t(n_t^*) + \hat{V}_{t+1}\left(S_t - n_t^*,\ \tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+\right)\right] \\
\leq\ & \mathbb{E}_{X_t}\left[\left(X_t - n_t^*\delta - Y_t\right)_+ + g_t(n_t^*) + V_{t+1}\left(S_t - n_t^*,\ Y_t + \left(X_t - n_t^*\delta - Y_t\right)_+\right)\right] \\
=\ & V_t(S_t,\ Y_t).
\end{aligned}$$

Here, we use the fact that

$$\tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+ = \tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+$$

by considering the following 3 cases:

(i) $X_t - n_t^*\delta - \tilde{Y}_t \geq 0$ and $X_t - n_t^*\delta - Y_t \geq 0$,

$\tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+ = \tilde{X}_t - n_t^*\tilde{\delta} = \tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+$;

(ii) $X_t - n_t^*\delta - \tilde{Y}_t < 0$ and $X_t - n_t^*\delta - Y_t < 0$,

$\tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+ = \tilde{Y}_t = \tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+$;

(iii) $X_t - n_t^*\delta - \tilde{Y}_t < 0$ and $X_t - n_t^*\delta - Y_t \geq 0$,

$\tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+ = \tilde{Y}_t = \tilde{X}_t - n_t^*\tilde{\delta} = \tilde{Y}_t + \left(\tilde{X}_t - n_t^*\tilde{\delta} - \tilde{Y}_t\right)_+$.

On the other hand, we verify that $\hat{V}_t(S_t,\, \tilde{Y}_t) \geq V_t(S_t,\, Y_t) - K(T-t+1)$ for all S_t and Y_t. At time $T+1$, $\hat{V}_{T+1}(S_{T+1},\, \lceil Y_{T+1} \rceil_K) = V_{T+1}(S_{T+1},\, Y_{T+1}) = 0$. Suppose the statement holds for t+1. Let $\hat{n}_t^*$ be the optimal solution to DP (1.3.4), then

$$
\begin{aligned}
& \hat{V}_t\left(S_t,\, \tilde{Y}_t\right) \\
= \;& \mathbb{E}_{X_t}\left[\left(X_t - \hat{n}_t^*\delta - \tilde{Y}_t\right)_+ + g_t(\hat{n}_t^*) + \hat{V}_{t+1}\left(S_t - \hat{n}_t^*,\, \tilde{Y}_t + \left(\tilde{X}_t - \hat{n}_t^*\tilde{\delta} - \tilde{Y}_t\right)_+\right)\right] \\
\geq \;& \mathbb{E}_{X_t}\left[\left(X_t - \hat{n}_t^*\delta - Y_t\right)_+ + g_t(\hat{n}_t^*) + \hat{V}_{t+1}\left(S_t - \hat{n}_t^*,\, \tilde{Y}_t + \left(\tilde{X}_t - \hat{n}_t^*\tilde{\delta} - \tilde{Y}_t\right)_+\right)\right] - K \\
\geq \;& \mathbb{E}_{X_t}\left[\left(X_t - \hat{n}_t^*\delta - Y_t\right)_+ + g_t(\hat{n}_t^*) + V_{t+1}\left(S_t - \hat{n}_t^*,\, Y_t + \left(X_t - \hat{n}_t^*\delta - Y_t\right)_+\right)\right] - K(T-t) - K \\
\geq \;& V_t(S_t,\, Y_t) - K(T-t+1).
\end{aligned}
$$

Therefore, we have

$$V_t(S_t, Y_t) - K(T-t+1) \leq \hat{V}_t(S_t,\, \tilde{Y}_t) \leq V_t(S_t, Y_t)$$

for all S_t and Y_t. □

This lemma indicates that $|\hat{V}_t - V_t| \leq K(T-t+1) < \varepsilon M$ for all S_t, Y_t. Therefore, the two value functions are close point-wise. Next, we show that the error in using the sample average

as opposed to the expectation with respect to the true distribution is small. In particular, we use Chernoff bounds to prove the following lemma.

Lemma 1.3.3. *If the number of sample $N = \tilde{O}(T^2/\varepsilon^2)$, then for any S_t and Y_t, $t = 1, 2, \cdots, T$, ,*

$$\hat{V}_t(S_t, \tilde{Y}_t) - 2K(T-t+1) \leq \bar{V}_t(S_t, \tilde{Y}_t) \leq \hat{V}_t(S_t, \tilde{Y}_t) + 2K(T-t+1)$$

with probability at least $1 - 1/T^3 S_1^3$.

Recall **Chernoff bound**:

Let $X_1, \cdots, X_N$ be i.i.d random variables with $X_i \in [0,1],\ \forall i$ and define $X = \sum_{i=1}^{N} X_i$. Let $\mathbb{E}[X_i] = \mu$. Then

$$Pr\left(\left|\frac{X}{N} - \mu\right| \geq \varepsilon\right) \leq 2e^{-\varepsilon^2 N}, \text{ where } \mathbb{E}[X_i] = \mu, \forall i.$$

Proof. At time T,

$$\begin{aligned} \hat{V}_T(S_T, \tilde{Y}_T) &= \mathbb{E}_{X_T}\left[\left(X_T - S_T\delta - \tilde{Y}_T\right)_+\right] \\ \bar{V}_T(S_T, \tilde{Y}_T) &= \frac{1}{N_T}\sum_{i=1}^{N_T}\left[\left(X_T^{(i)} - S_T\delta - \tilde{Y}_T\right)_+\right] \end{aligned}$$

Note that if we do not execute any contract, the total execution cost is zero and the value function V_1 is equal to the peak load X_{max}, which provides an upper bound on V_1 under the optimal policy. It follows that

$$\mathbb{P}(V_t > M) \leq \mathbb{P}(V_1 > M) \leq \mathbb{P}(X_{max} > M) < \eta,\ 1 \leq t \leq T.$$

This shows that M is an upper bound on the value function at time t with a failure probability of η.

We prove the induction hypothesis: for any S_t and Y_t, with the number of samples stated in the lemma,

$$\hat{V}_t(S_t,\tilde{Y}_t) - \frac{2\varepsilon M(T-t+1)}{T} \leq \bar{V}_t(S_t,\tilde{Y}_t) \leq \hat{V}_t(S_t,\tilde{Y}_t) + \frac{2\varepsilon M(T-t+1)}{T}$$

with probability at least $1 - \frac{T-t+1}{2T^3S_1^3}$.

Take $N_T = \frac{3T^2}{\varepsilon^2} log(2TS_1)$. We know from Chernoff Bound that,

$$Pr\left(\left|\frac{\bar{V}_T(S_T,\tilde{Y}_T)}{M} - \frac{V_T(S_T,\tilde{Y}_T)}{M}\right| \geq \frac{\varepsilon}{T}\right) \leq 2e^{-\varepsilon^2 N_T/T^2} \leq \frac{1}{4T^3S_1^3}$$

Therefore, $\hat{V}_T(S_T,\tilde{Y}_T) - \frac{\varepsilon M}{T} \leq \bar{V}_T(S_T,\tilde{Y}_T) \leq \hat{V}_T(S_T,\tilde{Y}_T) + \frac{\varepsilon M}{T}$ with probability $\geq 1 - \frac{1}{2T^3S_1^3}$.

For any strategy n_t, define

$$\begin{aligned}
\hat{V}_t(S_t,\tilde{Y}_t)_{n_t} &\triangleq \mathbb{E}_{X_t}\left[\left(X_t - n_t\delta - \tilde{Y}_t\right)_+ + \hat{V}_{t+1}\left(S_t - n_t,\ \left\lceil \tilde{Y}_t + \left(X_t - n_t\delta - \tilde{Y}_t\right)_+ \right\rceil_K\right)\right], \\
\bar{V}_t(S_t,\tilde{Y}_t)_{n_t} &\triangleq \frac{1}{N_t}\sum_{i=1}^{N_t}\left[\left(X_t^{(i)} - n_t\delta - \tilde{Y}_t\right)_+ + \bar{V}_{t+1}\left(S_t - n_t,\ \left\lceil \tilde{Y}_t + \left(X_t^{(i)} - n_t\delta - \tilde{Y}_t\right)_+ \right\rceil_K\right)\right].
\end{aligned}$$

Note that $\hat{V}_t(S_t,\tilde{Y}_t) = \hat{V}_t(S_t,\tilde{Y}_t)_{\hat{n}_t^*}$ and $\bar{V}_t(S_t,\tilde{Y}_t) = \bar{V}_t(S_t,\tilde{Y}_t)_{\bar{n}_t^*}$, where $\hat{n}_t^*$ and $\bar{n}_t^*$ are the optimal

solutions to DP (1.3.4) and (1.3.3) respectively. For any S_t and Y_t,

$$\begin{aligned}
& \left|\bar{V}_t(S_t,\tilde{Y}_t)_{n_t}-\hat{V}_t(S_t,\tilde{Y}_t)_{n_t}\right| \\
\leq\ & \left|\frac{1}{N_t}\sum_{i=1}^{N_t}\left(\left(X_t^{(i)}-n_t\delta\right)-\tilde{Y}_t\right)_+-\mathbb{E}_{X_t}\left[\left((X_t-n_t\delta)-\tilde{Y}_t\right)_+\right]\right| \\
& +\left|\frac{1}{N_t}\sum_{i=1}^{N_t}\bar{V}_{t+1}\left(S_t-n_t,\ \left\lceil\tilde{Y}_t+\left(X_t^{(i)}-n_t\delta-\tilde{Y}_t\right)_+\right\rceil_K\right)\right. \\
& \left.-\mathbb{E}_{X_t}\left[\bar{V}_{t+1}\left(S_t-n_t,\ \left\lceil\tilde{Y}_t+\left(X_t-n_t\delta-\tilde{Y}_t\right)_+\right\rceil_K\right)\right]\right| \\
& +\left|\mathbb{E}_{X_t}\left[\bar{V}_{t+1}\left(S_t-n_t,\ \left\lceil\tilde{Y}_t+\left(X_t-n_t\delta-\tilde{Y}_t\right)_+\right\rceil_K\right)\right]\right. \\
& \left.-\mathbb{E}_{X_t}\left[\hat{V}_{t+1}\left(S_t-n_t,\ \left\lceil\tilde{Y}_t+\left(X_t-n_t\delta-\tilde{Y}_t\right)_+\right\rceil_K\right)\right]\right| \\
\leq\ & \frac{\varepsilon M}{T}+\frac{\varepsilon M}{T}+\frac{2\varepsilon M(T-t)}{T} \\
=\ & \frac{2\varepsilon M(T-t+1)}{T}
\end{aligned}$$

with probability $\geq 1-\frac{T-t}{2T^3S_1^3}-\frac{1}{4T^3S_1^3}-\frac{1}{4T^3S_1^3}>1-\frac{T-t+1}{2T^3S_1^3}$. We thus have

$$\begin{aligned}
\bar{V}_t(S_t,\tilde{Y}_t) &= \bar{V}_t(S_t,\tilde{Y}_t)_{\bar{n}_t^*}\leq\bar{V}_t(S_t,\tilde{Y}_t)_{\hat{n}_t^*}\leq\hat{V}_t(S_t,\tilde{Y}_t)_{\hat{n}_t^*}+\frac{2\varepsilon M(T-t+1)}{T} \\
\bar{V}_t(S_t,\tilde{Y}_t) &= \bar{V}_t(S_t,\tilde{Y}_t)_{\bar{n}_t^*}\geq\hat{V}_t(S_t,\tilde{Y}_t)_{\bar{n}_t^*}-\frac{2\varepsilon M(T-t+1)}{T} \\
&\geq \hat{V}_t(S_t,\tilde{Y}_t)_{\hat{n}_t^*}-\frac{2\varepsilon M(T-t+1)}{T}
\end{aligned}$$

with probability $\geq 1-\frac{T-t+1}{2T^3S_1^3}$. Therefore, for all S_t and Y_t,

$$\hat{V}_t(S_t,\tilde{Y}_t)-\frac{2\varepsilon M(T-t+1)}{T}\leq\bar{V}_t(S_t,\tilde{Y}_t)\leq\hat{V}_t(S_t,\tilde{Y}_t)+\frac{2\varepsilon M(T-t+1)}{T}$$

with probability $\geq 1 - \frac{1}{T^2 S_1^2}$ □

Therefore, if the number of samples N is sufficiently large, then with high probability, the SAA sampling error is small: $|\bar{V}_t - \hat{V}_t| < 2K(T-t+1) < \varepsilon M$. Note that by taking a union bound over all states, we have that the value functions $\bar{V}$ and $\hat{V}$ are close point-wise with probability at least $(1 - 1/(TS_1))$. Finally, we prove that approximated true cost function $\bar{U}_t$ is a good approximation of $\bar{V}_t$, i.e., the error $|\bar{U}_t - \bar{V}_t|$ is small. In particular, we have the following lemma.

Lemma 1.3.4. *If the number of sample $N = \tilde{O}(\frac{T^2}{\varepsilon^2})$, then for any S_t and Y_t, $t = 1, 2, \cdots, T$,*

$$\bar{V}_t(S_t,\, \tilde{Y}_t) - K(T-t+1) \leq \bar{U}_t(S_t,\, Y_t) \leq \bar{V}_t(S_t,\, \tilde{Y}_t) + 2K(T-t+1)$$

with probability $\geq 1 - 1/T^3 S_1^3$.

Proof. We prove the induction hypothesis: for any S_t and Y_t, with the number of samples stated in the lemma,

$$\bar{V}_t(S_t,\, \tilde{Y}_t) - K(T-t+1) \leq \bar{U}_t(S_t,\, Y_t) \leq \bar{V}_t(S_t,\, Y_t) + 2K(T-t+1)$$

with probability $\geq 1 - \frac{T-t+1}{2T^3 S_1^3}$. At time $T+1$, $\bar{U}_{T+1}(S_{T+1},\, Y_{T+1}) = \bar{V}_{T+1}(S_{T+1},\, \lceil Y_{T+1} \rceil_K) = 0$.

Suppose the statement holds for t+1. Let $\bar{n}_t^*$ be the optimal solution to DP (1.3.3), then

$$\begin{aligned}
& \bar{U}_t(S_t,\, Y_t) \\
= \;& \mathbb{E}_{X_t}\left[(X_t-\bar{n}_t^*\delta-Y_t)_+ + g_t(\bar{n}_t^*)+\bar{U}_{t+1}\left(S_t-\bar{n}_t^*,\, Y_t+(X_t-\bar{n}_t^*\delta-Y_t)_+\right)\right] \\
\leq \;& \mathbb{E}_{X_t}\left[(X_t-\bar{n}_t^*\delta-Y_t)_+ + g_t(\bar{n}_t^*)+\bar{V}_{t+1}\left(S_t-\bar{n}_t^*,\, \left\lceil Y_t+(X_t-\bar{n}_t^*\delta-Y_t)_+\right\rceil_K\right)\right]+2K(T-t) \\
\leq \;& \mathbb{E}_{X_t}\left[\left(X_t-\bar{n}_t^*\delta-\tilde{Y}_t\right)_+ + g_t(\bar{n}_t^*)+\bar{V}_{t+1}\left(S_t-\bar{n}_t^*,\, \left\lceil \tilde{Y}_t+\left(X_t-\bar{n}_t^*\delta-\tilde{Y}_t\right)_+\right\rceil_K\right)\right] \\
& +K(2(T-t)+1)
\end{aligned}$$

with probability $\geq 1-\frac{T-t}{2T^3S_1^3}$. Let

$$f=\mathbb{E}_{X_t}\left[\left(X_t-\bar{n}_t^*\delta-\tilde{Y}_t\right)_+ + g_t(\bar{n}_t^*)+\bar{V}_{t+1}\left(S_t-\bar{n}_t^*,\, \left\lceil \tilde{Y}_t+\left(X_t-\bar{n}_t^*\delta-\tilde{Y}_t\right)_+\right\rceil_K\right)\right]$$

Also, we know from definition that

$$\begin{aligned}
& \bar{V}_t\left(S_t,\tilde{Y}_t\right) \\
= \;& \frac{1}{N_t}\sum_{i=1}^{N_t}\left[\left(X_t^{(i)}-\bar{n}_t^*\delta-\tilde{Y}_t\right)_+ + g_t(\bar{n}_t^*)+\bar{V}_{t+1}\left(S_t-\bar{n}_t^*,\left\lceil \tilde{Y}_t+\left(X_t^{(i)}-\bar{n}_t^*\delta-\tilde{Y}_t\right)_+\right\rceil_K\right)\right]
\end{aligned}$$

By Chernoff bound,

$$Pr\left(\left|\frac{\bar{V}_t(S_t,\tilde{Y}_t)}{M}-\frac{f}{M}\right|\geq\frac{\varepsilon}{T}\right)\leq 2e^{-\varepsilon^2N_t}\leq\frac{1}{4T^3S_1^3}$$

Therefore, for any S_t and Y_t,

$$f - \frac{\varepsilon M}{T} \leq \bar{V}_t(S_t, \tilde{Y}_t) \leq f + \frac{\varepsilon M}{T}$$

with probability $\geq 1 - \frac{1}{2T^3 S_1^3}$. Recall that $K = \frac{\varepsilon M}{T}$. We have for any S_t and Y_t,

$$\bar{U}_t(S_t,\, Y_t) \leq f + K(2(T-t)+1) \leq \bar{V}_t(S_t,\, \tilde{Y}_t) + 2K(T-t+1)$$

with probability $\geq 1 - \frac{T-t}{2T^3 S_1^3} - \frac{1}{4T^3 S_1^3} \geq 1 - \frac{T-t+1}{2T^3 S_1^3} \geq 1 - \frac{1}{T^2 S_1^2}$.

We can prove the other direction analogously. At time $T+1$, $\bar{U}_{T+1}(S_{T+1},\, Y_{T+1}) = \bar{V}_{T+1}(S_{T+1},\, \lceil Y_{T+1} \rceil_K) =$ 0. Suppose the statement holds for t+1. For any S_t, Y_t,

$$\begin{aligned}
& \bar{U}_t(S_t,\, Y_t) \\
= \;& \mathbb{E}_{X_t}\left[(X_t - \bar{n}_t^* \delta - Y_t)_+ + g_t(\bar{n}_t^*) + \bar{U}_{t+1}\left(S_t - \bar{n}_t^*,\, Y_t + (X_t - \bar{n}_t^* \delta - Y_t)_+\right)\right] \\
\geq \;& \mathbb{E}_{X_t}\left[(X_t - \bar{n}_t^* \delta - Y_t)_+ + g_t(\bar{n}_t^*) + \bar{V}_{t+1}\left(S_t - \bar{n}_t^*,\, \left\lceil Y_t + (X_t - \bar{n}_t^* \delta - Y_t)_+ \right\rceil_K\right)\right] - K(T-t) \\
\geq \;& \mathbb{E}_{X_t}\left[\left(X_t - \bar{n}_t^* \delta - \tilde{Y}_t\right)_+ + g_t(\bar{n}_t^*) + \bar{V}_{t+1}\left(S_t - \bar{n}_t^*,\, \left\lceil \tilde{Y}_t + \left(X_t - \bar{n}_t^* \delta - \tilde{Y}_t\right)_+ \right\rceil_K\right)\right] - K(T-t) \\
= \;& f - K(T-t) \\
\geq \;& \bar{V}_t(S_t,\, \tilde{Y}_t) - K(T-t+1)
\end{aligned}$$

with probability $\geq 1 - \frac{T-t+1}{2T^3 S_1^3} \geq 1 - \frac{1}{T^2 S_1^2}$. □

We are now ready to prove Theorem 1.3.1.

Proof. At time $t = 1$, $Y_1 = 0$. We know from Lemma 2 that for all S_1,

$$\bar{V}_1(S_1, 0) \leq \hat{V}_1(S_1, 0) + 2KT$$

with probability at least $1 - 1/(TS_1)$. Similarly, from Lemma 3 we have that for all S_1,

$$\bar{U}_1(S_1, 0) \leq \bar{V}_1(S_1, 0) + 2KT$$

From Lemmas 1, 2 and 3, we have

$$\bar{U}_1(S_1, 0) \leq \bar{V}_1(S_1, 0) + 2KT \leq \hat{V}_1(S_1, 0) + 4KT \leq V_1(S_1, 0) + 4KT$$

with probability at least $1 - 2/(TS_1)$ for all S_1.

Let $V_1^0(.)$ be the value function when there is no execution cost, i.e. $p = 0$. We have $V_1(S_1, 0) \geq V_1^0(S_1, 0)$. From Theorem 1.2.1 and Assumptions 2, the expected peak load is at least $(1 - 1/e)M/2$ and the average amount of reduction $S_1\delta/T$ is at most aM.

$$V_1(S_1, 0) \geq V_1^0(S_1, 0) \geq \mathbb{E}[X_{max}] - \frac{S_1}{T}\delta \geq cM,$$

for some constant c. With $K = \varepsilon M/T$, we conclude that with probability at least $1 - 2/(TS_1) - \eta$,

$$\bar{U}_1(S_1, 0) \leq \left(1 + \tilde{O}(\varepsilon)\right) V_t(S_1, 0).$$

□

1.4 Extensions to More General Demand

In this section, we show that our algorithm can be easily adapted to more general demand models and objective functions to give near-optimal execution policy.

1.4.1 Markovian Demand

A Markovian model of demand $\{X_t\}_{t=0}^{T-1}$ is more realistic since demand in period $t+1$ is significantly related to demand in period t. With the Markovian assumption, the demand distribution in period $t+1$ only depends on demand in period t. Therefore, we can add one more state variable X_t for the demand in current period in the dynamic programming formulation to track the future demand distribution.

The total number of states is still polynomial and we can use our SAA approximate DP based approach as in Algorithm 2.2 to compute the near-optimal strategy.

We make the following four mild assumptions for the case with Markovian demand.

Assumption 1. Demand X_t is positive, i.e. $X_t \geq \tau$ for $\tau > 0$, and is bounded for any t.

The bound assumption is without loss of generality if we allow a small failure probability η. Let $C = \max_t \mathbb{E}[X_t]$. By Markov inequality, we have $\mathbb{P}(X_t > M) < \mathbb{E}[X_t]/M$. Let $M = TC/\eta$. It follows that $\mathbb{P}(X_{max} > M) \leq \sum_t P(X_t > M) < \sum_t \mathbb{E}[X_t]/M \leq TC/M = \eta$, where $X_{max} = \max_{1 \leq t \leq T} X_t$, is the peak demand.

Assumption 2. $\mathbb{P}(X_{max} > M/2) \geq c$ for some $c > 0$.

Algorithm 1.2 FPTAS for a ε - optimal solution (Markovian Demand)

1: Given $\varepsilon > 0$, let $\Delta = \varepsilon M/T$.
2: Define the terminal value $\bar{V}_{T+1}(S_{T+1}, Y_{T+1}, X_T) = V_{T+1}(S_{T+1}, Y_{T+1}, X_T) = 0$.
3: **for** $t = T \to 1$ **do**
3: solve the following DP for the optimal strategy $\bar{n}_t^*$ by SAA:

$$\bar{V}_t(S_t, Y_t, X_{t-1}) = \min_{0 \le n_t \le S_t} \left\{ \frac{1}{N_t} \sum_{i=1}^{N_t} \left[\left(X_t^{(i)} - n_t\delta - Y_t \right)_+ + g_t(n_t) + \bar{V}_{t+1}\left(S_t - n_t, Y_t + \left(X_t^{(i)} - n_t\delta - Y_t \right)_+, X_t^{(i)} \right) \right] \right\}, \quad (1.4.1)$$

where

N_t : number of samples of X_t.

$X_t^{(i)} \sim X_t | X_{t-1}$: i^{th} sample of X_t based on the Markovian demand process.

4: **end for**
5: **Return:** $\{\bar{n}_t^* : t = 1, \cdots, T\}$

We require a lower bound on the probability that the peak demand is at least half of the maximum bound of the demand. This assumption is not too restrictive. It allows us to get a lower bound on expected peak load,

$$\mathbb{E}(X_{max}) \quad \geq \quad c\frac{M}{2}.$$

Assumption 3. $S_1\delta/T \leq a\mathbb{E}(X_{max})$, for some $a < 1$.

We assume that the average amount of reduction from executing contracts is a small constant fraction of the expected maximum demand. This assumption is reasonable because demand-response contracts are used to manage only peak loads and random variability which is only a small fraction of the total demand.

Assumption 4. The Markovian demand X_t have the following property,

$$\frac{\min(X_{t-1}, X'_{t-1})}{\max(X_{t-1}, X'_{t-1})} f(x|X'_{t-1}) \leq f(x|X_{t-1}) \leq \frac{\max(X_{t-1}, X'_{t-1})}{\min(X_{t-1}, X'_{t-1})} f(x|X'_{t-1}).$$

If X_{t-1} and X'_{t-1} are close to each other, the conditional distribution functions are also close. This characterizes the fact that the demand at time t is expected to depend heavily on the demand at time $t-1$.

We can prove the following theorem for the SAA algorithm.

Theorem 1.4.1. *Suppose the number of sample $N = O(T^2/\varepsilon^2)$ in computing the optimal strategy for the discretized DP, then the cost of this optimal strategy is (1+ε) - optimal, i.e.,*

$$\bar{U_1}(S_1, 0) \leq (1+\varepsilon) V_1(S_1, 0)$$

with probability at least $1 - O\left(\frac{1}{TS_1}\right) - \eta$.

The proof of the theorem is similar to the case with i.i.d. demand. We provide the details of the proof in the Appendix.

1.4.2 Multiple Period Contracts

We can also offer contracts where an execution of a contract can reduce load for multiple consecutive periods. Since contracts provide reduction in more periods, previously executed contracts can last up to $\gamma - 1$ periods. The state space of the DP needs to maintain the number of active contracts for each of $\gamma - 1$ future periods. The state variable in this setting is (S_t, Y_t, Δ_{p_t}), where $\Delta_{p_t} \in \mathbb{R}^{\gamma-1}$

is a vector which tracks the active contracts in $\gamma - 1$ periods. Therefore, for constant γ, the number of states is polynomial and Algorithm 2.2 can be easily adapted to obtain near-optimal execution policies for this case.

After we define this $\gamma + 1$-dimension state variable, we can have the DP formulation for the problem and use the SAA approximate DP based algorithm.

1.4.3 Rejected Execution and Lead Times

We can also consider contracts where the customer can reject to reduce load with certain probability and pay a penalty. For instance, if the rejection is modeled as a binomial process, we can introduce a binomial random variable $B_t \sim bin(n_t, p)$ to indicate the number of the contracts that get executed and actually reduce load. Then we can compute the expected realized load conditioning on B_t and our algorithm can be adapted in a straightforward manner. We can also incorporate lead times between the contract execution request and the actual load reduction. In this case, we compute decision for period t in period $t - L$. There is no change in the state space of the DP.

Therefore, our SAA based dynamic programming approach is quite powerful and can be adapted to handle many operational constraints that can arise in practice.

1.5 Computational Study

In this section, we present a computational study to test the performance of our SAA based dynamic programming algorithm and contrast with the theoretical bounds. We also analyze the structure of the near-optimal policy computed by our algorithm.

N	$\hat{V}_1^{\varepsilon}(S_1,0)$	R.E.	$\bar{U}_1^{\varepsilon}(S_1,0)$	R.E.
100	42.99	0.20%	43.41	0.67%
200	43.09	0.02%	43.24	0.26%
400	43.14	0.14%	43.19	0.16%
800	43.05	0.07%	43.17	0.10%
1600	43.11	0.05%	43.11	0.05%
3200	43.09	0.02%	43.13	0.02%
6400	43.11	0.05%	43.14	0.03%
12800	43.08	-	43.13	-

Table 1.1: Relative Error of Value Function and True Cost for $\varepsilon = 0.1$

N	$\hat{V}_1^{\varepsilon}(S_1,0)$	R.E.	$\bar{U}_1^{\varepsilon}(S_1,0)$	R.E.
100	43.10	0.21%	43.06	0.16%
200	43.10	0.20%	43.28	0.34%
400	43.14	0.12%	43.26	0.28%
800	43.19	0.01%	43.21	0.19%
1600	43.18	0.02%	43.13	0.02%
3200	43.20	0.02%	43.18	0.10%
6400	43.20	0.02%	43.15	0.04%
12800	43.19	-	43.13	-

Table 1.2: Relative Error of Value Function and True Cost for $\varepsilon = 0.05$

1.5.1 Dependence on the Sample Size

We show that with $\tilde{O}(T^2/\varepsilon^2)$ samples, Algorithm 2.2 guarantees a $(1+\varepsilon)$-approximation of optimal policy with high probability. In practice, we may be able to obtain a $(1+\varepsilon)$-approximation with significantly smaller numbers of samples. We conduct computational experiments to study the effect of number of samples on the performance of our algorithm. In particular, we compare the value function of SAA approximated DP, $\hat{V}_1(S_1,0)$, and the cost of the policy computed by the DP, $\bar{U}_1(S_1,0)$, with respect to the sample size. For our numerical experiment, we use the following parameters: $S_1 = 30, T = 30, \delta = 1$ and there is a linear execution cost of 0.01 per contract. The load $\overset{i.i.d.}{\sim} Unif(0,45)$.

Tables 1.11 and 1.2, show the value function and cost function as the sample size increases for different level of accuracy ε. We consider the value corresponding to the highest number of samples $N = 12800$ as the best approximation of the true value and compute the relative error with respect to the best approximation for different sample sizes. Tables 1.11 and 1.2 shows that both the DP value function and the true cost function of the DP policy converge rapidly as the

sample size increases. The relative error is less than 1% which is small compared with ε. This indicates that the approximation algorithm performs well even with a smaller sample size. From the theoretical bound, we would require around 1000000 samples to compute a $(1+\varepsilon)$-approximation for $\varepsilon = 0.01$. Thus, we can use much smaller number of samples in our SAA approximated DP base algorithm to obtain a (1+ε)-approximation of the optimal policy in practice.

1.5.2 True Cost Distribution

Given a near-optimal policy, the cost of policy $\bar{U}_1(S_1,0)$ might be different from the value function of SAA approximated DP, $\hat{V}_1(S_1,0)$. Table 1.3 presents the comparison between the true cost under the approximated policy with the approximate DP value function for different ε. We also give the 99% confidence interval of true cost function and compare with the DP value function.

ε	$\hat{V}_1^{\varepsilon}(S_1,0)$	$\bar{U}_1^{\varepsilon}(S_1,0)$	99% C.I.	R.E.
0.15	42.90	43.11	(39.39, 46.83)	0.49%
0.10	43.05	43.11	(39.37, 46.85)	0.15%
0.05	43.18	43.12	(39.32, 46.93)	0.14%
0.02	43.22	43.11	(39.35, 46.88)	0.02%
0.01	43.21	43.11	(39.48, 46.74)	0.02%

Table 1.3: True Cost Distribution

Table 1.3 shows that the relative error between the value function and the true cost is at most 0.493% and is much smaller than accuracy level ε. We also observe that the relative error decreases as ε decreases. This result agrees with Lemmas 2 and 3 for our algorithm which indicate that the error $|\bar{V}_t - \hat{V}_t|$ and $|\bar{U}_t - \bar{V}_t|$ are both small with respect to the level of accuracy. The discretized value

function $\hat{V}_1(S_1, 0)$ is inside the confidence interval in all the cases and is a good approximation of the true cost of the DP policy.

1.5.3 Structure of Policy

We also study the structure of the near-optimal policy computed by our approximate SAA based DP algorithm. We plot the approximated decision n_t with respect to the state variable S_t and Y_t for $t = T-1$ and $T-2$ in Figures 1.1 and 1.2.

We observe from Figure 1.1 and 1.2 that the optimal number of contracts to execute is almost linear in S_t and Y_t and remains constant after it reaches a threshold. Therefore, we can approximate the policy by a affine function of the states with some rounding to obtain a integer decision on the number of contracts. Such a functional approximation is quite useful as it provides a compact representation of the approximate execution policy. Moreover, it provides important insights into valuation, pricing and designing of the demand-response contracts.

1.5.4 Affine Approximation

Based on our observation of the policy structure, we consider an affine approximation of the approximate near-optimal policy. For any sample path ω, we define the approximate decision function $n_t(\omega)$ as an affine function of the state $X_j(\omega)$, $j = 1, \cdots, t$,

$$n_t(\omega) = a_0^t + \sum_{j=1}^{t-1} a_j^t X_j(\omega),$$

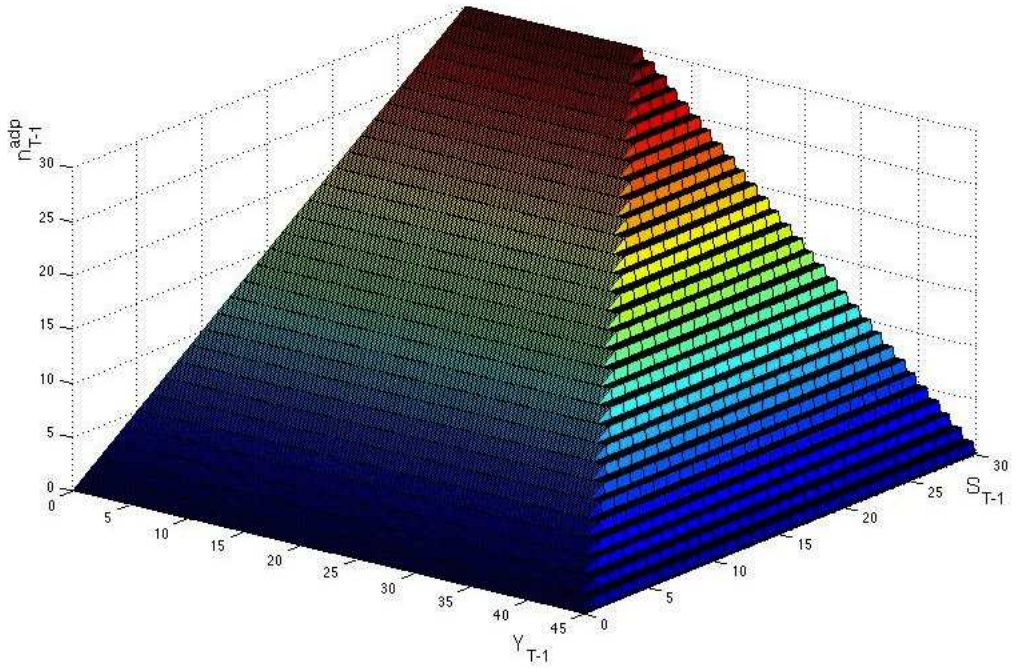

Figure 1.1: Near-optimal Policy at Period T-1

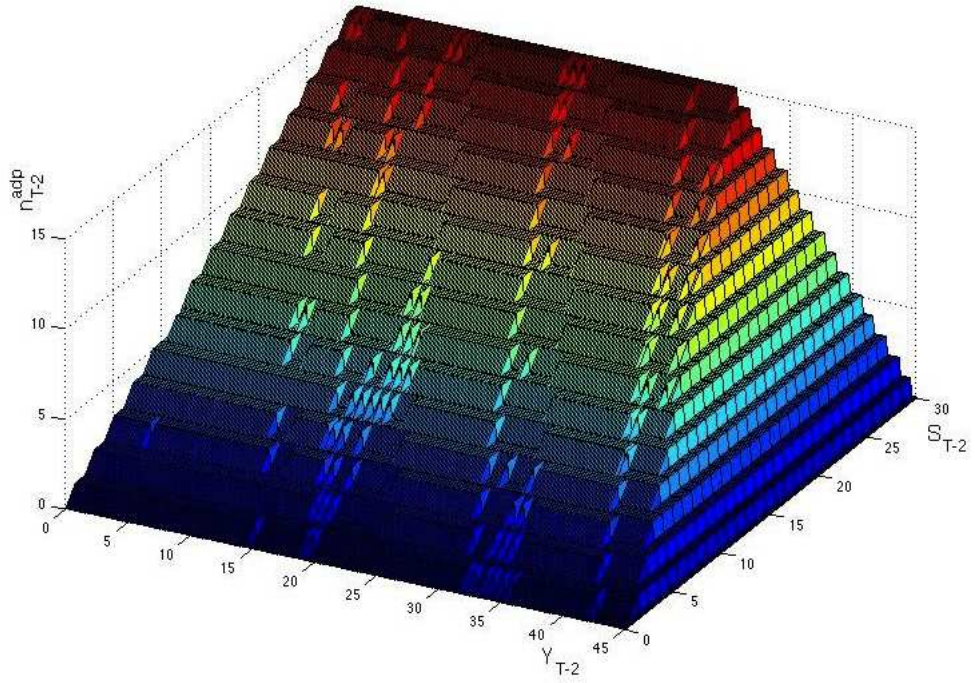

Figure 1.2: Near-optimal Policy at Period T-2

where $a_j's$ are the unknown coefficients. The approximate execution decision only depends on the previous demands. Our goal is to solve for the coefficients $a_j's$ such that the total expected execution cost is minimized. Since we are working with sample paths, we approximate the expectation by sample average and formulate the optimal execution problem as follows,

$$\begin{aligned} \min \quad & \frac{1}{N}\left(Y_T(\omega) + g_t(n_t(\omega))\right) \\ s.t. \quad & n_t(\omega) = a_0^t + \sum_{j=1}^{t-1} a_j^t X_j(\omega), \\ & \sum_{t=1}^{T} n_t(\omega) \leq S_1, \\ & Y_t(\omega) \geq X_t(\omega) - n_t(\omega), \\ & Y_t(\omega) \geq Y_{t-1}(\omega), \\ & Y_t(\omega) \geq 0. \end{aligned} \tag{1.5.1}$$

We compare the true cost $\bar{U_1}(S_1)$ with DP approximation and affine approximation for i.i.d. demand. Recall that the true cost is the sum of the peak value and the total execution cost. For our numerical experiment, we approximate up to the accuracy $\varepsilon = 0.05$ and fix the time period, $T = 15$, and the number of sample paths, $N = 1000$, which is sufficient according to the computational experiment in previous section. We use a linear execution cost function, so there is an execution cost of p per contract. We test for different value of S_1 and the per unit execution cost p. p is chosen to achieve both the trivial and non-trivial optimal strategies. To be more specific, in our instance, if $p = 0.08$, we will not execute any contract as the marginal cost is higher than the peak reduction. On the contrary, if $p = 0.04$, we will use up all the contracts to reduce the peak load. We

show the results in Table 1.4 for loads $X_t \overset{i.i.d.}{\sim} Unif[0,5]$, in Table 1.5 for $X_t \overset{i.i.d.}{\sim} N(2.5,2)$ truncated on $[0,5]$ and in Table 1.6 for $X_t \overset{i.i.d.}{\sim} N(2.5,0.4)$ truncated on $[0,5]$.

(S_1,p)	$\bar{U}_1^{DP}(S_1)$ (S.E.)	$\bar{U}_1^{Aff}(S_1)$ (S.E.)	Rel. Err.
(45, 0.08)	4.64(0.29)	4.70(0.29)	1.29%
(45, 0.06)	4.34(0.31)	4.41(0.31)	1.61%
(45, 0.04)	3.47(0.32)	3.51(0.31)	1.15%
(30, 0.08)	4.64(0.30)	4.70(0.29)	1.29%
(30, 0.06)	4.43(0.33)	4.51(0.31)	1.81%
(30, 0.04)	3.87(0.32)	3.91(0.31)	1.03%
(15, 0.08)	4.64(0.30)	4.70(0.29)	1.29%
(15, 0.06)	4.54(0.33)	4.61(0.31)	1.54%
(15, 0.04)	4.27(0.32)	4.31(0.31)	0.94%

Table 1.4: Comparison of $\bar{U}_1(S_1)$ with DP approximation and affine approximation for uniform load.

(S_1,p)	$\bar{U}_1^{DP}(S_1)$ (S.E.)	$\bar{U}_1^{Aff}(S_1)$ (S.E.)	Rel. Err.
(45, 0.08)	4.38(0.39)	4.41(0.41)	0.68%
(45, 0.06)	4.03(0.40)	4.13(0.42)	2.48%
(45, 0.04)	3.19(0.44)	3.20(0.44)	0.31%
(30, 0.08)	4.37(0.39)	4.42(0.40)	1.14%
(30, 0.06)	4.14(0.39)	4.25(0.45)	2.66%
(30, 0.04)	3.61(0.40)	3.63(0.41)	0.55%
(15, 0.08)	4.37(0.39)	4.42(0.40)	1.14%
(15, 0.06)	4.25(0.41)	4.34(0.44)	2.12%
(15, 0.04)	3.98(0.43)	4.04(0.43)	1.51%

Table 1.5: Comparison of $\bar{U}_1(S_1)$ with DP approximation and affine approximation for $N(2.5,2)$ truncated on $[0,5]$.

We see from Table 1.5 and Table 1.4 that the true costs with DP approximation and affine approximation are very close to each other for both distributions, although the peak values and the total execution costs may not be close respectively under the two strategies. This is reasonable as we are approximating the execution strategy using a linear affine function, instead of a piecewise affine function. Note that in the case if the optimal strategy is trivial, the affine approximation is

(S_1, p)	$\bar{U}_1^{DP}(S_1)$ (S.E.)	$\bar{U}_1^{Aff}(S_1)$ (S.E.)	Rel. Err.
(45, 0.08)	3.58(0.34)	3.61(0.36)	0.84%
(45, 0.06)	3.24(0.33)	3.32(0.36)	2.48%
(45, 0.04)	2.34(0.33)	2.35(0.36)	0.43%
(30, 0.08)	3.58(0.34)	3.61(0.35)	0.84%
(30, 0.06)	3.31(0.33)	3.41(0.34)	3.02%
(30, 0.04)	2.76(0.33)	2.81(0.36)	0.55%
(15, 0.08)	3.56(0.34)	3.60(0.33)	1.12%
(15, 0.06)	3.44(0.34)	3.52(0.33)	2.33%
(15, 0.04)	3.17(0.33)	3.21(0.35)	1.26%

Table 1.6: Comparison of $\bar{U}_1(S_1)$ with DP approximation and affine approximation for $N(2.5, 0.4)$ truncated on $[0, 5]$

even closer to the DP approximation. Since affine approximation does not depend on ε while the running time of DP approximate increases significantly as ε decreases, affine approximation is a good alternative to compute the execution strategy when a small ε is required.

Note that we allow execution of fractional contract in the affine approximation. In order to make a more fair comparison, we round the solution from affine approximation algorithm in the following way. Given the number of contract in time t, n_t, we execute $\lceil n_t \rceil$ with probability $n_t - \lfloor n_t \rfloor$, and $\lfloor n_t \rfloor$, with probability $n_t - \lceil n_t \rceil$. Further, in the true execution of the contracts, we assume that the number of contracts available at the very beginning is the maximum multiple of the time period which is not greater than the actual number of contracts. This will ensure that we won't use up more contracts than needed because we could take the ceiling of the affine approximation solution. We call this heuristic randomize rounding of affine approximation with modified contracts.

We also compare our DP approximation with the static solution of the optimization problem (1.3.1), which is obtained by solving the integer program.

	DP	Affine (continuous)		Affine(randomized)		Static
(S_1, p)	$\bar{U}_1^{DP}(S_1)$	$\bar{U}_1^{Aff}(S_1)$	Rel. Err.	$\bar{U}_1^{H}(S_1)$	Rel. Err.	$\bar{U}_1^{S}(S_1)$
(70, 0.06)	4.21(0.26)	4.28(0.29)	1.66%	4.29(0.28)	1.90%	
(65, 0.06)	4.20(0.26)	4.30(0.29)	2.38%	4.28(0.28)	1.90%	
(60, 0.06)	4.23(0.28)	4.31(0.31)	1.89%	4.31(0.31)	1.89%	4.23(0.26)
(59, 0.06)	4.23(0.30)	4.37(0.39)	3.31%	4.38(0.31)	3.55%	
(58, 0.06)	4.23(0.31)	4.61(0.69)	8.98%	4.38(0.30)	3.55%	
(57, 0.06)	4.24(0.32)	4.92(0.99)	16.04%	4.39(0.30)	3.54%	
(55, 0.06)	4.26(0.33)	5.14(1.27)	20.66%	4.40(0.30)	3.29%	
(53, 0.06)	4.27(0.31)	5.03(1.16)	17.80%	4.39(0.29)	2.81%	
(50, 0.06)	4.30(0.33)	4.75(0.99)	10.47%	4.38(0.32)	1.86%	
(45, 0.06)	4.32(0.32)	4.42(0.33)	2.32%	4.42(0.33)	2.32%	4.35(0.30)
(43, 0.06)	4.32(0.32)	4.67(0.67)	8.10%	4.48(0.29)	3.70%	
(40, 0.06)	4.36(0.33)	4.95(0.86)	12.84%	4.48(0.31)	2.75%	
(35, 0.06)	4.39(0.31)	4.85(0.82)	10.48%	4.46(0.31)	1.59%	
(30, 0.06)	4.42(0.31)	4.50(0.33)	2.26%	4.50(0.33)	1.81%	
(20, 0.06)	4.49(0.30)	4.56(0.36)	1.56%	4.58(0.3)	2.00%	

Table 1.7: Comparison of $\bar{U}_1(S_1)$ with DP approximation and affine approximation for $N(2.5, 0.4)$ truncated on $[0, 5]$.

	DP	Static	
(S_1, p)	$\bar{U}_1^{DP}(S_1)$	$\bar{U}_1^{S}(S_1)$	Rel. Err.
(45, 0.09)	4.26(0.41)	4.32(0.37)	1.41%
(35, 0.09)	4.29(0.49)	4.38(0.49)	2.10%
(25, 0.09)	4.37(0.52)	4.47(0.51)	2.29%
(15, 0.09)	4.49(0.54)	4.57(0.51)	1.78%

Table 1.8: Comparison of $\bar{U}_1(S_1)$ with DP approximation and static approximation for Markovian demand.

1.5.5 Comparison of i.i.d. demand and Markovian stationary demand.

In this section, we present a computational study to compare the i.i.d. demand and Markovian stationary demand. We also analyze the structure of the near-optimal policy computed by our algorithm. We consider a polynomial number of discretized value for X_t and Y_t and define Q to be

p = 0.06	DP	Affine (randomized)		Static	
S_1	$\bar{U}_1^{DP}(S_1)$	$\bar{U}_1^{H}(S_1)$	Rel. Err.	$\bar{U}_1^{S}(S_1)$	Rel. Err.
90	9.11(1.17)	9.54(0.91)	4.72%	9.70(1.44)	6.48%
75	9.12(1.30)	9.57(1.08)	4.93%	9.70(1.44)	6.36%
70	9.13(1.34)	9.57(1.13)	4.82%	9.70(1.44)	6.24%
60	9.19(1.38)	9.60(1.25)	4.46%	9.70(1.44)	5.55%
55	9.24(1.42)	9.60(1.28)	3.90%	9.70(1.44)	5.00%
50	9.29(1.34)	9.61(1.33)	3.44%	9.70(1.44)	4.41%
45	9.37(1.31)	9.63(1.36)	2.77%	9.70(1.44)	3.52%
40	9.38(1.29)	9.64(1.36)	2.77%	9.70(1.44)	3.41%
35	9.43(1.26)	9.65(1.39)	2.33%	9.70(1.44)	2.86%
30	9.47(1.27)	9.67(1.42)	2.11%	9.70(1.44)	2.43%
25	9.51(1.27)	9.68(1.40)	1.78%	9.71(1.44)	2.10%
20	9.56(1.28)	9.69(1.40)	1.36%	9.71(1.44)	1.57%
15	9.59(1.30)	9.70(1.44)	1.15%	9.71(1.44)	1.25%

Table 1.9: Comparison of $\bar{U}_1(S_1)$ with DP approximation and static approximation for i.i.d. demand X_t.

the transition matrix of the discretized demand process.

$$Q_{ij} = P(X_{t+1} = j | X_t = i), \quad \sum_j Q_{ij} = 1.$$

The demand in the Markovian model is no longer independent. We consider a simple example where X_t's are all identically uniformly distributed. Since the Markov chain is stationary, we shall have $\sum_i Q_{ij} = 1$ with uniform demand, which indicates that Q is a doubly stochastic matrix. We further impose the following monotonicity assumption on the transition matrix,

$$\beta Q_{ij_1} \leq Q_{ij_2}, \ \forall j_1 > j_2 \geq i$$

$$\beta Q_{ij_1} \leq Q_{ij_2}, \ \forall j_1 < j_2 \leq i$$

p = 0.06	DP	Affine (randomized)		Static	
S_1	$\bar{U}_1^{DP}(S_1)$	$\bar{U}_1^{H}(S_1)$	Rel. Err.	$\bar{U}_1^{S}(S_1)$	Rel. Err.
90	8.34(0.93)	8.67(1.06)	3.96%	8.77(1.13)	5.16%
75	8.34(0.97)	8.67(1.07)	3.96%	8.77(1.13)	5.16%
70	8.34(1.01)	8.67(1.13)	3.96%	8.77(1.13)	5.16%
60	8.34(1.02)	8.68(1.13)	4.08%	8.77(1.13)	5.16%
55	8.35(1.00)	8.68(1.12)	3.95%	8.77(1.13)	5.03%
50	8.37(1.01)	8.70(1.12)	3.94%	8.77(1.13)	4.78%
45	8.38(1.00)	8.72(1.14)	4.06%	8.77(1.13)	4.65%
40	8.39(1.02)	8.72(1.14)	3.93%	8.77(1.13)	4.53%
35	8.41(1.02)	8.73(1.14)	3.80%	8.77(1.13)	4.28%
30	8.42(1.03)	8.75(1.14)	3.92%	8.77(1.13)	4.15%
25	8.48(1.03)	8.77(1.15)	3.42%	8.78(1.16)	3.54%
20	8.53(1.05)	8.76(1.15)	2.70%	8.78(1.16)	2.93%
15	8.59(1.04)	8.77(1.15)	2.10%	8.78(1.16)	2.21%

Table 1.10: Comparison of $\bar{U}_1(S_1)$ with DP approximation and static approximation for i.i.d. demand X_t.

N	$V(S_1,0)$ - i.i.d. demand	$V(S_1,0)$ - Markovian demand
2000	3.8354624	2.9465

Table 1.11: Comparison of i.i.d. demand and Markovian stationary demand.

The monotonicity assumption captures the fact that the demand in period $t+1$ is more likely to stay close to the demand in period t.

For our numerical experiment, we use the following parameters: $S_1 = 3, T = 3, \delta = 1$. The load $\sim Unif(0,6)$. The cost of executing one contract is 0.1. We see in table 1.11 that the cost of the near-optimal policy is lower with the Markovian demand than with i.i.d demand, because we get more information about the load level given all the demand history, which helps us to make a better decision.

In Figure 1.3 and 1.4, we compare the structure of the near-optimal policy from our SAA algorithm with respect to the current load level. With the same S_t and Y_t, we tend to execute the

policy instead of leaving for the future when the current demand is high, because we know that it is highly likely that the load will keep at a high level.

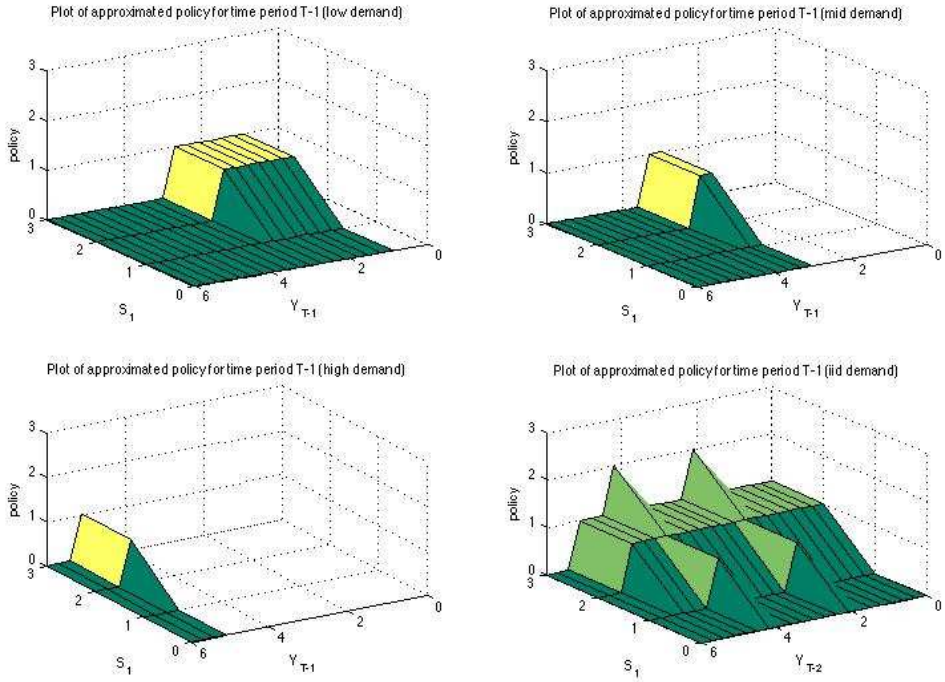

Figure 1.3: Near-optimal Policy at Period T-1

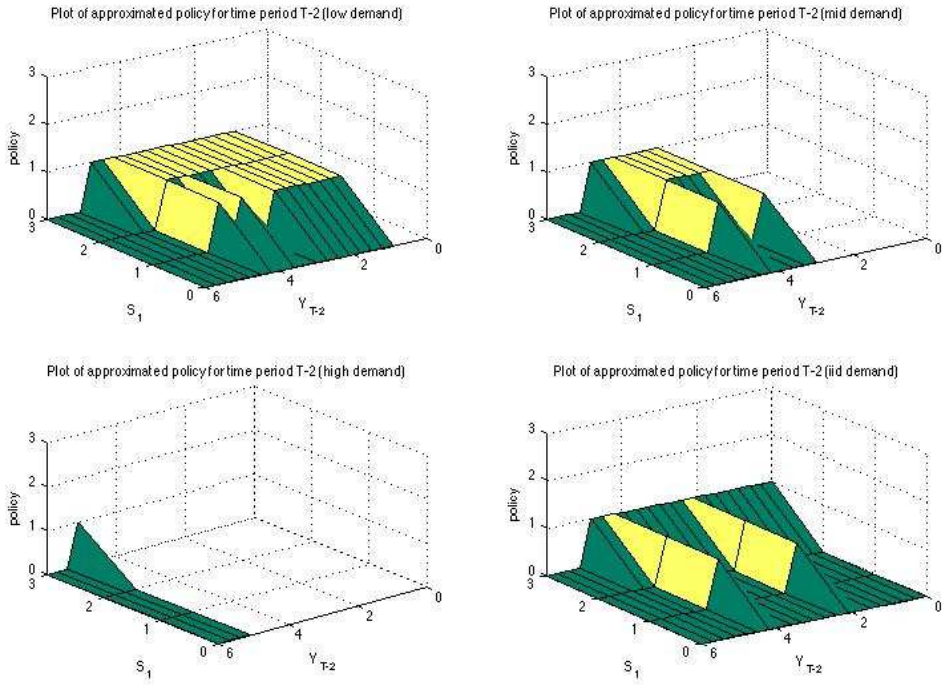

Figure 1.4: Near-optimal Policy at Period T-2

Chapter 2

Coflow Scheduling

In this chapter, we study the coflow scheduling problem which is NP-hard and present the first polynomial-time approximation algorithms for this problem.

2.1 Introduction

2.1.1 Background and Motivation

With the explosive growth of data-parallel computation frameworks such as MapReduce [16], Hadoop [1, 9, 43], Spark [51], Google Dataflow [2], etc., modern data centers are able to process large-scale data sets at an unprecedented speed. A key factor in materializing this efficiency is parallelism: many applications written in these frameworks alternate between computation and communication stages, where a typical computation stage produces many pieces of intermediate data for further processing, which are transferred between groups of servers across the network. Data transfer within a communication stage involves a large collection of parallel flows, and a

computation stage often cannot start until all flows within a preceding communication stage have finished [14, 17].

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

To faithfully capture application-level communication requirements, Chowdhury and Stoica [13] introduced the *coflow* abstraction, defined to be a collection of parallel flows with a common performance goal. Effective scheduling heuristics were proposed in [15] to optimize coflow completion times. In this chapter, we are interested in developing scheduling algorithms with provable performance guarantees, and our main contribution is a deterministic polynomial-time $64/3$-approximation algorithm and a randomized polynomial-time $(8+16\sqrt{2}/3)$-approximation algorithm, for the problem of minimizing the total weighted completion time of coflows with zero release times. These are the first $O(1)$-approximation algorithms for this problem.

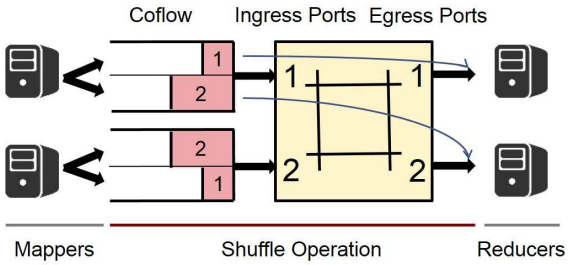

Figure 2.1: A MapReduce application in a 2×2 network

2.1.2 Preliminaries

System Model

The network

In order to describe the coflow scheduling problem, we first need to specify the conceptual model of the datacenter network. Similar to [15], we abstract out the network as one giant *non-blocking* switch [4, 7, 26, 38] with *m ingress ports* and *m egress ports* – which we call an $m \times m$ network switch – where *m* specifies the network size. Ingress ports represent physical or virtual links (e.g., Network Interface Cards) where data is transferred from servers to the network, and egress ports represent links through which servers receive data. We assume that the transfer of data within the network switch is *instantaneous*, so that any data that is transferred out of an ingress port is immediately available at the corresponding egress port. There are capacity constraints on the ingress and egress ports. For simplicity, we assume that all ports have unit capacity, i.e., one unit of data can be transferred through an ingress/egress port per unit time. See Figure 2.1 for an example of a 2×2 datacenter network switch. In the sequel, we sometimes use the terms *inputs* and *outputs* to mean ingress and egress ports respectively.

Coflows

A *coflow* is defined as a collection of parallel flows with a common performance goal. We assume that all flows within a coflow arrive to the system at the same time, the *release time* of the coflow. To illustrate, consider the shuffling stage of a MapReduce application with 2 mappers and 2 reducers that arrives to a 2×2 network at time 0, as shown in Figure 2.1. Both mappers need to transfer intermediate data to both reducers. Therefore, the shuffling stage consists of $2 \times 2 = 4$ parallel flows, each one corresponding to a pair of ingress and egress ports. For example, the size of the flow that needs to be transferred from input 1 to output 2 is 2. In general, we use an $m \times m$ matrix $D = \left(d_{ij}\right)_{i,j=1}^{m}$ to represent a coflow, in a $m \times m$ network. d_{ij} denotes the size of the flow to be transferred from input i to output j. We also assume that flows consist of discrete data units, so their sizes are integers.

Scheduling constraints

Since each input can transmit at most one data unit and each output can receive at most one data unit per time slot, a *feasible* schedule for a single time slot can be described by a *matching* between the inputs and outputs. When an input is matched to an output, a corresponding data unit (if available) is transferred across the network. To illustrate, consider again the coflow in Figure 2.1, given by the matrix $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$. When this coflow is the only one present, it can be completed in 3 time slots, using matching schedules described by $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Here entry 1 indicates a connection between the corresponding input and output, so for example, the first

matching connects input 1 with output 1, and input 2 with output 2. When multiple coflows are present, it is also possible for a matching to involve data units from different coflows.

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

Problem statement

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

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

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

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

$$\sum_{t=1}^{C_k} y(i,j,k,t) \geq d_{ij}^{(k)}, \text{ for } i,j = 1,\ldots,m, k = 1,\ldots,n; \tag{2.1.1}$$

$$\sum_{k=1}^{n} \sum_{j'=1}^{m} y(i,j',k,t) \leq 1, \text{ for } i = 1,\ldots,m, \ \forall t; \tag{2.1.2}$$

$$\sum_{k=1}^{n} \sum_{i'=1}^{m} y(i',j,k,t) \leq 1, \text{ for } j = 1,\ldots,m, \ \forall t; \tag{2.1.3}$$

$$y(i,j,k,t) = 0 \text{ if } t < r_k, \text{ for } i,j = 1,\ldots,m, \ \forall t,k; \tag{2.1.4}$$

$$y(i,j,k,t) \text{ binary}, \ \forall i,j,t,k. \tag{2.1.5}$$

The load constraints (2.1.1) state that all processing requirements of flows need to be met upon the completion of each coflow. (2.1.2) and (2.1.3) are the matching constraints. The release time constraints (2.1.4) guarantee that coflows are being served only after they are released in the system. Note that this mathematical program is not an integer linear programming formulation because variables C_k are in the limit of the summation.

The coflow scheduling problem (O) generalizes some well-known scheduling problems. First, when $m = 1$, it is easy to see that coflow scheduling is equivalent to single-machine scheduling with release times with the objective of minimizing the total weighted completion time, where preemption is allowed. The latter problem is strongly NP-hard [29], which immediately implies the NP-hardness of problem (O) with general release times.

The coflow scheduling problem (O) is also closely related with the concurrent open shop problem [3, 40], as observed in [15]. When all the coflows are given by *diagonal* matrices, coflow scheduling is equivalent to a concurrent open shop scheduling problem [15]. To see the equivalence, recall the problem setup of concurrent open shop. A set of n jobs are released into a system with m parallel machines at different times. For $k \in \{1, 2, \cdots, n\}$, job k is released at time r_k and requires $p_i^{(k)}$ units of processing time on machine i, for $i \in \{1, 2, \ldots, m\}$. The completion time of job k, which we denote by C_k, is the time that its processing requirements on all machines are completed. Let $z(i,k,t)$ be the indicator variable which equals 1 if job k is served on machine k at time t, and 0 otherwise. Let w_k be positive weight parameters, $k = 1, 2, \ldots, n$. Then, we can formulate a program (CO) for concurrent open shop scheduling.

$$
\begin{aligned}
\text{(CO)} \qquad \text{Minimize} \quad & \sum_{k=1}^{n} w_k C_k \\
\text{subject to} \quad & \sum_{t=1}^{C_k} z(i,k,t) \geq p_{ik}, \forall i,k; \\
& z(i,k,t) = 0 \text{ if } t < r_k, \forall i,k,t; \\
& z(i,k,t) \in \{0,1\}, \forall i,k,t.
\end{aligned}
$$

Now consider the following coflow scheduling problem. In an $m \times m$ network, each port has unit capacity, and for each $k \in \{1, \cdots, n\}$, $D^{(k)}$ is a diagonal matrix with diagonal entries given by $p_i^{(k)}$, $i \in \{1, \cdots, m\}$. Then, when $i \neq j$, we always have $y(i, j, k, t) = 0$. If we rewrite $y(i, i, k, t)$ as $z(i, k, t)$, program (O) is equivalent to (CO).

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

Lemma 1. *Problem (O) is NP-hard for* $m \geq 2$.

Although there are similarities between the concurrent open shop and the coflow scheduling problem, there are also key differences which make the coflow scheduling problem a more challenging one. First, coflow scheduling involves *coupled resources* – the inputs and outputs are coupled by the scheduling constraints. As a result, the coflow scheduling problem has also been called *concurrent open shop with coupled resources* in [15]. Second, for concurrent open shop, there exists an optimal schedule in which jobs can be processed in the same order on all machines, i.e., the schedule is a *permutation* schedule [3]. In contrast, permutation schedules need not be optimal for coflow scheduling [15]. LP-relaxations based on completion time variables, which have been used for concurrent open shop (see e.g., [32]) use permutation schedules in a crucial way and do not immediately extend to the coflow scheduling problem.

2.1.3 Main Results

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

Theorem 1. *When all the coflows are released into the system at time 0, there exists a deterministic polynomial time* $64/3$*-approximation algorithm, i.e.*

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

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

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

Our deterministic (described in Algorithm 2.2) and randomized algorithms combine ideas from combinatorial scheduling and matching theory along with some new insights. First, as with many other scheduling problems (see e.g. [21]), particularly for average completion time, we relax the problem formulation (O) to a polynomial-sized interval-indexed linear program (LP). The relaxation involves both dropping the matching constraints (2.1.2) and (2.1.3), and using intervals to make the LP polynomial sized. We then solve this LP, and use an optimal solution to the LP to obtain an ordered list of coflows. Then, we use this list to derive an actual schedule. To do so, we

partition coflows into a polynomial number of groups, based on the minimum required completion times of the ordered coflows, and schedule the coflows in the same group as a single coflow using matchings obtained from an integer version of the Birkhoff-von Neumann decomposition theorem (Lemma 5 and Algorithm 2.1).

The analysis of the algorithm couples techniques from the two areas in interesting ways. We analyze the interval-indexed linear program using tools similar to those used for other average completion time problems, especially concurrent open shop. The interval-indexed rather than time-indexed formulation is necessary to obtain a polynomial time algorithm, and we use a lower bound based on a priority-based load calculation (see Lemma 4). We also show how each coflow can be completed by a time bounded by a constant times the optimal completion time, via a clever grouping and decomposition of the coflow matrices. Here a challenge is to decompose the not necessarily polynomial length schedule into a polynomial number of matchings.

2.1.4 Related work

The coflow abstraction was first proposed in [13], although the idea was present in a previous paper [14]. Chowdhury et al. [14] observed that the optimal processing time of a coflow is exactly equal to its *load*, when the network is scheduling only a single coflow, and built upon this observation to schedule data transfer in a datacenter network. Chowdhury et al. [15] introduced the coflow scheduling problem with zero release times, and provided effective scheduling heuristics. They also observed the connection of coflow scheduling with concurrent open shop, established the NP-hardness of the coflow scheduling problem, and showed via a simple counter-example how permutation schedules need not be optimal for coflow scheduling.

There is a great deal of success over the past 20 years on combinatorial scheduling to minimize average completion time, see e.g. [21, 36, 37, 44]. This line of works typically uses a linear programming relaxation to obtain an ordering of jobs and then uses that ordering in some other polynomial-time algorithm. There has also been much work on shop scheduling, which we do not survey here, but note that traditional shop scheduling is not "concurrent". In the language of our problem, that would mean that traditionally, two flows in the same coflow could *not* be processed simultaneously. The recently studied concurrent open shop problem removes this restriction and models flows that can be processed in parallel. There have been several results showing that even restrictive special cases are NP-hard [3, 12, 20, 46]. There were several algorithms with super-constant (actually at least m, the number of machines) approximation ratios, e.g., [3, 30, 46, 48]. Recently there have been several constant factor approximation algorithms using LP-relaxations. Wang and Cheng [48] used an interval-indexed formulation. Several authors have observed that a relaxation in completion time variables is possible [12, 20, 30], and Mastrolilli et al. [32] gave a primal-dual 2-approximation algorithm and showed stronger hardness results. Relaxations in completion time variables presume the optimality of permutation schedules, which does not hold for the coflow scheduling problem. Thus, our work builds on the formulation in Wang and Cheng [48], even though their approach does not yield the strongest approximation ratio.

The design of our scheduling algorithms relies crucially on a fundamental result (Lemma 5 in this paper) concerning the decomposition of nonnegative integer-valued matrices into permutation matrices, which states that such a matrix can be written as a sum of ρ permutation matrices, where ρ is the maximum column and row sum. This result is closely related to the classical Birkhoff-von Neumann theorem [8], and has been stated in different forms and applied in different contexts. For

an application in scheduling theory, see e.g., [28]. For applications in communication networks, see e.g., [11, 31, 33, 42].

2.2 Linear Program (LP) Relaxation

In this section, we present an interval-indexed linear program relaxation of the scheduling problem (O), which produces a lower bound on $\sum_{k=1}^{n} w_k C_k(OPT)$, the optimal value of the total weighted completion time, as well as an ordering of coflows for our approximation algorithms (§2.2.1). We then define and analyze the concepts of *maximum total input/output loads*, which respect the ordering produced by the LP, and relate these concepts to $C_k(OPT)$ (§2.2.2). These relations will be used in the proofs of our main results in §2.3.3.

2.2.1 Two Linear Program Relaxations

From the discussion in §2.1.2, we know that problem (O) is NP-hard. Furthermore, the formulation in (2.1.1) - (2.1.5) is not immediately of use, since it is at least as hard as an integer linear program. We can, however, formulate an *interval-indexed linear program* (LP) by relaxing the following components from the original formulation. (i) First, we develop new load constraints (see (2.2.3) and (2.2.4)), by relaxing the matching constraints (2.1.2) and (2.1.3) and the load constraints (2.1.1), and formulate a *time-indexed linear program.* (The matching constraints will be enforced in the actual scheduling algorithm.) The time-indexed LP has been used many times (e.g. [5, 21]) but typically for non-shop scheduling problems. Note that in order to use it for our problem, we drop the explicit matching constraints. We call this (LP-EXP) below. (ii) Second,

in order to get a polynomial sized formulation, we divide time (which may not be polynomially bounded) into a set of geometrically increasing intervals. We call this an *interval-indexed integer program*, and it is also commonly used in combinatorial scheduling. In doing so, we have a weaker relaxation than the time-indexed one, but one that can be solved in polynomial time. We then relax the interval-indexed integer program to a linear program and solve the linear program.

To implement relaxation (i), let us examine the load and matching constraints (2.1.1) – (2.1.3). Constraints (2.1.2) and (2.1.3) imply that in each time slot t, each input/output can process at most one data unit. Thus, the total amount of work that can be processed by an input/output by time t is at most t. For each time slot t and each $k = 1, 2, \ldots, n$, let $z_t^{(k)} \in \{0, 1\}$ be an indicator variable of the event that coflow k completes in time slot t. Then

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

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

$$\sum_{s=1}^{t} \sum_{k=1}^{n} \sum_{j'=1}^{m} d_{ij'}^{(k)} z_s^{(k)} \leq t, \quad \text{for all } i = 1, 2, \ldots, m, \tag{2.2.1}$$

$$\sum_{s=1}^{t} \sum_{k=1}^{n} \sum_{i'=1}^{m} d_{i'j}^{(k)} z_s^{(k)} \leq t, \quad \text{for all } j = 1, 2, \ldots, m, \tag{2.2.2}$$

which are the load constraints on the inputs and outputs.

To complete relaxation (i), we require an upper bound on the time needed to complete all coflows in an optimal scheduling algorithm. To this end, note that the naive algorithm which sched-

ules one data unit in each time slot can complete processing all the coflows in $T = \max_k\{r_k\} + \sum_{k=1}^{n}\sum_{i,j=1}^{m} d_{ij}^{(k)}$ units of time, and it is clear that an optimal scheduling algorithm can finish processing all coflows by time T. Taking into account constraints (2.2.1) and (2.2.2), and relaxing the integer constraints $z_t^{(k)} \in \{0,1\}$ into the corresponding linear constraints, we can formulate the following linear programming relaxation (LP-EXP) of the coflow scheduling problem (O).

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

$$\sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{j'=1}^{m} d_{ij'}^{(k)} z_s^{(k)} \le t, \text{ for } i = 1,\ldots,m,\ t = 1,\ldots,T; \tag{2.2.3}$$

$$\sum_{s=1}^{t}\sum_{k=1}^{n}\sum_{i'=1}^{m} d_{i'j}^{(k)} z_s^{(k)} \le t, \text{ for } j = 1,\ldots,m,\ t = 1,\ldots,T; \tag{2.2.4}$$

$$z_t^{(k)} = 0 \text{ if } r_k + \sum_{j'=1}^{m} d_{ij'}^{(k)} > t \text{ or } r_k + \sum_{i'=1}^{m} d_{i'j}^{(k)} > t; \tag{2.2.5}$$

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

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

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

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

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

$$\sum_{u=1}^{l} \sum_{k=1}^{n} \sum_{j'=1}^{m} d_{ij'}^{(k)} x_u^{(k)} \leq \tau_l, \text{ for } i = 1, \ldots, m, \; l = 1, \ldots, L; \tag{2.2.6}$$

$$\sum_{u=1}^{l} \sum_{k=1}^{n} \sum_{i'=1}^{m} d_{i'j}^{(k)} x_u^{(k)} \leq \tau_l, \text{ for } j = 1, \ldots, m, \; l = 1, \ldots, L; \tag{2.2.7}$$

$$x_l^{(k)} = 0 \text{ if } r_k + \sum_{j'=1}^{m} d_{ij'}^{(k)} > \tau_l \text{ or } r_k + \sum_{i'=1}^{m} d_{i'j}^{(k)} > \tau_l; \tag{2.2.8}$$

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

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

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

Lemma 2. *The optimal value of the linear program (LP) is a lower bound on the optimal total weighted completion time $\sum_{k=1}^{n} w_k C_k(OPT)$ of coflow scheduling problem (O).*

Proof. Consider an optimal schedule of problem (O) and set $x_u^{(k)} = 1$ if coflow k completes within the uth time interval. This is a feasible solution to problem (LP) with the load and capacity constraints (2.2.6) and (2.2.7) and the feasibility constraint (2.2.8) all satisfied. Moreover, since coflow

k completes within the uth interval, the coflow completion time is at least τ_{u-1}. Hence, the objective value of the feasible solution constructed is no more than the optimal total weighted completion time $\sum_{k=1}^{n} w_k C_k(OPT)$. □

Since the constraint matrix in problem (LP) is of size $O((n+m)\log T)$ by $O(n\log T)$ and the maximum size of the coefficients is $O(\log T)$, the number of bits of input to the problem is $O(n(m+n)(\log T)^3)$. The interior point method can solve problem (LP) in polynomial time [27].

From an optimal solution to (LP), we can obtain an ordering of coflows, and use this order in our scheduling algorithms (see e.g., Algorithm 2.2). To do so, let an optimal solution to problem (LP) be $\bar{x}_l^{(k)}$ for $k = 1,\ldots,n$ and $l = 1,\ldots,L$. The relaxed problem (LP) computes an approximated completion time

$$\bar{C}_k = \sum_{l=1}^{L} \tau_{l-1} \bar{x}_l^{(k)} \tag{2.2.9}$$

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

$$\bar{C}_1 \leq \bar{C}_2 \leq \ldots \leq \bar{C}_n. \tag{2.2.10}$$

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

Wang and Cheng [48] gave a $16/3$-approximation algorithm for the concurrent open shop problem using a similar interval-indexed linear program. Our algorithms are more involved because we also have to address the matching constraints, which do not appear in the concurrent open shop problem.

2.2.2 Maximum Total Input / Output Loads

Here we define the *maximum total input/output loads,* respecting the ordering and indexing of (3.4.2). For each k, $k = 1,2,\ldots n$, define the *maximum total input load* I_k, the *maximum total output load* J_k and the *maximum total load* V_k by

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

$$\text{and} \quad V_k = \max\{I_k, J_k\} \tag{2.2.11}$$

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

Lemma 3. *For $k = 1,2,\ldots,n$, let $\tilde{C}^{(k)}$ be the time that all coflows $1,\ldots,k$ complete, where the indexing respects the order* (3.4.2)*. Then, under any scheduling algorithm,*

$$V_k \leq \tilde{C}^{(k)} \tag{2.2.12}$$

for all k simultaneously.

The following lemma, which states that with a proper ordering of the coflows, V_k is a $16/3$-

approximation of the optimal $C_k(OPT)$ for all k simultaneously, is crucial for the proof of our main results in the next section.

Lemma 4. *Let $\bar{C}_k$ be computed from problem (LP) by Eq. (2.2.9) and be indexed such that (3.4.2) is satisfied. Then $V_k \leq (16/3)C_k(OPT),\ \ k = 1,\ldots,n$.*

Proof. Proof of Lemma 4 The idea of the proof follows Wang and Cheng [48]. Suppose that $\tau_{u-1} < \bar{C}_k \leq \tau_u$ for some u. We consider the following three cases.

Case (1) $\bar{C}_k < \frac{5\tau_{u-1}}{4}$.

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

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

Therefore,

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

Let $g^* = \text{argmin}_{1\leq g\leq k} \sum_{l=1}^{u} \bar{x}_l^{(g)}$. We know from Eq. (2.2.11) that

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

We then have

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

Using the constraints (2.2.6) and (2.2.7), we have

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

Case (2) $\frac{5\tau_{u-1}}{4} \leq \bar{C}_k < \frac{3\tau_{u-1}}{2}$.

Define a sequence $\alpha_h, h = 1, 2, \ldots$, by

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

Note that a simple induction shows that for all h, $\alpha_h > 0$, and $\sum_{q=1}^{h} \alpha_q < 1/2$. Furthermore,

$\sum_{q=1}^{h} \alpha_q \to 1/2$ as $h \to \infty$. Thus, there exists some h such that

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

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

$$\begin{aligned}\left(1+\sum_{q=1}^{h}\alpha_q\right)\tau_{u-1} \geq \bar{C_k} \geq \bar{C_g} &= \sum_{l=1}^{L}\tau_{l-1}\bar{x_l^{(g)}} \\ &> \tau_u \sum_{l=u+1}^{L}\bar{x_l^{(g)}} = \tau_u\left(1-\sum_{l=1}^{u}\bar{x_l^{(g)}}\right),\end{aligned}$$

and

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

It follows that,

$$\begin{aligned}V_k &< \frac{\tau_u}{\sum_{l=1}^{u}\bar{x_l^{(g^*)}}} < \frac{2\tau_u}{1-\sum_{q=1}^{h}\alpha_q} = \frac{4\tau_{u-1}}{1-\sum_{q=1}^{h}\alpha_q} \\ &< \frac{4\bar{C_k}}{(1-\sum_{q=1}^{h}\alpha_q)(1+\sum_{q=1}^{h-1}\alpha_q)}.\end{aligned}$$

By the definition of α_q, we have

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

Therefore,

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

Case (3) $\bar{C_k} \geq \frac{3\tau_{u-1}}{2}$.

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

$$\begin{aligned}\tau_u > \bar{C_k} \geq \bar{C_g} &= \sum_{l=1}^{L} \tau_{l-1}\bar{x_l^{(g)}} > \tau_{u+1} \sum_{l=u+2}^{L} \bar{x_l^{(g)}} \\ &= \tau_{u+1}\left(1 - \sum_{l=1}^{u+1} \bar{x_l^{(g)}}\right) = 2\tau_u\left(1 - \sum_{l=1}^{u+1} \bar{x_l^{(g)}}\right),\end{aligned}$$

and hence

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

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

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

We know from Lemma 2 that $\bar{C_k} \leq C_k(OPT)$ and the result follows.

2.3 Approximation Algorithms

In this section, we describe a deterministic and a randomized polynomial time scheduling algorithm, with approximation ratios of $64/3$ and $8 + 16\sqrt{2}/3$ respectively, when all coflows are released at time 0. Both algorithms are based on the idea of efficiently scheduling coflows according to the ordering (3.4.2) produced by (LP). To fully describe the scheduling algorithms, we first

present some preliminaries on the minimal amount of time to finish processing an arbitrary coflow using only matching schedules (Algorithm 2.1 and §2.3.1). Algorithm 2.1 will be used crucially in the design of the approximation algorithms, and, as we will see, it is effectively an integer version of the famous Birkhoff-von Neumann theorem [8], and hence the name *Birkhoff-von Neumann decomposition*. Details of the main algorithms are provided in §2.3.2, and we supply proofs of the complexities and approximation ratios in §2.3.3.

2.3.1 Birkhoff-von Neumann Decomposition

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

$$\rho(D) = \max\left\{ \max_{i=1,\ldots,m}\left\{ \sum_{j'=1}^{m} d_{ij'} \right\}, \max_{j=1,\ldots,m}\left\{ \sum_{i'=1}^{m} d_{i'j} \right\} \right\}. \tag{2.3.1}$$

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

Lemma 5. *There exists a polynomial time algorithm which finishes processing coflow D in $\rho(D)$ time slots (using the matching schedules), were it to be scheduled alone.*

Algorithm 2.1 describes a polynomial-time scheduling algorithm that can be used to prove Lemma 5. The idea of the algorithm is as follows. For a given coflow matrix $D = (d_{ij})_{i,j=1}^{m}$, we first augment it to a "larger" matrix $\tilde{D}$, whose row and column sums are all equal to $\rho(D)$ (Step 1). In each iteration of Step 1, we increase one entry of D such that at least one more row or column

sums to ρ. Therefore, at most $2m-1$ iterations are required to get $\tilde{D}$. We then decompose $\tilde{D}$ into *permutation matrices* that correspond to the matching schedules (Step 2). More specifically, at the end of Step 2, we can write $\tilde{D} = \sum_{u=1}^{U} q_u \Pi_u$, so that Π_u are permutation matrices, $q_u \in \mathbb{N}$ are such that $\sum_{u=1}^{U} q_u = \rho(D)$, and $U \leq m^2$. The key to this decomposition is Step 2 (ii), where the existence of a perfect matching M can be proved by a simple application of Hall's matching theorem [22]. We now consider the complexity of Step 2. In each iteration of Step 2, we can set at least one entry of $\tilde{D}$ to zero, so that the algorithm ends in m^2 iterations. The complexity of finding a perfect matching in Step 2 (ii) is $O(m^3)$, e.g., by using the maximum bipartite matching algorithm. Since both Steps 1 and 2 of Algorithm 2.1 has polynomial-time complexity, the algorithm itself has polynomial-time complexity.

If we divide both sides of the identity $\tilde{D} = \sum_{u=1}^{U} q_u \Pi_u$ by $\rho(D)$, then $\tilde{D}/\rho(D)$ is a doubly stochastic matrix, and the coefficients $q_u/\rho(D)$ sum up to 1. Therefore, $\tilde{D}/\rho(D)$ is a convex combination of the permutation matrices Π_u. Because of this natural connection of Algorithm 2.1 with the Birkhoff-von Neumann theorem [8], we call the algorithm the Birkhoff-von Neumann decomposition. Lemma 5 has been stated in slightly different forms, see e.g., Theorem 1 in [28], Theorem 4.3 in [42], and Fact 2 in [33].

2.3.2 Approximation Algorithms

Here we present our deterministic and randomized scheduling algorithms. The deterministic algorithm is summarized in Algorithm 2.2, which consists of 2 steps. In Step 1, we solve (LP) to get the approximated completion time $\bar{C}_k$ for coflow ordering. Then, in Step 2, for each $k = 1, 2, \ldots, n$, we compute the maximum total load V_k of coflow k, and identify the time interval $(\tau_{r(k)-1}, \tau_{r(k)}]$ that it

Algorithm 2.1 Birkhoff-von Neumann Decomposition

Data: A single coflow $D = \left(d_{ij}\right)_{i,j=1}^{m}$.
Result: A scheduling algorithm that uses at most a polynomial number of different matchings.
Step 1: Augment D to a matrix $\tilde{D} = \left(\tilde{d}_{ij}\right)_{i,j=1}^{m}$, where $\tilde{d}_{ij} \geq d_{ij}$ for all i and j, and all row and column sums of $\tilde{D}$ are equal to $\rho(D)$.
Let $\eta = \min\left\{\min_i\left\{\sum_{j'=1}^{m} d_{ij'}\right\}, \min_j\left\{\sum_{i'=1}^{m} d_{i'j}\right\}\right\}$ be the minimum of row sums and column sums, and let $\rho(D)$ be defined according to Equation (2.4.2).
$\tilde{D} \leftarrow D$.
while $(\eta < \rho)$ **do**
  $i^* \leftarrow \text{argmin}_i \sum_{j'=1}^{m} \tilde{D}_{ij'}$; $j^* \leftarrow \text{argmin}_j \sum_{i'=1}^{m} \tilde{D}_{i'j}$.

$$\tilde{D} \leftarrow \tilde{D} + pE,$$

  where $p = \min\{\rho - \sum_{j'=1}^{m} \tilde{D}_{i^*j'}, \rho - \sum_{i'=1}^{m} \tilde{D}_{i'j^*}\}$, $E_{ij} = 1$ if $i = i^*$ and $j = j^*$, and $E_{ij} = 0$ otherwise.
  $\eta \leftarrow \min\left\{\min_i\left\{\sum_{j'=1}^{m} \tilde{D}_{ij'}\right\}, \min_j\left\{\sum_{i'=1}^{m} \tilde{D}_{i'j}\right\}\right\}$
Step 2: Decompose $\tilde{D}$ into permutation matrices Π.
while $(\tilde{D} \neq 0)$ **do**
  (i) Define an $m \times m$ binary matrix G where $G_{ij} = 1$ if $\tilde{D}_{ij} > 0$, and $G_{ij} = 0$ otherwise, for $i, j = 1, \ldots, m$.
  (ii) Interpret G as a bipartite graph, where an (undirected) edge (i, j) is present if and only if $G_{ij} = 1$. Find a perfect matching M on G and define an $m \times m$ binary matrix Π for the matching by $\Pi_{ij} = 1$ if $(i, j) \in M$, and $\Pi_{ij} = 0$ otherwise, for $i, j = 1, \ldots, m$.
  (iii) $\tilde{D} \leftarrow \tilde{D} - q\Pi$, where $q = \min\{\tilde{D}_{ij} : \Pi_{ij} > 0\}$. Process coflow D using the matching M for q time slots. More specifically, process dataflow from input i to output j for q time slots, if $(i, j) \in M$ and there is processing requirement remaining, for $i, j = 1, \ldots, m$.

belongs to, where τ_l are defined in §2.2.1. All the coflows that fall into the same time interval are combined into and treated as a single coflow, and processed using Algorithm 2.1 in §2.3.1.

The randomized scheduling algorithm follows the same steps as the deterministic one, except for the choice of the time intervals in Step 2 of Algorithm 2.2. Define the *random* time points τ'_l by $\tau'_0 = 0$, and $\tau'_l = T_0 a^{l-1}$, where $a = 1 + \sqrt{2}$, and $T_0 \sim Unif[1, a]$ is uniformly distributed between 1 and a. Having picked the random time points τ'_l, we then proceed to process the coflows in a similar fashion to the deterministic algorithm. Namely, for each $k = 1, 2, \ldots, n$, we identify the

Algorithm 2.2 Deterministic LP-based Approximation

Data: Coflows $\left(d_{ij}^{(k)}\right)_{i,j=1}^{m}$, for $k = 1,\ldots,n$.
Result: A scheduling algorithm that uses at most a polynomial number of different matchings.
Step 1: Given n coflows, solve the linear program (LP). Let an optimal solution be given by $\bar{x}_l^{(k)}$, for $l = 1,2,\ldots,L$ and $k = 1,2,\ldots,n$. Compute the approximated completion time C_k by Eq. (2.2.9). Order and index the coflows according to (3.4.2).
Step 2: Compute the maximum total load V_k for each k by (2.2.11). Suppose that $V_k \in (\tau_{r(k)-1}, \tau_{r(k)}]$ for some function $r(\cdot)$ of k. Let the range of function $r(\cdot)$ consist of values $s_1 < s_2 < \ldots < s_P$, and define the sets $S_u = \{k : \tau_{s_u-1} < V_k \leq \tau_{s_u}\}$, $u = 1,2,\ldots,P$.
$u \leftarrow 1$.
while $u \leq P$ **do**
After all the coflows in set S_u are released, schedule them as a single coflow with transfer requirement $\sum_{k\in S_u} d_{ij}^{(k)}$ from input i to output j and finish processing the coflow using Algorithm 2.1.
$u \leftarrow u+1$;

(random) time interval $(\tau'_{r'(k)-1}, \tau'_{r'(k)}]$ that coflow k belongs to, and for all coflows that belong to the same time interval, they are combined into a single coflow and processed using Algorithm 2.1.

2.3.3 Proofs of Main Results

We now establish the complexity and performance properties of our algorithms. We first provide the proof of Theorem 1 in detail. We then establish Theorem 2.

Proofs of Theorem 1 Let $C_k(A)$ be the completion time of coflow k under the deterministic scheduling algorithm (Algorithm 2.2). The following proposition will be used to prove Theorem 1.

Proposition 1. *For all $k = 1,2,\ldots,n$, when $r_k = 0$, the coflow completion time $C_k(A)$ satisfies*

$$C_k(A) \leq 4V_k, \tag{2.3.2}$$

where we recall that r_k is the release time of coflow k, and the total loads V_k are defined in Eq. (2.2.11).

Proof. Recall the notation used in Algorithm 2.2. For any coflow $k \in S_u$, $V_k \leq \tau_{s_u}$. By Lemma 5, we know that all coflows in the set S_u can be finished processing within τ_{s_u} units of time. Define $\overline{\tau}_0 = 0$ and $\overline{\tau}_u = \overline{\tau}_{u-1} + \tau_{s_u}$, $u = 1, 2, \ldots, P$. A simple induction argument shows that under Algorithm 2.2, $C_k(A)$, the completion time of coflow k, satisfies $C_k(A) \leq \overline{\tau}_u$, if $k \in S_u$.

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

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

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

$$C_k(A) \leq 2\tau_{r(k)} \leq 4V_k.$$

□

The proof of Theorem 1 is now a simple consequence of Lemmas 2 and 4 and Proposition 1.

Proof of Theorem 1. By Proposition 1 and Lemma 4, we have

$$C_k(A) \leq 4V_k \leq \frac{64}{3} C_k(OPT).$$

It follows that

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

where the second inequality follows from Lemma 2.

We now consider the running time of Algorithm 2.2. The program (LP) in Step 1 can be solved in polynomial time, as discussed in §2.2.1. Thus, it suffices to show that Step 2 runs in polynomial time. Since there are only $O(\log T)$, a polynomial number of intervals of the form $(\tau_{l-1}, \tau_l]$, it now suffices to show that for each $u = 1, 2, \ldots, P$, Algorithm 2.1 completes processing all coflows in the set S_u in polynomial time, where we recall the definition of S_u in Step 2 of Algorithm 2.2. But this follows from Lemma 5. Thus, Algorithm 2.2 runs in polynomial time. □

Before proceeding to the proofs of Theorem 2, let us provide some remarks on the upper bounds (2.3.2) in Proposition 1. Recall that Ineq. (2.3.2) hold simultaneously for all k. Then, a natural question is whether the upper bounds in (2.3.2) are tight. We leave this question as future work, but provide the following observation for now. By inspecting the proof of Proposition 1, it is easy to see that in fact, $\tilde{C}^{(k)}(A) \leq 4V_k$ for all k, where $\tilde{C}^{(k)}(A)$ is the completion time of coflows $1, 2, \ldots, k$ under our algorithm. Compared this with the lower bounds (2.2.12) in Lemma 3, we see that the upper bounds are off by a factor of at most 4. The lower bounds (2.2.12) cannot be achieved simultaneously for all k; this fact can be demonstrated through a simple counter-example. The counter-example involves 2 coflows $D^{(1)}$ and $D^{(2)}$ in a network with 3 inputs and 3 outputs

(each having unit capacity). Suppose that $D^{(1)}$ and $D^{(2)}$ are given by

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

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

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

Proposition 2. *Let $C_k(A')$ be the* random *completion time of coflow k under the randomized*

scheduling algorithm described in §2.3.2. Then, for all $k = 1, 2, \ldots, n$,

$$\mathbb{E}[C_k(A')] \leq \left(\frac{3}{2} + \sqrt{2}\right) V_k. \tag{2.3.3}$$

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

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

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

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

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

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

Similar to the proof of Proposition 1, we can establish that if $k \in S_u$, then $C_k(A') \leq \overline{\tau}'_u$. Thus, with

$a = 1+\sqrt{2}$,

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

□

Proof of Theorem 2. By Proposition 2 and Lemma 4, we have $\mathbb{E}[C_k(A')] \le \left(3/2+\sqrt{2}\right)V_k < \left(8+16\sqrt{2}/3\right)C_k(OPT)$. It follows that

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

where the second inequality follows from Lemma 2. □

2.4 General Network and Capacities

In this section, we generalize our approximation algorithms to handle the case where the size of the inputs and outputs are different and the numbers m^I and m^O capacity constraints on the inputs and outputs are not unit. We consider a general $m^I \times m^O$ network, where numbers m^I and m^O specify the input/output size. For $i \in \{1,2,\cdots,m^I\}$, let λ_i be a positive integer that specifies the capacity on input i. Similarly, for $j \in \{1,2,\cdots,m^O\}$, let μ_j be a positive integer that specifies the capacity on output j. We also write $\lambda = (\lambda_1,\cdots,\lambda_{m^I})$ and $\mu = (\mu_1,\cdots,\mu_{m^O})$ for the capacity vectors. With

general capacities, problem (LP-EXP) now becomes

$$
\begin{aligned}
\text{(LP-EXP-G)}\quad \text{Minimize } & \sum_{k=1}^{n} w_k \sum_{t=1}^{T} t z_t^{(k)} \\
\text{subject to } & \sum_{s=1}^{t} \sum_{k=1}^{n} \sum_{j'=1}^{m^O} d_{ij'}^{(k)} z_s^{(k)} \le \lambda_i t, \forall i, t; \\
& \sum_{s=1}^{t} \sum_{k=1}^{n} \sum_{i'=1}^{m^I} d_{i'j}^{(k)} z_s^{(k)} \le \mu_i t, \forall j, t; \\
& z_t^{(k)} = 0 \text{ if } r_k + \sum_{j'=1}^{m^O} \frac{d_{ij'}^{(k)}}{\lambda_i} > t \\
& \qquad \text{or } r_k + \sum_{i'=1}^{m^I} \frac{d_{i'j}^{(k)}}{\mu_j} > t; \\
& \sum_{t=1}^{T} z_t^{(k)} = 1, \forall k; \quad z_t^{(k)} \ge 0, \forall k, t.
\end{aligned}
$$

Here the index variable i ranges over the set $\{1,2,\cdots,m^I\}$, j ranges over $\{1,2,\cdots,m^O\}$, and k ranges over $\{1,2,\cdots,n\}$.

For each $k \in \{1,2,\cdots n\}$, V_k is the sum of the loads of the lower numbered coflows normalized by port capacity and is defined as follows:

$$
\begin{aligned}
I_k &= \max_{i=1,\ldots,m^I} \left\{ \sum_{j'=1}^{m^O} \sum_{g=1}^{k} \frac{d_{ij'}^{(g)}}{\lambda_i} \right\}, \\
J_k &= \max_{j=1,\ldots,m^O} \left\{ \sum_{i'=1}^{m^I} \sum_{g=1}^{k} \frac{d_{i'j}^{(g)}}{\mu_j} \right\}, \\
\text{and} \quad V_k &= \max\{I_k, J_k\}, \qquad (2.4.1)
\end{aligned}
$$

2.4.1 Coflow Matrix Decomposition

We now prove a generalized version of Lemma 5 with general sizes of the network and general capacities on inputs/outputs.

The following notation will be used in the sequel. We denote the ith row sum of a $m^r \times m^c$ matrix A by[1]

$$S_i^r(A) = \sum_{j'=1}^{m^c} A_{ij'}, \text{ for } i \in \{1, \cdots, m^r\},$$

and the jth column sum by

$$S_j^c(A) = \sum_{i'=1}^{m^r} A_{i'j}, \text{ for } j \in \{1, \cdots, m^c\}.$$

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

$$\rho(D) = \max \left\{ \max_{1 \le i \le m^I} \frac{S_i^r(D)}{\lambda_i}, \max_{1 \le j \le m^O} \frac{S_j^c(D)}{\mu_j} \right\}. \tag{2.4.2}$$

Note that for each i, $S_i^r(D)$ is the total processing requirement of coflow D on input i, and for each j, $S_j^c(D)$ is that on output j. By the capacity constraints, $\rho(D)$ is a universal lower bound on the completion time of coflow D, were it to be scheduled alone. Since $\rho(D)$ need not be an integer, and times are slotted, $\lceil \rho(D) \rceil$ is also a (slightly tigher) universal lower bound, where $\lceil x \rceil$ is the smallest integer that is no less than x, for any $x \in \mathbb{R}$. Theorem 3 states that $\lceil \rho(D) \rceil$ is tight.

[1]We introduce the additional notation m^r and m^c, because we will often manipulate matrices that need not be of size $m^I \times m^O$ in subsequent parts of the paper.

Theorem 3. *There exists a polynomial time algorithm that finishes processing coflow D in $\bar{\rho}(D) = \lceil \rho(D) \rceil$ time slots (using only feasible schedules from $\mathcal{S}$), were it to be scheduled alone.*

Algorithm 2.3 describes a polynomial-time scheduling algorithm that is used to prove Theorem 3. The idea of the algorithm is as follows. For any $m^I \times m^O$ coflow matrix $D = (d_{ij})$, we first augment it to a $\left(m^I + m^O\right) \times \left(m^I + m^O\right)$ matrix $\tilde{D}$, whose row and column sums are equal to $\bar{\rho}(D)$ times the corresponding capacities. We then decompose $\tilde{D}$ into a sum of *integer-valued matrices* that correspond naturally to some feasible schedules. More specifically, at the end of the algorithm, we can write $\tilde{D} = \sum_{p=1}^{P} q_p \Pi_p$, so that Π_p are integer-valued matrices, $q_p \in \mathbb{N}$ are such that $\sum_{p=1}^{P} q_p = \bar{\rho}(D)$, and P is polynomial in the input size.

To prove Theorem 3, we will use extensively properties of totally unimodular matrices [19, 23, 24, 41, 52].

Definition 1. *A matrix A is totally unimodular if the determinant of every square submatrix of A has value -1, 0 or 1.*

A well-known fact is that the incidence matrix of a bipartite graph is totally unimodular [23], which we collect in Proposition 3 for easy reference. For completenes, recall that the $|V| \times |E|$ incidence matrix A of a simple, undirected graph $G = (V, E)$ with vertex set V and edge set E is defined to be

$$A_{v,e} = \begin{cases} 1 & \text{if } v \in e; \\ 0 & \text{if } v \notin e. \end{cases}$$

Proposition 3. *(Heller & Tompkins (1956)) Let $G = (V, E)$ be a bipartite graph, and let A be the unoriented incidence matrix. Then, A is totally unimodular.*

Algorithm 2.3 Generalized Birkhoff-von Neumann decomposition

Data: A single coflow $D = \left(d_{ij} : i = 1,\ldots,m^I, j = 1,\ldots,m^O\right)$.
Result: A scheduling algorithm that uses at most a polynomial number of distinct schedules.
Augment D to an $(m^I + m^O) \times (m^I + m^O)$ matrix $\tilde{D}$,

$$\tilde{D} = \begin{pmatrix} D & diag\left\{\bar{\rho}(D)\lambda_i - S_i^r(D)\right\} \\ diag\left\{\bar{\rho}(D)\mu_j - S_j^c(D)\right\} & D^T \end{pmatrix};$$

$p \leftarrow 1$;
while $\rho(\tilde{D}) > 1$ **do**

Find an extreme point x^p of a polytope $\mathcal{P}$ in $\mathbb{R}_+^{(m^I+m^O)^2}$ defined as follows:

$$\begin{aligned} \mathcal{P} = \Big\{ & x : x \in \mathbb{R}_+^{(m^I+m^O)^2}, \text{ such that} \\ & X \leq \tilde{D}, \quad X \in \mathbb{R}_+^{(m^I+m^O)\times(m^I+m^O)}, \\ & S_k^r(X) = b_k^r, \quad k \in \{1,\ldots,m^I+m^O\}, \\ & S_l^c(X) = b_l^c, \quad l \in \{1,\ldots,m^I+m^O\}, \\ & x = vec(X)^T \Big\}. \end{aligned}$$

Construct an $(m^I + m^O) \times (m^I + m^O)$ matrix $\tilde{\Pi}^p$ from x^p such that $x^p = vec(\tilde{\Pi}^p)^T$, $k,l \in \{1,\cdots,m^I+m^O\}$;
$q_p \leftarrow \left\lfloor \min_{i,j}\left\{\tilde{D}_{kl}/\tilde{\Pi}_{kl}^p; \tilde{\Pi}_{kl}^p > 0\right\}\right\rfloor$;
Let Π^p be the upper $m^O \times m^I$ submatrices of $\tilde{\Pi}^p$. Transfer Π_{ij}^p data units from input i to output j for q_p time slots;
$\tilde{D} \leftarrow \tilde{D} - q_p\tilde{\Pi}^p$;
$p \leftarrow p+1$;

We also need the following classic result on totally unimodular matrices [24].

Proposition 4. *(Hoffman & Kruskal (1956)) Let A be a totally unimodular matrix. Consider the linear program given by* $\max c^T x : Ax = b,\ 0 \leq x \leq u$, *where all components of b are integers. Then, every vertex solution of the LP has all components being integers.*

We are now ready to present the following result on the decomposition of an arbitrary coflow matrix, from which Theorem 3 follows immediately.

Proposition 5. *Let D be an arbitrary $m^I \times m^O$ coflow matrix. Recall the capacity vectors $\lambda \in \mathbb{Z}_+^{m^I}$ and $\mu \in \mathbb{Z}_+^{m^O}$, the load $\rho(D)$ defined in* (2.4.2)*, and that $\bar{\rho}(D) = \lceil \rho(D) \rceil$. Then, we can write D as a linear combination of a polynomial number of feasible schedules, so that*

$$D = \sum_{p=1}^{P} q_p \Pi^p, \quad \bar{\rho}(D) = \sum_{p=1}^{P} q_p, \tag{2.4.3}$$

where P is polynomial in input size, and for every $p \in \{1, \cdots, P\}$, $q_p \in \mathbb{Z}_+$. Furthermore, for each p, $\Pi^p \in \mathcal{S}$ is a feasible schedule, i.e., it is a nonnegative $m^I \times m^O$ matrix with integer entries, and satisfies $S_i^r(\Pi^p) \leq \lambda_i$ for $i \in \{1, \cdots, m^I\}$ and $S_j^c(\Pi^p) \leq \mu_j$ for $j \in \{1, \cdots, m^O\}$.

Proof. If $\bar{\rho}(D) = 1$, then D itself is a feasible schedule, so we are done. When $\bar{\rho}(D) > 1$, we augment matrix D to an $(m^I + m^O) \times (m^I + m^O)$ integer-valued matrix $\tilde{D}$ defined as follows.

$$\tilde{D} = \begin{pmatrix} D & diag\left(\bar{\rho}(D)\lambda_i - S_i^r(D)\right) \\ diag\left(\bar{\rho}(D)\mu_j - S_j^c(D)\right) & D^T \end{pmatrix},$$

where for an m-dimensional vector v, $diag(v)$ is the $m \times m$ diagonal matrix with diagonal entries given by v. Let $b^r = (\lambda, \mu) = (\lambda_1, \cdots, \lambda_{m^I}, \mu_1, \cdots, \mu_{m^O})$ and $b^c = (\mu, \lambda)$. Then, $\tilde{D}$ has row sums given by $\bar{\rho}(D)b^r$ and column sums given by $\bar{\rho}(D)b^c$, and $\rho(\tilde{D}) = \bar{\rho}(D)$.

Consider the following polytope in $\mathbb{R}_+^{\left(m^I+m^O\right)^2}$:

$$\begin{aligned}\mathcal{P} = \mathcal{P}(\tilde{D}) = \Big\{ & x \in \mathbb{R}_+^{\left(m^I+m^O\right)^2} : X \text{ is a nonnegative} \\ & \left(m^I+m^O\right) \times \left(m^I+m^O\right) \text{ matrix}, \\ & X \leq \tilde{D} \text{ component-wise}, \\ & S_k^r(X) = b_k^r, \quad k \in \{1,\cdots,m^I+m^O\}, \\ & S_l^c(X) = b_l^c, \quad l \in \{1,\cdots,m^I+m^O\}, \\ & \text{and } x = vec(X)' \Big\}.\end{aligned}$$

Here, $vec(X)$ is the vectorization of matrix X, where $vec(\cdot)$ is a linear transformation that converts the matrix into a column vector, and for a vector v, v' is its transpose. We can view $x_{l\left(m^I+m^O-1\right)+k}$ as the load variable for arc (k,l) in a complete $\left(m^I+m^O\right) \times \left(m^I+m^O\right)$ bipartite graph, and write the polytope $\mathcal{P}$ in the form $\left\{ x \in \mathbb{R}_+^{\left(m^I+m^O\right)^2} : Ax = b, 0 \leq x \leq u \right\}$, where A is the incidence matrix of the graph, $b = (b^r, b^c)'$ and $u = vec(\tilde{D})'$. It follows from Proposition 4 that all vertex solutions of the linear program $\max 0 : Ax = b, 0 \leq x \leq u$ are integral, if any exists. Since $S_k^r(\tilde{D}/\rho(\tilde{D})) = b_k^r$ and $S_l^c(\tilde{D}/\rho(\tilde{D})) = b_l^c$, the LP is always feasible. Therefore, the polytope $\mathcal{P}$ has integral vertices.

Pick an extreme point x^* of $\mathcal{P}$ and define a matrix $\tilde{\Pi}^*$ such that $x^* = vec(\tilde{\Pi}^*)'$. Then, $\tilde{\Pi}^*$ is a feasible schedule. We uptime $\tilde{D}$ by subtracting $q^*\tilde{\Pi}^*$ from $\tilde{D}$, where q^* is defined to be

$$q^* = \left\lfloor \min \left\{ \frac{\tilde{D}_{kl}}{\tilde{\Pi}^*_{kl}} : \tilde{\Pi}^*_{kl} > 0 \right\} \right\rfloor.$$

The uptimed matrix $\tilde{D} := \tilde{D} - q^*\tilde{\Pi}^*$ remain nonnegative, because $\tilde{D}_{kl} - q^*\tilde{\Pi}^*_{kl} \geq 0, \forall k,l$, by the

definition of q^*. For each k and l, $\tilde{\Pi}^*_{kl} \leq \tilde{D}_{kl}$ by definition, so q^* is a positive integer. Since $\tilde{\Pi}^*$ has row sums equal to b^r and column sums equal to b^c, the row sums and column sums of the uptimed matrix become $(\bar{\rho}(D) - q^*)b^r$ and $(\bar{\rho}(D) - q^*)b^c$, respectively. For the uptimed matrix $\tilde{D}$, if $\rho(\tilde{D}) > 1$, then we can find an extreme point of the polytope $\mathcal{P}(\tilde{D})$ defined on the new $\tilde{D}$, until row sums and column sums become zero. We will then end up with P feasible schedules $\tilde{\Pi}_1, \tilde{\Pi}_2, \cdots, \tilde{\Pi}_P$ and P positive integer coefficients $q_1, q_2, \cdots, q_P$, such that

$$\tilde{D} = \sum_{p=1}^{P} q_p \tilde{\Pi}^p, \quad \text{and} \quad \bar{\rho}(D) = \sum_{p=1}^{P} q_p.$$

We now show that P is polynomial in input size. Towards this end, claim that in each iteration of subtraction $q^*\tilde{\Pi}^*$ from $\tilde{D}$, at least one entry in $\tilde{D}$ will get reduced by at least a half and be upper bounded by the corresponding capacity. To see this, suppose that k^* and l^* are such that $q^* = \left\lfloor \tilde{D}_{k^*l^*} / \tilde{\Pi}^*_{k^*l^*} \right\rfloor > 0$. Since $\tilde{\Pi}^*_{k^*l^*} \leq \min\{b^r_{k^*}, b^c_{l^*}\}$, we must have $\tilde{D}_{k^*l^*} - q^*\tilde{\Pi}^*_{k^*l^*} < \min\{b^r_{k^*}, b^c_{l^*}\}$ and $\tilde{D}_{k^*l^*} - q^*\tilde{\Pi}^*_{k^*l^*} < \tilde{D}_{k^*l^*}/2$. Therefore, all of the entries in $\tilde{D}$ will be reduced to 0 in $O((log(b_M) + 1)\left(m^I + m^O\right)^2)$ rounds, where b_M is the maximum capacity on the inputs and outputs. P is thus polynomial in input. To conclude the proof of Proposition 5, take the upper $m^I \times m^O$ submatrices of $\tilde{\Pi}_1$, $\tilde{\Pi}_2$, $\cdots$, $\tilde{\Pi}_P$ to get $\Pi_1, \Pi_2, \ldots, \Pi_P$, and we obtain the decomposition $D = \sum_{p=1}^{P} q_p \Pi^p$. □

We are now ready to prove Theorem 3.

Proof of Theorem 3. Given a coflow matrix D, we can find a decomposition of $D = \sum_{p=1}^{P} q_p \Pi^p$ by Proposition 5, which gives us a polynomial number of feasible schedules to finish processing coflow D in $\bar{\rho}(D)$ time slots. To be more specific, for $p = 1, \ldots, P$, we transfer Π^p_{ij} data units from input i to output j for q_p time slots to finish all the job requirements. Since Π^p satisfies the

constraints $\Pi^p \leq D$, $S_i^r(\Pi^p) \leq \lambda_i$ and $S_j^r(\Pi^p) \leq \mu_j$, the schedule is feasible. See Algorithm 2.3 for a complete description of the decomposition/scheduling algorithm.

We now consider the complexity of the algorithm. In each iteration, we solve a linear program of size $O(m^I + m^O)$ by $O((m^I + m^O)^2)$ for an extreme point, and the maximum size of the coefficients is 1, so the running time is polynomial using the interior point method. Since the number of iterations is polynomial in input size as stated in Proposition 5, our scheduling algorithm can finish processing coflow D in polynomial time. □

To schedule coflows in a general network, we still follow our approximation algorithm described in Algorithm 2.2, except that we solve (LP-EXP-G) in the ordering step, compute the normalized load V_k according to equation 2.4.1 and schedule the grouped coflows using Algorithm 2.3. The proofs of Proposition 1 and Theorem 1 are exactly the same as in §2.3.3, with a minor modification required in the proof of Lemma 4. We provide the proof of Lemma 4 under the general networks settings in Appendix B for completeness.

Chapter 3

Experimental Analysis of Approximation Algorithms for Coflow Scheduling

3.1 Introduction

In this chapter, we carry out a systematic experimental study on the practical performance of several coflow scheduling algorithms, including our approximation algorithms developed in Chapter 2. Our experiments are conducted on real-world data gathered from Facebook and extensive simulated data, where we compare our approximation algorithm and its modifications to several other scheduling algorithms in an offline setting, and evaluate their relative performances, and compare them to an LP-based lower bound. The algorithms that we consider in this chapter are characterized by several main components, such as the coflow order in which the algorithms follow, the grouping of the coflows, and the backfilling rules. We study the impact of each such component on the algorithm performance, and demonstrate the robust and near-optimal performance of our

approximation algorithm and its modifications in the offline setting, under the case of zero release times as well as general release times. We also consider online variants of the offline algorithms, and show that the online version of our approximation algorithm has near-optimal performance on real-world data and simulated instances.

3.1.1 Overview of Experiments

Since our LP-based algorithm consists of an ordering and a scheduling stage, we are interested in algorithmic variations for each stage and the performance impact of these variations. More specifically, we examine the impact of different ordering rules, coflow grouping and backfilling rules, in both the offline and online settings. In this chapter, we conduct a comprehensive study by considering various ordering and backfilling rules, and examining the performance of algorithms on general instances in addition to real-world data. We also consider the offline setting with general release times, and online extensions of algorithms.

Workload

Our evaluation uses real-world data, which is a Hive/MapReduce trace collected from a large production cluster at Facebook [14, 15, 39], as well as extensive simulated instances.

For real-world data, we use the same workload as described in [15] The workload is based on a Hive/MapReduce trace at Facebook that was collected on a 3000-machine cluster with 150 racks, so the datacenter in the experiments can be modeled as 150×150 network switch (and each coflow represented by a 150×150 matrix). The cluster has a 10:1 core-to-rack oversubscription ratio with a total bisection bandwidth of 300Gbps. Therefore, each ingress/egress port has a capacity of

1Gbps, or equivalently 128MBps. We select the time unit to be 1/128 second accordingly so that each port has the capacity of 1MB per time unit. We filter the coflows based on the number of non-zero flows, which we denote by M', and we consider three collections of coflows, filtered by the conditions $M' \geq 25$, $M' \geq 50$ and $M' \geq 100$, respectively. As pointed out in [15], coflow scheduling algorithms may be ineffective for very sparse coflows in real datacenters, due to communication overhead, so we investigate the performance of our algorithms for these three collections.

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

3.1.2 Main Findings

Our main experimental findings are as follows:

- Algorithms with coflow grouping consistently outperform those without grouping. Similarly, algorithms that use backfilling consistently outperform those that do not use backfilling. The benefit of backfilling can be further improved by using a balanced backfilling rule (see §3.2.2 for details).

- The performance of the LP-based algorithm and its extensions is relatively robust, and among the best compared with those that use other simpler ordering rules, in the offline setting.
- In the offline setting with general release times, the magnitude of inter-arrival times relative to the processing times can have complicated effects on the performance of various algorithms. (see §3.3.1 for details).
- The LP-based algorithm can be extended to an online algorithm and has near-optimal performance.

3.2 Offline Algorithms; Zero Release Time

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

3.2.1 Ordering heuristics

An intelligent ordering of coflows in the ordering stage can substantially reduce coflow completion times. We consider the following five greedy ordering rules, in addition to the LP-based order (3.4.2), and study how they affect algorithm performance.

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

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

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

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

Definition 5. *The Earliest Completion Time first (ECT) heuristic generates a sequence of coflow one at a time; each time it selects as the next coflow the one that would be completed the earliest*[1].

3.2.2 Scheduling via Birkhoff-von Neumann decomposition, backfilling and grouping

The derivation of the actual sequence of schedules in the scheduling stage of our LP-based algorithm relies on two key ideas: scheduling according to an optimal (Birkhoff-von Neumann)

[1]These completion times depend on the scheduling rule used. Thus, ECT depends on the underlying scheduling algorithm. In §3.2.2, the scheduling algorithms are described in more detail.

decomposition, and a suitable grouping of the coflows. It is reasonable to expect grouping to improve algorithm performance, because it may consolitime skewed coflow matrices to form more balanced ones that can be scheduled more efficiently. Thus, we compare algorithms with grouping and those without grouping to understand its effect. The particular grouping procedure that we consider here Step 2 of Algorithm 2.2, and basically groups coflows into geometrically increasing groups, based on aggregate demand. Coflows of the same group are treated as a single, *aggregated* coflow, and this consolitimed coflow is scheduled according to the Birkhoff-von Neumann decomposition (Algorithm 2.1).

Backfilling is a common strategy used in scheduling for computer systems to increase resource utilization (see, e.g. [15]). While it is difficult to analytically characterize the performance gain from backfilling in general, we evaluate its performance impact experimentally. We consider two backfilling rules, described as follows. Suppose that we are currently scheduling coflow D. The schedules are computed using the Birkhoff-von Neumann decomposition, which in turn makes use of a related, *augmented* matrix $\tilde{D}$, that is component-wise no smaller than D. The decomposition may introduce unforced idle time, whenever $D \neq \tilde{D}$. When we use a schedule that matches input i to output j to serve the coflow with $D_{ij} < \tilde{D}_{ij}$, and if there is no more service requirement on the pair of input i and output j for the coflow, we backfill in order from the flows on the same pair of ports in the subsequent coflows. When grouping is used, backfilling is applied to the aggregated coflows. The two backfilling rules that we consider – which we call *backfilling* and *balanced backfilling* – are only distinguished by the *augmentation* procedures used, which are, respectively, the augmentation used in Step 1 of Algorithm 2.1 and the balanced augmentation described in Algorithm 3.1.

Algorithm 3.1 Balanced Coflow Augmentation

Data: A single coflow $D = \left(d_{ij}\right)_{i,j=1}^{m}$.
Result: A matrix $\tilde{D} = \left(\tilde{d}_{ij}\right)_{i,j=1}^{m}$ with equal row and column sums, and $D \leq \tilde{D}$.
Let ρ be the load of D.
$p_i \leftarrow \rho - \sum_{j'=1}^{m} d_{ij'}$, for $i = 1, 2, \ldots, m$.
$q_i \leftarrow \rho - \sum_{i'=1}^{m} d_{i'j}$, for $j = 1, 2, \ldots, m$.
$\Delta \leftarrow m\rho - \sum_{i=1}^{m}\sum_{j=1}^{m} d_{ij}$.
$d'_{ij} = \lfloor d_{ij} + p_i q_i / \Delta \rfloor$.
Augment $D' = (d'_{ij})$ to a matrix $\tilde{D}$ with equal row and column sums (see Step 1 of Algorithm 2.1).

The balanced augmentation (Algorithm 3.1) results in less skewed matrices than the augmentation step in Algorithm 2.1, since it first "spreads out" the unevenness among the components of a coflow. To illustrate, let

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

Under the balanced augmentation, D is augmented to B and under the augmentation of Algorithm 2.1, D is augmented to C.

3.2.3 Scheduling Algorithms and Metrics

We consider 30 different scheduling algorithms, which are specified by the ordering used in the ordering stage, and the actual sequence of schedules used in the scheduling stage. We consider 6 different orderings described in §3.2.1, and the following 5 cases in the scheduling stage:

- (a) without grouping or backfilling, which we refer to as the base case;

- (b) without grouping but with backfilling;
- (c) without grouping but with balanced backfilling;
- (d) with grouping and with backfilling;
- (e) with grouping and with balanced backfilling.

We will refer to these cases often in the rest of the chapter. Our LP-based algorithm (Algorithm 2.2) corresponds to the combination of LP-based ordering and case (d).

For ordering, six different possibilities are considered. We use H_A to denote the ordering of coflows by heuristic A, where A is in the set $\{$FIFO, STPT, SMPT, SMCT, ECT$\}$, and H_{LP} to denote the LP-based coflow ordering.

3.2.4 Performance of Algorithms on Real-world Data

We compute the total weighted completion times for all 6 orders in the 5 different cases (a) – (e) described in §3.2.3, through a set of experiments on filtered coflow data. We present representative comparisons of the algorithms here.

Figure 3.1 plots the total weighted completion times as percentages of the base case (a), for the case of equal weights. Grouping and backfilling both improve the total weighted completion time with respect to the base case for all 6 orders. In addition to the reduction in the total weighted completion time from backfilling, which is up to 7.69%, the further reduction from grouping is up to 24.27%, while the improvement from adopting the balanced backfilling rule is up to 20.31%. For 5 non-arbitrary orders (excluding FIFO), scheduling with both grouping and balanced backfilling (i.e., case (e)) gives the smallest total weighted completion time.

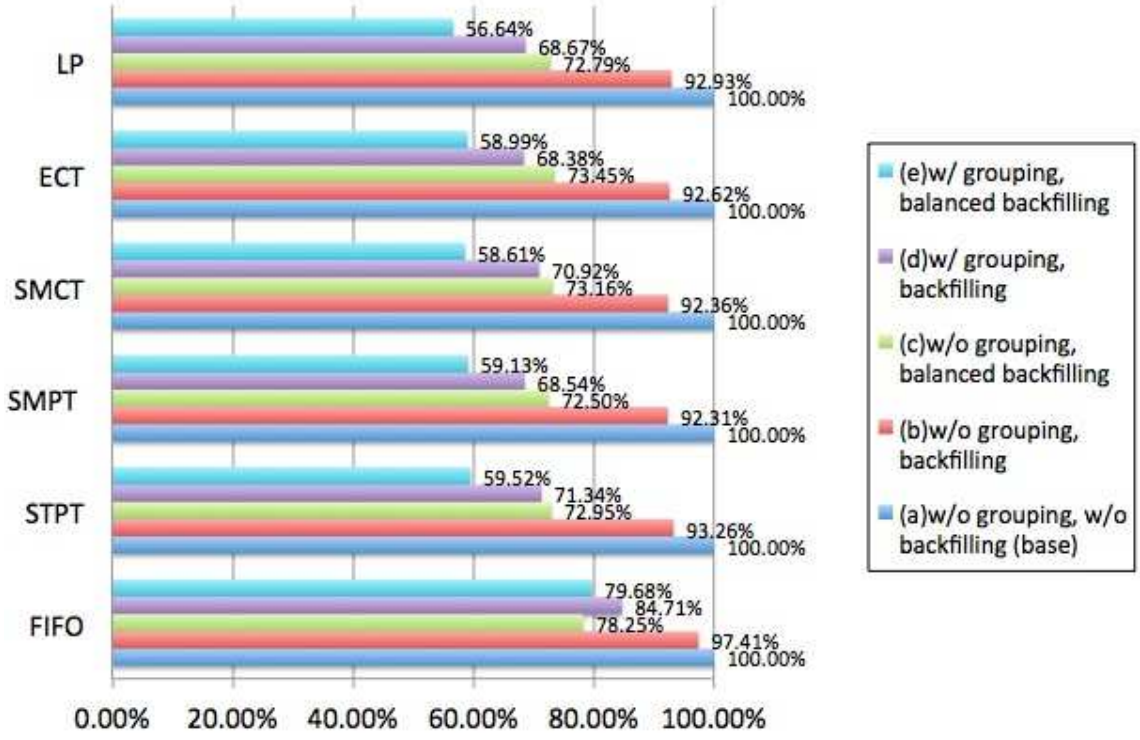

Figure 3.1: Comparison of total weighted completion times normalized using the base case (e) for each order. Data are filtered by $M' \geq 50$. Weights are equal.

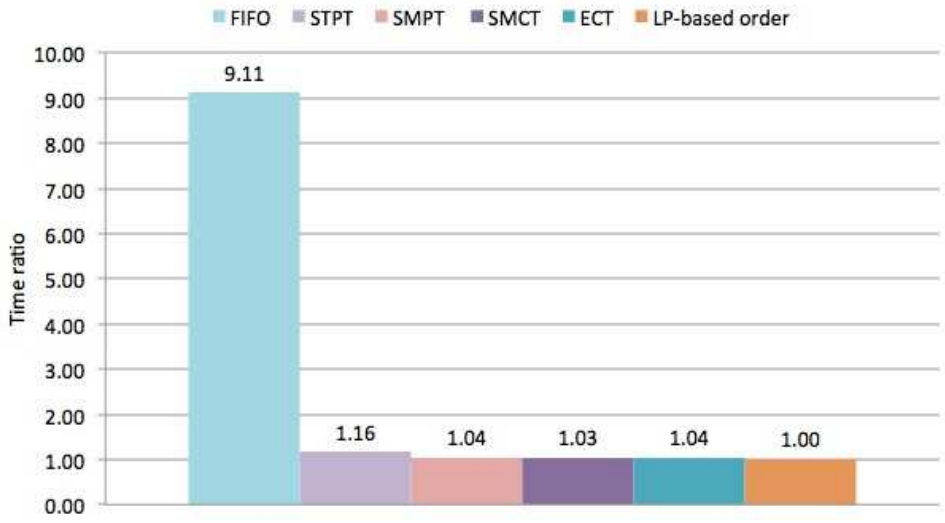

Figure 3.2: Comparison of 6 orderings with zero release times on Facebook data. Data is filtered by $M' \geq 50$.

We then compare the performances of different coflow orderings. Figure 3.2 shows the comparison of total weighted completion times evaluated on filtered coflow data in case (e) where the scheduling stage uses both grouping and balanced backfilling. Compared with H_{FIFO}, all other ordering heuristics reduce the total weighted completion times of coflows by a ratio between 7.88 and 9.11, with H_{LP} performing consistently better than other heuristics.

Cost of Matching

The main difference between our coflow scheduling problem and the well-studied concurrent open shop problem we discussed in §2.1.4 is the presence of matching constraints on paired resources, i.e. inputs and outputs, which is the most challenging part in the design of approximation algorithms. Since our approximation algorithm handles matching constraints, it is more complicated than scheduling algorithms for concurrent open shop problem. We are interested in how much we lose by imposing these matching constraints.

To do so, we generate two sets of coflow data from the Facebook trace. For each coflow k, let the coflow matrix $D^{(k)}$ be a diagonal matrix, which indicates that coflow k only has processing requirement from input i to output i, for $i = 1, \ldots, m$. The processing requirement $D^{(k)}_{i,i}$ is set to be equal to the sum of all dataflows of coflow k in the Facebook trace that require processing from input i. We then construct coflow matrix $\tilde{D}^{(k)}$ such that $\tilde{D}^{(k)}$ is not diagonal and has the same row sum and column sum as $D^{(k)}$. The details of the generation is described as in Algorithm 3.2.

Algorithm 3.2 Construction of coflow data

Data: A single diagonal coflow $D = \left(d_{ij}\right)_{i,j=1}^{m}$.

Result: Another coflow $\tilde{D} = \left(\tilde{d}_{ij}\right)_{i,j=1}^{m}$, such that row and column sums of the two matrices are all equal.

Let $\eta(\tilde{D}) = \sum_{i,j=1}^{m} \tilde{d}_{ij}$ be the sum of all entries in $\tilde{D}$. Similarly, $\eta(D) = \sum_{i=1}^{m} d_{ii}$.

$\tilde{D} \leftarrow 0$.

while $(\eta(\tilde{D}) < \eta(D))$ **do**

$S_i \leftarrow \{i : \sum_{j'=1}^{m} \tilde{D}_{ij'} < d_{ii}\}$; $S_j \leftarrow \{j : \sum_{i'=1}^{m} \tilde{D}_{i'j} < d_{jj}\}$. Randomly pick i^* from set S_i and j^* from set S_j. $\tilde{D} \leftarrow \tilde{D} + pE$, where $p = \min\{d_{i^*i^*} - \sum_{j'=1}^{m} \tilde{D}_{i^*j'}, d_{j^*j^*} - \sum_{i'=1}^{m} \tilde{D}_{i'j^*}\}$, $E_{ij} = 1$ if $i = i^*$ and $j = j^*$, and $E_{ij} = 0$ otherwise.

$\eta \leftarrow \sum_{i,j=1}^{m} \tilde{d}_{ij}$

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

3.2.5 Performance of Algorithms on General Instances

In previous sections, we present the experimental results of several algorithms on the Facebook trace. In order to examine the consistency of the performance of these algorithms, we consider more instances, including examples where certain algorithms behave badly. See §3.2.5 for instances under which some algorithms have approximation ratio $\Omega(\sqrt{m})$, where m is the size of the network.

General instances

We present comparison of total weighted completion time for 6 orderings and 5 cases on general simulated instances as described in §3.1.1, in Appendix Tables C.1 to C.5, normalized with respect to the LP-based ordering in case (c), which performs best on all of the instances. We have the similar observation from the general instances that both grouping and backfilling reduce the completion time. However, under balanced backfilling, grouping does not improve performance much. Both grouping and balanced backfilling form less skewed matrices that can be scheduled more efficiently, so when balanced backfilling is used, the effect of grouping is less pronounced. It is not clear whether case (c) with balanced backfilling only is in general better than case (e) with both balanced backfilling and grouping, as we have seen Facebook data on which case (e) gives the best result. As for the performance of the orderings, on the one hand, we see in Table C.3 very close time ratios among all the non-arbitrary orderings on instances 6 - 30, and a better performance of H_{ECT} on sparse instances 1 - 5 over other orderings; on the other hand, there are also instances where ECT performs poorly (see §3.2.5 for details). Besides their performance, the running times of the algorithms that we consider are also important. The running time of an algorithm consists of two main parts; computing the ordering and computing the schedule. On a Macbook Pro with 2.53 GHz two processor cores and 6G memory, the five ordering rules, FIFO, STPT, SMPT, SMCT and ECT, take less than 1 second to compute, whereas the LP-based order can take up to 90 seconds. Scheduling with backfilling can be computed in around 1 minute, whereas balanced backfilling computes the schedules with twice the amount of time, beause the balanced augmented matrices

have more non-zero entries. Besides improving performance, grouping can also reduce the running time by up to 90%.

Bad instances for greedy heuristics

We consider the following examples which illustrate instances on which certain ordering heuristics do not perform well.

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

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

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

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

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

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

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

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

3.3 Offline Algorithms; General Release Times

In this section, we examine the performances of the same class of algorithms and heuristics as that studied in Section 3.2, when release times can be general. We first extend descriptions of various heuristics to account for release times.

The FIFO heuristic computes a coflow order according to non-decreasing release time r. (Note that when all release times are distinct, FIFO specifies a unique ordering on coflows, instead of any

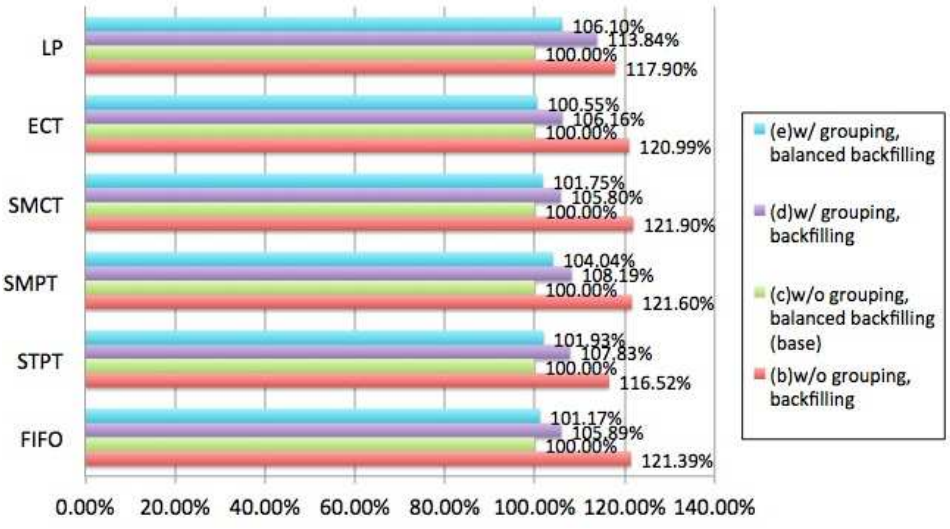

Figure 3.3: Comparison of total weighted completion times normalized using the base case (c) for each order. Data are filtered by $M' \geq 50$. Weights are equal.

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

‘ We compute the total weighted completion time for 6 orderings (namely, the LP-based order-

Figure 3.4: Comparison of 6 orderings with general release times on Facebook data. Data is filtered by $M' \geq 50$.

ing (3.4.2) and the orderings from definitions with release times and cases (b) - (e) (recall the description of these cases at the beginning of Section 3.2.3), normalized with respect to the LP-based ordering in case (c). The results for Facebook data are illustrated in Figure 3.3 and Figure 3.4. For general instances, we generate the coflow inter-arrival times from uniform distribution [1, 100] and present the ratios in Tables C.6 to C.9 in the Appendix. As we can see from e.g., Figure 3.3, the effects of backfilling and grouping on algorithm performance are similar to those noted in §3.2.5, where release times are all zero. The STPT and LP-based orderings STPT appear to perform the best among all the ordering rules (see Figure 3.4), because the magnitudes of release times have a greater effect on FIFO, SMPT, SMCT and ECT than they do on STPT.

By comparing Figures 3.2 and 3.4, we see that ECT performs much worse than it does with common release times. This occurs because with general release times, ECT only schedules a coflow after a preceding coflow completes, so it does not backfill. While we have kept the ECT

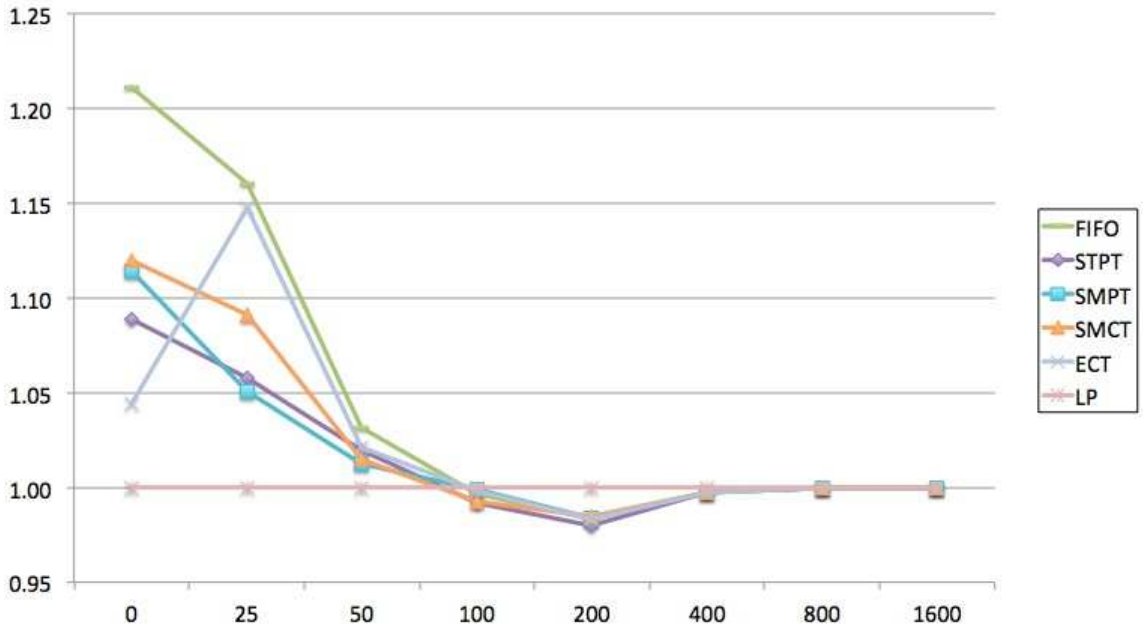

Figure 3.5: Number of flow is 16

ordering heuristic simple and reasonable to compute, no backfilling implies larger completion times, hence the worse performance.

3.3.1 Convergence of heuristics with respect to release times

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

As we can see from Figures 3.5 to 3.6, all the heuristics converge to FIFO as the inter-arrival

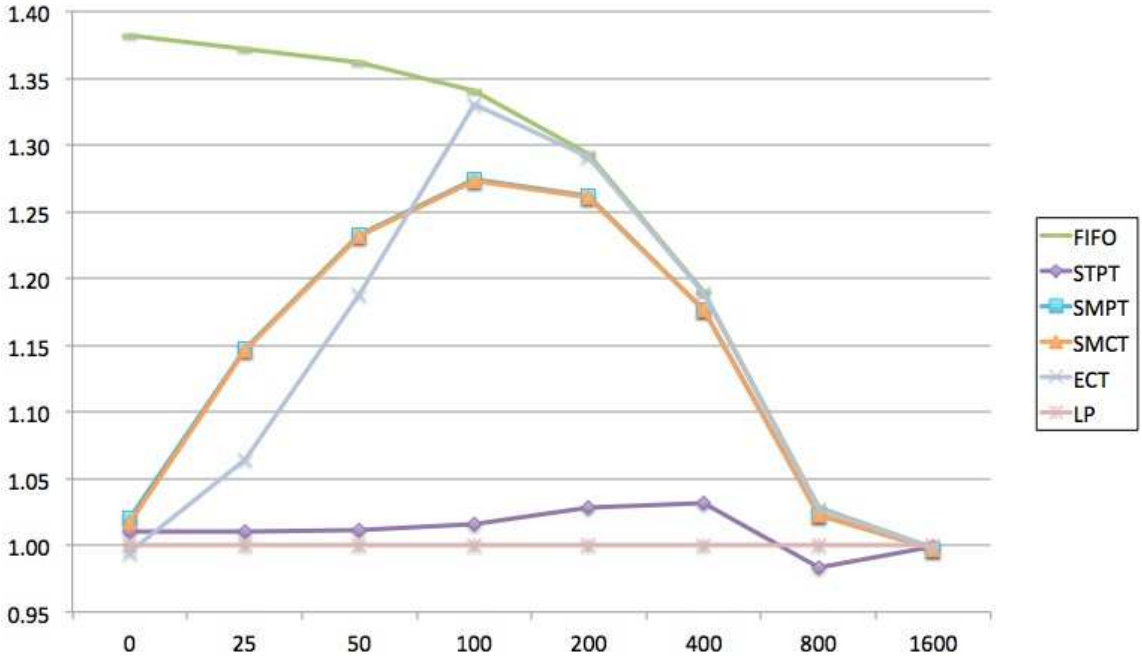

Figure 3.6: Number of flow is 256

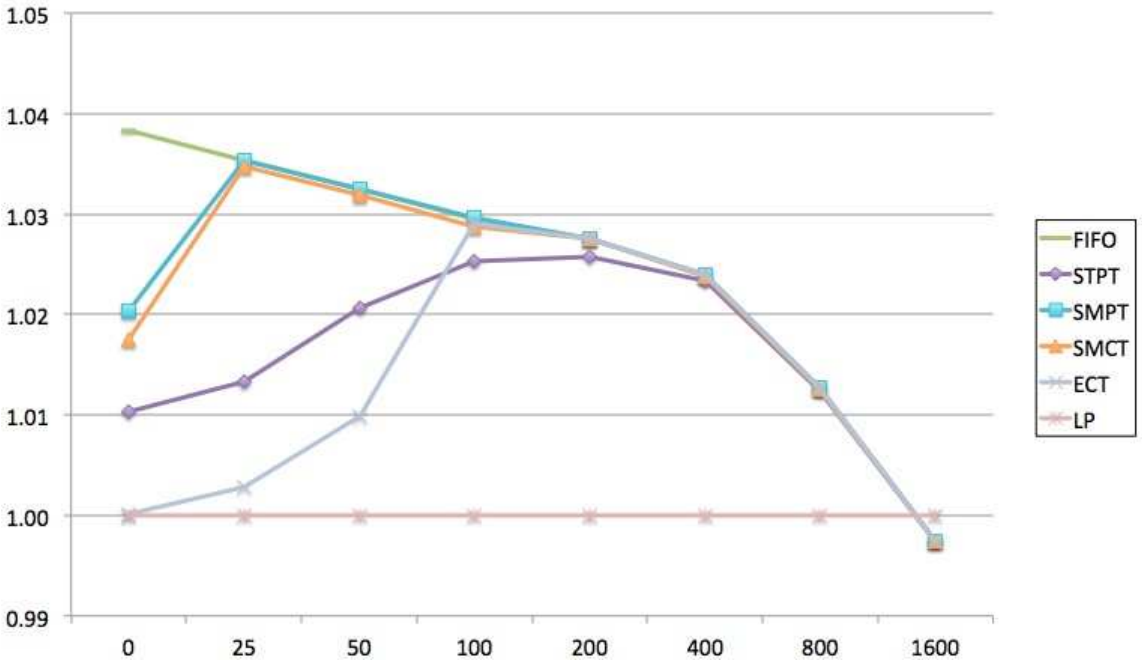

Figure 3.7: Number of flow is uniform in $[16, 256]$

time increases. This is reasonable as the release times dominate the ordering when they are large. The speed of convergence is higher in 3.5 where the coflow matrices in the instance are sparse and release times are more influential in all heuristics. On the contrary, when the coflow matrices are dense, release times weigh less in heuristics, which converge slower to FIFO as shown in 3.6. We also note that for heuristics other than FIFO, the relative performance of an ordering heurstic with respect to the LP-based order may deteriorate and then improve, as we increase the inter-arrival times. This indicates that when inter-arrival times are comparable to the coflow sizes, they can have a significant impact on algorithm performance and the order obtained.

3.4 Online Algorithm

We have discussed the experimental results of our LP-based algorithm and several heuristics in the offline setting, where the complete information of coflows is revealed at time 0. In reality, information on coflows (i.e., flow sizes) is often only revealed at their release times, i.e., in an online fashion. It is then natural to consider online modifications of the offline algorithms considered in earlier sections. We proceed as follows. For the ordering stage, upon each coflow arrival, we re-order the coflows according to their remaining processing requirements. We consider all six ordering rules described in §3.2. For example, the LP-based order is modified upon each coflow arrival, by re-solving the (LP) using the remaining coflow sizes (and the newly arrived coflow) at the time. For the scheduling stage, we use case (c) the balanced backfilling rule without grouping, because of its good performance in the offline setting. See Algorithm 3.3 for an example of online approximation algorithms which uses LP-base order.

Algorithm 3.3 Online LP-based Approximation

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

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

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

$$\bar{C}_k = \sum_{l=1}^{L} \tau_{l-1} \bar{x}_l^{(k)}. \tag{3.4.1}$$

Order and index the coflows according to

$$\bar{C}_1 \leq \bar{C}_2 \leq \ldots \leq \bar{C}_{n_a}. \tag{3.4.2}$$

Step 2: Schedule the coflows in order using Algorithm 2.1 until an release of a new coflow. Uptime the job requirement with the remaining job for each coflow in the system and go back to Step 1.

We compare the performance of the online algorithms and we compare the online algorithms to the offline algorithms. We improve the time ratio for all the orderings except FIFO by allowing re-ordering and preemption in the online algorithm compared with the static offline version. Note that we do not preempt with FIFO order. While several ordering heuristics perform as well as LP-based ordering in the online algorithms, a natural question to ask is how close H_A's are to the optimal, where $A \in \{STPT, SMPT, SMCT, ECT, LP\}$. In order to get a tight lower bound of the coflow scheduling problem, we solve (LP-EXP) for sparse instances. Since it is extremely time

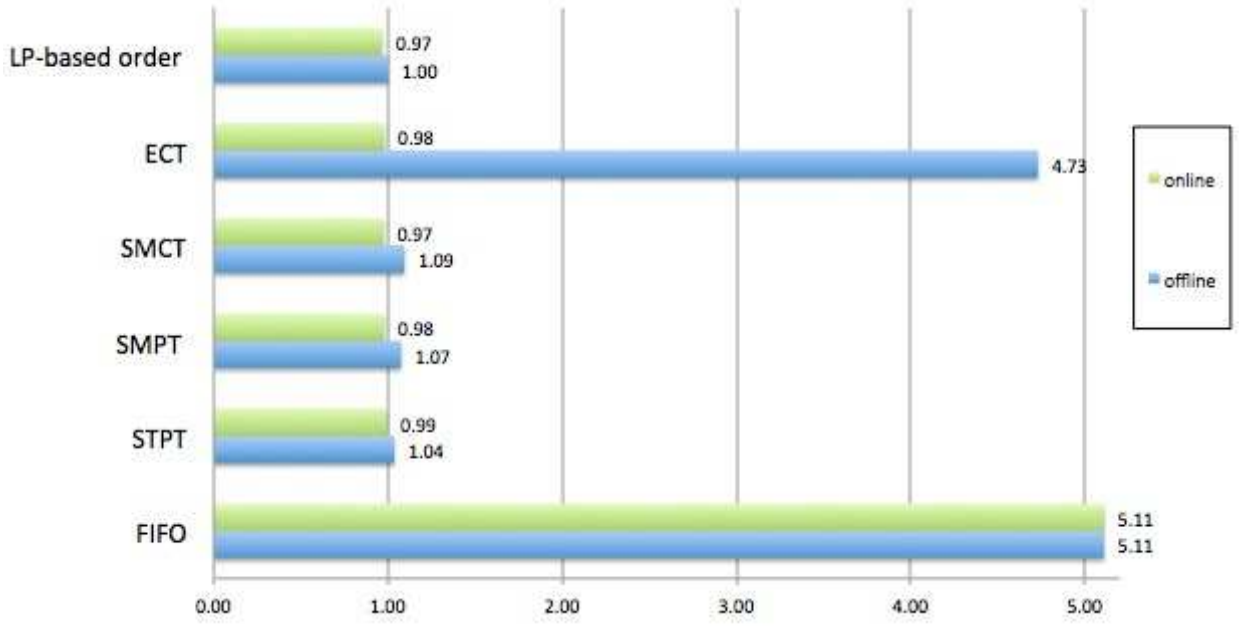

Figure 3.8: Comparison of total weighted completion times with respect to the base case for each order under the offline and online algorithms. Data are filtered by $M' \geq 50$. Weights are equal.

consuming to solve (LP-EXP) for dense instances, we consider a looser lower bound, which is computed as follows. We first aggregate the job requirement on each input and output and solve a single machine scheduling problem for the total weighted completion time, on each input/output. The lower bound is obtained by taking the maximum of the results. The lower bounds are shown in the last column of Table C.11. The ratio of the lower bound over the weighted completion time under H_{LP} is in the range of 0.91 to 0.97, which indicates that it provides a good approximation of the optimal.

Chapter 4

Conclusions

We propose and study several approximation algorithms for solving demand-response contract execution and coflow scheduling problems in this thesis.

In Chapter 1, we present a data-driven near-optimal algorithm for fairly general variants of the demand-response contract execution problems under mild assumptions. Our algorithm is a sample average approximation (SAA) based dynamic program over the multi-period planning horizon. The main theoretical contribution is to provide a sample complexity bound on the number of demand samples required to compute a $(1+\varepsilon)$-approximate policy for the multi-period problem. We show that our SAA algorithm is quite general and can be easily adapted to many different variants of the objective function. Moreover, our computational study suggests that we can achieve a $(1+\varepsilon)$-approximation with a significantly smaller number of samples than what is implied from the theoretical bounds. We also study the structure of optimal policy for the special case of i.i.d. demand and no execution cost and show that a static solution is optimal in this case. For the more general case, our computations experiments show that the near-optimal policy can be well approx-

imated by a affine function of the state with appropriate rounding. This would give a compact representation of the solution policy that makes it more applicable in practice as a control policy.

In Chapter 2, we give the first $O(1)$-approximation algorithms for minimizing the total weighted completion time of coflows in a datacenter network, when all the coflows are released at time 0. Beyond the obvious question of proving an approximation ratio in the presence of general release time and improving the approximation ratio, this work opens up several additional interesting directions in coflow scheduling, such as the consideration of other metrics and the addition of other realistic constraints, such as precedence constraints. We are particularly interested in minimizing weighted coflow *processing* time (usually called *flow time* in the literature), which is harder to approximate (the various hardness results from single machine schedule will clearly carry over), but may lead to the development and understanding of better algorithms, possibly by considering resource augmentation.

In Chapter 3, we perform comprehensive experiments to evaluate different scheduling algorithms for the problem of minimizing the total weighted completion time of coflows in a datacenter network. We also generalize our algorithms to an *on-line* version for them to work in real-time. For additional interesting directions in experimental analysis of coflow scheduling algorithms, we would like to come up with structured approximation algorithms that take into consideration other metrics and the addition of other realistic constraints, such as precedence constraints, and distributed algorithms that are more suitable for implementation in a data center. These new algorithms can be used to design other implementable, practical algorithms.

Bibliography

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

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

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

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

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

[6] Ross Baldick, Sergey Kolos, and Stathis Tompaidis. Interruptible electricity contracts from an electricity retailer's point of view: valuation and optimal interruption. *Operations Research*, 54(4):627–642, 2006.

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

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

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

[10] Douglas W. Caves and Joseph A. Herriges. Optimal dispatch of interruptible and curtailable service options. *Operations Research*, 40(1):104–112, 1992.

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

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

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

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

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

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

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

[18] Murat Fahrioglu and Fernando L. Alvarado. Designing incentive compatible contracts for effective demand management. *IEEE Transactions on Power System*, 15(4):1255–1260, 2000.

[19] Satoru Fujishige. A system of linear inequalities with a submodular function on $\{0, \pm 1\}$ vectors. *Linear Algebra and Its Applications*, 63(1):253–266, 1984.

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

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

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

[23] I. Heller and C.B. Tompkins. An extension of a theorem of dantzig's. *Linear Inequalities and Related Systems, Annals of Mathematics Studies*, 38:247–254, 1956.

[24] A.J. Hoffman and J.B. Kruskal. Integral boundary points of convex polyhedra. *Linear Inequalities and Related Systems, Annals of Mathematics Studies*, 38:223–246, 1956.

[25] Rajnish Kamat and Shmuel S. Oren. Exotic options for interruptible electricity supply contracts. *Operations Research*, 50(5):835–850, 2002.

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

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

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

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

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

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

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

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

[34] Shmuel S. Oren. Integrating real and financial options in demand-side electricity contracts. *Decision Support Systems*, 30(3):279–288, 2001.

[35] Shmuel S. Oren and Stephen A. Smith. Design and management of curtailable electricity service to reduce annual peaks. *Operations Research*, 40(2):213–228, 1992.

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

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

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

[39] Zhen Qiu, Cliff Stein, and Yuan Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 294–303, 2015.

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

[41] P. D. Seymour. Decomposition of regular matroids. *Journal of Combinatorial Theory, Series B*, 28(3):305–359, 1980.

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

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

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

[45] Todd P. Strauss and Shmuel S. Oren. Priority pricing of interruptible electric power with an early notification option. *Energy Journal*, 14(2):175–195, 1993.

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

[47] Le Anh Tuan and Kankar Bhattacharya. Competitive framework for procurement of interruptible load services. *IEEE Transactions on Power System*, 18(2):889–897, 2003.

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

[49] C. W. Yu, S. Zhang, T. S. Chung, and K. P. Wong. Modelling and evaluation of interruptible-load programmes in electricity markets. *IEE Proceedings - Generation Transmission and Distribution*, 152(5):581–588, 2005.

[50] C. W. Yu, S. H. Zhang, L. Wang, and T. S. Chung. Analysis of interruptible electric power in deregulated power systems. *Electric Power Systems Research*, 77(5-6):637–645, 2007.

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

[52] Thomas Zaslavsky. Signed graphs. *Discrete Applied Mathematics*, 4(1):47–74, 1982.

Appendix A

Proof of Theorem 1.4.1

There are two sources of error. One error comes from a discretized state space which allows us to reduce the state space size to polynomial. The other error is due to SAA approximation instead of expectation. We show in Lemma A.0.1 that the discretization error is small. In particular, we show that $|\hat{V}_t - V_t|$ is close point-wise. Next, we show that sampling error is small. In particular, if the number of samples $N = O(T^2/\varepsilon^2)$, we use Chernoff bounds to show that with high probability, the SAA sampling error $|\bar{V}_t - \hat{V}_t|$ is small. Finally, we prove in Lemma A.0.3 that approximated true cost function $\bar{U}_t$ is a good approximation of $\bar{V}_t$, i.e., the error $|\bar{U}_t - \bar{V}_t|$ is small.

Lemma A.0.1. ***(Discretization error)*** *For all S_t, Y_t and X_{t-1}, $t = 1, 2, \cdots, T$,*

$$\left|\hat{V}_t(S_t,\ \lceil Y_t \rceil_\Delta,\ \lceil X_{t-1} \rceil_\varepsilon) - V_t(S_t,\ Y_t,\ X_{t-1})\right| \leq (1 + 2\varepsilon(T-t+1))V_t(S_t,\ Y_t,\ X_{t-1}) + \Delta(T-t+1).$$

Proof. Proof of Lemma A.0.1 We prove the lemma by induction. At time $T+1$,

$$\hat{V}_{T+1}(S_{T+1},\ \lceil Y_{T+1}\rceil_\Delta,\ \lceil X_T\rceil_\varepsilon) = V_{T+1}(S_{T+1},\ Y_{T+1},\ X_T) = 0.$$

We first show that $\hat{V}_t(S_t,\ \lceil Y_t\rceil_\Delta,\ \lceil X_{t-1}\rceil_\varepsilon) \le (1+2\varepsilon(T-t+1))V_t(S_t,\ Y_t,\ X_{t-1})$. Suppose the statement holds for t+1. Let n_t^* be the optimal solution to DP with the value function V_t, then

$$\begin{aligned}
&\hat{V}_t\left(S_t,\ \lceil Y_t\rceil_\Delta,\ \lceil X_{t-1}\rceil_\varepsilon\right)\\
\le\ & \mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\left[(X_t - n_t^*\delta - \lceil Y_t\rceil_\Delta)_+ + g_t(n_t^*) + \hat{V}_{t+1}\left(S_t - n_t^*,\ \left\lceil \lceil Y_t\rceil_\Delta + (X_t - n_t^*\delta - \lceil Y_t\rceil_\Delta)_+\right\rceil_\Delta,\ \lceil X_t\rceil_\varepsilon\right)\right]\\
\le\ & \mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\left[(X_t - n_t^*\delta - Y_t)_+\right] + g_t(n_t^*)\\
&+\mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\left[\hat{V}_{t+1}\left(S_t - n_t^*,\ \left\lceil Y_t + (X_t - n_t^*\delta - Y_t)_+\right\rceil_\Delta,\ \lceil X_t\rceil_\varepsilon\right)\right] \qquad \text{(A.1)}\\
\le\ & (1+\varepsilon)\mathbb{E}_{X_t|X_{t-1}}\left[(X_t - n_t^*\delta - Y_t)_+\right] + g_t(n_t^*)\\
&+(1+\varepsilon)\mathbb{E}_{X_t|X_{t-1}}\left[\hat{V}_{t+1}\left(S_t - n_t^*,\ \left\lceil Y_t + (X_t - n_t^*\delta - Y_t)_+\right\rceil_\Delta,\ \lceil X_t\rceil_\varepsilon\right)\right] \quad \text{(change of distribution)} \qquad \text{(A.2)}\\
\le\ & (1+\varepsilon)\mathbb{E}_{X_t|X_{t-1}}\left[(X_t - n_t^*\delta - Y_t)_+\right] + g_t(n_t^*)\\
&+(1+\varepsilon)(1+2\varepsilon(T-t))\mathbb{E}_{X_t|X_{t-1}}\left[V_{t+1}\left(S_t - n_t^*, Y_t + (X_t - n_t^*\delta - Y_t)_+,\ X_t\right)\right] \qquad \text{(induction)} \qquad \text{(A.3)}\\
\le\ & (1+2\varepsilon(T-t+1))V_t(S_t,\ Y_t,\ X_{t-1}).
\end{aligned}$$

where inequality (A.1) follows from the following case analysis:

Case 1: $X_t - n_t^*\delta - \lceil Y_t\rceil_\Delta \ge 0$ and $X_t - n_t^*\delta - Y_t \ge 0$,

$$\left\lceil \lceil Y_t\rceil_\Delta + (X_t - n_t^*\delta - \lceil Y_t\rceil_\Delta)_+\right\rceil_\Delta = \lceil X_t - n_t^*\delta\rceil_\Delta = \left\lceil Y_t + (X_t - n_t^*\delta - Y_t)_+\right\rceil_\Delta;$$

Case 2: $X_t - n_t^*\delta - \lceil Y_t \rceil_\Delta < 0$ and $X_t - n_t^*\delta - Y_t < 0$,

$$\left\lceil \lceil Y_t \rceil_\Delta + (X_t - n_t^*\delta - \lceil Y_t \rceil_\Delta)_+ \right\rceil_\Delta = \lceil \lceil Y_t \rceil_\Delta \rceil_\Delta = \lceil Y_t \rceil_\Delta = \left\lceil Y_t + (X_t - n_t^*\delta - Y_t)_+ \right\rceil_\Delta;$$

Case 3: $X_t - n_t^*\delta - \lceil Y_t \rceil_\Delta < 0$ and $X_t - n_t^*\delta - Y_t \geq 0$,

$$\begin{aligned} \left\lceil \lceil Y_t \rceil_\Delta + (X_t - n_t^*\delta - \lceil Y_t \rceil_\Delta)_+ \right\rceil_\Delta &= \lceil \lceil Y_t \rceil_\Delta \rceil_\Delta = \lceil Y_t \rceil_\Delta = \lceil X_t - n_t^*\delta \rceil_\Delta \\ &= \left\lceil Y_t + (X_t - n_t^*\delta - Y_t)_+ \right\rceil_\Delta. \end{aligned}$$

and (A.2) can be shown using Assumption 4 for the Markovian demand. Given a function h (.) of X_t, for all Y_t, X_{t-1} and n_t,

$$\begin{aligned} & \left| \mathbb{E}_{X_t|\lceil X_{t-1} \rceil_\varepsilon}[h(X_t)] - \mathbb{E}_{X_t|X_{t-1}}[h(X_t)] \right| \\ = & \left| \int h(x)(f(x|\lceil X_{t-1} \rceil_\varepsilon) - f(x|X_{t-1}))dx \right| \\ \leq & \int h(x)\,|f(x|\lceil X_{t-1} \rceil_\varepsilon) - f(x|X_{t-1})|\,dx \\ \leq & \left(\frac{\max(\lceil X_{t-1} \rceil_\varepsilon,\, X_{t-1})}{\min(\lceil X_{t-1} \rceil_\varepsilon,\, X_{t-1})} - 1 \right) \int h(x) f(x|X_{t-1})dx \\ \leq & \varepsilon \int h(x) f(x|X_{t-1})dx \\ \leq & \varepsilon \mathbb{E}_{X_t|X_{t-1}}[h(X_t)]. \end{aligned}$$

On the other hand, we verify that $\hat{V}_t(S_t,\, \lceil Y_t \rceil_\Delta,\, \lceil X_{t-1} \rceil_\varepsilon) \geq (1 - 2\varepsilon(T-t+1))V_t(S_t,\, Y_t,\, X_{t-1}) - \Delta(T-t+1)$ for all S_t, Y_t and X_{t-1}. At time $T+1$, $\hat{V}_{T+1}(S_{T+1},\, \lceil Y_{T+1} \rceil_\Delta,\, \lceil X_T \rceil_\varepsilon) = V_{T+1}(S_{T+1},\, Y_{T+1},\, X_T) = 0$. Suppose the statement holds for t+1. Let $\hat{n}_t^*$ be the optimal solution to DP with value function

$\hat{V}_t$, then

$$
\begin{aligned}
& \hat{V}_t\left(S_t,\ \lceil Y_t\rceil_\Delta,\ \lceil X_{t-1}\rceil_\varepsilon\right) \\
= \ & \mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\left[(X_t-\hat{n}_t^*\delta-\lceil Y_t\rceil_\Delta)_+\right]+g_t(\hat{n}_t^*) \\
& +\mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\left[\hat{V}_{t+1}\left(S_t-\hat{n}_t^*,\ \left\lceil\lceil Y_t\rceil_\Delta+(X_t-\hat{n}_t^*\delta-\lceil Y_t\rceil_\Delta)_+\right\rceil_\Delta,\ \lceil X_t\rceil_\varepsilon\right)\right] \\
\geq \ & \mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\left[(X_t-\hat{n}_t^*\delta-Y_t)_+\right]-\Delta+g_t(\hat{n}_t^*) \\
& +\mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\left[\hat{V}_{t+1}\left(S_t-\hat{n}_t^*,\ \left\lceil Y_t+(X_t-\hat{n}_t^*\delta-Y_t)_+\right\rceil_\Delta,\ \lceil X_t\rceil_\varepsilon\right)\right] \\
\geq \ & (1-\varepsilon)\mathbb{E}_{X_t|X_{t-1}}\left[(X_t-\hat{n}_t^*\delta-Y_t)_+\right]-\Delta+g_t(\hat{n}_t^*) \\
& +(1-\varepsilon)\mathbb{E}_{X_t|X_{t-1}}\left[\hat{V}_{t+1}\left(S_t-\hat{n}_t^*,\ \left\lceil Y_t+(X_t-\hat{n}_t^*\delta-Y_t)_+\right\rceil_\Delta,\ \lceil X_t\rceil_\varepsilon\right)\right] \\
\geq \ & (1-\varepsilon)\mathbb{E}_{X_t|X_{t-1}}\left[(X_t-\hat{n}_t^*\delta-Y_t)_+\right]-\Delta+g_t(\hat{n}_t^*) \\
& +(1-\varepsilon)\left(\mathbb{E}_{X_t|X_{t-1}}\left[(1-2\varepsilon(T-t))\left(V_{t+1}\left(S_t-\hat{n}_t^*, Y_t+(X_t-\hat{n}_t^*\delta-Y_t)_+,\ X_t\right)\right)-\Delta(T-t)\right]\right) \\
\geq \ & (1-2\varepsilon(T-t+1))V_t(S_t,\ Y_t,\ X_{t-1})-\Delta(T-t+1).
\end{aligned}
$$

Therefore, we have

$$\left|\hat{V}_t(S_t,\ \lceil Y_t\rceil_\Delta,\ \lceil X_{t-1}\rceil_\varepsilon)-V_t(S_t,\ Y_t,\ X_{t-1})\right|\leq 2\varepsilon(T-t+1)V_t(S_t,\ Y_t,\ X_{t-1})+\Delta(T-t+1)$$

for all S_t, Y_t and X_{t-1}. □

We use Chernoff bound to show that $\hat{V}$ and $\overline{V}$ are close to each other. Since the value functions

$\hat{V}$ and $\bar{V}$ are only defined for the discretized variables $\lceil Y_t \rceil_\Delta$ and $\lceil X_{t-1} \rceil_\Delta$ on the grid, we write Y_t and X_{t-1} instead for simplicity.

Lemma A.0.2. ***(Sampling error)*** *If the number of sample $N = O(\frac{T^2}{\varepsilon^2})$, then for any S_t, Y_t and X_{t-1},*

$$\left|\bar{V}_t(S_t,\, Y_t,\, X_{t-1}) - \hat{V}_t(S_t,\, Y_t,\, X_{t-1})\right| \leq \frac{2(T-t+1)}{T}\Delta T$$

with probability at least $1 - \frac{1}{T^2 S_1^2}$.

Proof. Proof of Lemma A.0.2 At time $T+1$,

$$\hat{V}_{T+1}(S_{T+1}, Y_{T+1}, X_T) = \bar{V}_{T+1}(S_{T+1}, Y_{T+1}, X_T) = 0$$

Note that if we do not execute any contract, the total execution cost is zero and the value function V_1 is equal to the peak load X_{max}, which provides an upper bound on V_1 under the optimal policy. It follows that

$$\mathbb{P}(V_t > M) \leq \mathbb{P}(V_1 > M) \leq \mathbb{P}(X_{max} > M) < \eta,\ 1 \leq t \leq T.$$

Hence, M is also an upper bound on the value function with a failure probability of η.

We prove the induction hypothesis: for any S_t and Y_t, with the number of samples stated in the lemma,

$$\left|\bar{V}_t(S_t, Y_t, X_{t-1}) - \hat{V}_t(S_t, Y_t, X_{t-1})\right| \leq \frac{2(T-t+1)}{T}\Delta T$$

with probability at least $1-\frac{T-t+1}{2T^3S_1^3}$. Take $N=\frac{3T^2}{\varepsilon^2}log(2TS_1)$. We know from Chernoff Bound that,

$$Pr\left(\left|\frac{\bar{V}_T(S_T,Y_T,X_{T-1})}{M}-\frac{V_T(S_T,Y_T,X_{T-1})}{M}\right|\geq\frac{\varepsilon}{T}\right)\leq 2e^{-\varepsilon^2N_t/(T^2)}\leq\frac{1}{4T^3S_1^3}$$

Therefore, $|\hat{V}_T(S_T,Y_T,X_{T-1})-\bar{V}_T(S_T,Y_T,X_{T-1})|\leq\varepsilon M/T$ with probability $\geq 1-\frac{1}{4T^3S_1^3}$.

For any strategy n_t, define

$$\begin{aligned}
\hat{V}_t(S_t,Y_t,X_{t-1})_{n_t} &\triangleq \mathbb{E}_{X_t|X_{t-1}}\Bigg[(X_t-n_t\delta-Y_t)_+ \\
&\quad +\hat{V}_{t+1}\left(S_t-n_t,\ \lceil Y_t+(X_t-n_t\delta-Y_t)_+\rceil_\Delta,\lceil X_t\rceil_\varepsilon\right)\Bigg], \\
\bar{V}_t(S_t,Y_t,X_{t-1})_{n_t} &\triangleq \frac{1}{N_t}\sum_{i=1}^{N_t}\Bigg[\left(X_t^{(i)}-n_t\delta-Y_t\right)_+ \\
&\quad +\bar{V}_{t+1}\left(S_t-n_t,\ \left\lceil Y_t+\left(X_t^{(i)}-n_t\delta-Y_t\right)_+\right\rceil_\Delta,\ \left\lceil X_t^{(i)}\right\rceil_\varepsilon\right)\Bigg].
\end{aligned}$$

Note that $\hat{V}_t(S_t,Y_t,X_{t-1})=\hat{V}_t(S_t,Y_t,X_{t-1})_{\hat{n}_t^*}$ and $\bar{V}_t(S_t,Y_t,X_{t-1})=\bar{V}_t(S_t,Y_t,X_{t-1})_{\bar{n}_t^*}$, where $\hat{n}_t^*$ and $\bar{n}_t^*$ are the optimal solutions to DP with value function $\hat{V}$ and DP respectively. For any S_t , Y_t and

X_{t-1},

$$
\begin{aligned}
& \left|\bar{V}_t(S_t,Y_t,X_{t-1})_{n_t} - \hat{V}_t(S_t,Y_t,X_{t-1})_{n_t}\right| \\
\leq\ & \left|\frac{1}{N_t}\sum_{i=1}^{N_t}\left(X_t^{(i)} - n_t\delta - Y_t\right)_+ - \mathbb{E}_{X_t|X_{t-1}}\left[(X_t - n_t\delta - Y_t)_+\right]\right| \\
& + \left|\frac{1}{N_t}\sum_{i=1}^{N_t}\bar{V}_{t+1}\left(S_t - n_t,\ \left\lceil Y_t + \left(X_t^{(i)} - n_t\delta - Y_t\right)_+\right\rceil_\Delta,\ \left\lceil X_t^{(i)}\right\rceil_\varepsilon\right)\right. \\
& \left. - \mathbb{E}_{X_t|X_{t-1}}\left[\bar{V}_{t+1}\left(S_t - n_t,\ \left\lceil Y_t + (X_t - n_t\delta - Y_t)_+\right\rceil_\Delta, \lceil X_t\rceil_\varepsilon\right)\right]\right| \\
& + \left|\mathbb{E}_{X_t|X_{t-1}}\left[\bar{V}_{t+1}\left(S_t - n_t,\ \left\lceil Y_t + (X_t - n_t\delta - Y_t)_+\right\rceil_\Delta, \lceil X_t\rceil_\varepsilon\right)\right]\right. \\
& \left. - \mathbb{E}_{X_t|X_{t-1}}\left[\hat{V}_{t+1}\left(S_t - n_t,\ \left\lceil Y_t + (X_t - n_t\delta - Y_t)_+\right\rceil_\Delta, \lceil X_t\rceil_\varepsilon\right)\right]\right| \\
\leq\ & \frac{\varepsilon M}{T} + \frac{\varepsilon M}{T} + \frac{2(T-t)}{T}\Delta T \\
=\ & \frac{2\varepsilon M(T-t+1)}{T} \\
=\ & \frac{2(T-t+1)}{T}\Delta T
\end{aligned}
$$

with probability $\geq 1 - \frac{T-t}{2T^3S_1^3} - \frac{1}{4T^3S_1^3} - \frac{1}{4T^3S_1^3} > 1 - \frac{T-t+1}{2T^3S_1^3}$. We thus have

$$
\begin{aligned}
\bar{V}_t(S_t,Y_t,X_{t-1}) &= \bar{V}_t(S_t,Y_t,X_{t-1})_{\bar{n}_t^*} \leq \bar{V}_t(S_t,Y_t,X_{t-1})_{\hat{n}_t^*} \\
&\leq \hat{V}_t(S_t,Y_t,X_{t-1})_{\hat{n}_t^*} + \frac{2(T-t+1)}{T}\Delta T \\
\bar{V}_t(S_t,Y_t,X_{t-1}) &= \bar{V}_t(S_t,Y_t,X_{t-1})_{\bar{n}_t^*} \geq \hat{V}_t(S_t,Y_t,X_{t-1})_{\bar{n}_t^*} - \frac{2(T-t+1)}{T}\Delta T \\
&\geq \hat{V}_t(S_t,Y_t,X_{t-1})_{\hat{n}_t^*} - \frac{2(T-t+1)}{T}\Delta T
\end{aligned}
$$

with probability $\geq 1 - \frac{T-t+1}{2T^3 S_1^3}$. Therefore, for any S_t and Y_t,

$$\left|\bar{V}_t(S_t, Y_t, X_{t-1}) - \hat{V}_t(S_t, Y_t, X_{t-1})\right| \leq \frac{2(T-t+1)}{T}\Delta T$$

with probability $\geq 1 - \frac{1}{T^2 S_1^2}$ □

Lemma A.0.3. ***(True cost of approximation)*** *If the number of sample $N = O(\frac{T^2}{\varepsilon^2})$, then for any S_t, Y_t and X_{t-1},*

$$|\bar{U}_t(S_t,\, Y_t,\, X_{t-1}) - \bar{V}_t(S_t,\, \lceil Y_t \rceil_\Delta,\, \lceil X_{t-1} \rceil_\varepsilon)| \leq 2\varepsilon(T-t+1)\bar{V}_t(S_t,\, Y_t,\, X_{t-1}) + 3\Delta(T-t+1)$$

with probability $\geq 1 - \frac{1}{T^2 S_1^2}$.

Proof. Proof of Lemma A.0.3 We prove the induction hypothesis: for any S_t and Y_t, with the number of samples stated in the lemma,

$$|\bar{U}_t(S_t,\, Y_t,\, X_{t-1}) - \bar{V}_t(S_t,\, \lceil Y_t \rceil_\Delta,\, \lceil X_{t-1} \rceil_\varepsilon)| \leq 2\varepsilon(T-t+1)\bar{V}_t(S_t,\, Y_t,\, X_{t-1}) + 3\Delta(T-t+1)$$

with probability $\geq 1 - \frac{T-t+1}{2T^3 S_1^3}$. At time $T+1$, $\bar{U}_{T+1}(S_{T+1},\, Y_{T+1},\, X_T) = \bar{V}_{T+1}(S_{T+1},\, \lceil Y_{T+1} \rceil_\Delta,\, \lceil X_T \rceil_\Delta) =$

0. Suppose the statement holds for t+1. Let $\bar{n}_t^*$ be the optimal solution to DP, then

$$
\begin{aligned}
& \bar{U}_t(S_t,\, Y_t,\, X_{t-1}) \\
= \;& \mathbb{E}_{X_t|X_{t-1}}\left[(X_t-\bar{n}_t^*\delta-Y_t)_+ + g_t(\bar{n}_t^*) + \bar{U}_{t+1}\left(S_t-\bar{n}_t^*,\, Y_t+(X_t-\bar{n}_t^*\delta-Y_t)_+,\, X_t\right)\right] \\
\leq \;& \mathbb{E}_{X_t|X_{t-1}}\Bigg[(X_t-\bar{n}_t^*\delta-Y_t)_+ + g_t(\bar{n}_t^*) \\
& +(1+2\varepsilon(T-t))\bar{V}_{t+1}\left(S_t-\bar{n}_t^*,\, \left\lceil Y_t+(X_t-\bar{n}_t^*\delta-Y_t)_+\right\rceil_\Delta,\, \lceil X_t\rceil_\varepsilon\right)+3\Delta(T-t+1)\Bigg] \\
\leq \;& (1+\varepsilon)\mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\Bigg[(X_t-\bar{n}_t^*\delta-\lceil Y_t\rceil_\Delta)_+ + \Delta + g_t(\bar{n}_t^*) \\
& +(1+2\varepsilon(T-t))\bar{V}_{t+1}\left(S_t-\bar{n}_t^*,\, \left\lceil \lceil Y_t\rceil_\Delta+(X_t-\bar{n}_t^*\delta-\lceil Y_t\rceil_\Delta)_+\right\rceil_\Delta,\, \lceil X_t\rceil_\varepsilon\right)+3\Delta(T-t+1)\Bigg]
\end{aligned}
$$

with probability $\geq 1-\frac{T-t}{2T^3S_1^3}$. Let

$$
\begin{aligned}
F \;=\; & \mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\Bigg[(X_t-\bar{n}_t^*\delta-\lceil Y_t\rceil_\Delta)_+ + g_t(\bar{n}_t^*) \\
& +\bar{V}_{t+1}\left(S_t-\bar{n}_t^*,\, \left\lceil \lceil Y_t\rceil_\Delta+(X_t-\bar{n}_t^*\delta-\lceil Y_t\rceil_\Delta)_+\right\rceil_\Delta,\, \lceil X_t\rceil_\varepsilon\right)\Bigg]
\end{aligned}
$$

Also, we know from definition that

$$
\begin{aligned}
& \bar{V}_t\left(S_t, \lceil Y_t\rceil_\Delta,\, \lceil X_{t-1}\rceil_\varepsilon\right) \\
= \;& \frac{1}{N_t}\sum_{i=1}^{N_t}\Bigg[\left(X_t^{(i)}-n_t^*\delta-\lceil Y_t\rceil_\Delta\right)_+ + g_t(n_t^*) \\
& +\bar{V}_{t+1}\left(S_t-n_t,\, \left\lceil \lceil Y_t\rceil_\Delta+\left(X_t^{(i)}-n_t\delta-\lceil Y_t\rceil_\Delta\right)_+\right\rceil_\Delta,\, \left\lceil X_t^{(i)}\right\rceil_\varepsilon\right)\Bigg]
\end{aligned}
$$

By Chernoff bound,

$$Pr\left(\left|\frac{\bar{V}_t(S_t,\lceil Y_t\rceil_\Delta,\ \lceil X_{t-1}\rceil_\varepsilon)}{M}-\frac{F}{M}\right|\geq\frac{\varepsilon}{T}\right)\leq 2e^{-\varepsilon^2 N_t}\leq\frac{1}{4T^3S_1^3}$$

Therefore, for any S_t, Y_t and X_{t-1},

$$F-\frac{\varepsilon M}{T}\leq\bar{V}_t(S_t,\lceil Y_t\rceil_\Delta,\ \lceil X_{t-1}\rceil_\varepsilon)\leq F+\frac{\varepsilon M}{T}$$

with probability $\geq 1-\frac{1}{2T^3S_1^3}$. Recall that $\Delta=\frac{\varepsilon M}{T}$. We have for any, Y_t and X_{t-1},

$$\bar{U}_t(S_t,\ Y_t,\ X_{t-1})\leq(1+2\varepsilon(T-t+1))\bar{V}_t(S_t,\ \lceil Y_t\rceil_\Delta,\ \lceil X_{t-1}\rceil_\varepsilon)+3(T-t+1)\Delta$$

with probability $\geq 1-\frac{T-t}{2T^3S_1^3}-\frac{1}{4T^3S_1^3}\geq 1-\frac{T-t+1}{2T^3S_1^3}\geq 1-\frac{1}{T^2S_1^3}$.

We can prove the other direction analogously. At time $T+1$, $\bar{U}_{T+1}(S_{T+1},\ Y_{T+1},\ X_T)=\bar{V}_{T+1}(S_{T+1},\ \lceil Y_{T+1}\rceil_\Delta,\ \lceil X_T\rceil$

0. Suppose the statement holds for t+1. For any S_t, Y_t,

$$
\begin{aligned}
& \bar{U}_t(S_t,\, Y_t,\, X_{t-1}) \\
= \;& \mathbb{E}_{X_t|X_{t-1}}\left[(X_t - \bar{n}_t^*\delta - Y_t)_+ + g_t(\bar{n}_t^*) + \bar{U}_{t+1}\left(S_t - \bar{n}_t^*,\, Y_t + (X_t - \bar{n}_t^*\delta - Y_t)_+,\, X_t\right)\right] \\
\geq \;& \mathbb{E}_{X_t|X_{t-1}}\Bigg[(X_t - \bar{n}_t^*\delta - \lceil Y_t\rceil_\Delta)_+ + g_t(\bar{n}_t^*) \\
& + (1-2\varepsilon(T-t))\bar{V}_{t+1}\left(S_t - \bar{n}_t^*,\, \left\lceil Y_t + (X_t - \bar{n}_t^*\delta - Y_t)_+\right\rceil_\Delta,\, \lceil X_t\rceil_\varepsilon\right) - 3\Delta(T-t)\Bigg] \\
\geq \;& (1-\varepsilon)\mathbb{E}_{X_t|\lceil X_{t-1}\rceil_\varepsilon}\Bigg[(X_t - \bar{n}_t^*\delta - \lceil Y_t\rceil_\Delta)_+ + g_t(\bar{n}_t^*) \\
& + (1-2\varepsilon(T-t))\bar{V}_{t+1}\left(S_t - \bar{n}_t^*,\, \left\lceil \lceil Y_t\rceil_\Delta + (X_t - \bar{n}_t^*\delta - \lceil Y_t\rceil_\Delta)_+\right\rceil_\Delta\right) - 3\Delta(T-t)\Bigg] \\
\geq \;& (1-2\varepsilon(T-t+1))\bar{V}_t(S_t,\, \lceil Y_t\rceil_\Delta,\, \lceil X_{t-1}\rceil_\varepsilon) - 3\Delta(T-t+1)
\end{aligned}
$$

with probability $\geq 1 - \frac{T-t+1}{2T^3S_1^3} \geq 1 - \frac{1}{T^2S_1^3}$. Therefore, for any S_t and Y_t,

$$|\bar{U}_t(S_t,\, Y_t,\, X_{t-1}) - \bar{V}_t(S_t,\, \lceil Y_t\rceil_\Delta,\, \lceil X_{t-1}\rceil_\varepsilon)| \leq 2\varepsilon(T-t+1)\bar{V}_t(S_t,\, Y_t,\, X_{t-1}) + 3\Delta(T-t+1)$$

with probability $\geq 1 - \frac{1}{T^2S_1^2}$ □

We are now ready to prove the main theorem.

Proof. Theorem 1.4.1 At time $t = 1$, $Y_1 = 0$. From Lemma 2, we know that for

$$\bar{V}_1(S_1, 0) \leq \hat{V}_1(S_1, 0) + 2T\Delta$$

with probability at least $1 - 1/(TS_1)$ for all S_1.

From Lemmas A.0.1 - A.0.3, we have

$$\bar{U}_1(S_1,0) \le (1+2\varepsilon T)\bar{V}_1(S_1,0)+3T\Delta \le (1+2\varepsilon T)\hat{V}_1(S_1,0)+6T\Delta \le (1+5\varepsilon T)V_1(S_1,0)+8T\Delta,$$

$$\bar{U}_1(S_1,0) \ge (1-2\varepsilon T)\bar{V}_1(S_1,0)-3T\Delta \ge (1-2\varepsilon T)\hat{V}_1(S_1,0)-6T\Delta \ge (1-5\varepsilon T)V_1(S_1,0)-8T\Delta,$$

with probability at least $1-1/(TS_1)-1/(TS_1)=1-2/(TS_1)$ for all S_1.

Let $V_1^0(.)$ be the value function when there is no execution cost, i.e. $p=0$. We have $V_1(S_1,0) \ge V_1^0(S_1,0)$. From Theorem 1.2.1 and Assumptions 2 and 3, the expected peak load is at least $cM/2$ and the average amount of reduction $S_1\delta/T$ is at most aM.

$$\hat{V}_1(S_1,0) \ge V_1^0(S_1,0) \ge \mathbb{E}\left[X_{max}\right] - \frac{S_1}{T}\delta \ge c_1 M$$

for some constant c_1. Therefore, with probability at least $1-2/(TS_1)-\eta$

$$\bar{U}_1(S_1,0) \le \left(1+5\varepsilon T+\frac{16e}{(e-1)(1-a)}\varepsilon\right)V_t(S_1,0) \le (1+O(\varepsilon))V_t(S_1,0),$$

$$\bar{U}_1(S_1,0) \ge \left(1-5\varepsilon T-\frac{16e}{(e-1)(1-a)}\varepsilon\right)V_t(S_1,0) \ge (1-O(\varepsilon))V_t(S_1,0).$$

□

Appendix B

Proof of Lemma 4 with general network size and input/output capacities

Suppose that $\tau_{u-1} < \bar{C_k} \leq \tau_u$ for some u. We consider the following three cases.

Case (1): $\bar{C_k} < \frac{5\tau_{u-1}}{4}$.

For any $g \in \{1, \cdots, k\}$, we have

$$\begin{aligned}\frac{5\tau_{u-1}}{4} > \bar{C_k} \geq \bar{C_g} &= \sum_{l=1}^{L} \tau_{l-1} \bar{x_l^{(g)}} \geq \tau_u \sum_{l=u+1}^{L} \bar{x_l^{(g)}} \\ &= \tau_u \left(1 - \sum_{l=1}^{u} \bar{x_l^{(g)}}\right) = 2\tau_{u-1} \left(1 - \sum_{l=1}^{u} \bar{x_l^{(g)}}\right).\end{aligned}$$

Thus, $\sum_{l=1}^{u} \bar{x_l^{(g)}} > \frac{3}{8}$.

Let $g^* = \text{argmin}_{1\le g\le k}\sum_{l=1}^{u}\bar{x}_l^{(g)}$. We know from Eq. (2.2.11) that

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

We then have

$$\begin{aligned}
V_k &= \max\left\{\max_i\left\{\left(\sum_{j'=1}^{m}\sum_{g=1}^{k}\frac{d_{ij'}^{(g)}}{\lambda_i}\right)\left(\sum_{l=1}^{u}\bar{x}_l^{(g^*)}\right)\right\},\right.\\
&\qquad\left.\max_j\left\{\left(\sum_{i'=1}^{m}\sum_{g=1}^{k}\frac{d_{i'j}^{(g)}}{\mu_j}\right)\left(\sum_{l=1}^{u}\bar{x}_l^{(g^*)}\right)\right\}\right\}\Bigg/\sum_{l=1}^{u}\bar{x}_l^{(g^*)}\\
&\le \max\left\{\max_i\left\{\sum_{j'=1}^{m}\sum_{g=1}^{k}\left(\frac{d_{ij'}^{(g)}}{\lambda_i}\sum_{l=1}^{u}\bar{x}_l^{(g)}\right)\right\},\right.\\
&\qquad\left.\max_j\left\{\sum_{i'=1}^{m}\sum_{g=1}^{k}\left(\frac{d_{i'j}^{(g)}}{\mu_j}\sum_{l=1}^{u}\bar{x}_l^{(g)}\right)\right\}\right\}\Bigg/\sum_{l=1}^{u}\bar{x}_l^{(g^*)}\\
&\le \max\left\{\max_i\left\{\sum_{l=1}^{u}\sum_{j'=1}^{m}\sum_{g=1}^{n}\frac{d_{ij'}^{(g)}\bar{x}_l^{(g)}}{\lambda_i}\right\},\right.\\
&\qquad\left.\max_j\left\{\sum_{l=1}^{u}\sum_{i'=1}^{m}\sum_{g=1}^{n}\frac{d_{i'j}^{(g)}\bar{x}_l^{(g)}}{\mu_j}\right\}\right\}\Bigg/\sum_{l=1}^{u}\bar{x}_l^{(g^*)}.
\end{aligned}$$

Using the constraints (2.2.1) and (2.2.2), we have

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

Case (2): $\frac{5\tau_{u-1}}{4} \leq \bar{C_k} < \frac{3\tau_{u-1}}{2}$.

Define a sequence $\alpha_h, h = 1, 2, \cdots$, by

$$\alpha_1 = \frac{1}{4},$$
$$\alpha_h = 1 - \sum_{q=1}^{h-1} \alpha_q - \frac{1-\alpha_1}{1+\sum_{q=1}^{h-1}\alpha_q}, h = 2, 3, \cdots.$$

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

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

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

$$\left(1 + \sum_{q=1}^{h} \alpha_q\right) \tau_{u-1} \geq \bar{C_k} \geq \bar{C_g} = \sum_{l=1}^{L} \tau_{l-1} \bar{x_l^{(g)}}$$
$$> \tau_u \sum_{l=u+1}^{L} \bar{x_l^{(g)}} = \tau_u \left(1 - \sum_{l=1}^{u} \bar{x_l^{(g)}}\right),$$

and

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

It follows that,

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

By the definition of α_q, we have

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

Therefore,

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

Case (3): $\bar{C}_k \geq \frac{3\tau_{u-1}}{2}$.

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

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

and hence

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

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

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

We know from Lemma 2 that $\bar{C}_k \leq C_k(OPT)$ and the result follows.

Appendix C

Tables

We present the total weighted completion time ratios with respect to the base cases for general instances in Tables C.1 to C.11.

Table C.1: General instances with zero release time, (a) without backfill and without grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	2.33	2.22	2.06	2.12	2.15	2.26
2	m	2.49	2.39	2.18	2.29	2.38	2.40
3	m	2.43	2.29	2.15	2.24	2.29	2.36
4	m	2.41	2.23	2.11	2.21	2.22	2.28
5	m	2.47	2.24	2.09	2.19	2.19	2.21
6	m^2	1.28	1.25	1.24	1.25	1.26	1.26
7	m^2	1.26	1.25	1.23	1.24	1.26	1.26
8	m^2	1.29	1.26	1.24	1.24	1.26	1.26
9	m^2	1.27	1.27	1.24	1.25	1.27	1.27
10	m^2	1.27	1.26	1.23	1.24	1.26	1.26
11	Unif[m, m^2]	1.91	1.61	1.60	1.60	1.61	1.61
12	Unif[m, m^2]	1.93	1.64	1.63	1.63	1.65	1.66
13	Unif[m, m^2]	2.04	1.62	1.61	1.62	1.63	1.63
14	Unif[m, m^2]	1.98	1.56	1.55	1.56	1.57	1.57
15	Unif[m, m^2]	1.88	1.58	1.56	1.57	1.59	1.59
16	Unif[m, m^2]	2.05	1.57	1.56	1.57	1.58	1.58
17	Unif[m, m^2]	1.97	1.58	1.57	1.58	1.59	1.59
18	Unif[m, m^2]	2.03	1.65	1.64	1.65	1.66	1.66
19	Unif[m, m^2]	2.04	1.57	1.56	1.57	1.58	1.58
20	Unif[m, m^2]	2.12	1.66	1.65	1.66	1.68	1.67
21	Unif[m, m^2]	1.94	1.66	1.64	1.64	1.66	1.67
22	Unif[m, m^2]	2.08	1.64	1.63	1.63	1.65	1.65
23	Unif[m, m^2]	1.98	1.60	1.59	1.60	1.61	1.61
24	Unif[m, m^2]	2.14	1.69	1.67	1.68	1.70	1.69
25	Unif[m, m^2]	2.02	1.65	1.64	1.64	1.66	1.67
26	Unif[m, m^2]	2.17	1.68	1.67	1.68	1.70	1.70
27	Unif[m, m^2]	1.86	1.59	1.58	1.58	1.61	1.61
28	Unif[m, m^2]	1.90	1.61	1.59	1.60	1.62	1.62
29	Unif[m, m^2]	2.22	1.72	1.71	1.71	1.74	1.73
30	Unif[m, m^2]	1.97	1.59	1.58	1.58	1.60	1.60

Table C.2: General instances with zero release time, (b) with backfill and without grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	1.37	1.36	1.35	1.43	1.33	1.36
2	m	1.58	1.40	1.50	1.53	1.52	1.41
3	m	1.50	1.36	1.41	1.44	1.41	1.45
4	m	1.56	1.41	1.37	1.35	1.33	1.48
5	m	1.59	1.44	1.37	1.43	1.38	1.48
6	m^2	1.04	1.01	1.02	1.02	1.01	1.01
7	m^2	1.05	1.03	1.03	1.02	1.01	1.01
8	m^2	1.04	1.02	1.02	1.02	1.01	1.01
9	m^2	1.03	1.02	1.02	1.02	1.01	1.01
10	m^2	1.03	1.03	1.03	1.03	1.01	1.01
11	Unif[m, m^2]	1.35	1.06	1.06	1.05	1.06	1.05
12	Unif[m, m^2]	1.35	1.05	1.06	1.06	1.05	1.06
13	Unif[m, m^2]	1.45	1.05	1.06	1.06	1.06	1.04
14	Unif[m, m^2]	1.42	1.05	1.05	1.06	1.05	1.05
15	Unif[m, m^2]	1.33	1.05	1.05	1.06	1.05	1.05
16	Unif[m, m^2]	1.48	1.05	1.06	1.06	1.06	1.05
17	Unif[m, m^2]	1.42	1.04	1.05	1.05	1.04	1.05
18	Unif[m, m^2]	1.42	1.06	1.06	1.08	1.06	1.07
19	Unif[m, m^2]	1.47	1.04	1.05	1.05	1.05	1.04
20	Unif[m, m^2]	1.45	1.06	1.07	1.05	1.06	1.06
21	Unif[m, m^2]	1.33	1.06	1.08	1.07	1.05	1.06
22	Unif[m, m^2]	1.47	1.06	1.06	1.06	1.07	1.06
23	Unif[m, m^2]	1.43	1.05	1.07	1.06	1.05	1.05
24	Unif[m, m^2]	1.47	1.06	1.07	1.07	1.05	1.06
25	Unif[m, m^2]	1.44	1.05	1.08	1.06	1.06	1.06
26	Unif[m, m^2]	1.51	1.07	1.08	1.09	1.08	1.08
27	Unif[m, m^2]	1.30	1.05	1.07	1.06	1.05	1.05
28	Unif[m, m^2]	1.46	1.08	1.38	1.38	1.45	1.07
29	Unif[m, m^2]	1.29	1.06	1.24	1.24	1.28	1.05
30	Unif[m, m^2]	1.46	1.05	1.37	1.38	1.45	1.05

Table C.3: General instances with zero release time, (c) with balanced backfill and without grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	1.04	1.00	1.03	0.98	0.89	1.00
2	m	1.09	1.04	1.03	1.03	0.89	1.00
3	m	1.08	1.00	1.02	1.03	0.92	1.00
4	m	1.11	1.01	1.04	0.98	0.91	1.00
5	m	1.10	1.03	0.97	1.00	0.89	1.00
6	m^2	1.03	1.01	1.01	1.02	1.00	1.00
7	m^2	1.05	1.02	1.03	1.02	1.00	1.00
8	m^2	1.04	1.01	1.02	1.01	1.00	1.00
9	m^2	1.02	1.01	1.02	1.02	1.00	1.00
10	m^2	1.03	1.03	1.03	1.02	1.00	1.00
11	Unif[m, m^2]	1.31	1.01	1.01	1.02	1.00	1.00
12	Unif[m, m^2]	1.29	1.01	1.03	1.02	0.99	1.00
13	Unif[m, m^2]	1.42	1.01	1.02	1.02	0.99	1.00
14	Unif[m, m^2]	1.39	1.01	1.02	1.02	1.00	1.00
15	Unif[m, m^2]	1.30	1.01	1.03	1.02	1.00	1.00
16	Unif[m, m^2]	1.44	1.01	1.02	1.01	0.99	1.00
17	Unif[m, m^2]	1.40	1.01	1.02	1.01	0.99	1.00
18	Unif[m, m^2]	1.37	1.01	1.02	1.02	1.00	1.00
19	Unif[m, m^2]	1.42	1.01	1.02	1.01	1.00	1.00
20	Unif[m, m^2]	1.41	1.01	1.02	1.02	1.00	1.00
21	Unif[m, m^2]	1.28	1.00	1.02	1.02	1.00	1.00
22	Unif[m, m^2]	1.42	1.01	1.02	1.01	1.00	1.00
23	Unif[m, m^2]	1.38	1.01	1.02	1.02	1.00	1.00
24	Unif[m, m^2]	1.42	1.00	1.02	1.02	0.99	1.00
25	Unif[m, m^2]	1.40	1.00	1.02	1.02	0.99	1.00
26	Unif[m, m^2]	1.44	1.01	1.02	1.02	1.00	1.00
27	Unif[m, m^2]	1.26	1.02	1.03	1.02	1.00	1.00
28	Unif[m, m^2]	1.30	1.03	1.03	1.02	1.00	1.00
29	Unif[m, m^2]	1.48	1.01	1.02	1.02	0.99	1.00
30	Unif[m, m^2]	1.41	1.01	1.01	1.01	1.00	1.00

Table C.4: General instances with zero release time, (d) with backfill and with grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	1.25	1.21	1.21	1.19	1.11	1.08
2	m	1.26	1.14	1.22	1.20	1.14	1.04
3	m	1.18	1.10	1.18	1.20	1.14	1.14
4	m	1.31	1.19	1.33	1.20	1.11	1.07
5	m	1.30	1.18	1.13	1.19	1.08	1.05
6	m^2	1.37	1.36	1.38	1.38	1.34	1.36
7	m^2	1.37	1.35	1.37	1.35	1.35	1.34
8	m^2	1.39	1.36	1.35	1.37	1.35	1.33
9	m^2	1.39	1.37	1.36	1.36	1.35	1.35
10	m^2	1.36	1.37	1.38	1.36	1.34	1.35
11	Unif[m, m^2]	1.73	1.41	1.39	1.41	1.37	1.38
12	Unif[m, m^2]	1.75	1.44	1.43	1.43	1.42	1.45
13	Unif[m, m^2]	1.86	1.40	1.41	1.43	1.38	1.34
14	Unif[m, m^2]	1.87	1.39	1.41	1.40	1.40	1.37
15	Unif[m, m^2]	1.68	1.43	1.43	1.41	1.35	1.37
16	Unif[m, m^2]	1.88	1.36	1.39	1.40	1.38	1.38
17	Unif[m, m^2]	1.82	1.37	1.38	1.41	1.37	1.37
18	Unif[m, m^2]	1.88	1.41	1.43	1.42	1.39	1.42
19	Unif[m, m^2]	1.87	1.41	1.41	1.41	1.40	1.41
20	Unif[m, m^2]	1.92	1.42	1.44	1.42	1.41	1.39
21	Unif[m, m^2]	1.81	1.43	1.45	1.47	1.40	1.40
22	Unif[m, m^2]	1.91	1.41	1.43	1.39	1.41	1.42
23	Unif[m, m^2]	1.80	1.42	1.42	1.39	1.39	1.37
24	Unif[m, m^2]	1.89	1.40	1.44	1.44	1.38	1.40
25	Unif[m, m^2]	1.82	1.42	1.43	1.42	1.41	1.42
26	Unif[m, m^2]	2.00	1.50	1.49	1.48	1.46	1.44
27	Unif[m, m^2]	1.69	1.42	1.44	1.41	1.39	1.42
28	Unif[m, m^2]	1.71	1.44	1.43	1.44	1.38	1.39
29	Unif[m, m^2]	2.08	1.49	1.46	1.48	1.46	1.43
30	Unif[m, m^2]	1.79	1.44	1.42	1.43	1.38	1.41

Table C.5: General instances with zero release time, (e) with balanced backfill and with grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	1.18	1.14	1.15	1.13	1.07	1.04
2	m	1.19	1.09	1.16	1.15	1.10	1.00
3	m	1.13	1.05	1.12	1.16	1.10	1.09
4	m	1.23	1.14	1.21	1.15	1.07	1.05
5	m	1.23	1.12	1.08	1.13	1.05	1.01
6	m^2	1.36	1.35	1.35	1.35	1.33	1.34
7	m^2	1.35	1.34	1.36	1.35	1.33	1.32
8	m^2	1.38	1.36	1.35	1.36	1.34	1.33
9	m^2	1.37	1.35	1.35	1.35	1.33	1.33
10	m^2	1.35	1.36	1.36	1.36	1.33	1.34
11	Unif[m, m^2]	1.69	1.36	1.35	1.37	1.35	1.34
12	Unif[m, m^2]	1.70	1.38	1.39	1.39	1.37	1.36
13	Unif[m, m^2]	1.83	1.34	1.36	1.37	1.35	1.32
14	Unif[m, m^2]	1.83	1.35	1.37	1.36	1.36	1.34
15	Unif[m, m^2]	1.65	1.35	1.37	1.37	1.33	1.33
16	Unif[m, m^2]	1.86	1.32	1.35	1.34	1.33	1.33
17	Unif[m, m^2]	1.77	1.34	1.35	1.36	1.33	1.33
18	Unif[m, m^2]	1.84	1.36	1.38	1.36	1.35	1.37
19	Unif[m, m^2]	1.85	1.36	1.35	1.36	1.35	1.36
20	Unif[m, m^2]	1.87	1.38	1.42	1.40	1.39	1.36
21	Unif[m, m^2]	1.77	1.38	1.40	1.41	1.36	1.36
22	Unif[m, m^2]	1.88	1.36	1.37	1.35	1.36	1.37
23	Unif[m, m^2]	1.78	1.37	1.37	1.36	1.35	1.34
24	Unif[m, m^2]	1.85	1.35	1.38	1.39	1.35	1.37
25	Unif[m, m^2]	1.79	1.39	1.39	1.40	1.38	1.37
26	Unif[m, m^2]	1.98	1.44	1.44	1.45	1.43	1.40
27	Unif[m, m^2]	1.67	1.37	1.38	1.36	1.37	1.35
28	Unif[m, m^2]	1.69	1.40	1.39	1.39	1.37	1.35
29	Unif[m, m^2]	2.04	1.43	1.41	1.43	1.41	1.39
30	Unif[m, m^2]	1.78	1.38	1.37	1.38	1.34	1.36

Table C.6: General instances with general release times, (b) with backfill and without grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	1.33	1.28	1.32	1.32	1.33	1.30
2	m	1.32	1.34	1.33	1.32	1.32	1.33
3	m	1.37	1.38	1.37	1.36	1.37	1.34
4	m	1.28	1.27	1.24	1.31	1.30	1.29
5	m	1.31	1.26	1.34	1.32	1.37	1.33
6	m^2	1.04	1.03	1.04	1.04	1.03	1.01
7	m^2	1.03	1.02	1.03	1.03	1.03	1.01
8	m^2	1.03	1.02	1.03	1.03	1.03	1.01
9	m^2	1.02	1.02	1.02	1.02	1.02	1.01
10	m^2	1.03	1.03	1.03	1.03	1.03	1.01
11	Unif[m, m^2]	1.44	1.07	1.36	1.36	1.44	1.06
12	Unif[m, m^2]	1.45	1.08	1.38	1.36	1.44	1.05
13	Unif[m, m^2]	1.37	1.06	1.29	1.29	1.32	1.04
14	Unif[m, m^2]	1.43	1.07	1.38	1.37	1.43	1.07
15	Unif[m, m^2]	1.38	1.07	1.31	1.32	1.37	1.05
16	Unif[m, m^2]	1.42	1.05	1.35	1.35	1.41	1.05
17	Unif[m, m^2]	1.37	1.07	1.29	1.29	1.36	1.04
18	Unif[m, m^2]	1.33	1.05	1.25	1.26	1.31	1.04
19	Unif[m, m^2]	1.25	1.04	1.21	1.21	1.24	1.03
20	Unif[m, m^2]	1.43	1.06	1.36	1.36	1.44	1.06
21	Unif[m, m^2]	1.38	1.04	1.30	1.30	1.38	1.04
22	Unif[m, m^2]	1.40	1.06	1.30	1.31	1.38	1.05
23	Unif[m, m^2]	1.35	1.05	1.30	1.29	1.34	1.05
24	Unif[m, m^2]	1.36	1.04	1.28	1.28	1.35	1.04
25	Unif[m, m^2]	1.45	1.06	1.36	1.37	1.40	1.06
26	Unif[m, m^2]	1.45	1.06	1.35	1.37	1.42	1.04
27	Unif[m, m^2]	1.38	1.08	1.29	1.30	1.36	1.06
28	Unif[m, m^2]	1.39	1.07	1.31	1.31	1.38	1.05
29	Unif[m, m^2]	1.42	1.07	1.38	1.37	1.42	1.05
30	Unif[m, m^2]	1.38	1.05	1.32	1.31	1.37	1.05

Table C.7: General instances with general release times, (c) with balanced backfill and without grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	1.03	1.10	1.07	1.06	1.03	1.03
2	m	1.03	1.07	1.07	1.04	1.03	1.06
3	m	1.09	1.08	1.09	1.09	1.09	1.07
4	m	1.03	1.02	1.05	1.01	1.03	0.99
5	m	1.02	1.07	1.09	1.05	1.06	1.05
6	m^2	1.34	1.34	1.34	1.34	1.33	1.32
7	m^2	1.33	1.31	1.33	1.32	1.32	1.33
8	m^2	1.32	1.33	1.32	1.32	1.32	1.32
9	m^2	1.32	1.31	1.32	1.32	1.32	1.32
10	m^2	1.31	1.33	1.31	1.32	1.31	1.32
11	Unif[m, m^2]	1.75	1.34	1.68	1.68	1.76	1.31
12	Unif[m, m^2]	1.73	1.36	1.68	1.67	1.76	1.36
13	Unif[m, m^2]	1.70	1.36	1.61	1.60	1.59	1.31
14	Unif[m, m^2]	1.79	1.33	1.69	1.69	1.74	1.33
15	Unif[m, m^2]	1.68	1.36	1.60	1.60	1.67	1.33
16	Unif[m, m^2]	1.75	1.40	1.65	1.69	1.73	1.36
17	Unif[m, m^2]	1.67	1.35	1.58	1.58	1.66	1.35
18	Unif[m, m^2]	1.63	1.34	1.58	1.59	1.64	1.34
19	Unif[m, m^2]	1.53	1.35	1.50	1.49	1.53	1.33
20	Unif[m, m^2]	1.75	1.38	1.65	1.65	1.74	1.35
21	Unif[m, m^2]	1.73	1.33	1.61	1.59	1.66	1.30
22	Unif[m, m^2]	1.68	1.34	1.62	1.61	1.68	1.34
23	Unif[m, m^2]	1.66	1.35	1.61	1.61	1.66	1.36
24	Unif[m, m^2]	1.64	1.34	1.58	1.58	1.64	1.29
25	Unif[m, m^2]	1.75	1.36	1.64	1.65	1.66	1.33
26	Unif[m, m^2]	1.73	1.38	1.66	1.65	1.70	1.32
27	Unif[m, m^2]	1.71	1.36	1.60	1.59	1.66	1.34
28	Unif[m, m^2]	1.69	1.36	1.59	1.60	1.69	1.35
29	Unif[m, m^2]	1.77	1.35	1.70	1.71	1.75	1.33
30	Unif[m, m^2]	1.68	1.35	1.62	1.60	1.68	1.33

Table C.8: General instances with general release times, (d) with backfill and with grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	1.03	1.10	1.07	1.06	1.03	1.03
2	m	1.03	1.07	1.07	1.04	1.03	1.06
3	m	1.09	1.08	1.09	1.09	1.09	1.07
4	m	1.03	1.02	1.05	1.01	1.03	0.99
5	m	1.02	1.07	1.09	1.05	1.06	1.05
6	m^2	1.34	1.34	1.34	1.34	1.33	1.32
7	m^2	1.33	1.31	1.33	1.32	1.32	1.33
8	m^2	1.32	1.33	1.32	1.32	1.32	1.32
9	m^2	1.32	1.31	1.32	1.32	1.32	1.32
10	m^2	1.31	1.33	1.31	1.32	1.31	1.32
11	Unif[m, m^2]	1.75	1.34	1.68	1.68	1.76	1.31
12	Unif[m, m^2]	1.73	1.36	1.68	1.67	1.76	1.36
13	Unif[m, m^2]	1.70	1.36	1.61	1.60	1.59	1.31
14	Unif[m, m^2]	1.79	1.33	1.69	1.69	1.74	1.33
15	Unif[m, m^2]	1.68	1.36	1.60	1.60	1.67	1.33
16	Unif[m, m^2]	1.75	1.40	1.65	1.69	1.73	1.36
17	Unif[m, m^2]	1.67	1.35	1.58	1.58	1.66	1.35
18	Unif[m, m^2]	1.63	1.34	1.58	1.59	1.64	1.34
19	Unif[m, m^2]	1.53	1.35	1.50	1.49	1.53	1.33
20	Unif[m, m^2]	1.75	1.38	1.65	1.65	1.74	1.35
21	Unif[m, m^2]	1.73	1.33	1.61	1.59	1.66	1.30
22	Unif[m, m^2]	1.68	1.34	1.62	1.61	1.68	1.34
23	Unif[m, m^2]	1.66	1.35	1.61	1.61	1.66	1.36
24	Unif[m, m^2]	1.64	1.34	1.58	1.58	1.64	1.29
25	Unif[m, m^2]	1.75	1.36	1.64	1.65	1.66	1.33
26	Unif[m, m^2]	1.73	1.38	1.66	1.65	1.70	1.32
27	Unif[m, m^2]	1.71	1.36	1.60	1.59	1.66	1.34
28	Unif[m, m^2]	1.69	1.36	1.59	1.60	1.69	1.35
29	Unif[m, m^2]	1.77	1.35	1.70	1.71	1.75	1.33
30	Unif[m, m^2]	1.68	1.35	1.62	1.60	1.68	1.33

Table C.9: General instances with general release times, (e) with balanced backfill and with grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	0.97	0.98	0.96	0.96	0.97	0.95
2	m	0.95	0.96	0.97	0.96	0.95	0.98
3	m	1.02	1.02	1.02	1.02	1.02	0.97
4	m	0.92	0.91	0.91	0.91	0.92	0.91
5	m	0.97	0.98	0.98	0.97	0.98	0.96
6	m^2	1.34	1.33	1.34	1.34	1.33	1.31
7	m^2	1.33	1.31	1.33	1.32	1.32	1.33
8	m^2	1.32	1.32	1.31	1.32	1.32	1.32
9	m^2	1.31	1.30	1.31	1.31	1.31	1.31
10	m^2	1.31	1.32	1.32	1.32	1.31	1.31
11	Unif[m, m^2]	1.72	1.32	1.66	1.66	1.74	1.29
12	Unif[m, m^2]	1.74	1.35	1.66	1.64	1.74	1.32
13	Unif[m, m^2]	1.64	1.35	1.60	1.60	1.58	1.32
14	Unif[m, m^2]	1.76	1.32	1.67	1.67	1.72	1.30
15	Unif[m, m^2]	1.68	1.34	1.60	1.60	1.67	1.31
16	Unif[m, m^2]	1.74	1.37	1.64	1.67	1.72	1.31
17	Unif[m, m^2]	1.65	1.32	1.58	1.58	1.65	1.32
18	Unif[m, m^2]	1.63	1.31	1.53	1.53	1.64	1.32
19	Unif[m, m^2]	1.52	1.35	1.49	1.49	1.52	1.31
20	Unif[m, m^2]	1.74	1.34	1.65	1.65	1.74	1.30
21	Unif[m, m^2]	1.72	1.31	1.60	1.59	1.65	1.29
22	Unif[m, m^2]	1.66	1.31	1.60	1.60	1.66	1.30
23	Unif[m, m^2]	1.67	1.33	1.61	1.60	1.67	1.32
24	Unif[m, m^2]	1.63	1.32	1.57	1.57	1.60	1.28
25	Unif[m, m^2]	1.73	1.35	1.63	1.63	1.64	1.31
26	Unif[m, m^2]	1.71	1.34	1.65	1.64	1.69	1.31
27	Unif[m, m^2]	1.68	1.32	1.58	1.58	1.65	1.31
28	Unif[m, m^2]	1.66	1.33	1.55	1.56	1.64	1.33
29	Unif[m, m^2]	1.78	1.34	1.70	1.69	1.75	1.30
30	Unif[m, m^2]	1.68	1.34	1.61	1.60	1.67	1.31

Table C.10: Offline algorithm on general instances with release times, (c) with balanced backfill and without grouping

Instance	No. of flows in each coflow	FIFO	STPT	SMPT	SMCT	ECT	LP-based
1	m	0.99	0.98	0.98	0.98	1.00	1.00
2	m	0.99	0.99	0.99	0.99	0.99	1.00
3	m	1.01	0.99	1.00	0.99	1.01	1.00
4	m	0.97	0.97	0.97	0.98	0.97	1.00
5	m	0.98	0.97	0.97	0.98	0.98	1.00
6	m^2	1.02	1.02	1.02	1.02	1.02	1.00
7	m^2	1.02	1.02	1.02	1.02	1.02	1.00
8	m^2	1.04	1.02	1.03	1.03	1.03	1.00
9	m^2	1.02	1.02	1.02	1.02	1.03	1.00
10	m^2	1.04	1.03	1.04	1.04	1.04	1.00
11	Unif[m, m^2]	1.40	1.01	1.33	1.33	1.39	1.00
12	Unif[m, m^2]	1.32	1.01	1.25	1.25	1.31	1.00
13	Unif[m, m^2]	1.39	1.01	1.31	1.31	1.38	1.00
14	Unif[m, m^2]	1.35	1.02	1.29	1.29	1.34	1.00
15	Unif[m, m^2]	1.40	1.02	1.33	1.33	1.40	1.00
16	Unif[m, m^2]	1.27	1.01	1.20	1.20	1.26	1.00
17	Unif[m, m^2]	1.42	1.01	1.33	1.33	1.40	1.00
18	Unif[m, m^2]	1.33	1.01	1.27	1.27	1.33	1.00
19	Unif[m, m^2]	1.42	1.02	1.34	1.34	1.41	1.00
20	Unif[m, m^2]	1.40	1.02	1.34	1.34	1.40	1.00
21	Unif[m, m^2]	1.35	1.01	1.27	1.27	1.34	1.00
22	Unif[m, m^2]	1.30	1.03	1.25	1.25	1.30	1.00
23	Unif[m, m^2]	1.35	1.02	1.29	1.28	1.35	1.00
24	Unif[m, m^2]	1.34	1.02	1.28	1.28	1.33	1.00
25	Unif[m, m^2]	1.30	1.01	1.24	1.24	1.29	1.00
26	Unif[m, m^2]	1.30	1.02	1.24	1.24	1.29	1.00
27	Unif[m, m^2]	1.28	1.01	1.22	1.22	1.27	1.00
28	Unif[m, m^2]	1.35	1.01	1.27	1.27	1.34	1.00
29	Unif[m, m^2]	1.32	1.02	1.25	1.25	1.31	1.00
30	Unif[m, m^2]	1.28	1.01	1.23	1.23	1.28	1.00

Table C.11: Online algorithm on general instances with release times, (c) with balanced backfill and without grouping

Instance	No. of flows	FIFO	STPT	SMPT	SMCT	ECT	LP-based	Lower bound
1	m	0.99	0.95	0.94	0.97	0.93	0.96	0.88
2	m	0.99	0.98	0.97	0.97	0.94	0.96	0.89
3	m	1.01	0.98	0.96	0.97	0.93	0.97	0.88
4	m	0.97	0.96	0.95	0.96	0.92	0.94	0.88
5	m	0.98	0.95	0.94	0.94	0.91	0.93	0.87
6	m^2	1.02	1.00	1.01	1.01	0.99	0.99	0.94
7	m^2	1.02	1.01	1.02	1.02	1.00	1.00	0.94
8	m^2	1.04	1.01	1.02	1.01	1.00	1.00	0.94
9	m^2	1.02	1.01	1.01	1.01	0.99	0.99	0.94
10	m^2	1.04	1.01	1.03	1.02	1.00	1.00	0.96
11	Unif[m, m^2]	1.40	1.00	1.00	1.00	0.99	0.99	0.92
12	Unif[m, m^2]	1.32	1.00	1.01	1.01	0.99	1.00	0.91
13	Unif[m, m^2]	1.39	0.99	1.01	1.00	0.98	0.99	0.90
14	Unif[m, m^2]	1.35	1.00	1.01	1.00	0.99	0.99	0.92
15	Unif[m, m^2]	1.40	1.00	1.01	1.00	0.99	1.00	0.94
16	Unif[m, m^2]	1.27	1.00	1.01	1.02	1.00	1.00	0.95
17	Unif[m, m^2]	1.42	1.00	1.00	1.00	0.99	1.00	0.91
18	Unif[m, m^2]	1.33	1.00	1.01	1.01	0.99	0.99	0.93
19	Unif[m, m^2]	1.42	1.01	1.02	1.01	1.00	1.00	0.91
20	Unif[m, m^2]	1.40	0.99	1.01	1.00	0.99	0.99	0.92
21	Unif[m, m^2]	1.35	0.99	1.00	1.01	0.99	0.99	0.91
22	Unif[m, m^2]	1.30	1.00	1.02	1.02	1.00	1.00	0.93
23	Unif[m, m^2]	1.35	1.00	1.01	1.01	0.99	0.99	0.94
24	Unif[m, m^2]	1.34	1.00	1.01	1.01	0.99	0.99	0.93
25	Unif[m, m^2]	1.30	1.00	1.00	1.00	0.99	0.99	0.91
26	Unif[m, m^2]	1.30	1.01	1.02	1.02	0.99	1.00	0.94
27	Unif[m, m^2]	1.28	1.00	1.01	1.01	0.99	0.99	0.93
28	Unif[m, m^2]	1.35	1.01	1.01	1.01	0.99	0.99	0.94
29	Unif[m, m^2]	1.32	1.01	1.01	1.01	0.99	1.00	0.93
30	Unif[m, m^2]	1.28	1.00	1.02	1.01	0.99	0.99	0.93