ABSTRACT

Title of dissertation:	ALGORITHMIC APPROACHES TO REDUCING RESOURCE COSTS IN DATA CENTERS
	Koyel Mukherjee, Doctor of Philosophy, 2013
Dissertation directed by:	Professor Samir Khuller Department of Computer Science

A substantial portion of resource costs incurred by data centers relate to energy costs, with cooling energy and equipment powering energy accounting for a major fraction. Other major costs incurred by data centers, is due to huge data transmission volume and resultant network bandwidth consumption. In this dissertation, we study problems inspired by the needs to reduce energy consumption and network bandwidth billing costs in data centers.

A significant amount of data center cooling energy is wasted due to thermal imbalance and hot spots. In order to prevent hot spots, it is desirable to schedule the workload in a data center in a thermally aware manner, assigning jobs to machines not just based on local load of the machines, but based on the overall thermal profile of the data center. This is challenging because of the spatial cross-interference between machines, where a job assigned to a machine may impact not only that machine's temperature, but also nearby machines due to directional cooling mechanisms currently used in most data centers and subsequent hot air recirculation effects. We define the notion of effective load of a machine, that captures this effect and analyze several different models for two natural (both strongly NP-hard) optimization problems: 1) maximizing the profit of scheduled jobs under a cooling energy budget, and a resultant maximum temperature limit; 2) minimizing the maximum temperature on any machine while scheduling all jobs. For the first problem, we give a $\frac{1}{2} - O(\epsilon)$ approximation for profit maximization on all three models. For the second problem we give a 2 approximation offline algorithm and a 3 competitive online algorithm for a single rack of machines, where the approximation factor approaches $\frac{4}{3}$ and the competitive ratio approaches 2, respectively as the cross-interference falls off. The analysis of all these algorithms is tight.

Apart from cooling issues, servers consume energy while running; hence, shutting down some will reduce energy consumption. In this context, we consider two problems that have been studied in the literature: 1) the active time problem and 2) the busy time problem, The goal in both cases is to minimize the total time the machines are 'on', however, in the active time model, we have access to a single machine whereas in the busy time model we have access to unlimited number of machines. The machines have bounded capacity and the jobs have release times, deadlines and arbitrary processing lengths. For the active time problem, we give a 3 approximation algorithm for non-unit length jobs with integral preemption and show our analysis is tight. We give a 2 approximation algorithm via LP rounding, and also show that the integrality gap of the LP is 2. For the busy time model, we give a 3 approximation algorithm which improves the best known result of 4. We consider the preemptive problem as well and give new algorithms.

Data centers need to transmit a huge volume of data every day, and the resultant network bandwidth consumption costs are extremely high. Frequently, Internet Service Providers charge for Internet use either based on the peak bandwidth usage in any slot in a billing cycle, or according to a percentile (often the 95th percentile) cost model. As a result, an enterprise could save on billing costs by optimizing these measures by delaying some traffic, if possible. However, in reality, traffic is of different types, where some cannot be delayed, and some traffic (such as ftp, bulk data transfer) can be delayed. We provide an optimal offline algorithm for the percentile problem when jobs can have variable delay. We also consider the online problem of minimizing the maximum bandwidth. There exists a tight e-competitive online algorithm for the general problem, where the delay allowed for certain jobs can be arbitrarily large and time is considered to be continuous. We consider smaller values of delay and discrete time slots, since in practice we may not want to delay traffic too much. We give new lower bounds, which are much better than e, on the competitive ratio of online algorithms for several values of delay, and propose and analyze online algorithms with better upper bounds than e for small delay.

ALGORITHMIC APPROACHES TO REDUCING RESOURCE COSTS IN DATA CENTERS

by

Koyel Mukherjee

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2013

Dedication

Dedicated to Ma, Baba, Archi, Tutun and Jeannie, with all my love.

Acknowledgments

The walk to Ph.D. was long, with God guiding me and holding my hand all along the way, and throughout, it was the constant support of Ma, Baba, Archi and Tutun that kept me going. I owe this dissertation completely to my father Dr. Kishalay Bikash Mukherjee, my mother Dr. Shyamali Mukherjee, my husband Dr. Archisman Majumdar, my sister Rupsa Mukherjee and my advisor Professor Dr. Samir Khuller.

My parents helped me realize my dream of graduate education in the United States, when I first joined the University of Texas at Arlington for MS in Electrical Engineering. I got a wonderful job offer at the end of my course of study, but I wanted to study more! My family encouraged me all the way. I joined the Ph.D. program of the Department of Computer Science, in the University of Maryland, College Park.

I am extremely fortunate to have had Professor Samir Khuller as my advisor, who has guided and molded me in every way. His patience with me while I came up to speed from nearly zero background knowledge was infinite. It has been a truly wonderful and enlightening journey as his student and to him I owe all my knowledge in the field of Algorithms. It is due to his continued guidance and faith on me that I am able to graduate with a Ph.D in Computer Science. My gratitude to him can never know any bound.

Throughout the course of my Ph.D., my family has held me together with all their care, support, understanding, and faith on me. Their sacrifices and constant support have made this dissertation possible. When self-doubts plagued my mind, they bore the brunt of my negativity, restored my faith on myself and helped me sail through. Ma and Baba have helped me hold strong, and their unwavering faith on me has made me successful. I have got married during the course of my Ph.D., and for a major part of these past three years, which were also the first three years of my marriage, Archi and I have had to stay apart. Throughout these difficult times of staying away, Archi has been a pillar of support holding me together, not once complaining, bearing my depressive outbursts, and helping me get over the darkness to see the light again.

I would like to acknowledge Jessica Chang, my friend and colleague, with whom I have spent several hours laboring together on algorithmic problems. Working with her has helped me to achieve better clarity of thought and writing, and I am truly grateful for all the times she took the extra effort of picking me up and dropping me home, since I did not have a car. My lab mates Manish Purohit and Kanthi Kiran Sarpatwar, have become two of my treasured friends, and I will forever remember all their help on the day of my defense, alleviating much of my stress. I also want to mention our ex-lab mate Tom Chan, who is a valued friend as well, and the good times and fun hours that I have spent with Jessica, Kanthi, Manish and Tom will remain cherished as beautiful moments in my memory.

I would like to thank Professor Dr. Amol Deshpande, with whom I had the good fortune to collaborate during my graduate studies. Not only was the experience enriching, but I also learned several latex tricks from him! I want to thank all my committee members, Professor Samir Khuller, Professor Amol Deshpande, Professor William Gasarch, Professor David Mount and Professor Gang Qu, for their time, help and valued suggestions regarding my dissertation and defense. I was also fortunate to have got the opportunity to collaborate with Dr. Leana Golubchik and Dr. Yuan Yao which was an enriching experience. I would like to acknowledge my mentors during the internships I did during my graduate studies, from each of whom I learned some thing new that enriched my knowledge and thinking. This includes Dr. Venkatesan Chakaravarthy, Dr. Dinesh Garg, Dr. Sachin Garg, Dr. Sambuddha Roy, Dr. Yogish Sabharwal and Dr. Shrutivandana Sharma. I also want to mention that a part of my graduate studies was funded by NSF grants: CCF-1217890 and CCF-0937865.

I would like to end by thanking God once again above all.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Over the last decade, there has been a dramatic proliferation of data centers. Not only have they increased in number, but existing data centers have expanded in size [71]. With the ubiquitousness of the Internet, increasing popularity of social media, and creation of billions of terabytes of new data [53, 59], the reliance on data centers is likely to increase. The increasing popularity of cloud-computing facilities, which are offered and hosted by data centers is further contributing to the tremendous growth rate of data centers.

A rising concern accompanying the growth and proliferation of data centers has been their energy usage. In fact the energy consumption of data centers has been compared to that of a small town! This is a significant problem not limited to the viewpoint of computer scientists, since it has a direct bearing on the environment. According to a report by the EPA [72], in 2006, data centers consumed 61 billion KWh of electricity, which is 1.5% of the total US electricity consumption, and cost the US $4.5 billion, and it has increased manifold since then. In fact EPA had projected the energy consumption of data centers over the period 2006-2011 (see Figure 1.1).

In recent years, companies like Amazon, Google, Microsoft and Yahoo! have

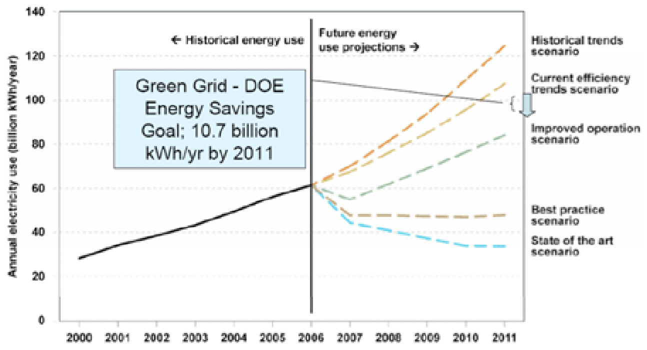


Figure 1.1: EPA prepared graph providing future projections of data center energy consumption based on historical trends [72]

made large investments in massive data centers supporting cloud services. Greenberg et al. [31] attempt to give a breakup of the huge operational costs of cloud supporting data centers. Over 55% of the costs arise from cooling, power distribution and equipment powering and networking. In this dissertation we take a multi-pronged algorithmic approach towards reducing some of the resource costs in data centers arising from the above issues. We describe the problems in detail in the following sections, and give a brief overview of our approaches and results.

1.1 Cooling Energy Costs: The Thermal Scheduling Problem

Modern data centers consist of thousands of computers closely packed in a dense space. The power consumption is increasing with the increase in number and computing power of servers. However, this also results in higher heat dissipation by the servers. It is essential to maintain the temperatures of servers and computing equipment below a certain limit in order to maintain reliability and reduce circuit failures. A significant portion of the energy cost of a data center is the cost incurred in cooling the machines in the data center [5, 19].

According to Moore et al. [54], data centers spend one-half to one Watt to power the cooling infrastructure for every Watt of power spent for computation. Figure 1.2 illustrates this by a pie chart.

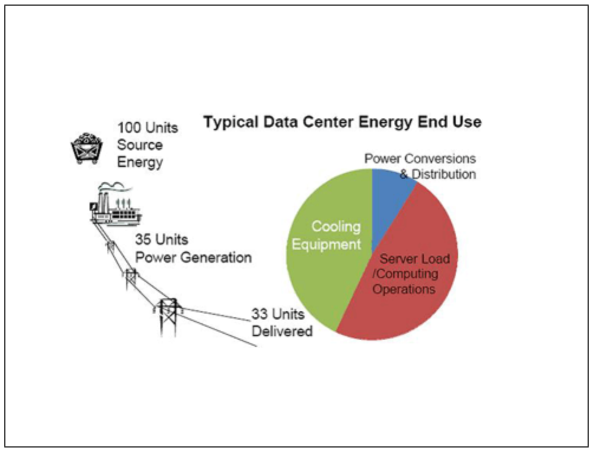


Figure 1.2: Distribution of energy consumed by data center [20].

Data center energy management, and in particular cooling strategies have emerged as a primary challenge. The energy cost of cooling is directly driven by the *supply temperature* (denoted by T_{sup}) of the cold air being blown in to cool the data center – the incoming air is often kept at a lower than necessary temperature to prevent hotspots from forming since those can damage the hardware. It has also been observed that servers near the top of a rack often run hotter and are subject to higher failure rates [33]. Thermal balancing through judicious task scheduling can lead to fewer hotspots and thus lower overall cooling costs and lower failure rates. A reduction of even a few percent in the supply temperature could have a drastic impact on the overall cooling costs.

This has led to much work in better cooling technologies as well as reactive and proactive scheduling policies in recent years. Exotic technologies like heat pipes, liquid cooling, and immersion have been proposed and are effective to some extent, but these techniques are expensive and do not scale with technology [29].

The potential of savings by deploying thermal aware scheduling instead of exotic new cooling technologies can be illustrated by the following example providing some representative numbers [54]. In a standard $30,000$ square foot data center with 1000 standard computing racks, each consuming 10 kW, the initial cost of purchasing and installing the computer room air conditioning (CRAC) units is \$2-\$5 million. This results in an average electricity cost of \$100/MWhr, which translates to \$4-\$8 million annual cost for cooling alone. If through intelligent thermal management, the data center can run the same computational workload with the same cooling configuration, while maintaining an ambient room temperature that is $5°C$ cooler, the CRAC power consumption will reduce by 20%-40% resulting in a \$1-\$3 million savings in annual cooling costs [54].

It has been observed that the heat generated by jobs running on a machine raises its own temperature as well as the temperatures of nearby machines due to recirculation effects. Such effects are well-documented both for data centers [63,70]. In addition, the *geometry* of the data center plays a significant role in determining the cross-effect parameters, which are often asymmetric. For instance, in a standard raised-floor data center where the cold air is blown in through vents in the floor, the load on a machine closer to the ground is likely to impact the temperature of the machines above it, but not vice versa. Thus the temperature of a machine not only depends on its local load but also on the load of nearby machines.

In the first part of the dissertation, we consider the thermal scheduling problem, motivated by increasing power density and consequent cooling considerations in data centers. We propose thermally-aware scheduling algorithms of jobs in data centers, such that either the maximum temperature of the machines while executing a given set of jobs is minimized, or the number or profit of jobs assigned is maximized while keeping the maximum temperature below a certain limit. Traditional scheduling models and algorithms do not provide a satisfactory answer to these problems. The key differentiating factor here from all of the prior work in scheduling is the notion of *spatial cross-interference* or cross effects, which arises due to thermal effects.

First, we consider the fractional problem, where jobs can be arbitrarily split between machines. We give optimal fractional strategies for minimizing the effective load (in other words, the temperature of machines) while scheduling the entire load, as well as for the dual problem of maximizing the number of jobs scheduled when there is a budget on the maximum temperature, or effective load of every machine, determined by the temperature of the cold air supply [55]. We then extend our work to multiple racks of machines, and the more realistic and complex problem requiring jobs to be integrally assigned to machines [56]. This is strongly NP-hard, generalizing the multiple knapsack problem for the budgeted version, and the minimum makespan problem for the dual problem of minimizing the maximum temperature. We analyze several different models, and give $\frac{1}{2} - O(\epsilon)$ approximation algorithms for the profit maximization problem on all of the models. For the minimization of maximum temperature problem, we give a 2 approximation algorithm for a single rack of machines, which approaches $\frac{4}{3}$ as the spatial cross-interference factor falls off. We also give an online algorithm for the temperature minimization problem with a competitive ratio 3. The competitive ratio approaches 2 with the decrease in spatial cross-interference. The analysis of all the algorithms is tight.

1.2 Equipment Powering Costs: The Busy-Time and Active-Time Problems

Cooling is however not the only source of energy consumption. The servers consume energy while running; hence, shutting down some will reduce energy consumption. This gives rise to another fundamental scheduling problem motivated by energy issues in a cloud computing context.

The majority of servers in a typical data center run at or below 20% utilization most of the time, yet still draw full power during the process [62]. The cost of powering such severely underutilized servers can account for a huge chunk of an organization's energy bill [52]. This also leads to increased cooling costs as well as higher energy footprint. Consolidating underutilized servers to a fewer number of servers will reduce energy and support costs. In fact, as Figure 1.3 shows, improving server utilization has the highest potential for increasing energy efficiency.

Recent progress in virtualization has facilitated the consolidation of multiple virtual machines (VMs) into fewer hosts. As a consequence, many computers can be shut off, resulting in substantial power savings. Today, products such as Citrix

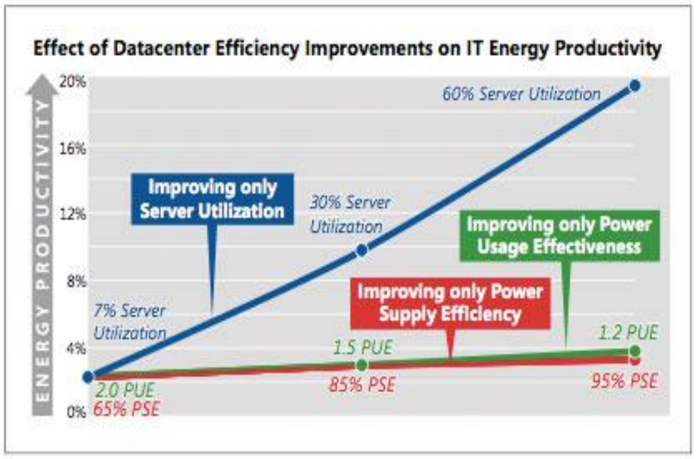


Figure 1.3: Graph illustrating the potential for energy efficiency gains by server utilization improvement [52].

XenServer an VMware Distributed Resource Scheduler (DRS) offer VM consolidation as a feature.

1.2.1 The Busy Time Problem

Motivated by the above, we consider a batch scheduling problem, where broadly, our goal is to batch jobs in an effective manner so that the servers or machines that need to be 'on' simultaneously is minimized[1]. This has been studied in literature as the "busy-time" problem, and not only cleanly captures energy related issues, but also has connections several key problems in optical network design, perhaps most notably in the minimization of the fiber costs of Optical Add Drop Multiplexers (OADMs) [22, 23, 25, 74]. In this model, jobs have a certain processing requirement,

[1]Some parts of this work have been done jointly with Jessica Chang, and these can also be found in her thesis [6].

and a specified time window within which they need to be scheduled. Every machine, which can be thought of as a virtual server, has a limited capacity for the number of jobs that can run on it simultaneously. The goal is to batch the jobs in a feasible manner, so that all jobs get processed, machine capacities are not violated while the total runtime or busy-time of all machines is minimized. There is no restriction on the number of machines that can be active at the same time in this model. We improve the approximation factor of the known/existing algorithms for this model for the non-preemptive version of the problem. For the special case where the jobs do not have any flexibility of assignment within their feasible window, a factor 4 approximation algorithm was known so far, given by Flammini et al. [22]. We show that a 2 approximation algorithm for this problem is implied by the works of Alicherry and Bhatia [1] and Kumar and Rudra [42] for related problems. We then consider the general problem where jobs have flexibility of assignment within their feasible windows. For this problem, a 4 approximation algorithm was given by Khandekar et al. [39] and also a natural extension of the algorithms of Alicherry and Bhatia, and Kumar and Rudra give a 4 approximation. We give a 3 approximation algorithm for this problem [8] and also show that the factor 3 is tight asymptotically. Furthermore, we also consider the pre-emptive version of this model, and give an exact algorithm when machine capacities are unbounded and a 2 approximation algorithm for bounded capacity.

1.2.2 The Active Time Problem

We then consider a related problem, active time, which was introduced by Chang et al. [7]. The setting is almost the same as in the busy time problem, except that in this model, we do not have access to an unlimited number of machines. There is a single machine in which jobs need to be scheduled and as in the busy time model, the number of jobs which can run simultaneously on the machine is limited. Jobs have associated sizes or processing requirements and release times and deadlines, which define the feasible windows within they need to be processed. We assume there exists a feasible schedule since determining whether one exists is an easy problem. The goal is to minimize the total time when any job is running on the machine, since during all such times, the machine needs to be active, hence draw power. Chang et al. [7] considered the special case of unit jobs and gave a polynomial time algorithm. The problem of non-unit jobs with preemption allowed at integral time boundaries was considered by Chang [6] for which a 5 approximation algorithm was presented. We improve the analysis of this algorithm to improve the bound to 3, which we show to be tight. We then give a 2 approximation algorithm for this problem using LP rounding [8] and also show that the integrality gap of this LP is 2.

1.3 Network Bandwidth Costs: The Percentile Problem and the Min-Max Problem

The previous sections dealt with the energy costs of data centers and possible optimization solutions. However, as we pointed out earlier, energy is not the only source of the huge operational costs of data centers. A large part of the resource costs incurred by data centers is due to the huge volume of data transmission and the resultant network bandwidth consumption costs. Generally, Cloud Service Providers (CSP) operate the cloud infrastructure over multiple data centers (DCs), which are connected by very high capacity data links [58]. Most of the time, these links are leased by the CSPs from network operators. Google, Amazon and other operators of large data centers that host cloud computing applications need to provide services and synchronize processed data across multiple locations, hence transmit a huge volume of data over these links daily, and consequently pay the Internet Service Providers (ISP's) for the heavy bandwidth usage. Multimedia content providers such as Akamai need to replicate video clips over servers located in different regions for optimized content delivery. This also results in huge data chunks being transmitted over the high speed leased links, and cost the senders a significant amount of money on a daily basis.

Cost accounting for data transfers is performed differently from other utilities such as energy where we are billed based on total volume used over a period of time. For network bandwidth cost, two widely used rules for charging are 1) peak bandwidth usage, and 2) the 95th percentile rule [44]. Over each billing cycle (say a day), the cycle is broken into "slots" (for example one minute period of time) and the bandwidth usage per slot is sampled. In other words, the billing is based on a vector $\mathbf{x}$ where x_i is the volume of traffic sent over the i^{th} slot. In the peak bandwidth charging scheme, one is charged for the maximum bandwidth utilization over any slot in one billing cycle. In the 95th percentile charging scheme, one is not charged on the maximum, but on the 95th percentile of this vector for billing.

The CSPs end up paying for peak usage or 95th percentile of it; however a lot of bandwidth capacity is left unused over the inter-datacenter links [58]. The links between the data centers are used to carry both client traffic and inter-datacenter traffic. The CSP has no control over the client traffic which has to be sent over the link as soon as it arrives. However, the inter-datacenter traffic is primarily composed of delay tolerant jobs, such as backup traffic, database operations, video processing, analytics, etc. The performance requirements of such data transfers typically allow for some transmission delay, within which the data chunks have to be delivered. Such traffic in fact occupies as much as 40% of the inter-datacenter link bandwidth [58]. As a result, an enterprise could save on billing costs by optimizing the billing measures by delaying some traffic, if possible. For example, video replications over servers at different locations do not have to be carried out exactly at 5:00PM daily. All that is needed is that they are completed, for instance, before midnight. This provides opportunities for data senders to optimize data transfers by attempting to reduce the 95th percentile or peak bandwidth usage.

1.3.1 The Percentile Problem

We first consider the percentile problem, both online and offline variants of it. In this setting, the data transfer link has a certain capacity for peak bandwidth, which cannot be exceeded at any point of time. Specifically, we consider a model where (data transfer) jobs come with processing requirements (associated sizes) and also with a specified delay, within which they must be completed. The delay varies with jobs. We provide an optimal polynomial time offline algorithm for minimizing the 95^{th} or any percentile of the job transmission vector over a billing cycle for the case when unit sized data chunks can be transmitted independently of each other, and different jobs are allowed different amounts of delay [40]. We had earlier studied the special case where all jobs are allowed uniform delay [28] (This problem was also considered by Yao [76]). Our new result not only generalizes, but also subsumes both of the above mentioned earlier works [28, 76]. When the data can have non-uniform sizes (in other words, dependencies), the offline problem becomes strongly NP-hard. We give a fully polynomial time approximation algorithm for a special case. However, for the online variant of the problem, we had earlier shown that no deterministic online algorithm can have a bounded competitive ratio [28,76].

1.3.2 The Min-Max Problem

We then consider the online problem of minimizing the maximum bandwidth, which we call the min-max problem. Interestingly, the problem of peak bandwidth usage minimization has connections to the energy minimization problem. Consider a setting where there is a single processor, which can run at variable speeds, and jobs that need to be processed have associated release times and deadlines. For a feasible schedule, we might need to adjust the speed of the processor, in order to finish processing all jobs. However, the power consumed by the processor is directly proportional to the speed at which they are running, and the speed is dictated by release times, deadlines and processing requirements of the jobs so as to ensure feasibility. This model was introduced and analyzed by Yao et al. [75] and since then both the online and offline problems of energy consumption minimization in this model has been widely studied in literature [2, 3]. Now consider time to be discrete and the machine as a time slot, and the volume of data transferred in the time slot as the amount of processing done by the machine at that instant. The data requiring transmission has associated sizes, release times and deadlines, determined by the time slot they arrive in and the allowed delay. Hence, the optimal offline solution minimizing the peak bandwidth consumption over any time slot is the same as the optimal offline algorithm minimizing the maximum speed of (and hence, maximum power consumed by) a processor over a time period ensuring feasibility. This algorithm was given by Yao et al. [75] for the energy minimization problem (assuming the energy function is convex). For the online problem an e-competitive online algorithm was given by Bansal et al. [2] for the general problem, where the delay allowed for certain jobs can be arbitrarily large and time is considered to be continuous. We consider the problem of smaller values of delay and discrete time slots, since in practice we may not want to delay traffic or jobs too much. We first studied this problem for the special case of uniform delay, where all jobs can be delayed by the same amount [28]. We now give improved lower bounds on the competitive ratio of online algorithms for several values of uniform delay, as well as variable delay, and give new lower bounds for arbitrarily large discrete delay which is still small compared to the entire time period [40]. The lower bounds are much better than the best known bound of e of Bansal et al. [2] for the variable delay problem, and also directly improve the bounds presented by Yao [76] for the uniform delay problem. In addition we give an online algorithm matching the lower bound for $D \in \{0, 1\}$, where D is the maximum allowed delay for any job, and a 2 competitive algorithm for $D \in \{0, 1, 2\}$ [40], both of which are better than the best known upper bound of e.

1.4 Outline of the Dissertation

In the following chapters, we formally define the models that we have broadly outlined in the above sections, and then we present our algorithms, results and analysis on these models.

We formally define the thermal model in Chapter 2, and present our results for a single rack of machines in Chapter 3. In Chapter 4, we analyze the more complex problem of multiple racks of machines.

We introduce the busy time model in Chapter 5 and give improved approximation algorithms for the non-preemptive model, as well as new algorithms for the preemptive model. In Chapter 6, we introduce the active time model and present new and improved results for non-unit length jobs with integral preemption.

In Chapter 7, we define the network bandwidth problem of percentile billing and present optimal and near optimal offline algorithms for this problem. Finally, we define the min-max problem in Chapter 8, and present improved lower and upper bounds for the online version of this problem. We conclude the dissertation in Chapter 9.

Chapter 2: The Thermal Scheduling Problem

In this chapter, we formally define the thermal models and the optimization problems we consider in the first part of the dissertation. We highlight how our model abstracts the physical observations presented in the systems literature, and also present on overview of related work in this area. We use the terms *machines* and *processors* interchangeably. We also use the terms *jobs* and *items* interchangeably.

2.1 Thermal Model

The key consideration in thermal management of data centers is to ensure that the temperature of any processor does not cross the red-line temperature T_{red}. We base our model on the abstract heat circulation model suggested by Tang et al. [67–70], Mukherjee et al. [57], and Varsampoulous et al. [73].

According to this model, under a steady state assumption (see below), the temperature of the cold air T_{sup} required for maintaining the temperature of machine i below the red-line is defined by the following constraint, $T_i = T_{sup} + \mathbf{D_i} \cdot \mathbf{L}$ where T_i is the temperature of machine i, and $\mathbf{D_i}$ is the i^{th} row of the heat distribution matrix $\mathbf{D}$, and $\mathbf{L}$ is the load or power vector. The vector $\mathbf{L} = \{L_1, \cdots, L_m\}$, where m is the number of machines, denotes the loads on the machines in terms of the *power* consumed by the jobs assigned to the machine.

The matrix $\mathbf{D}$ represents how the heat or load of any machine j affects machine i (called *cross-interference*). Since the temperature of no machine should exceed T_{red}, we have that: $T_{sup} + \max_{i\in[1,\ldots,m]} \mathbf{D_i} \cdot \mathbf{L} \leq T_{red}$. Thus, given a set of jobs, our goal is to schedule them so as to either:

1. given a constraint on $\max_{i\in[1,\ldots,m]} \mathbf{D_i} \cdot \mathbf{L}$, maximize the number of assigned jobs

2. maximize T_{sup} (or equivalently, minimize $\max_{i\in[1,\ldots,m]} \mathbf{D_i} \cdot \mathbf{L}$)

For each job, we assume that we can estimate the power that will be consumed to execute it on a machine; this can be computed using the estimated resources required to execute the job, its time duration, and standard system power modeling techniques [16].

The total energy consumption is obtained by adding the energy for processing and the energy for cooling, and can be modeled as: $E = L(1 + \frac{1}{CoP(T_{sup})})$, where $L = \Sigma L_i$ is the total load on all the machines, and CoP (*coefficient of performance*) is a super-linear function of the supply temperature.

The efficiency of cooling units is characterized by the *coefficient of performance (CoP)* [57,67–69,73], which is typically a super-linear function of the required supply temperature, T_{sup}. The CoP denotes the ratio of heat removed to the work done to remove heat.

As with much of the prior work on thermal scheduling, we assume that the system is in steady state. In other words, we assume the jobs are long-lived, and analyze the system state when all the jobs have arrived and the temperatures have stabilized. The *size* of a job in this setting refers to the power consumed per unit time to process its entire computation on any machine, instead of its length. The heat dissipation due to a job is therefore the product of the power requirement and the time duration over which we are examining the state of the system. This time duration would thus be the same for all jobs, only the power needed may vary.

2.1.1 Effective Load

We formulate the problem of thermal scheduling in terms of minimizing what we call the "effective load" on a machine. Effective load on a machine is a linear combination of the load of the machine itself and the load of other machines. Specifically, given that the load of machine i is L_i, and the effect of machine j's load on machine i is captured through the **cross-interference coefficient** D_{ij}, the effective load EL_i is computed as follows: $EL_i = \sum_j D_{ij} L_j$ where $0 \leq D_{ij} \leq 1$ and $D_{ii} = 1$. Our optimization problem of minimizing the maximum temperature can now be seen as minimizing maximum effective load instead, an easier quantity to reason about.

2.2 One Dimensional Cross-interference Model

We assume an asymmetric model of spatial cross-interference that tries to capture the nature of the heat flow in a data center. Here we consider a model of machines in a linear array with the cold air blowing from one end. This models the behavior of a single rack with the cold air blowing from the floor [63] and the warm air moving into a vent in the ceiling and back to the HVAC unit.

In this simple model, which we call the *one-dimensional model*, heat is recirculated in one direction. The i^{th} machine is affected only by the heat recirculated from the machines located below it, closer to the source of the cold air. We number machines from bottom to top, in increasing order from the cold air source. Machine i is only affected by machines $j \leq i$.

We assume the heat falls off in an exponential manner. Specifically, the heat felt by a machine i due to machine j is a fraction $\frac{1}{K^d}$ of the load (heat) of j, where $d = |i - j|$ is the distance between i and j and K is a constant > 1. More formally, $D_{i,j} = \frac{1}{K^{|i-j|}}$. For technical reasons we assume that $K \geq 2$. In any case, this is a reasonable assumption, otherwise the heat effect felt by a machine due to its immediate neighbor in the rack can contribute to more than half of the total heat it can handle so as not to violate T_{red}, which is unrealistic.

The number of machines in a rack is $m \geq 2$. The *effective load* of the i^{th} machine, where $1 \leq i \leq m$, is given as

$$EL_i = \sum_{j=1}^{i} \frac{L_j}{K^{i-j}} \tag{2.2.1}$$

Figure 2.1 illustrates the model considered in the one dimensional case.

2.3 Two Dimensional Cross Interference Models

We then generalize the above 1-D model to a 2-D array. We consider not just one rack of machines, but a two dimensional grid, consisting of several adjacent racks, such that there is heat distribution among the machines of one rack as in

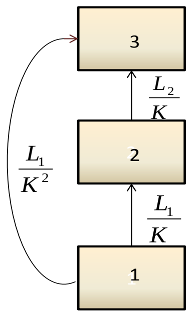


Figure 2.1: Heating effect in one dimensional model. Here, we show a rack with three machines.

the one-dimensional model, as well as heat distribution laterally between adjacent racks. We assume as before the cold air source is located at the bottom of the racks, so the heat flows upward along each rack. We also assume there is a cold air source at one end of the series of racks, so that the heat also flows laterally from one rack to another. The heating effect is felt by a machine from the machines located below it in the same rack, as well as from the machines located on the racks to its left, at the same or lower position on their respective racks.

The racks are numbered in an increasing order from the cold air source and also from bottom to top. In previous works [29,49,66] the lateral heat redistribution is considered much weaker than the vertical one. So, the temperatures of machines located in the same rack are more strongly coupled than those across racks. To capture this effect we define *three* models of heat recirculation in the lateral direction. In all these models, the number of machines in a single rack is $m_1 \geq 2$ and the number of racks is $m_2 \geq 1$. The total number of machines m is therefore $m_1 m_2$.

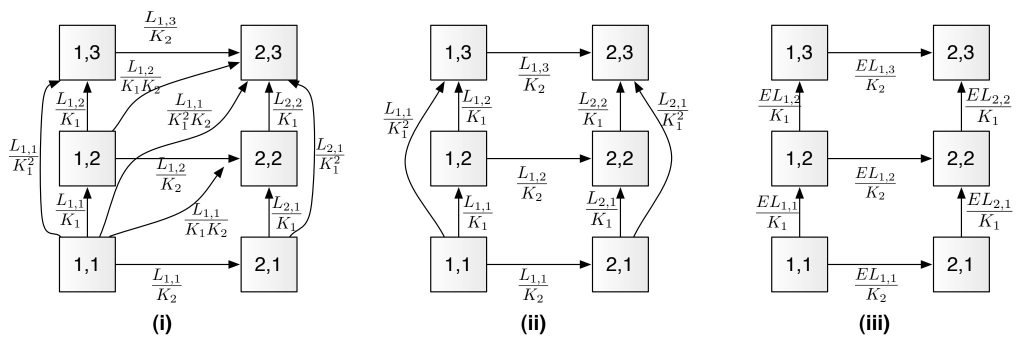


Figure 2.2: Models of heating effect shown for two racks each with three machines: (i) General model, (ii) Horizontal Sibling Model, (iii) Indirect Sibling Model.

The heat recirculation coefficient in the vertical direction is called K_1 and in the lateral direction as K_2. We assume $K_1 \geq 2$ and $K_2 \geq 2$.

2.3.1 2-D General Model:

In the first model, which we call the *general model*, for a machine located at j^{th} row of the i^{th} rack, where $1 \leq j \leq m_1$ and $1 \leq i \leq m_2$, we consider the effect of loads on all machines located in rows j or lower, of all racks numbered $[1 \ldots i]$. The vertical heat redistribution falls off as K_1^d, and the lateral redistribution falls off K_2^d, with $K_2 > K_1$ to model the weaker lateral effect compared to vertical one. The heat redistribution effect from the load on the $(i', j')^{th}$ machine on the $(i, j)^{th}$ machine is $\frac{L_{i'j'}}{{K_1}^{j-j'}\ {K_2}^{i-i'}}$, i.e., the effective load on the $(i, j)^{th}$ machine is:

$$EL_{i,j} = \sum_{i'=1}^{i} \sum_{j'=1}^{j} \frac{L_{i',j'}}{{K_1}^{j-j'}\ {K_2}^{i-i'}} \tag{2.3.1}$$

Fig. 2.2(i) illustrates the heating effect in the general model.

2.3.2 2-D Horizontal Sibling Model:

In the second model, which we call the *horizontal sibling model*, the heat redistribution in the vertical direction, along a rack, is same as the 1-D model, but the lateral redistribution is restricted to a single level. The vertical heat redistribution falls off exponentially as K_1^d, and the lateral redistribution falls off as K_2. In this model, the effective load on the $(i, j)^{th}$ machine is given as follows.

$$EL_{i,j} = \sum_{\ell=1}^{j} \frac{L_{i,\ell}}{{K_1}^{j-\ell}} + \frac{L_{i-1,j}}{K_2} \tag{2.3.2}$$

Fig. 2.2(ii) illustrates the heating effect in the horizontal sibling model.

2.3.3 2-D Indirect Sibling Model:

In the third model, which we call the *indirect sibling model*, the effective load on the $\left(i^{th}, j^{th}\right)$ machine is defined as

$$EL_{i,j} = L_{i,j} + \frac{EL_{i,j-1}}{K_1} + \frac{EL_{i-1,j}}{K_2} \tag{2.3.3}$$

Fig. 2.2(iii) illustrates the heating effect in the indirect sibling model. (This model was motivated as a generalization of the 1-D model. The generalization will become clear when reading Claim 1.)

2.4 Thermal Scheduling Problem

We are given a set of jobs $\mathcal{J}$, with job $j \in \mathcal{J}$ having a size s_j, where the size of a job denotes its power requirement, and hence, the thermal load caused by it on the assigned machine. The jobs may additionally have profits associated with them. In this case, on scheduling a job j, we can get a profit p_j. We are also given a set of machines, and a capacity c_i with each machine i which refers to the maximum temperature or effective load it can handle (this capacity definition holds for the maximization problem, where the cooling energy budget fixes T_{sup}; T_{red} is of course a system dependent constant). We are also provided a cross-interference model and the geometric configuration of the data center. Our goal is to either schedule all the jobs while minimizing the maximum effective load (assuming that all the jobs can be scheduled), or alternately, maximizing the number or profit of jobs that can be scheduled, given upper bounds on maximum effective load. We assume here that a job can be assigned to any machine. This is a reasonable assumption, if all the machines are from the same cluster, and all have the same processing capability. We denote the number of jobs in the given instance, i.e., $|\mathcal{J}|$ as n for the integral assignment problem. For the one-dimensional problem, studied in Chapter 3, we assume there are m machines arranged on a rack. For the two-dimensional problem, studied in Chapter 4, there are m_1 machines per rack and m_2 racks, hence in total, $m_1 m_2$ machines.

It is NP-hard to find the optimal scheduling policy for the thermal scheduling problem. We therefore relax the problem to the case when jobs are splittable between machines, find an optimal solution, and then use this solution to devise approximations for the integral assignment case. One can use LP to solve the fractional assignment problem, however, we provide combinatorial insights regarding the structure of such an optimum fractional solution, which we use to devise feasible packings for the integral case. One central contribution is to show how to "reduce" the problem to the multiple knapsack problem with appropriately defined capacities, so that we can safely ignore thermal constraints.

2.5 Related Work

Thermal scheduling in data centers has been an area of active research in recent years. Spatial cross-interference effects are well-documented (see, e.g., Schmidt and Cruz [63]). Moore et al. [54] suggest a set of heuristics for workload placement and scheduling for controlling hot spots. Tang et al. [67] proposes a cross-interference model to capture heat recirculation in a data center, and that model and its impact on task scheduling has been explored in a series of works since then [57, 68, 69, 73]. We introduce the thermal scheduling problem [55] allowing fractional assignments, following the thermal model proposed by them. Shi and Srivastava [65] also use a similar model, but focus on the storage units (disks) instead of the compute units (processors). Pakbaznia and Pedram [60], using a similar model, argue that server consolidation (choosing which servers are on) is critical in minimizing the power consumption, and address the combined problem of task scheduling and server consolidation.

Similar to our work here, much of the above work also makes a steady state assumption leading to a stationary temperature profile that is optimized. Some work has considered different cooling models and their impact on task scheduling [73]. Zhang et al. [77] take time into account and give an approximation algorithm for voltage frequency scaling. Fisher et al. [21] use the Fourier's cooling model to model cooling and heating phenomena, and develop algorithms for frequency scaling. Choi et al. [11] observe that the rise- and fall-times of on-chip temperatures are typically an order of magnitude larger than the OS scheduler ticks, and develop an OS-level scheduler to balance the heat and avoid formation of hotspots. Extending our approach to formally analyze temporal effects is a rich area for future work that we are planning to explore.

There have been several algorithmic and theoretical papers exploring the power on and off strategies [4,34]. Yao et al. [75] and Irani et al. [34] consider the problem of reducing the total energy consumed by controlling processor speed for a single processor. Bansal et al. [2] and Bansal and Pruhs [3] also consider the problem of minimizing energy and temperature for a single processor using speed scaling techniques. Our work is fundamentally different since we consider multiple processors with cross-interference and there is no speed control.

With increasing power density on multi-core chips and a trend toward 3D chip architectures [50] that tend to exhibit high temperatures, micro-level thermal management has seen much work in last few years. In addition to investigations into better cooling technologies, techniques have also been proposed to either reduce the power consumption locally through frequency and voltage scaling, or by dynamically redistributing the workload to handle hotspots. Gomaa et al. [29] propose a technique called *heat-and-run* that uses intelligent thread assignment, and thread migration to address the problem. Coskun et al. [14] use both voltage/frequency scaling and task migration to reduce the frequency of hotspots. Ge et al. [26] propose using local task swaps between neighboring cores to achieve thermal balancing; their thermal model looks very similar to the cross-interference model that we use in this work. Liu et al. [49] also give heuristic solutions to the problem we study. Li et al. [48] also look at thermal management in micro chips, but when jobs have precedence constraints. In an earlier work, Kursun et al. [43] examine the effects of task scheduling on thermal behavior and experimentally show that thermal-aware scheduling policies can alleviate on-chip temperatures. In recent work, Zhou et al. [78] propose and analyze several heuristics for thermal-aware scheduling in 3D chips.

Unlike us, prior work in spatial cross-interference has either presented heuristics or suggested using ILP solvers to solve the problem, and has not attempted to exploit the structure of the cross-interference matrix to design algorithms with worst case guarantees.

Chapter 3: Single Rack of Machines: The One Dimensional Model

In this chapter, we consider a single rack of machines (in essence, the one-dimensional model). First, we study the energy savings possible when fractional assignments are allowed, i.e., the load of a job can be split across multiple machines. Next we consider the more difficult problem of integral assignments. For both these cases, we consider two optimization problems: maximization of profit or number of assigned jobs under a cooling energy budget, and minimization of maximum temperature of any machine when all jobs need to be assigned.

3.1 Fractional Assignments

We know from Chapter 2, that higher the supply temperature, the lower is the cooling energy consumption. However, in order not to violate the red-line temperature, the supply temperature can be made higher only if the effective load on machines can be lowered. Hence, we want to minimize the maximum effective load on any machine. Since we are allowing fractional assignments, this implies that we should make the effective load across all machines uniform.

A naive scheduler on the other hand would spread the load uniformly across all the machines. However, due to the thermal cross effects, this leads to a skewed effective load that increases as we go left to right. The central question we consider is: *How should the load be distributed in order to minimize the maximum effective load?*

Let us first consider a simple motivating example when $m = 2$ and $K = 2$. Suppose we have an effective load capacity of one unit (the capacity being determined by T_{red} for a fixed T_{sup}). How much load can we assign if we distribute load uniformly? Note that if we assign $\frac{2}{3}$ units of load to each machine, then the effective load on the second machine is already 1 (its load plus half the load on the first machine). However if we assign one unit of load on the first machine, then we can assign $\frac{1}{2}$ unit of load on the second machine, and now both machines have an effective load of one unit. Thus we were able to assign *more* load by distributing the load in a non-uniform manner.

3.1.1 Minimization of Effective Load

For this problem, all jobs need to be fully assigned and we simply wish to minimize the maximum effective load $\max_{i\in[1,\ldots,m]} EL_i$. Let the total load (power requirement for computation of all the jobs over a unit time interval) be L, i.e., $\sum_{j\in\mathcal{J}} s_j = L$. Theorem 1 outlines the reduction in maximum effective load that a thermally aware scheduler can achieve compared to a naive scheduler uniformly splitting the load.

Theorem 1. *The reduction in effective load that a thermally aware scheduler can achieve compared to a naive load balanced strategy is* $\geq \frac{L}{m(K-1)+1}(1-\frac{1}{K})^{m-1} > 0$.

Before proving Theorem 1, we prove the following.

Claim 1. *The effective load of the i^{th} machine can be expressed as: $EL_i = L_i + \frac{EL_{i-1}}{K}$, where L_i is the local load of the i^{th} machine.*

Proof. This claim is proved by induction. For $i = 1$, it is trivially true, as there is no machine before machine 1 (i.e., $EL_0 = 0$). Hence, $EL_1 = L_1$. Clearly $EL_2 = L_2 + \frac{L_1}{K} = L_2 + \frac{EL_1}{K}$. By the induction hypothesis, let us assume, the claim is true for all $i \leq p, p > 1$. The effective load on the $p+1^{th}$ machine is $EL_{p+1} = L_{p+1} + \frac{L_p}{K} + \frac{L_{p-1}}{K^2} + \ldots + \frac{L_1}{K^p}$ by definition. Hence, by induction, $EL_{p+1} = L_{p+1} + \frac{EL_p}{K}$. □

Lemma 1. *An optimal strategy for minimizing the maximum* effective load *when fractional assignments are allowed, would result in uniform effective load of $EL = \frac{L}{m - \frac{m-1}{K}}$.*

Proof. An optimal strategy for minimizing the maximum effective load on any machine, would result in uniform effective load across all the machines, without loss of generality. This is because, by redistributing load from the machine with the highest effective load so as to smoothen out the effective load across all machines, the cost of the solution would not increase. Let us therefore assume that the optimal strategy makes the effective load on any machine i, $EL_i = EL \; \forall i \in [1, \ldots, m]$.

From Claim 1, the total load can be expressed as:

$L = EL_1 + \sum_{i=2}^{m} \left(EL_i - \frac{EL_{i-1}}{K} \right)$. Substituting EL for all EL_i, $i \in [1, \ldots, m]$, $EL = \frac{L}{m - \frac{m-1}{K}}$. □

An optimal strategy minimizing the maximum effective load would therefore, assign load $L_1 = EL$ on machine 1, and a load $L_i = EL\left(1 - \frac{1}{K}\right)$ on machine i for all $2 \leq i \leq m$, where $EL = \frac{L}{m - \frac{m-1}{K}}$.

Lemma 2. *An optimal strategy for minimizing the maximum* load *for a fractional assignment with total load L, would result in maximum effective load* $EL_{max} = \frac{L(1-\frac{1}{K^m})}{m(1-\frac{1}{K})}$.

Proof. An optimal strategy for minimizing the maximum load in a fractional assignment setting would distribute the load uniformly among the machines, hence $L_i = \frac{L}{m}$ for all $i \in [1, \ldots, m]$. Machine m would have the maximum effective load because of the cumulative heating effect from all machines below it. Hence, $EL_{max} = EL_m$, where $EL_m = \frac{L}{m}\left(\sum_{j=1}^{m} \frac{1}{K^{m-j}}\right) = \frac{L}{m}\frac{(1-\frac{1}{K^m})}{(1-\frac{1}{K})}$. □

Proof of Theorem 1:

Proof. Let the savings in maximum effective load be ΔEL. From Lemmas 1 and 2,

$$\begin{aligned}
\Delta EL &= \frac{L}{m}\left(\frac{1-\frac{1}{K^m}}{1-\frac{1}{K}}\right) - \frac{L}{m - \frac{m-1}{K}} \\
&= \frac{L\left(1 + \frac{1}{m(K-1)} - \frac{1}{K^m} - \frac{1}{K^m m(K-1)} - 1\right)}{m\left(1-\frac{1}{K}\right)\left(1 + \frac{1}{m(K-1)}\right)} \\
&= \frac{L\ K\left(\frac{1}{m(K-1)}\left(1 - \frac{1}{K^m}\right) - \frac{1}{K^m}\right)}{m(K-1)+1} \\
&= \frac{L\left(K^m - 1 - m(K-1)\right)}{K^{m-1}\left(m(K-1)+1\right)\left(m(K-1)\right)}
\end{aligned}$$

Writing $K^m = (1 + (K-1))^m$, and using binomial series (since $K > 1$), we get: $K^m = 1 + \binom{m}{1}(K-1) + \binom{m}{2}(K-1)^2 + ... + \binom{m}{m}(K-1)^m$. In other words, $K^m > 1 + m(K-1) + (K-1)^m$, when $m \geq 2$. Therefore, the savings in effective load can

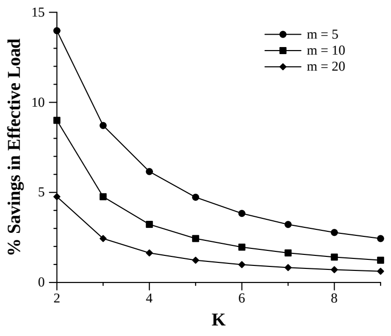


Figure 3.1: Percentage reduction in maximum effective load across all machines as K varies, for different values of m

be expressed as

$$\Delta EL > \frac{L(K-1)^m}{K^{m-1}(m(K-1)+1)(m(K-1))} \qquad = \frac{L}{m(K-1)+1}(1-\frac{1}{K})^{m-1}.$$

This proves the theorem. □

Consider the following example. Let the total load be 10, the number of machines in the rack be 7 and $K = 3$. The maximum effective load by the thermal aware strategy would be $\frac{10}{7-\frac{6}{3}} = \frac{10}{5} = 2$. The naive uniform load strategy would put a load of $\frac{10}{7}$ unit on every machine, such that the maximum effective load (on the last machine) would be $\frac{10}{7}\ \frac{1-\frac{1}{3^7}}{1-\frac{1}{3}}$, which is equal to 2.142. The percentage savings would be 6.6%.

In Figure 3.1 we show the percentage reduction in effective load for different values of m and K.

From Theorem 1, we can see that the savings fall off with increasing m. In hindsight, this is obvious because, as the number of machines piled up on one rack increases, and we can only increase the load of the first machine (near the cold air source), so the decrease in effective load will reduce. Of course the precise reduction is a function of the recirculation model we use.

The savings in maximum effective load translates into savings in energy of the cooling system. Let the maximum effective load without thermal aware scheduling be EL_{old} and the one with thermal savings be EL_{new}. We have already noted that that the energy consumed = energy for processing + energy for cooling is

$$E = L(1 + \frac{1}{CoP(T_{sup})}) = L(1 + \frac{1}{CoP(T_{red} - EL_{max})}) \qquad (3.1.1)$$

where L is the total load, and CoP is a super-linear function of the supply temperature. Let us assume a simple function for CoP: $CoP(T) = T^{1+\delta}$, for some $\delta > 0$. Therefore, the cooling energy consumption is given by $\frac{L}{(T_{red}-EL_{max})^{1+\delta}}$. Let us denote the cooling energy consumed by a naive strategy splitting the load uniformly be E_{old}, and that consumed by the optimal thermally aware strategy be E_{new}. and their difference by $\Delta E = E_{old} - E_{new}$. The next theorem outlines the fraction of savings in cooling energy by a thermally aware strategy.

Theorem 2. *The fraction of cooling energy that can be saved by an optimal thermally aware strategy is* $\frac{\Delta E}{E_{old}} > \frac{\Delta EL}{T_{red}-EL_{new}}$*, where,* ΔEL *is the savings in effective load,* EL_{new} *is the optimal effective load as generated by the optimal fractional strategy, and* T_{red} *is the red-line temperature.*

Proof. The difference in energy consumed is:
$\Delta E = L\left(\frac{1}{(T_{red}-EL_{old})^{1+\delta}} - \frac{1}{(T_{red}-EL_{new})^{1+\delta}}\right) = \frac{L}{(T_{red}-EL_{old})^{1+\delta}}\left[1 - \left(\frac{1-\frac{EL_{old}}{T_{red}}}{1-\frac{EL_{new}}{T_{red}}}\right)^{1+\delta}\right].$

$$\Delta E = E_{old}\left[1 - \left(\frac{1 - \frac{EL_{new}}{T_{red}} + \frac{EL_{new} - EL_{old}}{T_{red}}}{1 - \frac{EL_{new}}{T_{red}}}\right)^{1+\delta}\right]$$
$$= E_{old}\left[1 - \left(1 - \frac{\Delta EL}{T_{red} - EL_{new}}\right)^{1+\delta}\right]$$

Obviously, $(1-x)^{1+\delta} < 1 - x$, for any $0 < x < 1$ and $0 < \delta < 1$. Substituting, we get: $\frac{\Delta E}{E_{old}} > \left(1 - 1 + \frac{\Delta EL}{T_{red} - EL_{new}}\right) = \frac{\Delta EL}{T_{red} - EL_{new}}$. Hence the savings are directly proportional to the reductions in effective load. □

3.1.2 Maximization of Scheduled Load under Effective Load Constraint

Until now we assumed that the total load L needs to be scheduled, and tried to minimize the cooling energy consumption. The dual problem asks what is the maximum amount (or profit) of load that can be scheduled without violating the red-line temperature, when the supply temperature T_{sup} is fixed. Note that this fixes the effective load capacity of every machine, since both T_{sup} and T_{red} are now constants. We consider machines to be identical, hence all jobs can be assigned to all machines, and the effective load capacity of every machine is c, as determined by T_{sup} and T_{red}.

We next outline an optimal packing strategy for the fractional problem.

Lemma 3. *Let c be the capacity constraint for the processors. An optimal strategy packs the first machine completely, and all other machines to an extent of $c(1 - \frac{1}{K})$.*

Proof. First we show that the above packing strategy is thermally feasible. When the first machine is filled to c, the second machine can only be filled to capacity $c(1-\frac{1}{K})$ in order to not violate the effective load capacity c. If indeed machine 2 is packed up to $c(1-\frac{1}{K})$, note that its effective load becomes c, and hence it now creates an recirculated load of $\frac{c}{K}$ on machine 3. If all the machines $2 \leq j < i$ are filled to $c(1-\frac{1}{K})$, it can be seen from Claim 1 and induction, that the i^{th} machine can be filled up to $c(1-\frac{1}{K})$ without violating feasibility and if filled up to $c(1-\frac{1}{K})$ its effective load due to recirculated heat from earlier machines would be c. Hence this packing is feasible, and we prove its optimality in the following paragraph.

Suppose for the sake of contradiction, the optimal strategy does not pack the first machine to capacity c, but to $c' = c - \delta$, leaving some unused space. The total extra space (for assigning more load without violating feasibility) available across all the other machines because of this would be $\delta(\frac{1}{K} + \frac{1}{K^2} + ... + \frac{1}{K^{m-1}}) < \delta$ since $K \geq 2$. Hence we will never be able to pack more items by leaving unused space in the first machine, in the fractional case. Suppose the first machine is filled to c, and all machines $j < i, j \geq 2$ are filled to $c(1-\frac{1}{K})$, and the i^{th} machine is filled to capacity $c(1-\frac{1}{K}) - \delta$. Then as before, the extra capacity that would be available across all machines after i, would add up to less than δ, hence it would not be profitable to leave any unused space, as long as there is unassigned load. $\square$

Thus in the above setting, the maximum load L that can be scheduled without violating thermal constraints is $c\left(m - \frac{m-1}{K}\right)$.

Theorem 3. *The fraction of extra load we can assign by a thermal aware strategy is* $\frac{1-\frac{1}{K^m}}{m(K-1)} - \frac{1}{K^m}$.

Proof. The total load that can be assigned in the thermal aware strategy is clearly $\frac{c}{K} + (1 - \frac{1}{K})mc$. If we distribute the load L uniformly (naive scheduler), then as shown above the maximum effective load on machine m will be (given by Lemma 2) $\frac{L}{m}\frac{(1-\frac{1}{K^m})}{(1-\frac{1}{K})}$. Since this should be at most c, we get the desired bound on L. Putting it together with the bound on effective load in the thermal aware strategy we get the desired result. $\square$

Note that when $m = 2$ and $K = 2$ we get $\frac{1-\frac{1}{4}}{2} - \frac{1}{4} = \frac{1}{8}$. This is the improvement we observed in the prior example $(\frac{\frac{3}{2}-\frac{4}{3}}{\frac{4}{3}} = \frac{1}{8})$. When $m = 4$ we can assign an extra $\frac{11}{64}$ fraction of load.

3.2 Integral Assignments: The Maximum Profit Problem

We now consider the case where the jobs need to be integrally assigned to machines. In the maximum profit problem, the goal is to maximize the profit by scheduling as many jobs as possible without violating thermal constraints, when the supply temperature is fixed. As in the fractional case, the fixed supply temperature T_{sup} creates a budget or capacity constraint on every machine with respect to its effective load, which we denote as c for all machines. This capacity refers to effective load, hence the actual space available for accommodating the load assigned to a machine may be lower than c in order for the effective load not to exceed c, and violate thermal feasibility.

Recall that in the fractional case, the optimal strategy was to pack the first machine up to c, and each subsequent machine up to $c\left(1 - \frac{1}{K}\right)$, till we run out of machines or load to assign. Since now the jobs have to be assigned integrally, it is possible that the machines cannot be filled completely to occupy the entire space c in the first machine and $c(1 - \frac{1}{K})$ in the other machines. Suppose in some packing strategy, the jobs assigned to the first machine occupied a total space $c' < c$. Consequently, the space available in the second machine is now no longer limited to $c\left(1 - \frac{1}{K}\right)$ for maintaining the effective load capacity, but is slightly more. This extra space might allow us to fit an extra job in this machine. Hence the optimal packing may not have a straightforward pattern of $c, c(1-\frac{1}{K}), ..., c(1-\frac{1}{K})$ anymore. Let an optimal solution pack machine i, $(i \in [1, \ldots, m])$ up to a capacity c_i, where $0 \leq c_i \leq c$ and the effective load at i is $\leq c$ for all i.

Henceforth we refer to the sequence $c_1, c_2, ..., c_m$, as a "pattern" or "layout" of capacities, interchangeably. If $c_1 < c$, we refer to $\delta_1 = c - c_1$ as the gap left in the first machine by a strategy. Similarly, if $c_i < c(1 - \frac{1}{K})$ for $i > 1$, we refer to $\delta_i = c(1 - \frac{1}{K}) - c_i$ as the gap left in the i^{th} machine by the particular packing strategy.

An *underloaded machine* is one for which the *effective load* is at most $c - \Delta$ where Δ is the size of the largest job.

Lemma 4. *Among all optimal solutions, there is way to assign jobs so that if machine i is an underloaded machine, then all machines $j > i$ are also underloaded.*

Proof. Suppose this is not true. In other words there is an optimal solution that has an underloaded machine i and machine $j > i$ such that j is not underloaded. Select the smallest such numbered machine j. We can assume now that $j = i + 1$. Move jobs greedily from machine $i+1$ to machine i until machine i is not underloaded any more. (The assumption on Δ ensures that the packing on i remains feasible at each step.) If the total shifted load is s then the effective load on i goes up by s, and the load on $i+1$ goes down by s. However the effective load on $i+1$ goes up by $\frac{s}{K}$, but the drop in the real load of s compensates for this and the new effective load is only lower than the effective load initially. In a similar way, the effective load of all later machines also decreases. $\square$

NOTE: This lemma holds more generally, for *any function* where the effect of i on j is monotonically decreasing as j gets further away from i.

However, this does not mean, that for any set of jobs with arbitrary sizes, the capacity pattern of any optimal strategy is monotonically decreasing or of any regular form, like the fractional case. The optimal strategy might have a capacity pattern that is quite irregular, decreasing in the middle, and again rising at the ends, or any other arbitrary pattern, and this is the case even if we assume a limit on the sizes of the objects. In fact, when the sizes are allowed to be arbitrary, we can always construct examples, where the optimal strategy might require a completely arbitrary distribution.

The following examples show that even when the maximum allowed object size is fixed, the optimal capacity distribution might follow an irregular and non-monotonic pattern.

Example 1

Let the maximum object size be $\frac{c}{2}$, where c is the effective load limit of every processor. There are three processors. There are three objects of size $\frac{c}{2}$, and two objects of size $\frac{13c}{32}$. Let $K = 4$ in this case. It can be easily verified, that the only way to fit all these objects, and hence achieve maximum profit without violating the thermal constraints is to put two objects of size $\frac{c}{2}$ in processor 1, thus filling it to capacity, then placing the remaining object of size $\frac{c}{2}$ in processor 2, thus leaving a space of $\frac{c}{4}$ here, and putting the remaining two of size $\frac{13c}{32}$ on the third processor, thus filling it to its capacity of effective load. This does not violate the thermal constraints for any processor, as the first one does not have any thermal constraints, the second one still has available space, having an effective load of only $\frac{3c}{4}$, and the effective load on the third one (following the formulation proved in Claim 1) is $\frac{3c}{16} + \frac{13c}{32} + \frac{13c}{32} = c$. Thus the actual capacity pattern of the optimal strategy is $c, \frac{c}{2}, \frac{13c}{16}$.

The following example shows the pattern might be even more complicated having "gaps" (less than the size of any object) in two consecutive machines, which helps to accommodate an extra object in the third one. These gaps get created because objects need to be assigned completely to machines. However, as can be seen in the example, these gaps were necessary in order to fit all the objects without thermal violation.

Example 2

Let $K = 4$. There are 4 processors, we have one object of size c, 3 objects of size $\frac{11c}{48}$, 3 objects of size $\frac{41c}{192}$, and 4 objects of size $\frac{25c}{128}$. An optimal arrangement fitting them all is the object of size c in processor 1, 3 objects of size $\frac{11c}{48}$ in processor 2, 3 objects of size $\frac{41c}{192}$ in processor 3, and 4 objects of size $\frac{25c}{128}$ in processor 4. This gives an actual load capacity layout of $c, \frac{11c}{16}, \frac{41c}{64}, \frac{25c}{32}$. With a little effort it can be seen that this is an optimal packing. The optimal capacity pattern can therefore be quite arbitrary.

We next describe an algorithm for maximizing profit, given an effective load capacity of c and an instance of jobs, $\mathcal{J}$, where each job $j \in \mathcal{J}$ has a thermal size s_j, and profit p_j. Let us denote by Δ be the maximum size of any job: $\Delta = \max_{j \in \mathcal{J}} s_j$. In the following algorithm, we artificially set the actual *load* capacities of the machines to the values as dictated by an optimal fractional packing, and show an existential result, that there exists packing on the modified load capacities which is close to an optimal integral solution. We can now consider the machines with modified capacities as knapsacks, and the input instance $\mathcal{J}$ as a set of items with sizes and profits to be packed in the knapsacks without violating the capacities and maximizing the total profit of items packed. Hence, after setting the machine capacities, Algorithm 1 uses any of the polynomial time approximation schemes (PTAS) for the strongly hard multiple knapsack problem, due to Chekuri and Khanna [10] and Jansen [35], for generating a packing with minimal loss. Let μ be a lower bound on the number of jobs that can be packed in any machine i, once its load capacity has been artificially set to c_i. We assume $\mu \geq \lfloor \frac{c\left(1-\frac{1}{K}\right)}{\Delta} \rfloor \geq 1$. In practice it is much greater than 1 and the approximation guarantee would increase monotonically with μ_i.

Lemma 5. *Algorithm 1 produces a thermally feasible packing.*

Proof. The proof follows from Claim 1. □

Algorithm 1 Algorithm for the One Dimensional Model. Input: c, $\mathcal{J}$

```
1: Set the load capacity of machine 1, as $c_1 = c$, and the load capacity of any
   machine $i$, $i \in [2, \ldots, m]$, as $c_i = c\left(1 - \frac{1}{K}\right)$.
2: Run the PTAS for multiple knapsack using the modified machine capacities on
   the instance $\mathcal{J}$.
```

Theorem 4. *Algorithm 1 produces a* $\frac{\mu}{\mu+1} - O(\epsilon) \geq \frac{1}{2} - O(\epsilon)$ *approximation to an optimal solution for the maximum profit problem on any instance* $\mathcal{J}$*, in polynomial time for any fixed* $\epsilon > 0$.

Proof. First we consider the maximum cardinality (or unit profit) problem, and show that there exists a packing on the chosen capacity pattern in which we can miss out on at most one job per machine compared to an optimal solution. Let us consider an optimal solution OPT. OPT might pack some machines i to an extent lower than c_i. Let us refer to such machines as *underpacked.* Similarly, some machines j may be packed to an extent greater than c_j in OPT. We refer to such machines as *overpacked.* If the optimal solution has packed a machine i up to c'_i, it is said to have a *gap* of $\delta_i = c_i - c'_i$. The gaps may be zero, positive or negative, however, the gaps of interest to us are those which are positive, occurring in underpacked machines.

Let i be the first overpacked machine in OPT, and the sum of sizes of jobs packed in i in OPT be c'_i. (Clearly, for a feasible solution, if there exists any such i, $i > 1$). For this arrangement to be feasible, without violating thermal constraints, some machines $j < i$ must be underpacked with positive gaps.

We repack or reassign the jobs (in arbitrary order) from i to underpacked machines $j < i$ with gaps using some heuristic such as *First-Fit*, such that the thermal constraints are not violated. Specifically, as long as i is overpacked, and there exists a job k in i, such that $s_k \leq \delta_j$ for any underpacked machine $j < i$, we reassign k to j from i, and adjust the capacities and gaps accordingly. Note that by this reassignment, the total effective load on machine j does not exceed c, as $c_j - \delta_j + s_k \leq c_j$.

If i is no longer overpacked, then we move to the next overpacked machine after i and repeat this process. Otherwise, i is still overpacked, but we are unable to move any more jobs from i to any $j < i$ due to size constraints. This implies that among all the jobs remaining in i, the smallest size job $s_{i,min} > \delta_{max}$, where $\delta_{max} = \max_{j<i} \delta_j$, ($\delta_j$ denotes the gap in j after the reassignments). However, from Claim 1, this implies that the capacity to which i is packed currently must be $c'_i \leq c\left(1 - \frac{1}{K}\right) + \delta_{max}$. This is because, the extra space over and above c_i in i is the largest when all of the machines $[1, \ldots, (i-1)]$ are underpacked with each having a gap of δ_{max}. Therefore, the effective load in machine i is: $c'_i + \sum_{j<i} \frac{c_j - \delta_{max}}{K^{i-j}} \leq c$, the current packing being feasible. On substituting c_j for all $j < i$, we get the bound on c'_i for $K \geq 2$. Therefore, if we remove any job from i, it will become underpacked. We discard any arbitrary job from i and proceed to the next overpacked machine.

By the above procedure, the effective load on i can not increase. This is because, the new effective load created by a job of size k on being reassigned to a machine $j < i$, is $\frac{s_k}{K^{i-j}} < s_k$, where s_k is the drop in the local load of i. This implies that the feasibility in later machines $i + d, d \geq 1$ are not affected by this process. That is because, from Claim 1 and the definition of the one-dimensional model, we can express $EL_{i+d} = L_{i+d} + \sum_{j=i+1}^{i+d-1} \frac{L_j}{K^{i+d-j}} + \frac{EL_i}{K^d}$, and we have not changed the loads of machines $[(i+1), \ldots, (i+d-1)]$. Hence, we now move to the next overpacked machine after i and repeat the above procedure.

At the end, we would have no overpacked machines, and we have lost at most one job per machine. Since $\mu \geq 1$, any overpacked machine must have contained ≥ 2 jobs. Therefore, in the above procedure, when we discard a job from an overpacked machine, we will still have one job in it, and hence at the end, we get at least half the number of jobs packed by OPT. In the profit case, while discarding a job from an overpacked machine, we choose the least profit job from the machine, thus ensuring that for every job we discard, another distinct job of equal or greater profit remains in the packing, to which we can charge the profit of the discarded job. No job is charged more than once, since according to the repacking procedure described earlier, once a job is discarded from a machine, it is no longer overpacked. Hence, at the end, we get at least half the profit of OPT.

If we had access to an optimum multiple knapsack oracle, our approximation would have been $\geq \frac{\mu}{\mu+1} \geq \frac{1}{2}$. However, this being a strongly NP-hard problem, we use a PTAS for multiple knapsack([10], [35]) to get a $\frac{\mu}{\mu+1} - O(\epsilon) \geq \frac{1}{2} - O(\epsilon)$ approximation to OPT in polynomial time for any fixed $\epsilon > 0$.

$\square$

The factor of $\frac{1}{2}$ is asymptotically tight as can be seen in the following example. Let the effective load capacity of any machine be 1 and there are m machines and $2m$ jobs. The optimal solution packs 2 objects of size $\frac{1-\varepsilon}{2}$ in machine 1, 2 objects of size $\frac{1-\frac{1}{K}}{2} + \frac{\varepsilon}{4K}$ in machine 2, 2 objects of size $\frac{1-\frac{1}{K}}{2} + \frac{\varepsilon}{8K^2}$ in machine 3 and so on, till on machine m it packs 2 jobs of size $\frac{1-\frac{1}{K}}{2} + \frac{\varepsilon}{2^m K^{m-1}}$. Algorithm 1 would miss one object from each of machine $[2 \ldots m]$, hence it will give an approximation $\frac{m+1}{2m} = \frac{1}{2} + \frac{1}{m} \approx \frac{1}{2}$ when m is large.

3.3 Integral Assignments: The Minimum Thermal Makespan Problem

In this section we consider the dual problem where we need to assign all jobs and the objective is to minimize the maximum effective load or *thermal makespan* on any machine. We had earlier analyzed this problem where fractional assignments are allowed, while here we only allow integral assignments. We analyze both offline and online variants of this problem for a single rack of machines (1D model).

Formally, there are m machines and and n jobs. The objective is to *minimize* $\max_{i \in [1 \ldots m]} EL_i$ while scheduling all the jobs. This problem is obviously NP-hard for general m and n as it is a generalization of the minimum makespan problem.

Note that the maximum effective load may be on a machine that may not have the maximum load assigned among all machines. For example, suppose $m = n \geq 2$, $(m-1)$ of the jobs are of unit size and assigned to machines $[1 \ldots (m-1)]$, and one job is of size $1 - \epsilon$, assigned to machine m. The machine m therefore has the lowest load among all machines. However, the effective load on m is $\frac{1-\frac{1}{K^m}}{1-\frac{1}{K}} - \epsilon$. It can be verified that if $\epsilon < \frac{1}{K^{m-1}}$, then machine m has the largest effective load.

3.3.1 NP-hardness

The problem of minimizing maximum effective load obviously generalizes the minimum makespan problem, which is strongly NP-hard. However, when the number of jobs is less than or equal to the number of machines, the minimum makespan problem is not NP-hard, and any arbitrary arrangement of the jobs will be optimal.

However, the minimum effective load problem does not have any such trivial solution for the case of $n \leq m$. In fact, we show that the problem remains NP-hard even when $n = m$. The reduction is from another scheduling problem inspired by thermal issues, studied by Chrobak et al. [12]. In their model, they have a single machine, time is considered to be slotted and the input is a set of unit length jobs of unit profit, with release times, deadlines, and arbitrary heat contributions (analogous to heat sizes of jobs in our model). In order to get the profit of a job, it needs to be scheduled within its feasible window for a unit length of time. There is temporal drop-off of temperature, specifically, if at the time a job of heat size h is scheduled, the temperature of the machine is τ, then after the execution of the job, the temperature of the machine is $\frac{\tau+h}{2}$. They have a hard constraint on the maximum temperature that the machine can reach at any time step, which is normalized to 1, and the goal is to schedule as many jobs as possible without violating the temperature constraint at any time step. They show that even when all jobs have the same release times and deadlines, it is NP-hard to maximize the number of jobs scheduled, by a reduction from numerical 3D matching.

We show that minimizing the maximum effective load on any machine (in the one-dimensional model with $K = 2$) is NP-hard by a reduction from the problem studied by Chrobak et al. [12]. The crucial observation is that the time axis can be interpreted to be the space axis; specifically, every time slot can be interpreted as a separate machine. Chrobak et al. [12] show NP-hardness for an instance where there are m jobs, each with release time 0 and deadline m, and heat contribution h_j for job j. We create an instance of the minimum thermal makespan problem from this instance by creating m jobs, with each jobs heat size $s_j = \frac{h_j}{2}$, and m machines, with spatial cross-interference coefficient $K = 2$. It can be seen that if there exists a schedule for the instance of Chrobak et al. [12] with throughput m, there exists a schedule of jobs in the one-dimensional model such that the maximum effective load on any machine is 1. Similarly, if there exists a schedule in the one-dimensional model with maximum effective load on any machine $= 1$, there exists a schedule for the instance of Chrobak et al. of throughput $= m$.

Therefore, the following theorem follows from the work of Chrobak et al. [12].

Theorem 5. *The offline problem of minimizing the maximum effective load for the one-dimensional case is strongly NP-hard even when the number of jobs n is equal to the number of machines m.*

3.3.2 Offline Algorithm

We show that applying Graham's LPT scheduling [30] algorithm gives a $\max\left(\frac{K}{K-1}, \frac{4K-3}{3K-3}\right)$ approximation to the minimum thermal makespan problem. This analysis is tight both for $K = 2$ which is the minimum value of K, as well as asymptotically, since for no cross-effects, or $K \to \infty$, it is well known that LPT gives a $\frac{4}{3}$ approximation.

The algorithm is formalized below. The intuition for favoring lower indices is that these machines are closer to the source of cold air. We denote the load on machine k as L_k.

Algorithm 2 Algorithm using load as the decision metric

```
1: Sort and order jobs in a list in non-increasing sizes.
2: Assign the next job on the list to machine $k$ such that, $L_k \leq L_j \; \forall j \in [1 \ldots m]$
   and $L_k < L_p \; \forall p < k$.
```

Theorem 6. *Algorithm 2 achieves an approximation ratio of* $\max(\frac{K}{K-1}, (\frac{4K-3}{3K-3} + \frac{1}{3m}))$ *for* $2 \leq K < 3$ *and* $\frac{4K-3}{3K-3}$ *for* $K \geq 3$.

Proof. Let i be the machine with the maximum effective load after all jobs have been assigned. Machine i was assigned $p \geq 1$ jobs. Let the size of the last job assigned to this machine be $s_{i,p}$. We denote the optimal solution cost as OPT. Consider the iteration when $s_{i,p}$ was placed on i. The following hold in this iteration: 1) $L_i \leq L_j \; \forall j \in [1 \ldots m]$, 2) all jobs assigned so far are larger in size than $s_{i,p}$. So, if $p \geq 3$, obviously $OPT \geq 3s_{i,p}$. We will consider two cases separately: 1) $s_{i,p} > \frac{OPT}{3}$ and 2) $s_{i,p} \leq \frac{OPT}{3}$.

Case 1. $s_{i,p} > \frac{OPT}{3}$

As argued above, this case is possible only when $p = 1$ or $p = 2$. In this case, we will show that $EL_i \leq \frac{K}{K-1} OPT$.

Suppose $p = 2$; consider the iteration when $s_{i,2}$ was placed on i. The following claims are true at that iteration.

Claim 2. $L_j = s_{j,1} \geq s_{i,1}\ \forall j \leq i$

Proof. Algorithm 2 places $s_{i,2}$ on the machine with the minimum load, favoring lower indices. Hence, $L_j > s_{i,1}\ \forall j < i$. Moreover, each $j < i$ could have only received a single job so far, since Algorithm 2 considers jobs in non-increasing size order, and assigns them to lowest load machines, favoring lower indices. Hence, jobs larger than $s_{i,1}$ were placed on machines $j < i$. □

Claim 3. *Either* $L_k = s_{k,1} = s_{i,1}$ *or* $L_k \geq 2s_{i,2}\ \forall k > i$.

Proof. Since Algorithm 2 placed $s_{i,2}$ on i, all machines $L_k \geq s_{i,1}\ \forall k > i$. However, for each such machine, $s_{k,1} \leq s_{i,1}$. If some $k > i$ has only one job, then $s_{k,1} = s_{i,1}$, otherwise, $s_{i,2}$ would have been placed on k. On the other hand, if k has ≥ 2 jobs, since these jobs were placed earlier than $s_{i,2}$, $L_k \geq 2s_{i,2}$. □

Lemma 6. *If* $p = 1$ *or* $p = 2$, $L_i \leq OPT$.

Proof. It is obvious for $p = 1$. For $p = 2$, we use the above claims to prove the lemma. If $i = m$ then obviously $L_i = L_m \leq OPT$. This follows from Claim 2. Hence, let us assume $i < m$. Let the number of machines $k > i$ with single jobs be ℓ. Therefore, from Claim 2, we know that, in OPT, there are $i + \ell$ jobs of size $\geq s_{i,1}$. Let us call them *big* jobs. (Note that if $i + \ell = m$, once again, trivially, $L_i \geq OPT$, hence, we assume $i + \ell < m$.)

We know from Claim 3 that there are at least $2(m-(i+\ell))+1$ jobs of size $\geq s_{i,2}$ that OPT would need to assign. Let us call this set as *small* jobs. If the optimal solution paired any of the big jobs with the small jobs, then clearly, $L_i \leq OPT$. Hence, we assume that none of the big jobs were paired with any of the small jobs in the optimal solution. This implies that the small jobs must have been distributed among $(m-(i+\ell))$ machines. However, that requires at least one of these machines to have load $\geq 3s_{i,2}$, which implies, $OPT \geq 3s_{i,2}$, which is a contradiction. Therefore, $L_i \leq OPT$. □

From Claim 1, we know $EL_i = L_i + \frac{EL_{i-1}}{K}$. Since by assumption, $EL_i \geq EL_j \forall j$, $EL_{max} \leq L_i + \frac{EL_{max}}{K}$. Applying $L_i \leq OPT$, we get, $EL_{max} \leq \frac{K}{K-1} OPT$.

Case 2. $s_{i,p} \leq \frac{OPT}{3}$.

Consider the iteration in which $s_{i,p}$ was assigned to i. Let us denote the load on any machine j in this iteration as L'_j. Since LPT assigned $s_{i,p}$ to i for job $s_{i,p}$, $L'_i \leq L'_j \; \forall j$. Hence, $L'_i \leq \frac{\sum_{j=1}^{m} L'_j}{m}$. The total load being L, we have $L'_i \leq \frac{L-s_{i,p}}{m}$. The effective load on i is $EL_{max} \leq L'_i + s_{i,p} + \frac{EL_{max}}{K}$. Substituting for L'_i, we get

$$\begin{aligned} EL_{max} &\leq \frac{K}{K-1}\left(\frac{L-s_{i,p}}{m} + s_{i,p}\right) \\ &= \frac{K}{K-1}\left(\frac{L}{m} + s_{i,p}\left(1-\frac{1}{m}\right)\right). \end{aligned}$$

We know $s_{i,p} \leq \frac{OPT}{3}$. From Claim 1, the minimum effective load for the one-dimensional system when jobs can be distributed fractionally is $EL = \frac{L}{m-\frac{m-1}{K}}$. Therefore this is a lower bound on OPT. Applying these lower bounds on OPT, we get

$$EL_{max} < \frac{K}{K-1}OPT\left(1-\frac{1}{K}+\frac{1}{mK}+\frac{1}{3}-\frac{1}{3m}\right)$$
$$= OPT\left(1+\frac{K}{3(K-1)}-\frac{K-3}{3m(K-1)}\right).$$

Therefore, $EL_{max} \leq OPT\left(\frac{4K-3}{3K-3}+\frac{1}{3m}\right)$ for $2 \leq K < 3$ and $EL_{max} \leq OPT\left(\frac{4K-3}{3K-3}\right)$ for $K \geq 3$. $\square$

For $K = 2$, $\frac{K}{K-1} = 2 \geq \left(\frac{4K-3}{3K-3}+\frac{1}{3m}\right)$. For this case, there is a tight example. Let instance I have a very large number of machines $m \to \infty$. Let the number of jobs be very large n, however, $n << m$ and all jobs are of unit size. The optimal strategy would space out the jobs with one job on the first machine, one on the last machine, and the rest distributed sparsely such that, the effective load on any machine is $\leq (1+\epsilon)$, where $\epsilon \to 0$. This will be possible if $n << m$. However, our algorithm will place the jobs on the n consecutive machines, resulting on a maximum effective load on the n^{th} machine which is $\frac{1-\frac{1}{K^n}}{1-\frac{1}{K}} \approx \frac{K}{K-1} = 2$ for very large n. Hence the approximation is $\approx 2 - o(\epsilon)$.

For $K \geq 3$, $\frac{K}{K-1} \leq \frac{4K-3}{3K-3}$. For higher values of K, the approximation tends to $\frac{4}{3}$, which is a tight approximation factor for minimum makespan problem as well. Hence this analysis is tight.

3.3.3 Online Algorithm

Here we consider the above problem in an online setting. Specifically, we have m machines, and the jobs arrive in an online fashion. Once a job arrives, we have to assign it to a machine and the decision is irrevocable. The objective is to minimize the maximum effective load or the thermal makespan. We assume the jobs are long-lasting, and hence ignore any temporal effects.

We show Graham's List Scheduling algorithm gives a $\frac{2K-1}{K-1} - \frac{1}{m}$ approximation to the online problem of minimizing thermal makespan. The algorithm is simple. When a job arrives, assign it to the machine with the minimum load.

Theorem 7. *Graham's list scheduling algorithm gives a* $\frac{2K-1}{K-1} - \frac{1}{m}$ *approximation to the online problem of minimizing thermal makespan.*

Proof. Let the machine with the largest effective load EL_{max} be i. Let the last job assigned to this machine be s_i and the load on i before assigning s_i be L_i. We know $EL_{max} \leq L_i + s_i + \frac{EL_{max}}{K}$, or, $EL_{max} \leq \frac{K}{K-1}(L_i + s_i)$. Obviously, $s_i \leq OPT$. When s_i was assigned to i, $L_i \leq L_j \; \forall j \in [1 \ldots m]$. Hence, $L_i \leq \frac{\sum_{j \in [1 \ldots m]} L_j}{m}$. If the total load to be assigned is L, we have $L_i \leq \frac{L - s_i}{m}$. We know, from Claim 1, $OPT \leq \frac{L}{m - \frac{m-1}{K}}$. Applying the lower bounds on OPT, we get,

$$\begin{aligned} EL_{max} &\leq \frac{K}{K-1} OPT \left(1 - \frac{1}{K} + \frac{1}{mK} + 1 - \frac{1}{m}\right) \\ &\leq OPT \left(\frac{2K-1}{K-1} - \frac{1}{m}\right). \end{aligned}$$

□

The analysis is essentially tight, since it is well known that for no cross effects, or $K \to \infty$, list scheduling gives a $2 - \frac{1}{m}$ approximation, which is what we get asymptotically.

Chapter 4: Multiple Racks of Machines: Two Dimensional Models

In this chapter we discuss the three different two-dimensional thermal models, introduced in Chapter 2. For each of these models, we derive optimal fractional load distribution strategies so as to minimize maximum effective load, as we did in the previous chapter. From this analysis it is easy to derive the corresponding three theorems that bound: (a) how much extra load can be assigned due to our strategy compared to a naive load distribution strategy subject to a thermal constraint, (b) the reduction in maximum effective load on a machine for a given load that needs to be distributed, and (c) the savings in cooling costs. In Section 4.2 we discuss the NP-hard problem of maximizing profit of assigned jobs, in the presence of hard thermal constraints, when only integral assignments are allowed. We show how to develop approximation algorithms for assigning jobs by fixing the effective load capacities for each machine, which lets us now completely ignore thermal constraints and reduces the problem (as before) to a multiple knapsack problem.

4.1 Fractional Assignments

4.1.1 Two Dimensional General Model

In this model, the heat redistribution effect of the load on the $(i', j')^{th}$ machine on the $(i, j)^{th}$ machine is $\frac{L_{i'j'}}{{K_1}^{j-j'}\ {K_2}^{i-i'}}$. That is, the effective load on the $(i, j)^{th}$ machine is

$$EL_{i,j} = \sum_{i'=1}^{i}\sum_{j'=1}^{j} \frac{L_{i',j'}}{{K_1}^{j-j'}\ {K_2}^{i-i'}} \tag{4.1.1}$$

Claim 4. *The effective load $EL_{i,j}$ in for the general model can be given as $EL_{i,j} = L_{i,j} + \frac{EL_{i,j-1}}{K_1} + \frac{EL_{i-1,j}}{K_2} - \frac{EL_{i-1,j-1}}{K_1\ K_2}$.*

Proof. For the first rack, that is $i = 1$, the machines do not experience any lateral heating effect, since there are no racks beyond this rack. So the first rack machines are exactly like the 1-D case, where we already proved, $EL_j = L_j + \frac{EL_{j-1}}{K}$. Similarly, here it would be $EL_{1,j} = L_{1,j} + \frac{EL_{1,j-1}}{K_1}$.

From Equation (4.1.1), $EL_{i,j} = \sum_{i'=1}^{i}\sum_{j'=1}^{j} \frac{L_{i',j'}}{{K_1}^{j-j'}\ {K_2}^{i-i'}}$. Similarly, $EL_{i,j-1} = \sum_{i'=1}^{i}\sum_{j'=1}^{j-1} \frac{L_{i',j'}}{{K_1}^{j-j'-1}\ {K_2}^{i-i'}}$. Hence,

$$\begin{aligned} EL_{i,j} &= \frac{EL_{i,j-1}}{K_1} + \sum_{i'=1}^{i} \frac{L_{i',j}}{{K_2}^{i-i'}} \\ &= \frac{EL_{i,j-1}}{K_1} + L_{i,j} + \frac{L_{i-1,j}}{K_2} + \sum_{i'=1}^{i-2} \frac{L_{i',j}}{{K_2}^{i-i'}} \end{aligned} \tag{4.1.2}$$

Again, from Equation (4.1.1), $EL_{i-1,j} = \sum_{i'=1}^{i-1}\sum_{j'=1}^{j} \frac{L_{i',j'}}{{K_1}^{j-j'}\ {K_2}^{i-i'-1}}$. Therefore, $L_{i-1,j} = EL_{i-1,j} - \sum_{i'=1}^{i-1}\sum_{j'=1}^{j-1} \frac{L_{i',j'}}{{K_1}^{j-j'}\ {K_2}^{i-i'-1}} - \sum_{i'=1}^{i-2} \frac{L_{i',j}}{{K_2}^{i-i'-1}}$. Hence we can write, $L_{i-1,j} = EL_{i-1,j} - \frac{EL_{i-1,j-1}}{K_1} - \sum_{i'=1}^{i-1} \frac{L_{i',j}}{{K_2}^{i-i'-1}}$. Substituting in Equation (4.1.2), we have,

$$\begin{aligned} EL_{i,j} &= L_{i,j} + \frac{EL_{i,j-1}}{K_1} + \frac{EL_{i-1,j}}{K_2} - \frac{EL_{i-1,j-1}}{K_1\ K_2} - \sum_{i'=1}^{i-2} \frac{L_{i',j}}{{K_2}^{i-i'}} + \sum_{i'=1}^{i-2} \frac{L_{i',j}}{{K_2}^{i-i'}} \\ &= L_{i,j} + \frac{EL_{i,j-1}}{K_1} + \frac{EL_{i-1,j}}{K_2} - \frac{EL_{i-1,j-1}}{K_1\ K_2} \end{aligned}$$

Therefore, we have proved that $EL_{i,j} = L_{i,j} + \frac{EL_{i,j-1}}{K_1} + \frac{EL_{i-1,j}}{K_2} - \frac{EL_{i-1,j-1}}{K_1\ K_2}$ □

Lemma 7. *Any optimal strategy for minimizing the maximum effective load when fractional assignments are allowed would result in uniform effective load of* $\frac{L}{\left(m_1 - \frac{m_1-1}{K_1}\right)\left(m_2 - \frac{m_2-1}{K_2}\right)}$, *where L is the total load to be assigned.*

Proof. An optimal strategy for minimizing the maximum effective load for fractional assignments, would distribute the load so as to make the effective load uniform across all the machines. When the effective load is equal for all machines, then from Claim 4, for a machine $(i,j)\,, i > 1, j > 1$, the load is $L_{i,j} = EL(1 - \frac{1}{K_1} - \frac{1}{K_2} + \frac{1}{K_1 K_2})$, where EL is the uniform effective load across all machines. For the first rack $i = 1, j > 1$, $L_{1,j} = EL(1 - \frac{1}{K_1})$. For the first machine in each rack, that is $j = 1, i > 1$, $L_{i,1} = EL(1 - \frac{1}{K_2})$. For the first machine in the first rack $L_{1,1} = EL$. Summing up the load across all the machines, the total load L is equal to

$$\begin{gathered} L = EL\ \left(1 + (m_1 - 1)\left(1 - \frac{1}{K_1}\right) + (m_2 - 1)\left(1 - \frac{1}{K_2}\right) + \right. \\ \left. (m_1 - 1)(m_2 - 1)\left(1 - \frac{1}{K_1} - \frac{1}{K_2} + \frac{1}{K_1 K_2}\right)\right) \end{gathered}$$

Factorizing, we can see that the above equation is equivalent to

$$L = EL\ \left(1 + (m_1 - 1)\left(1 - \frac{1}{K_1}\right)\right)\ \left(1 + (m_2 - 1)\left(1 - \frac{1}{K_1}\right)\right)$$

In other words, $EL = \frac{L}{\left(m_1 - \frac{m_1-1}{K_1}\right)\left(m_2 - \frac{m_2-1}{K_2}\right)}$. □

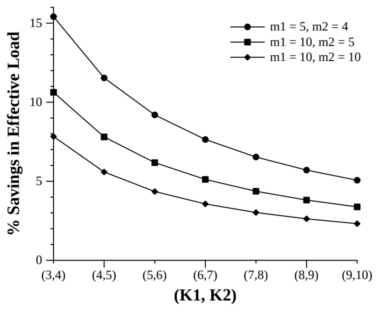


Figure 4.1: Percentage reduction in maximum effective load across all machines as $(K1, K2)$ vary, for different values of m_1, m_2

Lemma 8. *A naive strategy minimizing the maximum load for total load L, would split the load uniformly and result in maximum effective load*
$EL = \frac{L}{m_1\ m_2} \frac{1-\frac{1}{{K_1}^{m_1}}}{1-\frac{1}{K_1}} \frac{1-\frac{1}{{K_2}^{m_2}}}{1-\frac{1}{K_2}}$

Proof. A naive strategy which splits the load uniformly across all machines will result in maximum effective load at the $(m_2, m_1)^{th}$ position due to all the other machines in the grid. The effective load on (m_2, m_1) is

$$\begin{aligned} EL_{m_2,m_1} &= \sum_{i=1}^{m_2}\sum_{j=1}^{m_1} \frac{L}{m_1 m_2} \frac{1}{{K_2}^{m_2-i}\ {K_1}^{m_1-j}} \\ &= \frac{L}{m_1\ m_2} \frac{1-\frac{1}{{K_1}^{m_1}}}{1-\frac{1}{K_1}} \frac{1-\frac{1}{{K_2}^{m_2}}}{1-\frac{1}{K_2}} \end{aligned}$$

□

4.1.2 Two Dimensional Horizontal Sibling Model

In this section the machines in each rack are numbered starting from 0 and the racks are also numbered starting from 0. In this model, the effective load on the

$(i,j)^{th}$ machine is given as follows.

$$EL_{i,j} = L_{i,j} + \sum_{\ell=0}^{j-1} \frac{L_{i,\ell}}{{K_1}^{j-\ell}} + \frac{L_{i-1,j}}{K_2}.$$

Claim 5. *The effective load on the $(i,j)^{th}$ machine can be expressed as*

$$\begin{aligned} EL_{i,j} &= L_{i,j} + \frac{EL_{i,j-1}}{K_1} \\ &\quad - \sum_{l=0}^{i-1} \left(\frac{-1}{K_2}\right)^{i-l} \sum_{p=0}^{\min(j,i-l+1)} \left(\frac{-1}{K_1}\right)^{p} \binom{i-l+1}{p} EL_{l,j-p} \end{aligned}$$

Proof. We prove this claim by induction. The base case of $i = 0$ is already proved, by claim 1, where we proved that for a single rack of machines, the effective load on the j^{th} machine is given by $EL_j = L_j + \frac{EL_{j-1}}{K}$. For the first rack, $i = 0$, it is equivalent to a single rack of machines since there is no effect from the top. So replacing j by $(0,j)$ and K by K_1, we get $EL_{0,j} = L_{0,j} + \frac{EL_{0,j-1}}{K_1}$ which is exactly what is given by the formula for $i = 0$. Also, we can verify the case of $j = 0$ or the first column. By the definition of our model, the effective load on $(i,0)^{th}$ machine is given by $EL_{i,0} = L_{i,0} + \frac{L_{i-1,0}}{K_2}$. Let us assume the induction hypothesis is true for all $q \le i-1$, that is $EL_{q,0} = L_{q,0} - \sum_{l=1}^{q-1} \left(\frac{-1}{K_2}\right)^{q-l} EL_{l,0}$. So, by applying the induction hypothesis we get,

$$\begin{aligned} EL_{i,0} &= L_{i,0} + \frac{EL_{i-1,0}}{K_2} + \frac{1}{K_2} \sum_{l=0}^{i-2} \left(\frac{-1}{K_2}\right)^{i-l-1} EL_{l,0} \\ &= L_{i,0} - \sum_{i-1}^{l=0} \left(\frac{-1}{K_2}\right)^{i-l} EL_{l,0} \end{aligned}$$

This proves it for all $j = 0$.

Now, let us assume by strong induction hypothesis, that the claim is true for all machines of all racks before the i^{th} rack and all machines before the j^{th} machine of the i^{th} rack, that is the claim is true for (q, r), where $0 \leq q < i, 0 \leq r < m$, and (i, s), where $0 \leq s < j$. We will prove that in this case, the claim is true for $(i, j)^{th}$ machine as well. Since the base case of the first machine of any rack of i has already been proved, this would prove our claim by mathematical induction.

By the definition of the model, we have,

$$EL_{i,j} = L_{i,j} + \frac{L_{i,j-1}}{K_1} + \frac{L_{i,j-2}}{K_1^2} + ... + \frac{L_{i,0}}{K_1^j} + \frac{L_{i-1,j}}{K_2}$$

Substituting in the above equation, $L_{i,j-1} = EL_{i,j-1} - \sum_{l=0}^{j-2} \frac{L_{i,l}}{{K_1}^{j-1-l}} - \frac{L_{i-1,j-1}}{K_2}$, (which follows from the definition of the model) we get,

$$EL_{i,j} = L_{i,j} + \frac{EL_{i,j-1}}{K_1} - \frac{L_{i-1,j-1}}{K_1\ K_2} + \frac{L_{i-1,j}}{K_2} \tag{4.1.3}$$

Now, we apply the induction hypothesis and substitute for $L_{i-1,j-1}$ and $L_{i-1,j}$. By induction hypothesis,

$$\begin{aligned} L_{i-1,j} = EL_{i-1,j} &- \frac{EL_{i-1,j-1}}{K_1} \\ &- \sum_{l=0}^{i-2} \left(\frac{-1}{K_2}\right)^{i-l-1} \sum_{p=0}^{\min(j,i-l)} \left(\frac{-1}{K_1}\right)^p \binom{i-l}{p} EL_{l,j-p} \end{aligned} \tag{4.1.4}$$

$$\begin{aligned} L_{i-1,j-1} = EL_{i-1,j-1} &- \frac{EL_{i-1,j-2}}{K_1} \\ &- \sum_{i-2}^{l=0} \left(\frac{-1}{K_2}\right)^{i-l-1} \sum_{p=0}^{\min(j-1,i-l)} \left(\frac{-1}{K_1}\right)^p \binom{i-l}{p} EL_{l,j-1-p} \end{aligned} \tag{4.1.5}$$

Therefore, from equations 4.1.4 and 4.1.5, $\frac{L_{i-1,j}}{K_2} - \frac{L_{i-1,j-1}}{K_1 K_2} =$

$$\begin{aligned} &= \frac{EL_{i-1,j}}{K_2} - \frac{2EL_{i-1,j-1}}{K_1 K_2} + \frac{EL_{i-1,j-2}}{{K_1}^2 K_2} \\ &+ \frac{1}{K_2} \sum_{l=0}^{i-2} \left(\frac{-1}{K_2}\right)^{i-l-1} \left(\sum_{p=1}^{\min(j,i-l)} \left(\frac{-1}{K_1}\right)^p \left(\binom{i-l}{p} + \binom{i-l}{p-1} \right) EL_{l,j-p} \right) \\ &+ \frac{1}{K_2} \sum_{l=0}^{i-2} \left(\frac{-1}{K_2}\right)^{i-l-1} \left(EL_{l,j} + \left(\frac{-1}{K_1}\right)^{i-l+1} EL_{l,j-i+l-1} \right) \end{aligned} \tag{4.1.6}$$

In the above equation we assume $j \geq i - l + 1$ for all l. When $j = i - l$ or $j < i - l$, the same proof works with $i - l + 1$ replaced by j. The above equation can be simplified to

$$\frac{L_{i-1,j}}{K_2} - \frac{L_{i-1,j-1}}{K_1K_2} = \frac{1}{K_2}\sum_{q=0}^{2} EL_{i-1,j-q}\binom{2}{q}\left(\frac{-1}{K_1}\right)^q$$
$$+ \frac{1}{K_2}\sum_{l=0}^{i-2}\left(\frac{-1}{K_2}\right)^{i-l-1}\sum_{p=0}^{\min(j,i-l+1)}\left(\frac{-1}{K_1}\right)^p\binom{i-l+1}{p}EL_{l,j-p}$$

where we have used the identity $\binom{a}{b} + \binom{a}{b-1} = \binom{a+1}{b}$. Therefore we get

$$\frac{L_{i-1,j}}{K_2} - \frac{L_{i-1,j-1}}{K_1K_2} = -\sum_{l=0}^{i-1}\left(\frac{-1}{K_2}\right)^{i-l}\sum_{p=0}^{\min(j,i-l+1)}\left(\frac{-1}{K_1}\right)^p\binom{i-l+1}{p}EL_{l,j-p} \quad (4.1.7)$$

Hence substituting equation 4.1.7 in equation 4.1.3,

$$EL_{i,j} = L_{i,j} + \frac{EL_{i,j-1}}{K_1}$$
$$-\sum_{l=0}^{i-1}\left(\frac{-1}{K_2}\right)^{i-l}\sum_{p=0}^{\min(j,i-l+1)}\left(\frac{-1}{K_1}\right)^p\binom{i-l+1}{p}EL_{l,j-p}$$

Hence we have proved our claim by induction.

$\square$

When we want to minimize the effective load, as before we want to make it uniform all over. The load distribution for uniform effective load of EL all over would be:

$L_{0,0} = EL$, $L_{0,j} = EL\left(1 - \frac{1}{K_1}\right)$,
$L_{i,0} = EL\left(1 + \sum_{l=0}^{i-1}\left(\frac{-1}{K_2}\right)^{i-l}\right)$, and for $i > 0, j > 0$
$L_{i,j} = EL\left(1 - \frac{1}{K_1} + \sum_{l=0}^{i-1}\left(\frac{-1}{K_2}\right)^{i-l}\sum_{p=0}^{\min(j,i-l+1)}\binom{i-l+1}{p}\left(\frac{-1}{K_1}\right)^p\right)$.

The total load $L = \sum_{i=0}^{m_2-1} \sum_{j=0}^{m_1-1} L_{i,j}$. Summing up we get,

$$L = EL \sum_{i=0}^{m_2-1} \left(\frac{-1}{K_2}\right)^i (m_2 - i) \left((m_1 - i - 1) \left(1 - \frac{1}{K_1}\right)^{i+1} \right.$$
$$\left. + \sum_{j=0}^{i} (m_1 - j) \binom{i+1}{j} \left(\frac{-1}{K_1}\right)^j \right)$$

Therefore the effective load EL that results by thermal aware placement of load L is

$$EL = L / \left(\sum_{i=0}^{m_2-1} \left(\frac{-1}{K_2}\right)^i (m_2 - i) \right.$$
$$\left. \left((m_1 - i - 1) \left(1 - \frac{1}{K_1}\right)^{i+1} + \sum_{j=0}^{i} (m_1 - j) \binom{i+1}{j} \left(\frac{-1}{K_1}\right)^j \right) \right)$$

Distributing the load uniformly as $\frac{L}{m_1 m_2}$ on every machine, the maximum effective load will result on the topmost machine of every rack after the 0^{th} rack, and is denoted as $EL_{ob} = \frac{L}{m_1 m_2} \left[1 + \frac{1}{K_1} + \frac{1}{K_1{}^2} + \ldots + \frac{1}{K_1{}^{m_1-1}} + \frac{1}{K_2} \right]$.

$$EL_{ob} = \frac{L}{m_1 m_2} \left[\frac{1 - \frac{1}{K_1{}^{m_1}}}{1 - \frac{1}{K_1}} + \frac{1}{K_2} \right]$$
$$= \frac{L}{m_1 m_2} \left[\frac{K_1{}^{m_1} - 1}{K_1{}^{m_1-1} (K_1 - 1)} + \frac{1}{K_2} \right]$$

4.1.3 Two Dimensional Indirect Sibling Model

In this model, unlike the above two models, we define the effective load on a machine in terms of the *effective* loads of their neighbors, not the actual loads. The effective load of machine (i, j) depends on the effective load of its two immediate neighbors: machine $(i, j-1)$ on the same rack, and machine $(i-1, j)$ in the adjacent rack in the same row as (i, j). Specifically,

$$EL_{i,j} = L_{i,j} + \frac{EL_{i,j-1}}{K_1} + \frac{EL_{i-1,j}}{K_2}. \qquad (4.1.8)$$

where $i > 1, j > 1$.

$$EL_{1,j} = L_{1,j} + \frac{EL_{1,j-1}}{K_1}. \tag{4.1.9}$$

where $j > 1$.

$$El_{i,1} = L_{i,1} + \frac{EL_{i-1,1}}{K_2} \tag{4.1.10}$$

where $i > 1$, and $EL_{1,1} = L_{1,1}$.

This is a natural way of defining effective load of machine. The effective load is a measure of the temperature of a machine, and the temperature of machine is affected by the temperature of its two immediate neighbors in this model.This recursive definition already takes into account the heat recirculated from previous machines. In fact, this generalizes the one-dimensional model in a way, because, in the 1-D model with a single rack, we had proved in Claim 1 that $EL_i = L_i + \frac{EL_{i-1}}{K}$.

Lemma 9. *An optimal strategy minimizing the effective load across all machines when the total load is L would result in an uniform effective load*
$EL = \frac{L\ K_1\ K_2}{m_1K_1+m_2K_2+m_1m_2(K_1K_2-K_1-K_2)}$.

Proof. An optimal strategy minimizing the effective load across all the machines would be making the effective load uniform across them, as discussed earlier. Let this uniform effective load be EL when the total load is L. We know that $L = \sum_{i=1}^{m_2} \sum_{j=1}^{m_1} L_{i,j}$. When the effective load is uniform everywhere and equal to EL, then by definition the following are true.

$L_{1,1} = EL$

$L_{1,j} = EL\left(1 - \frac{1}{K_1}\right)$ where $j \in [2 \ldots m_1]$

$L_{i,1} = EL\left(1 - \frac{1}{K_2}\right)$ where $i \in [2 \ldots m_2]$

$L_{i,j} = EL\left(1 - \frac{1}{K_1} - \frac{1}{K_2}\right)$ where $j \in [2 \ldots m_1]$ and $i \in [2 \ldots m_2]$.

Summing up the loads over all machines we get

$$\begin{aligned} L = EL\ &\left(1 + (m_1 - 1)\left(1 - \frac{1}{K_1}\right) + (m_2 - 1)\left(1 - \frac{1}{K_2}\right)\right. \\ &\left. +(m_1 - 1)(m_2 - 1)\left(1 - \frac{1}{K_1} - \frac{1}{K_2}\right)\right) \qquad (4.1.11) \\ = EL\ &\left(\frac{m_1}{K_2} + \frac{m_2}{K_1} + m_1 m_2\left(1 - \frac{1}{K_1} - \frac{1}{K_2}\right)\right) \end{aligned}$$

Hence, $EL = \frac{L\ K_1\ K_2}{m_1K_1 + m_2K_2 + m_1m_2(K_1K_2 - K_1 - K_2)}$. □

The following lemma gives the dependence of the effective load on machine (i, j) in terms of the actual loads of other machines.

Lemma 10. *The effective load on machine* (i, j) *in terms of the* loads *of the other machines is* $EL_{i,j} = \sum_{p=0}^{i-1} \sum_{q=0}^{j-1} \binom{p+q}{p} \frac{L_{i-p,j-q}}{K_1{}^p\ K_2{}^q}$.

Proof. We will prove this by induction. It is obvious that the expression holds for base case $i = 1, j = 1$ as $EL_{1,1} = L_{1,1}$.

For the machines in the first rack, $i = 1, j \in [2 \ldots m_1]$, let us assume the induction hypothesis that the expression holds for all $j \leq j_1,\ j_1 < m_1$. Therefore, $\forall j \in [2 \ldots j_1],\ EL_{1,j} = \sum_{q=0}^{j-1} \frac{L_{1,j-q}}{K_1{}^q}$ as $\binom{n}{0} = 1\ \forall n \in \mathbb{Z}^+$ and $K_2{}^0 = 1$.

By definition, $EL_{1,j_1+1} = L_{1,j_1+1} + \frac{EL_{1,j_1}}{K_1}$. Substituting the value of EL_{1,j_1} according to the induction hypothesis, we get,

$$\begin{aligned} EL_{1,j_1+1} &= L_{1,j_1+1} + \frac{\sum_{q=0}^{j_1-1} \frac{L_{1,j_1-q}}{K_1{}^q}}{K_1} \qquad (4.1.12) \\ &= \sum_{q=0}^{j_1} \frac{L_{1,j_1+1-q}}{K_1{}^q} \end{aligned}$$

which proves it for the first rack.

We now consider the machines in the first row of each rack, $i \in [1 \dots m_2],\ j = 1$. The base case is $(1, 1)$ which is true. Let us assume by induction hypothesis that the expression is true for all $i \leq i_1, j = 1$. Therefore, $\forall i \in [2 \dots i_1],\ EL_{i,1} = \sum_{p=0}^{i-1} \frac{L_{i-p,1}}{K_2^p}$ as $\binom{n}{0} = 1\ \forall n \in \mathbb{Z}^+$ and $K_1^0 = 1$.

By definition, $EL_{i_1+1,1} = L_{i_1+1,1} + \frac{EL_{i_1,1}}{K_2}$. Substituting the value of $EL_{i_1,1}$ according to the induction hypothesis, we get,

$$\begin{aligned} EL_{i_1+1,1} &= L_{i_1+1,1} + \frac{\sum_{p=0}^{i_1-1} \frac{L_{i_1-p,1}}{{K_2}^p}}{K_2} \\ &= \sum_{p=0}^{i_1} \frac{L_{i_1+1-p,1}}{{K_2}^p} \end{aligned} \tag{4.1.13}$$

which proves it for the first row of all racks.

We now apply induction on the racks. We have proved the expression is true for the first rack $i = 1$, which is the base case for racks. Let us assume as our induction hypothesis that the expression holds for all racks $i < i_1$, that is, for all machines in racks $[1 \dots i_1]$. We have also proved that for any rack i_1, the expression is true for the first machine $(i_1, 1)$. This is the base case for the rack i_1. Let us extend our induction hypothesis to hold for all machines $j < j_1$ in rack i_1. By definition, $EL_{i_1,j_1} = L_{i_1,j_1} + \frac{EL_{i_1,j_1-1}}{K_1} + \frac{EL_{i_1-1,j_1}}{K_2}$. Substituting from our induction hypothesis, we get

$$\begin{aligned} EL_{i_1,j_1} &= L_{i_1,j_1} \\ &\quad + \frac{1}{K_1} \left(\sum_{p=0}^{i_1-1} \sum_{q=0}^{j_1-2} \binom{p+q}{p} \frac{L_{i_1-p,j_1-1-q}}{{K_1}^q\ {K_2}^p} \right) \\ &\quad + \frac{1}{K_2} \left(\sum_{p=0}^{i_1-2} \sum_{q=0}^{j_1-1} \binom{p+q}{p} \frac{L_{i_1-1-p,j_1-q}}{{K_1}^q\ {K_2}^p} \right) \end{aligned} \tag{4.1.14}$$

The above equation can be written as

$$\begin{aligned} EL_{i_1,j_1} &= L_{i_1,j_1} \\ &+ \left(\sum_{p=0}^{i_1-1} \sum_{q=1}^{j_1-1} \binom{p+q-1}{p} \frac{L_{i_1-p,j_1-q}}{{K_1}^q\ {K_2}^p} \right) \\ &+ \left(\sum_{p=1}^{i_1-1} \sum_{q=0}^{j_1-1} \binom{p+q-1}{p-1} \frac{L_{i_1-p,j_1-q}}{{K_1}^q\ {K_2}^p} \right) \end{aligned} \tag{4.1.15}$$

$$\begin{aligned} EL_{i_1,j_1} &= L_{i_1,j_1} \\ &+ \left(\sum_{p=1}^{i_1-1} \sum_{q=1}^{j_1-1} \left(\binom{p+q-1}{p} + \binom{p+q-1}{p-1} \right) \frac{L_{i_1-p,j_1-q}}{{K_1}^q\ {K_2}^p} \right) \\ &+ \sum_{q=1}^{j_1-1} \binom{q-1}{0} \frac{L_{i_1,j_1-q}}{{K_1}^q} + \sum_{p=1}^{i_1-1} \binom{p-1}{p-1} \frac{L_{i_1-p,j_1}}{{K_2}^p} \end{aligned} \tag{4.1.16}$$

We know, $\binom{p+q-1}{p} + \binom{p+q-1}{p-1} = \binom{p+q}{p}$. Substituting in the above equation, we get

$$\begin{aligned} EL_{i_1,j_1} &= L_{i_1,j_1} \\ &+ \left(\sum_{p=1}^{i_1-1} \sum_{q=1}^{j_1-1} \binom{p+q}{p} \frac{L_{i_1-p,j_1-q}}{{K_1}^q\ {K_2}^p} \right) \\ &+ \sum_{q=1}^{j_1-1} \frac{L_{i_1,j_1-q}}{{K_1}^q} + \sum_{p=1}^{i_1-1} \frac{L_{i_1-p,j_1}}{{K_2}^p} \end{aligned} \tag{4.1.17}$$

where we have also used $\binom{n}{0} = 1$ and $\binom{n}{n} = 1, \forall n \in \mathbb{Z}^+$. In the above equation, the penultimate summation is for $p = 0, q \in [1 \dots j_1 - 1]$, and the last summation is for $q = 0, p \in [1 \dots i_1 - 1]$. The first summation is for $p \in [1 \dots i_1 - 1], q \in [1 \dots j_1 - 1]$ and $L_{i_1,j_1} = \binom{0}{0} \frac{L_{i_1,j_1}}{K_1^0\ K_2^0}$ corresponding to the term $p = 0, q = 0$. Combining and expressing in a compact manner, we get,

$$EL_{i_1,j_1} = \sum_{p=0}^{i_1-1} \sum_{q=0}^{j_1-1} \binom{p+q}{p} \frac{L_{i_1-p,j_1-q}}{{K_1}^q\ {K_2}^p} \tag{4.1.18}$$

This proves that the expression for effective load is true for any $(i,j)^{th}$ machine by induction. □

We would like to compare the savings provided by our thermally aware strategy to that of naive strategy which splits the *load* uniformly across all machines. Each machine now gets a load $\frac{L}{m_1 m_2}$. The effective load is obviously maximized on the machine (m_1, m_2).

Lemma 11. *A naive strategy which splits load uniformly across all machines results in a maximum effective load* $EL_{m_1,m_2} = \frac{L}{m_1\ m_2} \sum_{i=0}^{m_2-1} \sum_{j=0}^{m_1-1} \frac{\binom{(i+j)}{i}}{{K_1}^j\ {K_2}^i}$.

Proof. The proof of this lemma follows from Lemma 10 and the facts that the load over each machine is uniform and equal to $\frac{L}{m_1 m_2}$. □

4.2 Integral Assignments for 2D Models

Similar to the one-dimensional model, in this section, we assume there is a hard limit c on the effective load capacity for all machines. Our goal is to maximize the number or profit of jobs integrally assigned with respect to this thermal capacity constraint.

4.2.1 General Model

We first consider the **general model** and show that by fixing a (load) capacity pattern, and using the PTAS (polynomial time approximation scheme) for multiple knapsack by Chekuri and Khanna [10], or that by Jansen [35], we can get a $\frac{1}{2} - O(\epsilon)$ approximation to the optimal solution. We are given an instance of jobs, $\mathcal{J}$, where each job $j \in \mathcal{J}$ has a thermal size s_j, and profit p_j. Let us denote by Δ be the maximum size of any job: $\Delta = \max_{j\in\mathcal{J}} s_j$. We assume that $\Delta \leq$

$c\left(1-\frac{1}{K_1}\right)\left(1-\frac{1}{K_2}\right)$. Let $\mu = \lfloor \frac{c\left(1-\frac{1}{K_1}\right)\left(1-\frac{1}{K_2}\right)}{\Delta} \rfloor$. By assumption, $\mu \geq 1$ $\forall i$. We assume there are $m_1 \geq 2$ machines in each rack and $m_2 \geq 1$ racks.

Algorithm 3 Algorithm for General Model

```
1: Set the load capacities of the machines as follows: $c$ for the machine (1,1),
   $c\left(1-\frac{1}{K_1}\right)$ for machines $(1,j)$, $j \in [2 \dots m_1]$, $c\left(1-\frac{1}{K_2}\right)$ for machines
   $(i,1)$, $i \in [2 \dots m_2]$, and $c\left(1-\frac{1}{K_1}\right)\left(1-\frac{1}{K_2}\right)$ for the all other machines.
2: Run the PTAS for multiple knapsack using the modified machine capacities on
   the instance $\mathcal{J}$.
```

Lemma 12. *The packing produced by Algorithm 3 is thermally feasible for the general model.*

Proof. This follows from the relation between effective loads and actual loads of the machine, as given by Claim 4. □

Theorem 8. *When $K_2 \geq 1 + \frac{K_1}{K_1-2}$, Algorithm 3 produces a $\geq \frac{1}{2} - O(\epsilon)$ approximation to an optimal solution for the maximum profit problem on any instance $\mathcal{J}$, in polynomial time for any fixed $\epsilon > 0$.*

Proof. For proving this theorem, we initially assume that we have access to a multiple knapsack oracle that returns an optimal packing for a set of jobs (items) with sizes and profits, in machines (knapsacks) of certain capacities (which may not be the same for all machines). Let us consider an optimal solution OPT which maximizes the profit of jobs integrally assigned to the machines (with unmodified capacities). For ease of exposition, first we consider the maximum cardinality problem, (in other words, profit is unity for all jobs). We will to show that there exists a packing that loses at most one job per machine, as compared to OPT, when we fix the load capacities of the machines as $c_{1,1} = c$, $c_{1,j} = c\left(1 - \frac{1}{K_1}\right)$, $j \in [2 \dots m_1]$, $c_{i,1} = c\left(1 - \frac{1}{K_2}\right)$, $i \in [2 \dots m_2]$, $c_{i,j} = c\left(1 - \frac{1}{K_1}\right)\left(1 - \frac{1}{K_2}\right)$, $i \in [2 \dots m_2]$, $j \in [2 \dots m_1]$. Note that this choice of capacity pattern ensures that if any machine (i, j) is packed to capacity $c_{i,j}$, the thermal constraints would not be violated anywhere, provided, each machine (i', j') has been packed to an extent at most $c_{i',j'}$. This follows from Lemma 12.

In the optimal solution OPT, the machines may be packed to different capacities, which may or may not be equal to their modified load capacities as set by Algorithm 3. If the sum of the sizes of jobs assigned to a machine (i, j) in OPT exceeds the capacity $c_{i,j}$ (as set by Algorithm 3), we call it *overpacked.* On the other hand, if the sum of the job sizes assigned to (i, j) is lower than $c_{i,j}$, we call it *underpacked.* Let $\mathcal{O}$ be the set of overpacked machines and $\mathcal{U}$ be the set of underpacked machines in OPT.

Recall that for the one-dimensional case, we had shown how to derive a packing from OPT, respecting the artificially set load capacities of the machines, by losing at most one job per machine in a polynomial number of operations [55]. However, it is *not* necessary that we show how such a repacking can be done in polynomial time; our purpose is to prove that such a packing *exists* for proving the rest of the theorem.

For an overpacked machine $(i, j) \in \mathcal{O}$, let us number the jobs in (i, j) in non-decreasing size order. Let k be the largest index such that $\sum_{p=1}^{k} s_p \leq c_{i,j}$.

Obviously, the remaining jobs in (i,j) could be accommodated because of the extra space available due to the underpacked machines. Let us denote this set of jobs as $\mathcal{J}_{i,j}$ for an $(i,j) \in \mathcal{O}$, and similarly, we compute this set for all $(i,j) \in \mathcal{O}$. Suppose the load in (i,j) originally was $L_{i,j}$. If $S(\mathcal{J}_{i,j}) > L_{i,j} - c_{i,j}$, then discarding any job from $\mathcal{J}_{i,j}$ will ensure that the total size of the remaining jobs is $\leq L_{i,j} - c_{i,j}$. This follows from the manner in which $\mathcal{J}_{i,j}$ was initially chosen. Let us therefore discard any job from $\mathcal{J}_{i,j}$ to get the set $\mathcal{J}'_{i,j}$. On the other hand, if $S(\mathcal{J}_{i,j}) = L_{i,j} - c_{i,j}$, then $\mathcal{J}'_{i,j} = \mathcal{J}_{i,j}$. Since $S(\mathcal{J}'_{i,j}) \leq L_{i,j} - c_{i,j}$, the jobs in $\mathcal{J}'_{i,j}$ can be *completely* accommodated in the extra space available in machine (i,j) over and above $c_{i,j}$ due to gaps in underpacked machines. Note that the set $\mathcal{J}'_{i,j}$ may be empty as well. We compute these sets for all overpacked machines.

Let us denote by $\mathcal{J}^O = \cup_{(i,j)\in\mathcal{O}} \mathcal{J}'_{i,j}$, the set of jobs which can be *completely* accommodated in the extra space created by gaps (of underpacked machines) in the overpacked machines in OPT.

Now we consider a multiple knapsack problem defined as follows. For every underpacked machine $(i',j') \in \mathcal{U}$, let us consider the gap $\delta_{i',j'}$ in (i',j') as a knapsack of capacity $\delta_{i',j'}$. Let the items to be packed in this multiple knapsack problem be the set of jobs in $\mathcal{J}^O$. Considering each job $\ell \in \mathcal{J}^O$ to be an item of size s_ℓ and profit 1, we call our multiple knapsack oracle to pack these knapsacks optimally. If all the items or jobs have been successfully packed then we have our required packing where each machine (i,j) is packed to an extent $\leq c_{i,j}$, with the loss of at most one job per overpacked machine. Note that this repacking would not violate the thermal constraints by definition since the total size of jobs reassigned to a machine (i',j')

is $\leq \delta_{i',j'}$.

If all items or jobs could not be packed, let $\mathcal{J}_{rem}$ be the set of jobs which could not be assigned to any of the knapsacks. Let us denote by $\epsilon_{i',j'}$ the space left in knapsack (i',j') after this new packing. Let s_{min} be the smallest size of any job in $\mathcal{J}_{rem}$; we know $s_{min} > \epsilon_{max}$, where $\epsilon_{max} \geq \epsilon_{i',j'}$ for $(i',j') \in \mathcal{U}$.

Now we define the notion of *contribution* of a gap.

Definition 1. *An underpacked machine (i',j') packed up to $L_{i',j'} < c_{i',j'}$, gives rise to a gap $\delta_{i',j'} = c_{i',j'} - L_{i',j'}$. The* contribution *of this gap of size $\delta_{i',j'}$ in an overpacked machine (i,j) is $\frac{\delta_{i',j'}}{K_1{}^{j-j'} K_2{}^{i-i'}}$. In other words, it denotes the maximum extra space that this gap can produce in (i,j) for enabling a thermally feasible packing of jobs exceeding $c_{i,j}$.*

The total contribution of a gap $\delta_{i',j'}$ is its contribution summed over all machines, and is therefore an upper bound on the total extra space created by $\delta_{i',j'}$ over all machines. Let S denote the sum of the total contribution of all gaps in $\mathcal{U}$. For feasibility, the following must be true: $S \geq S(\mathcal{J}^O) = S(\mathcal{J}_{rem}) + S(\mathcal{J}^O \setminus \mathcal{J}_{rem})$. Therefore, $S(\mathcal{J}_{rem}) \leq S - S(\mathcal{J}^O \setminus \mathcal{J}_{rem})$.

The total contribution of each $\delta_{i',j'}$ in S is

$$C_{i',j'} = \delta_{i',j'} \left(\sum_{p=i'}^{m_2} \sum_{q=j'}^{m_1} \frac{1}{K_1{}^{q-j'} \; K_2{}^{p-i'}} \; - \; 1 \right)$$

since the contribution of $\delta_{i',j'}$ can only be on machines located higher up in the same rack, or on the same row or higher for racks to the right. Obviously, $S = \sum_{(i',j') \in \mathcal{U}} C_{i',j'}$. Writing $C_{i',j'}$ in terms of $\delta_{i',j'}$ and $\epsilon_{i',j'}$, we get

$$C_{i',j'} = (\delta_{i',j'} - \epsilon_{i',j'})\left(\sum_{p=i'}^{m_2}\sum_{q=j'}^{m_1}\frac{1}{{K_1}^{q-j'}\ {K_2}^{p-i'}} - 1\right)$$
$$+\ \epsilon_{i',j'}\left(\sum_{p=i'}^{m_2}\sum_{q=j'}^{m_1}\frac{1}{{K_1}^{q-j'}\ {K_2}^{p-i'}} - 1\right)$$

Therefore,

$$S = \sum_{(i',j')\in U}\left((\delta_{i',j'} - \epsilon_{i',j'})\left(\sum_{p=i'}^{m_2}\sum_{q=j'}^{m_1}\frac{1}{{K_1}^{q-j'}\ {K_2}^{p-i'}} - 1\right)\right.$$
$$\left.+\epsilon_{i',j'}\left(\sum_{p=i'}^{m_2}\sum_{q=j'}^{m_1}\frac{1}{{K_1}^{q-j'}\ {K_2}^{p-i'}} - 1\right)\right)$$

Since $K_2 \geq 1 + \frac{K_1}{K_1-2}$, it can be verified that $\left(\frac{K_1}{K_1-1}\right)\left(\frac{K_2}{K_2-1}\right) - \ 1 \ \leq 1$. Moreover, $S(\mathcal{J}^O \setminus \mathcal{J}_{rem}) = \sum_{(i',j')\in\mathcal{U}}(\delta_{i',j'} - \epsilon_{i',j'})$. Substituting $S(\mathcal{J}^O \setminus \mathcal{J}_{rem})$ and S, and using the relation $\left(\frac{K_1}{K_1-1}\right)\left(\frac{K_2}{K_2-1}\right) - \ 1 \ \leq 1$, we get

$$S(\mathcal{J}_{rem}) \leq \sum_{(i',j')\in\mathcal{U}}\left(\epsilon_{i',j'}\left(\sum_{p=i'}^{m_2}\sum_{q=j'}^{m_1}\frac{1}{{K_1}^{q-j'}\ {K_2}^{p-i'}} - 1\right)\right)$$
$$\leq \sum_{(i',j')\in\mathcal{U}}\epsilon_{max}\left(\sum_{p=i'}^{m_2}\sum_{q=j'}^{m_1}\frac{1}{{K_1}^{q-j'}\ {K_2}^{p-i'}} - 1\right)$$
$$\leq \epsilon_{max}$$

where the last inequality follows from the relation between K_1 and K_2. However, $s_{min} > \epsilon_{max}$ and $s_{min} \leq S(\mathcal{J}_{rem})$, which is a contradiction. Therefore, $\mathcal{J}_{rem} = \emptyset$.

Hence we have proved that there exists a packing with the chosen capacity pattern, which, if packed optimally would lose no more than one job per overpacked machine. Due to assumption on size, we know a machine which loses one job, has at least $\mu \geq 1$ jobs remaining in it. Hence, any optimal packing on this reduced instance will give a $\geq \frac{\mu}{\mu+1} \geq \frac{1}{2}$ approximation. By using the multiple knapsack PTAS for the packing, we lose at most an $O(\epsilon)$ factor.

If the jobs have variable profits, we choose the jobs in $\mathcal{J}_{i,j}$ as in the unit profit case by ordering jobs in non-decreasing size order. Let the jobs remaining in the machine after removing those in $\mathcal{J}_{i,j}$ be denoted as $\mathcal{J}_{i,j}^R$. If $S(\mathcal{J}_{i,j}) = L_{i,j} - c_{i,j}$, we set $\mathcal{J}'_{i,j} = \mathcal{J}_{i,j}$ and proceed. If however, $S(\mathcal{J}_{i,j}) > L_{i,j} - c_{i,j}$, we need to discard a job from $\mathcal{J}_{i,j}$ in order to get $\mathcal{J}'_{i,j}$. We first try discarding least profit job j from $\mathcal{J}_{i,j}$ to get $\mathcal{J}'_{i,j}$. If the sum of the profits of the jobs remaining in the machine (i, j) and those in $\mathcal{J}'_{i,j}$ together is greater than the profit of the job discarded: $P(\mathcal{J}'_{i,j}) + P(\mathcal{J}_{i,j}^R) \geq p_j$, we discard it safely and proceed with the resultant set $\mathcal{J}'_{i,j}$. Otherwise, it must be that $\mathcal{J}'_{i,j}$ is empty, and $\mathcal{J}_{i,j}$ had only consisted of j. Furthermore, $P(\mathcal{J}_{i,j}^R) < p_j$. In this case, we interchange the sets $\mathcal{J}_{i,j}^R$ and $\mathcal{J}_{i,j}$. By definition $s_j \leq \Delta \leq c_{i,j}$, hence it is feasible to make $\mathcal{J}_{i,j}^R = \{j\}$. Now, we order the jobs in $\mathcal{J}_{i,j}$ in non-increasing profit order, and let k' be the largest index such that $\sum_{x=1}^{k'} s_x \leq (L_{i,j} - c_{i,j})$. We set $\mathcal{J}'_{i,j}$ to be the jobs numbered $[1, \ldots, k']$ in $\mathcal{J}_{i,j}$, and discard the remaining jobs. Note that $\mathcal{J}'_{i,j}$ may be empty as in the unit profit case, however, we have ensured that the profit of the discarded jobs can be charged to the profit of the jobs remaining in $\mathcal{J}'_{i,j} \cup \mathcal{J}_{i,j}^R$.

As in the unit profit case, now $\mathcal{J}^O$ will be the set of jobs that can be accommodated completely in the extra space created by gaps. By the same packing arguments for the unit profit case, we know that the jobs in $\mathcal{J}^O$ can be feasibly packed in the gaps in the underpacked machines. This is a thermally feasible packing, which has a total profit at least half of the optimal packing that we started out with, since we had ensured that the profit of the discarded jobs can be charged to the profit of the remaining ones. Therefore, any optimal multiple knapsack packing oracle on the machines with modified load capacities would give $\geq \frac{1}{2}$ the profit of an optimal solution, and hence, using the multiple knapsack packing PTAS [10,35], we lose at most $O(\epsilon)$ of the profit. $\square$

4.2.2 Horizontal Sibling Model

We now consider the **horizontal sibling model** and show that by fixing a capacity pattern we can get a $\frac{1}{2} - O(\epsilon)$ approximation to an optimal solution maximizing the number or profit of jobs integrally assigned in the presence of a hard thermal constraint fixing the maximum effective load capacity of any machine at c. Let the number of racks be m_2, and the number of machines in rack be m_1. We are given an instance of jobs $\mathcal{J}$, where a job j has a size s_j and profit p_j.

Algorithm 4 Algorithm for Horizontal Sibling Model

```
1: Set the load capacity of the $(i,j)^{th}$ machine as:
   $c_{i,j} = c\left(1 - \frac{1}{K_1} + \sum_{\ell=0}^{i-2} \sum_{p=0}^{\min j-1, i-\ell} \left(\frac{-1}{K_2}\right)^{i-\ell-1} \left(\frac{-1}{K_1}\right)^{p} \binom{i+\ell}{p}\right)$.
2: Run the multiple knapsack PTAS on instance $\mathcal{J}$ using the machines with modified capacities.
```

Lemma 13. *The packing produced by the Algorithm 4 is thermally feasible for the horizontal sibling model.*

Proof. This follows from the proof of Claim 9. $\square$

Let us denote by Δ be the maximum size of any job: $\Delta = \max_{j \in \mathcal{J}} s_j$. Let c_{min} be the minimum value of any $c_{i,j}$ computed as above, and $\mu = \lfloor \frac{c_{min}}{\Delta} \rfloor$. We assume $\mu \geq 1$ as before which is reasonable for data centers. As already stated, μ is much larger in practice.

Theorem 9. *When $K_2 \geq 1 + \frac{1}{K_1 - 2}$, Algorithm 4 gives a $\frac{1}{2} - O(\epsilon)$ approximation to an optimal solution for the maximum profit problem for any instance $\mathcal{J}$ in polynomial time for any fixed $\epsilon > 0$.*

Proof. For proving this theorem, we assume that we have access to a multiple knapsack oracle that returns the optimal packing for an instance $\mathcal{J}$ maximizing the profit of jobs (items) assigned integrally to machines (knapsacks) of variable capacities. Let us consider an optimal solution OPT. First, we will consider the special case of unit profits; in other words, the maximum cardinality problem. We will show that there exists a packing that loses at most one job per machine, as compared to the OPT when we set the load capacities of the machines as:
$c_{i,j} = c \left(1 - \frac{1}{K_1} + \sum_{\ell=0}^{i-2} \sum_{p=0}^{\min j-1, i-\ell} \left(\frac{-1}{K_2}\right)^{i-\ell-1} \left(\frac{-1}{K_1}\right)^{p} \binom{i+\ell}{p}\right)$.

By choosing this capacity pattern we ensure that if any machine (i, j) is packed to capacity $c_{i,j}$, the thermal constraints would not be violated anywhere, provided, each machine (i', j') has been packed to an extent at most $c_{i',j'}$.

The rest of the proof, including notations, is similar to that of Theorem 8, and hence we do not repeat it here in details. The differences are mainly in the contribution of gaps, since that depends on the thermal cross-interference model being considered. We highlight these portions of the proof below:

The total contribution of a gap $\delta_{i',j'}$ (due to an underpacked machine $(i,j) \in \mathcal{U}$) is as follows:

$$C_{i',j'} \leq \delta_{i',j'} \left(\sum_{q=j'}^{m_1} \frac{1}{{K_1}^{q-j'}} + \frac{1}{K_2} \; - \; 1 \right) \tag{4.2.1}$$

since the contribution of $\delta_{i',j'}$ can only be on machines located higher up in the same rack, or on the machine in the same row of the adjacent rack to the right, if any. Obviously, the total space due to gaps is the sum of the total contributions of all the gaps: $S = \sum_{(i',j') \in \mathcal{U}} C_{i',j'}$. Writing $C_{i',j'}$ in terms of $\delta_{i',j'}$ and $\epsilon_{i',j'}$, we get

$$\begin{aligned} C_{i',j'} \leq & (\delta_{i',j'} - \epsilon_{i',j'}) \left(\sum_{q=j'}^{m_1} \frac{1}{{K_1}^{q-j'}} + \frac{1}{K_2} - 1 \right) \\ & + \epsilon_{i',j'} \left(\sum_{q=j'}^{m_1} \frac{1}{{K_1}^{q-j'}} + \frac{1}{K_2} - 1 \right) \end{aligned} \tag{4.2.2}$$

Therefore,

$$\begin{aligned} S = \sum_{(i',j') \in \mathcal{U}} & \left((\delta_{i',j'} - \epsilon_{i',j'}) \left(\sum_{q=j'}^{m_1} \frac{1}{{K_1}^{q-j'}} + \frac{1}{K_2} - 1 \right) \right. \\ & \left. + \epsilon_{i',j'} \left(\sum_{q=j'}^{m_1} \frac{1}{{K_1}^{q-j'}} + \frac{1}{K_2} - 1 \right) \right) \end{aligned} \tag{4.2.3}$$

Since $K_2 \geq 1 + \frac{1}{K_1 - 2}$ and $K_1 \geq 2$, it can be verified that $\frac{K_1}{K_1 - 1} + \frac{1}{K_2} - 1 \; \leq 1$. Moreover, after repacking, $S(\mathcal{J}^O \backslash \mathcal{J}_{rem}) = \sum_{(i',j') \in \mathcal{U}} (\delta_{i',j'} - \epsilon_{i',j'})$. Substituting $S(\mathcal{J}^O \setminus \mathcal{J}_{rem})$ and S,

$$\begin{aligned} S(\mathcal{J}_{rem}) & \leq \sum_{(i',j') \in \mathcal{U}} \left(\epsilon_{i',j'} \left(\sum_{q=j'}^{m_1} \frac{1}{{K_1}^{q-j'}} + \frac{1}{K_2} - 1 \right) \right) \\ & \leq \sum_{(i',j') \in \mathcal{U}} \epsilon_{max} \left(\sum_{q=j'}^{m_1} \frac{1}{{K_1}^{q-j'}} + \frac{1}{K_2} - 1 \right) \\ & \leq \epsilon_{max} \end{aligned} \tag{4.2.4}$$

where the last inequality follows from the relation between K_1 and K_2. However, $s_{min} > \epsilon_{max}$ and $s_{min} \leq S(J_{rem})$, which is a contradiction. Therefore, $J_{rem} = \emptyset$. For the variable profits case, the arguments are exactly similar to that in Theorem 8, hence not repeated here. □

4.2.3 Indirect Sibling Model

We now consider the **indirect sibling model** and show that by fixing a capacity pattern we can get at least a $\frac{1}{2} - O(\epsilon)$ approximation to an optimal solution maximizing the number or profit of jobs integrally assigned in the presence of a hard thermal constraint which fixes the maximum effective load capacity of any machine at c. Let the number of racks be m_2 racks, and the number of machines in each rack be m_1. We are given an instance of jobs $\mathcal{J}$ where each job j has a size s_j and a profit p_j.

Algorithm 5 Algorithm for Indirect Sibling Model

```
1: Set the load capacities of the machines as: $c_{1,1} = c$, $c_{1,j} = c\left(1 - \frac{1}{K_1}\right)$ for $j \geq 2$,
   $c_{i,1} = c\left(1 - \frac{1}{K_2}\right)$ for $i \geq 2$, and $c_{i,j} = c\left(1 - \frac{1}{K_1} - \frac{1}{K_2}\right)$ for $i \geq 2, j \geq 2$.
2: Run the multiple knapsack PTAS on instance $\mathcal{J}$ using the machines with modified capacities.
```

Lemma 14. *The packing produced by the above method is thermally feasible for the indirect sibling model.*

Proof. This follows from the definition of the indirect sibling model. □

Let us denote by Δ be the maximum size of any job: $\Delta = \max_{j \in \mathcal{J}} s_j$. We assume that $\Delta \leq c\left(1 - \frac{1}{K_1} - \frac{1}{K_2}\right)$. Let $\mu = \lfloor \frac{c\left(1 - \frac{1}{K_1} - \frac{1}{K_2}\right)}{\Delta} \rfloor) \geq 1$.

Theorem 10. *When* $\sum_{i=0}^{m_2-1} \sum_{j=0}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} \leq 2$, *Algorithm 5 gives a* $\frac{1}{2} - O(\epsilon)$ *approximation to an optimal solution for the maximum profit problem for an instance* $\mathcal{J}$ *in polynomial time for any fixed* $\epsilon > 0$.

Proof. This is similar to the proof of Theorem 8 and hence we do not repeat the details here. Below we highlight the differences, which are in the contribution of gaps, since that depends on the thermal model being considered. First we show the bound for the special case of unit profits, or the maximum cardinality problem. The notations are the same as in proof of Theorem 8. The total contribution of a gap $\delta_{i',j'}$ due to an underpacked machine $(i', j') \in \mathcal{U}$ is as follows:

$$C_{i',j'} \leq \delta_{i',j'} \left(\sum_{i=i'-1}^{m_2-1} \sum_{j=j'-1}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} \ - \ 1 \right)$$

since the contribution of $\delta_{i',j'}$ can only be on machines located higher up in the same rack, or on the machines in the same or higher row in the racks to the right, if any. The expression for $C_{i',j'}$ follows from Lemma 10. Obviously, the total space available due to gaps is the sum of their total contributions:: $S = \sum_{(i',j') \in \mathcal{U}} C_{i',j'}$. Writing $C_{i',j'}$ in terms of $\delta_{i',j'}$ and $\epsilon_{i',j'}$, we get

$$\begin{aligned} C_{i',j'} \leq (\delta_{i',j'} - \epsilon_{i',j'}) \left(\sum_{i=i'-1}^{m_2-1} \sum_{j=j'-1}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} \ - \ 1 \right) \\ + \epsilon_{i',j'} \left(\sum_{i=i'-1}^{m_2-1} \sum_{j=j'-1}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} \ - \ 1 \right) \end{aligned} \tag{4.2.5}$$

Therefore,

$$\begin{aligned} S = \sum_{(i',j')\in\mathcal{U}} \Bigg(&(\delta_{i',j'} - \epsilon_{i',j'}) \left(\sum_{i=i'-1}^{m_2-1} \sum_{j=j'-1}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} \ - \ 1 \right) \\ &+\epsilon_{i',j'} \left(\sum_{i=i'-1}^{m_2-1} \sum_{j=j'-1}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} \ - \ 1 \right) \Bigg) \end{aligned} \tag{4.2.6}$$

Since $\left(\sum_{i=0}^{m_2-1} \sum_{j=0}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} \ - \ 1 \right) \leq 1$, and $S(\mathcal{J}^O \backslash \mathcal{J}_{rem}) = \sum_{(i',j')\in\mathcal{U}} (\delta_{i',j'} - \epsilon_{i',j'})$ after the repacking, we get the following:

$$\begin{aligned} S(\mathcal{J}_{rem}) &\leq \sum_{(i',j')\in\mathcal{U}} \left(\epsilon_{i',j'} \left(\sum_{i=i'-1}^{m_2-1} \sum_{j=j'-1}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} - 1 \right) \right) \\ &\leq \sum_{(i',j')\in U} \epsilon_{max} \left(\sum_{i=i'-1}^{m_2-1} \sum_{j=j'-1}^{m_1-1} \frac{\binom{i+j}{i}}{{K_1}^j\ {K_2}^i} - 1 \right) \\ &\leq \epsilon_{max} \end{aligned} \tag{4.2.7}$$

where the last inequality follows from the relation between K_1 and K_2. However, $s_{min} > \epsilon_{max}$ and $s_{min} \leq S(J_{rem})$, which is a contradiction. Therefore, $J_{rem} = \emptyset$. The arguments for variable profits is exactly the same as in Theorem 8, hence not repeated here. □

Chapter 5: The Busy Time Problem

In this chapter we shift gears from the study of thermal energy costs and possible optimizations in data centers, and focus our attention to the energy costs for powering machines.

Towards this objective, we consider the busy time problem [22, 39] [1]. Traditionally, scheduling jobs on multiple parallel or batch machines has been studied mostly in the context of "job-related" metrics such as minimizing makespan, total completion time, flow time, tardiness and maximizing throughput under various deadline constraints. However, these objectives do not address the problem of reducing energy consumption. In an effort to capture some central issues in energy management in cloud computing contexts, the busy time measure was recently introduced [22], and since then studied in a number of subsequent works. We define the problem formally in the next section.

[1]Some parts of our work on the busy time problem have been done jointly with Jessica Chang and can also be found in her thesis [6].

5.1 Problem Definition

The input is a collection $\mathcal{J}$ of n jobs that need to be scheduled on a set of identical machines. Each job j has release time r_j, deadline d_j and required processing length p_j. Each job j needs to be scheduled non-preemptively within its feasible time window $[r_j, d_j)$. If $p_j = d_j - r_j$, we call them interval jobs. Otherwise, we call them flexible jobs. We also consider the preemptive version of the problem, where a job may be scheduled over multiple machines, without any penalty. The processing capacities of the machines are limited: at most g jobs can run simultaneously on a given machine at any given instant. A machine is *busy* at time t if there is at least one job running on the machine at t; otherwise the machine is idle. The time intervals during which a machine is busy is called its *busy time*. The objective is to find a feasible schedule or assignment of all the jobs on the machines so as to minimize the cumulative busy time over all the machines. This is known as the **busy time problem**. The schedule can potentially use an unbounded number of machines since each group is really a virtual machine. Figure 5.1 shows a collection of interval jobs and the corresponding packing that yields an optimal solution minimizing busy time.

5.2 Prior Work

The busy time problem on interval jobs is well-studied in the literature. Since each job's deadline for interval jobs is exactly its release time plus its processing

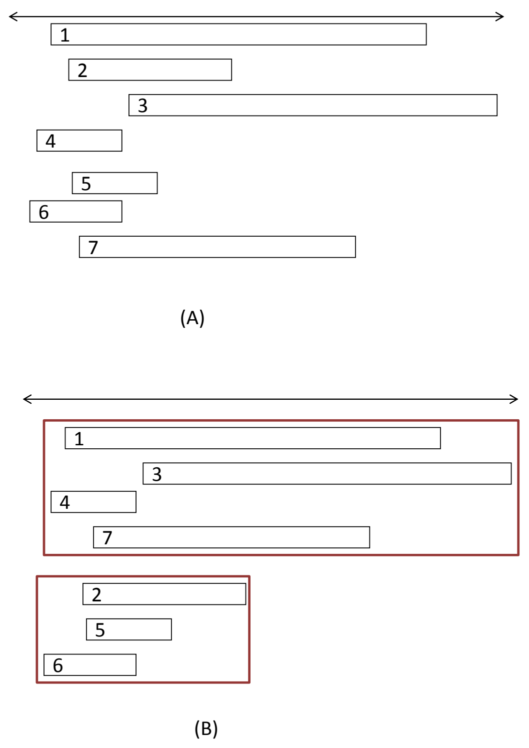


Figure 5.1: (A) Collection of interval jobs, numbered arbitrarily. (B) Optimal packing of the jobs on two machines with $g = 3$ minimizing total busy time.

time, there is no question about when it must start. The busy time problem is NP-hard [74] even when $g = 2$ for interval jobs. For interval jobs, we say the interval $[r_j, d_j)$ is the *span* of job j. The *span* of a job set $\mathcal{J}'$ is the union of the spans of jobs in $\mathcal{J}'$.

The busy time scheduling problem was introduced by Flammini et al. [22]. In this paper, they studied interval jobs. They present a very simple greedy algorithm FirstFit and demonstrate that it always produces a solution of busy time at most

4 times that of the optimal solution. The algorithm considers jobs in non-increasing order by length, greedily packing each job in the first group in which it fits. Additionally, they consider two special cases for which they present algorithms with improved approximation guarantees. The first case pertains to "proper intervals", where no job's interval is strictly contained in that of another. For instances of this type, they show that the greedy algorithm ordering jobs by release times is actually 2-approximate. The second special case involve instances whose corresponding interval graph is a clique - in other words, there exists a time t such that each interval $[r_j, d_j)$ contains it. In this case, a greedy algorithm also yields a 2-approximation.

It is not yet resolved whether minimizing busy time on clique instances or on proper instances is NP-hard. However, when the interval jobs are both proper *and* form a clique, a very simple dynamic program gives an optimal solution [51].

Khandekar et al. [39] consider the generalization in which each job has an associated width or "demand" on its machine. For any set of jobs assigned to the same machine, the cumulative demand of the active ones can be at most g at any time. The authors apply ideas from the analysis of FirstFit to this problem and obtain a 5-approximation. The main idea involves partitioning jobs into those of "narrow" and "wide" demand. Each wide job is assigned to its own machine, while FirstFit is applied to the set of narrow jobs. In addition, the authors give improved bounds for special cases of busy-time scheduling with jobs of unit demand. When the interval jobs form a clique, they provide a polynomial time approximation scheme (PTAS). They give a simple exact algorithm when the intervals of the jobs are laminar, i.e. two jobs' intervals intersect only if one interval is contained in the other. Khandekar et al. [39] also consider the generalization to the busy time problem where job windows need not be rigid, that is, they consider *flexible* jobs as well. They give a 5 approximation algorithm for this problem when jobs may additionally have non-unit demands. The approach is to first solve the problem for the case when the machine capacity is unbounded. They give an optimal polynomial time dynamic program for this problem. This essentially fixes the position of the jobs within their feasible windows. This serves as the input to the second stage of the algorithm, where now the jobs are treated as interval jobs (with possibly non-unit demands). Now, the algorithm for interval jobs with non-unit demands is used to give the 5 approximation. For unit demands, this result implies a 4 approximation algorithm for flexible jobs.

We only consider unit demands of jobs. Hence, for the rest of this chapter, when we refer to any job, it is implicit that its demand is unit.

5.3 Our Contributions

We first show that a 2 approximation algorithm for the busy time problem for interval jobs is implied by a 2 approximation algorithm given by Alicherry and Bhatia [1], for a coloring and routing problem on interval graphs, motivated by by the design of optical line systems. A 2 approximation algorithm developed by Kumar and Rudra [42] for a closely related problem called fiber minimization also implies a 2 approximation algorithm for the busy time problem on interval graphs.

We study the general busy time problem, where the jobs windows are not rigid, that is, where the jobs may be flexible. For this problem, we have already stated that Khandekar et al.'s result implies a 4 approximation algorithm. We show that a natural extension of the 2 approximation algorithm for interval jobs, gives a 4 approximation for the general busy time problem, following the same approach as Khandekar et al. [39] of fixing the windows of jobs via dynamic programming for the problem of unbounded g. We then develop a fast algorithm for interval jobs, which we call GreedyTracking. Again, using the approach of fixing the job windows for $g = \infty$, and using the output of the first phase as an input for GreedyTracking, we give a 3 approximation algorithm for the general busy time problem. We also show that the approximation factor of 3 is asymptotically tight for the algorithm.

For the preemptive version, we give a fast, *exact* algorithm for unbounded g. We then use this algorithm to give a 2 approximation algorithm for bounded g for the preemptive general busy time problem.

5.4 Related Work

Mertzios et al. [51] consider a dual problem to busy time minimization, the *resource allocation maximization version*, where the goal is to maximize the number of jobs scheduled without violating a budget constraint given in terms of busy time and the parallelism constraint. They show that the maximization version is NP-hard whenever the (busy time) minimization problem is NP-hard. They give a 6 approximation algorithm for clique instances and a polynomial time algorithm for proper clique instances for the maximization problem.

The online version of both the busy time minimization and resource allocation maximization was considered by Shalom et al. [64]. They prove a lower bound of g where g is the parallelism parameter, for any deterministic algorithm for general instances and give an $O(g)$ competitive algorithm. They also consider special cases, and show a lower bound of 2 and an upper bound of $(1 + \phi)$ for a one-sided clique instances (which is a special case of laminar cliques), where ϕ is the golden ratio. They also show that the bounds increase by a factor of 2 for clique instances. For the maximization version of the problem with parallelism g and busy time budget T, they show that any deterministic algorithm cannot be more than gT competitive. They give a 4.5 competitive algorithm for one-sided clique instances.

Flammini et al. [24] consider the problem of optimizing the cost of regenerators that need to be placed on light paths in optical networks, after every d nodes, to regenerate the signal. They show that the 4 approximation algorithm for minimizing busy time [22] solves this problem for a path topology and $d = 1$ and extend it to ring and tree topologies for general d.

Faigle et al. [18] consider the online problem of maximizing "busy time" but their objective function is totally different from ours. Their setting consists of a single machine and no parallelism. Their objective is to maximize the total length of intervals scheduled as they arrive online, such that at a given time, at most one interval job has been scheduled on the machine. They give a randomized online algorithm for this problem.

5.5 Notations and Preliminaries

Definition 2. *An instance $\mathcal{J}$ is said to be an* interval job *instance if for every job $j \in \mathcal{J}$, $d_j = r_j + p_j$.*

A job j is *active* on machine m at some time $t \in [r_j, d_j)$ if j is one of the jobs being processed by machine m at time t.

Definition 3. *The length of a time interval $I = [a, b)$ is denoted $\ell(I) = b - a$, For a single contiguous interval, the length is the same as its span, and hence may be referred to interchangeably as the span of I, $|Sp(I)|$. For a set of intervals $\mathcal{I}$, the length of $\mathcal{I}$ is $\ell(\mathcal{I}) = \sum_{I \in \mathcal{I}} \ell(I)$. The span of $\mathcal{I}$ is $Sp(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} I$.*

The length $\ell(j)$ (span $Sp(j)$, respectively) of an interval job j is the length (span, respectively) of $[r_j, d_j)$. Similarly, the length $\ell(\mathcal{J})$ (span $Sp(\mathcal{J})$, respectively) of a set $\mathcal{J}$ of interval jobs is the length (span, respectively) of the set of intervals $\{[r_j, d_j) : j \in \mathcal{J}\}$.

By abuse of notation, for flexible jobs, we sometimes denote p_j as $\ell(j)$. Similar to interval jobs, the length of any set of flexible jobs $\ell(\mathcal{J})$ is really the sum of the processing lengths of these jobs. Specifically, $\ell(\mathcal{J}) = \sum_{j \in \mathcal{J}} p_j$. Whether we are referring to the lengths of contiguous intervals, or the processing times of jobs will be clear from the context. Consider for a moment the non-preemptive variant of the problem. For the special case of interval jobs, we need to find a partition of the jobs into groups or bundles, such that in every bundle, there are at most g jobs active at any time t. We then schedule each bundle on a single machine. Let $\mathcal{B}_\kappa$

be the set of interval jobs assigned to bundle κ by some partitioning scheme. Then, the busy time of the machine on which the bundle κ will be scheduled is given by $|Sp(\mathcal{B}_\kappa)|$. Suppose we have partitioned all jobs into k feasible bundles (the feasibility respects the parallelism bound g as well as the release times and deadlines). Then the total cost of the solution is given by the cumulative busy time $\sum_{\kappa=1}^{k} |Sp(\mathcal{B}_\kappa)|$. The objective is to minimize this cost. We consider both the variants where g is unbounded and where $g < \infty$. For the *preemptive* version of the problem, the problem definition remains the same, the only difference being that the jobs can be processed preemptively across various machines.

To minimize busy time for flexible jobs, the difficulty lies not just in finding a partition of jobs, but also in deciding when each job j should be scheduled. We study both the preemptive and non-preemptive versions of this problem.

We denote the cost of the optimal solution of an instance $\mathcal{J}$ (of flexible or interval jobs) by $OPT(\mathcal{J})$. We denote by $OPT_\infty(\mathcal{J})$ the cost of the optimal solution for the instance $\mathcal{J}$ when unbounded parallelism is allowed.

Without loss of generality, the busy time of a machine is contiguous. If it is not, we can break it up into disjoint periods of contiguous busy time, assigning each of them to different machines, without increasing the total busy time of the solution.

The following lower bounds were introduced in [22] and hold trivially on any optimal solution for a given instance $\mathcal{J}$.

Observation 1. *For an instance $\mathcal{J}$, $OPT(\mathcal{J}) \geq \frac{\ell(\mathcal{J})}{g}$, where $g \geq 1$ and $\ell(\mathcal{J})$ denotes the sum of the processing lengths of the jobs in $\mathcal{J}$, interchangeably referred to as the* *mass of the set $\mathcal{J}$.*

This holds because in any machine, we can have at most g jobs active simultaneously.

Observation 2. *For a set $\mathcal{J}$ of flexible or interval jobs, the cost of an optimal solution $OPT(\mathcal{J}) \geq OPT_\infty(\mathcal{J})$.*

The above observation follows from the fact that if a lower cost solution exists for bounded g, then it is a feasible solution for unbounded g as well. If the jobs in $\mathcal{J}$ are interval jobs, then, $OPT_\infty(\mathcal{J}) = |Sp(\mathcal{J})|$.

However, the above lower bounds, individually, can be arbitrarily bad. For example, consider an instance of g disjoint unit length interval jobs. The mass bound would simply give a lower bound of 1, whereas the optimal solution pays g. Similarly, consider an instance of g^2 identical unit length interval jobs. The span bound would give a lower bound of 1, whereas the optimal solution has to open up g machines for unit intervals, paying g.

We introduce [8] a stronger lower bound, which we call the Demand Profile. In fact, the algorithm of Alicherry and Bhatia [1] as well as that of Kumar and Rudra [42] implicitly charge the demand profile. This lower bound holds for the case of interval jobs.

Definition 4. *Let $A(t)$ be the set of interval jobs which are active at time t. In other words, $A(t) = \{j : t \in [r_j, d_j)\}$. We say that $|A(t)|$ is the* raw demand *at time t, and define the* demand *at time t as $D(t) = \lceil \frac{|A(t)|}{g} \rceil$.*

Definition 5. *An interval within which no job begins or ends is called an* interesting *interval.*

By definition, the raw demand, and hence the demand, is uniform over an interesting interval because no job begins or ends within the interval. Let us represent the raw demand over an interesting interval I_i, as $A(I_i)$ and the demand as $D(I_i)$. Let $\mathcal{I}$ be the set of interesting intervals, $\mathcal{I} = I_1, I_2, \ldots, I_\ell$, where $\ell \leq 2n$, and $D(I_i) = D(t),\ \forall t \in I_i$. $\bigcup_{I_i \in \mathcal{I}} Sp(I_i) = Sp(\mathcal{J})$.

Definition 6. *We define the Demand Profile of an instance of interval jobs $\mathcal{J}$, $DeP(\mathcal{J})$ as the set of tuples $(I_i, D(I_i))$, where $I_i \in \mathcal{I}$.*

Note that the above definition expresses the Demand Profile in a linear number of tuples, even when the release times and deadlines of jobs, as well as their processing lengths are arbitrary (not polynomial).

We denote the cost of the demand profile $DeP(\mathcal{J})$ as $|DeP(\mathcal{J})|$. Specifically, $|DeP(\mathcal{J})| = \sum_{I_i \in \mathcal{I}} D(I_i)$.

Observation 3. *The demand profile of an instance is a lower bound on the cost of any feasible solution. Therefore $OPT(\mathcal{J}) \geq |DeP(\mathcal{J})|$.*

Proof. There are $|A(I_i)|$ jobs active within an interesting interval I_i. Then any feasible solution would have $\lceil \frac{|A(I_i)|}{g} \rceil$ machines busy during the interval I_i. Moreover, $Sp(\mathcal{J}) = Sp(\bigcup_{I_i \in \mathcal{I}} I_i)$. Hence, the proof follows. $\square$

5.6 Interval Jobs

Before describing the more general problem of flexible jobs, we discuss the simpler case of interval jobs, and outline the algorithms of Kumar and Rudra [42] and Alicherry and Bhatia [1], which improve the existing approximation algorithms for the busy time problem on interval jobs by giving a factor 2 approximation. We start by describing the algorithm of Kumar and Rudra [42] since the problem they consider is more closely related to the busy time problem.

5.6.1 Kumar and Rudra's Algorithm

Here we provide an overview of the algorithm of Kumar and Rudra [42] for fiber minimization problem which implies a 2 approximation for the busy time problem on interval jobs. The fiber minimization problem is as follows. An optical fiber network needs to satisfy the given set of requests, that need to be assigned to consecutive links or edges connected in a line. There are n of these links. Each request needs some links $[i, i+1, \ldots, j]$, where $1 \leq i < j \leq n$. Each segment of an optical fiber can support μ wavelengths over the consecutive links that it spans, and no two requests can be assigned the same wavelength on the same fiber, if they need to use the same link. We want a feasible assignment of the requests such that total length of optical fiber used is minimized. Notice, this is very similar to the busy time problem on interval jobs. Think of the requests as interval jobs. If a request needs the consecutive links $[i, i+1, \ldots, j]$, where $1 \leq i < j \leq n$, then this can be equivalently thought of as an interval job with release time i and deadline j, i.e., with a window $[i, j)$, with processing length $j - i$, in a discrete setting where time is slotted. The total number of links being n, the processing lengths of the jobs here is linear. In this case, we can think of each slot as an interesting interval (since jobs begin and end only at slots) and define the demand profile as the tuples $(i, D(i))$, where i is a time slot $\in [1, \ldots, n]$. Their algorithm proceeds in two phases. In the first phase they assign the jobs to levels within the demand profile (where the total number of levels equals the maximum raw demand at any point), and potentially allow for a limited infeasibility in this packing. Specifically, at most two jobs can be assigned to the same level anywhere within the demand profile. In the second phase, they give a feasible packing of the jobs, considering μ levels at a time, removing the infeasibility introduced earlier, but without exceeding the cost by more than a factor of 2. This is done as follows. For levels $\{(i-1)\mu + 1, \ldots, i\mu\}$, $(i \in \{1, \ldots, D_{max}\})$ where D_{max} is the maximum height of the demand profile), Kumar and Rudra [42] open two fibers, instead of one, and assign jobs to fibers, such that two jobs which were assigned to the same level in the demand profile, get assigned to separate fibers according to a simple parity based assignment. Their analysis assumes that the raw demand at every time slot t, $|A(t)|$ is a multiple of μ and charges to such a demand profile. It is clear that the demand profile gets charged at most twice, respecting the μ capacity constraint of the fibers.

The polynomial time complexity of the algorithm crucially depends on the fact that we have n links, and hence the job lengths being linear, we need to consider only a linear number of slots. The above does not hold for the busy time problem for interval jobs with arbitrary release times, deadlines and processing lengths. The number of time instants to consider may not be polynomial. However, the key observation is that even if the release times and deadlines of jobs are not integral, there can be at most $2n$ *interesting* intervals, such that no jobs begin or end within the interval. The demand profile is uniform over every interesting interval. Therefore, their algorithm can be applied to the busy time problem, with this simple modification, still maintaining the polynomial complexity. The assumption regarding multiple of μ (in the busy time case, this would be g) at every slot, would translate as a multiple of g jobs over every interesting interval. However, note that for an arbitrary instance, we can add dummy jobs spanning any interesting interval I_i where the raw demand $|A(I_i)|$ is not a multiple of g without changing the demand profile. Specifically, if $cg < |A(I_i)| < (c+1)g$, for some $c \geq 0$, then $DeP(I_i) = c+1$, hence adding $(c+1)g - |A(I_i)|$ jobs spanning I_i does not change the demand profile. Thus we can apply their algorithm on the busy time instance, where the demand profile is defined only interesting intervals and the demand everywhere is a multiple of g. The assignments to the fibers as done by their algorithm in Phase 2, will give the bundles for the busy time problem.

5.6.2 Alicherry and Bhatia's Algorithm

Now, we describe how the work of Alicherry and Bhatia [1], implies another, elegant algorithm with a 2 approximation for interval jobs. Alicherry and Bhatia study a generalized coloring and routing problem on interval and circular graphs, motivated by optical design systems. Though the problems they consider are not directly related to the busy time problem, we can use their techniques to develop the 2 approximation algorithm. Similar to Kumar and Rudra's work, their goal is to route certain requests, which require to be assigned to consecutive links or edges in the interval or circular graph. At each link, we color the requests assigned to that link. The colors are partitioned into sets, which are ordered, such that colors in the higher numbered sets cost more. The total cost of the solution is the sum of the costs of the highest colors used at all the links, and the objective is to minimize this cost. Though this problem seems quite different from the busy time problem on interval jobs, the one of the key observations is that the cost needs to be a monotonically non-decreasing function respecting the set order. It need not be a strictly increasing function. Hence, we can think of the sets numbered in a linear order, and give each set g colors. We set the number of all the colors in a set i as i. If $c \cdot g + k$ requests use a link, the cost of that link would be the cost of the highest color used at the link, which is $c + 1$. Hence, what we are really summing is the total cost of the demand profile defined on a set of interval jobs, which have integral release times, and deadlines, and linear processing lengths, since the number of links is n (part of the input). Therefore, a 2 approximation algorithm minimizing the cost is really providing a solution that costs at most twice the demand profile of this restricted instance. The technique used involves setting up a flow graph with a certain structure, depending on the current demands or requests as yet unassigned. It can be easily proved that the graph has a cut of size at least 2 everywhere if the demand everywhere is at least 2. Now, we find a flow of size two in this graph from the source to the sink. Each flow path will consist of a set of disjoint requests or demands (where the disjointness refers to the links they need to use), and the union of the two flows will reduce a demand of at least unity from every link. This is repeated till the demand is 0 or 1 everywhere.

As in Section 5.6.1, we use the following observation: the time slots can be considered to be interesting intervals for a set of interval jobs. The busy time instance with non-polynomial job lengths and arbitrary release times and deadlines has a linear number of interesting intervals, and hence we can think of our instance in this discretized setting. Therefore, we can apply their algorithm, modified accordingly, to our problem to get an algorithm within twice of the optimal solution. The algorithm will consider a busy time instance with the demand profile defined on interesting intervals and with a multiple of g jobs everywhere without any loss of generality. It will first open up two bundles. The flow graph is then set up as defined by Alicherry and Bhatia. For the first g iterations, the algorithm will find $2g$ flow paths (each consisting of disjoint interval jobs), the union of which removes at least a demand of g from everywhere. We assign g of these paths to one bundle and the remaining g to the other. Each flow path consists of disjoint jobs, hence, each bundle will have at most g jobs at time instant. Moreover, together, these bundles have removed a demand g from everywhere in the demand profile, hence they have charged the lowermost level (which is also the widest level) of the demand profile at most twice. The demand profile is now suitable modified after removing the jobs already assigned. Once again two bundles are opened, and the same procedure is performed for the next g iterations. This continues till the demand profile becomes empty everywhere, in other words, all jobs are assigned. The resultant bundles are feasible and charge the demand profile at most twice.

5.6.3 Lower bound

Though the upper bound of 2 was shown by Kumar and Rudra [42] and Alicherry and Bhatia [1] for their algorithms, a lower bound on the performance of the algorithms was not provided. Here we show that for both these algorithms, the approximation ratio obtained can be arbitrarily close to 2. Figure 5.2 shows an instance of interval jobs, for which both the algorithms implied by the work of Kumar and Rudra and Alicherry and Bhatia approach a factor of 2 of the optimal solution. In this example, $g = 2$ and there are two interval jobs of length 1, one interval job of length ϵ, one of length $\epsilon' < \epsilon$, and one of length $\epsilon - \epsilon'$. As required by the analysis of Kumar and Rudra and Alicherry and Bhatia, the demand everywhere is a multiple of g. A possible output by both algorithms (adapted to the busy time problem as described) has cost $2 + \epsilon$, whereas the optimal solution has cost $1 + \epsilon$. For $\epsilon \to 0$, the approximation factor approaches 2.

Theorem 11. *There exist* 2 *approximation polynomial time algorithms for the busy time problem on interval jobs. The approximation factor is tight.*

Proof. The proof follows from the discussions of Sections 5.6.1, 5.6.2, and 5.6.3. □

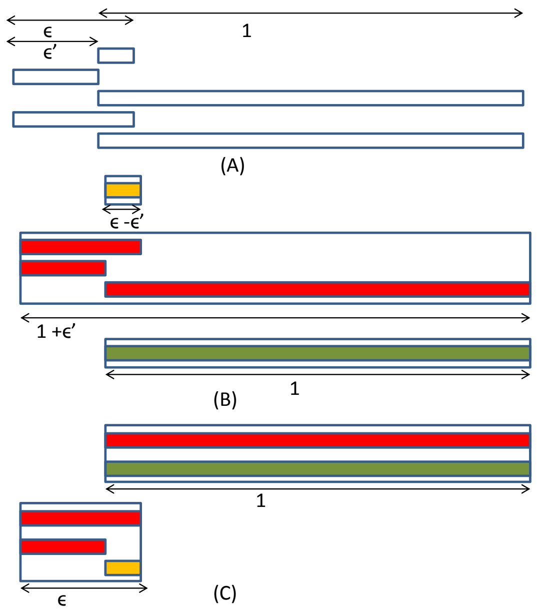


Figure 5.2: (A) An instance of interval jobs and $g = 2$. (B) A possible output by the algorithms of Kumar and Rudra [42] and Alicherry and Bhatia [1], of cost $= 2 + \epsilon$. (C) The optimal solution of cost $1 + \epsilon$.

5.7 Flexible Jobs

5.7.1 Prior 4 approximation

In this section we discuss the more general problem of flexible jobs. This problem was studied by Khandekar et al. [39], who refer to this problem as the real-time scheduling problem. They gave a 5 approximation for this problem when the jobs can have arbitrary widths. For the unit width jobs, their analysis can be modified to give a 4 approximation.

As a first step towards proving the 5-approximation for the general problem with flexible windows and non-unit width, Khandekar et al. [39] prove that if g is unbounded, then this problem is polynomial-time solvable. The output of their dynamic program essentially converts an instance of jobs with flexible windows to an instance of interval jobs (with rigid windows), by fixing the start and end times of every job.

Theorem 12. *[39] If g is unbounded, the real-time scheduling problem is polynomial-time solvable.*

From Theorem 12, the busy time of the output of the dynamic program on the set of flexible interval jobs $\mathcal{J}$ is equal to $OPT_\infty(\mathcal{J})$.

Once Khandekar et al. obtain the modified interval instance, they apply their 5 approximation algorithm for non-unit width interval jobs to get the final bound. However, for jobs with unit width, their algorithm and analysis can be modified without loss to apply the 4 approximation algorithm of Flammini et al. [22] for interval jobs with bounded g to get the final bound of 4.

The 2 approximation algorithm [42] for interval instance charges the demand profile, hence it is immediately not clear how to extend it to handle flexible jobs since the demand profile cannot be defined analogous to the interval case. One possible natural extension is to follow the approach of Khandekar et al., to convert a flexible instance to an interval instance, and then apply the algorithm to this modified instance. Furthermore, the algorithm of Kumar and Rudra assumes that the demand profile everywhere is a multiple of g. Hence, after modifying the instance to an interval instance, we need to add dummy jobs accordingly to interesting intervals to bring up their demands to multiples of g. However, there exists an instance where this algorithm will approach a factor of 4 of the optimal solution. This is the worst that it can do, since we prove in the following lemma that the demand profile of the modified instance of interval jobs is at most twice the demand profile of the optimal solution (note that once the jobs have been assigned in the optimal solution, their positions get fixed, and hence the demand profile can now be computed easily).

Lemma 15. *The demand profile of the output of the dynamic program converting the flexible jobs to interval jobs is at most* 2 *times the demand profile of an optimal solution structure.*

Proof. The objective function of the dynamic program (12) is to minimize the total busy time of a flexible job instance assuming g is unbounded. Since the dynamic program is optimal, it will pack as many jobs and as much length as possible together. Hence, if a job has a choice of being assigned to a spot where other jobs need to be assigned as well, then it will be assigned at that spot instead of at some other spot where no jobs need to be assigned. Therefore, at any level of the demand profile, we can charge it to the mass of the level below, and if it is the first level, we charge it to OPT_∞ bound. Hence, in total the optimal solution gets charged twice, once by the mass bound, and once by the span bound, giving a 2 approximation. □

There exists an instance of flexible jobs for which the demand profile output by the dynamic program of Khandekar et al. approaches 2 times the cost of the demand profile of the optimal solution structure. We have shown such an instance in Figure 5.3. The instance consists of the following types of jobs: one interval job of unit length, followed by $(g-1)$ disjoint sets of identical g interval jobs, where in the i^{th} set, each job is of length $1+i\epsilon$, $(i \in [1,\ldots,(g-1)])$. Apart from these, there are $g-1$ flexible jobs, where the i^{th} job is of length $1+i\epsilon$, where $i \in [1,\ldots,(g-1)]$ and has a feasible window spanning the the windows of the first $i+1$ disjoint sets of interval jobs, as shown in the figure. An optimal solution would pack the $g-1$ flexible jobs with the first interval job, and the remaining $(g-1)$ disjoint sets of identical g interval jobs in their respective windows, with a total busy time of $g+\left(\frac{g(g+1)}{2}-1\right)\epsilon$. The dynamic program however disregards capacity constraints of the machines, and simply tries to minimize the span of the solution. Hence, with a little effort it can be seen that the unique output of the dynamic program (as shown in Figure 5.3) would have a span of $g+\frac{g(g-1)}{2}\epsilon$, and the demand profile on imposing a capacity of g is of cost $2g-1+g(g-1)\epsilon$, which approaches 2 the cost of the optimal solution when $\epsilon \to 0$.

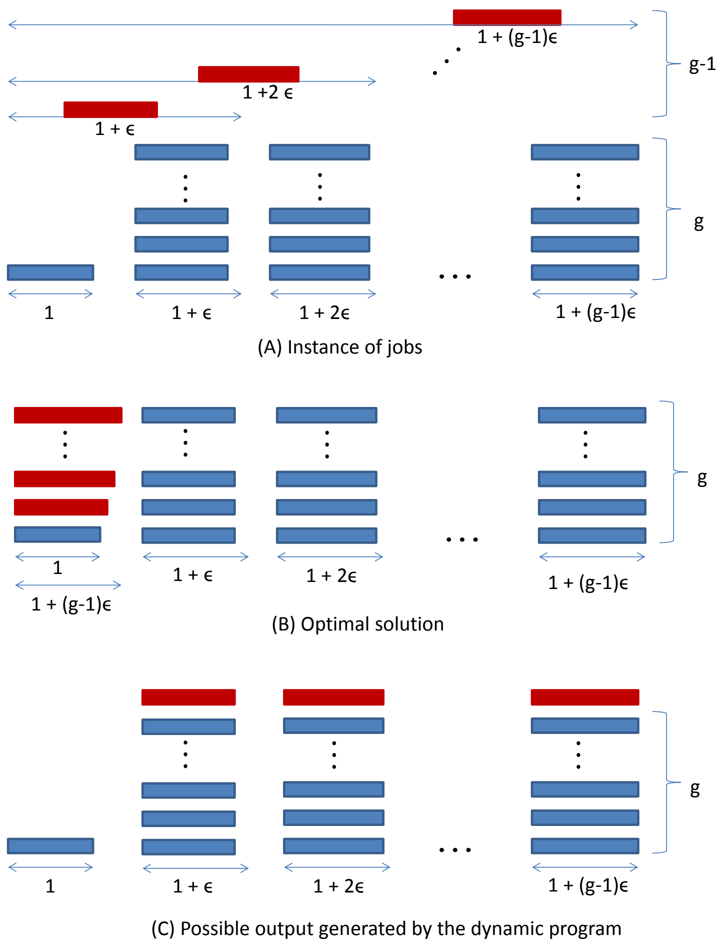


Figure 5.3: (A) An instance of interval and flexible jobs. (B)The optimal solution of busy time $g + \frac{g^2+g-2}{2}\epsilon$. (C)The output of the dynamic program of Khandekar et al. [39] of busy time $= 2g - 1 + g(g-1)\epsilon$.

Theorem 13. *A natural extension of the* 2 *approximation algorithm of Kumar and Rudra [42] (or the algorithm of Alicherry and Bhatia [1]) for the interval jobs problem, to the flexible jobs problem, gives an approximation of* 4*. This factor is tight.*

Proof. The approximation upper bound of 4 follows from Lemma 15 and Theorem 11. However, there is a tight example as well. In this example, we have an instance of interval and flexible jobs. The instance consists of a unit length interval job, followed by $g-1$ disjoint occurrences of the gadget shown in Figure 5.4. The gadget consists of g unit length interval jobs, $2g-2$ interval jobs of length ϵ, 2 interval jobs of length ϵ' and 2 jobs of length $\epsilon - \epsilon'$, as shown in the figure. There are $g-1$ unit length flexible jobs, each with windows spanning the windows of the union of all of the interval jobs.

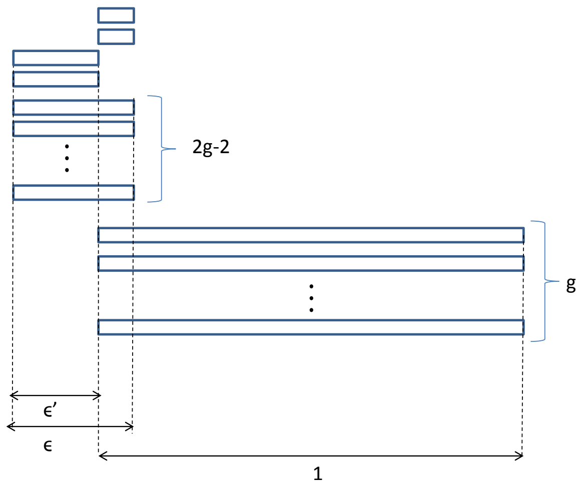


Figure 5.4: Here we show the gadget for the factor 4 example.

On running the dynamic program to minimize span, a possible output is when each of $g-1$ flexible jobs are packed along with the $g-1$ gadgets. For applying the algorithms of Kumar and Rudra (or Alicherry and Bhatia), we need to make sure the demand everywhere is a multiple of g. Hence we add $g-1$ dummy jobs of unit length coincident with the first unit length interval job, as well as with each of the $g-1$ gadgets with a flexible job. This is shown in Figure 5.5.

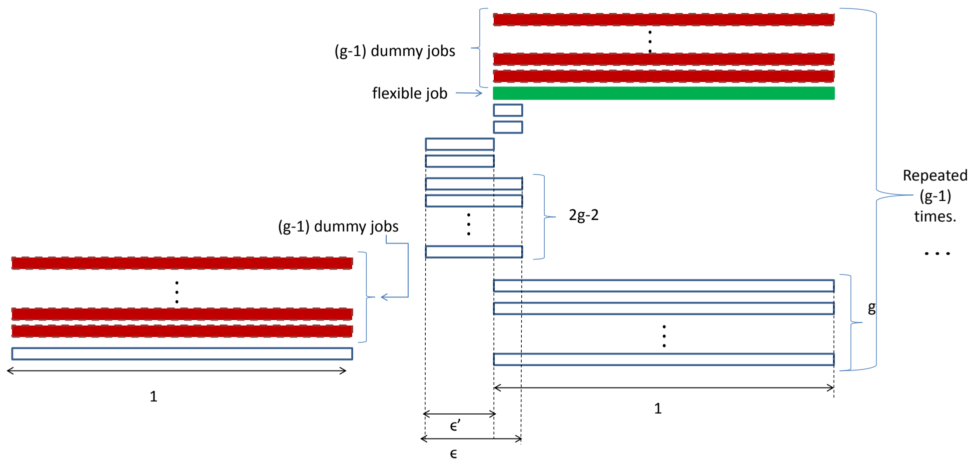


Figure 5.5: Output of the dynamic program on the instance of interval and flexible jobs for the factor 4 example.

Now, one possible run of the algorithm of Kumar and Rudra (or Alicherry and Bhatia) may result in the packing shown in Figure 5.6, of cost $1 + 4(g-1) + O(\epsilon)$, whereas the optimal solution is to pack the flexible jobs with the first unit length interval job, and pack all the identical unit length jobs together, with a total cost of $g + O(\epsilon)$. Hence the ratio approaches 4 for large g and small ϵ.

□

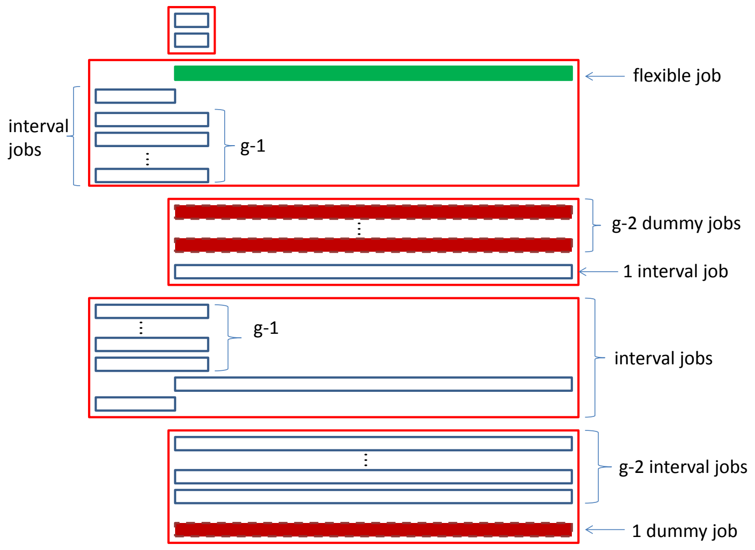


Figure 5.6: Here we show the busy time bundling produced by a possible run of Kumar and Rudra or Alicherry and Bhatia's algorithms on one gadget along with the flexible job and dummy jobs.

5.7.2 A 3 approximation algorithm

We now give a 3 approximation algorithm for the problem of flexible jobs. Let us consider a set of flexible jobs $\mathcal{J}'$. Analogous to Khandekar et al. [39], we first convert this instance to an instance of interval jobs by running the dynamic program on $\mathcal{J}'$. Let $\mathcal{J}$ be the resultant set of interval jobs on fixing the job windows according to the output of the dynamic program, and let $OPT_\infty(\mathcal{J}')$ denote the cost or busy time of the output of the dynamic program. From Observation 2, we know that $OPT_\infty(\mathcal{J}') \leq OPT(\mathcal{J}')$.

On the interval job instance $\mathcal{J}$, we run our algorithm, which we call GREEDY-TRACKING. Note that, now the span of the effective windows of every job becomes equal to its processing length. For an interval job j, we denote its window $[r_j, d_j)$ as the span of j: $Sp(j)$. For the rest of the section, we assume that our input consists of interval jobs.

Before we describe the algorithm, we define the notion of a *track*.

Definition 7. *A* track *of interval jobs is a set of interval jobs with disjoint spans.*

Given a feasible solution, one can think of each bundle $\mathcal{B}$ as the union of g individual tracks of jobs. The main idea behind the algorithm is to identify such tracks iteratively, bundling the first g tracks into a single bundle, the second g tracks into the second bundle, etc. FIRSTFIT [22] suffers from the fact that it greedily considers jobs one-by-one; GREEDYTRACKING is less myopic in that it identifies jobs whole tracks at a time.

In the i^{th} iteration, $i \geq 1$, the algorithm identifies a track $\mathcal{T}_i \subseteq \mathcal{J} \setminus \bigcup_{k=1}^{i-1} \mathcal{T}_k$ of maximum length $\ell(\mathcal{T}_i)$ and assigns it to bundle B_p, where $p = \lceil \frac{i}{g} \rceil$. One can find such a track efficiently via weighted interval scheduling algorithms [13]. We consider the lengths of the interval jobs as their weights, and find the maximum weight set of interval jobs with disjoint spans. If the final solution has κ bundles, the algorithm's total cost is $\sum_{i=1}^{\kappa} |Sp(\mathcal{B}_i)|$. The pseudocode for GREEDYTRACKING is provided in Algorithm 6[2].

We next prove a key property of GREEDYTRACKING: the span of any track is at least half that of the remaining unscheduled jobs. In particular, the span of any bundle is at most twice that of the first track to be assigned to it.

[2]This algorithm was also given by Chang [6], however the proof given here is shorter and simpler.

Algorithm 6 GreedyTracking. Inputs: $\mathcal{J}$, g.

```
1: $\mathcal{S} \leftarrow \mathcal{J}$, $i \leftarrow 1$.
2: while $\mathcal{S} \neq \emptyset$ do
3:     Compute the longest track $\mathcal{T}_i$ from $\mathcal{S}$ and assign it to bundle $B_{\lceil \frac{i}{g} \rceil}$.
4:     $\mathcal{S} \leftarrow \mathcal{S} \setminus \mathcal{T}_i$, $i \leftarrow i + 1$.
5: end while
6: return bundles $\{\mathcal{B}_p\}_{p=1}^{\lceil \frac{i-1}{g} \rceil}$
```

Lemma 16. *Let $\mathcal{T}_i$ be the ith track found by* GreedyTracking*, $i \geq 1$. Let $\mathcal{J}'_i \subseteq \mathcal{J}$ denote the set of unscheduled jobs $\mathcal{J} \setminus \bigcup_{k=1}^{i-1} \mathcal{T}_k$. Then $|Sp(\mathcal{J}'_i)| \leq 2 \cdot |Sp(\mathcal{T}_i)|$.*

Proof. In order to prove this, we first prove the following. There exists two tracks $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$, such that $\mathcal{T}_1^* \subseteq \mathcal{J}'_i$ and $\mathcal{T}_2^* \subseteq \mathcal{J}'_i$, $\mathcal{T}_1^* \cap \mathcal{T}_2^* = \emptyset$ and $Sp(\mathcal{T}_1^*) \cup Sp(\mathcal{T}_2^*) = Sp(\mathcal{J}'_i)$. Let us assume, by way of contradiction, that the above is not true. In other words, for every pair of disjoint tracks from the set of yet unscheduled jobs $\mathcal{J}'_i$, the union of their spans does not cover $Sp(\mathcal{J}'_i)$.

Let $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$ be two disjoint tracks from $\mathcal{J}'_i$, such that the union of their spans is maximum among all such tracks. By our assumption, $|Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)| < |Sp(\mathcal{J}'_i)|$. This implies that there exists an interval $I \in Sp(\mathcal{J}'_i)$, such that $I \notin Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. Let I be $[t_I, t'_I)$.

Clearly, no job $j \in \mathcal{J}'_i$ has a window $\subseteq [t_I, t'_I)$, by the maximality of $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. In fact, all jobs intersecting I, must intersect with some job in both $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$, because of the same reason.

In the following we prove that *no* such interval I can exist given our assumptions on $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$.

Let us first define the notion of *minimum replaceable set*.

Definition 8. *Consider a track $\mathcal{T}$ and an interval job j with window $[r_j, d_j)$. Let $j_f \in \mathcal{T}$ have the earliest deadline $d_{j_f} > r_j$ such that $r_{j_f} \leq r_j$. Let $j_\ell \in \mathcal{T}$ have the latest release time $r_{j_\ell} < d_j$, such that $d_{j_\ell} \geq d_j$. Then the set of jobs in $\mathcal{T}$ with windows in $[r_{j_f}, d_{j_\ell})$ is the* minimum replaceable set *$MRS(j, \mathcal{T})$ for j in $\mathcal{T}$. In other words, it is the set of jobs whose union has the minimum span, such that $\{\mathcal{T} \cup j\} \setminus MRS(j, \mathcal{T})$ is a valid track. If there exists no such job j_f (respectively, j_ℓ), then $MRS(j, \mathcal{T})$ would consist of jobs in the interval $[r_j, d_{j_\ell})$ (respectively, $[r_{j_f}, d_j)$). If there exists no such job j_f as well as j_ℓ, then $MRS(j, \mathcal{T}) = \emptyset$. (See Figure 5.7 for an example).*

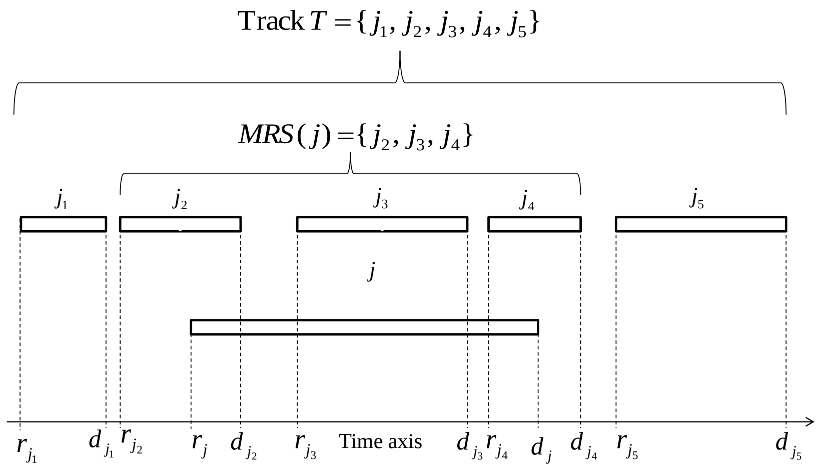
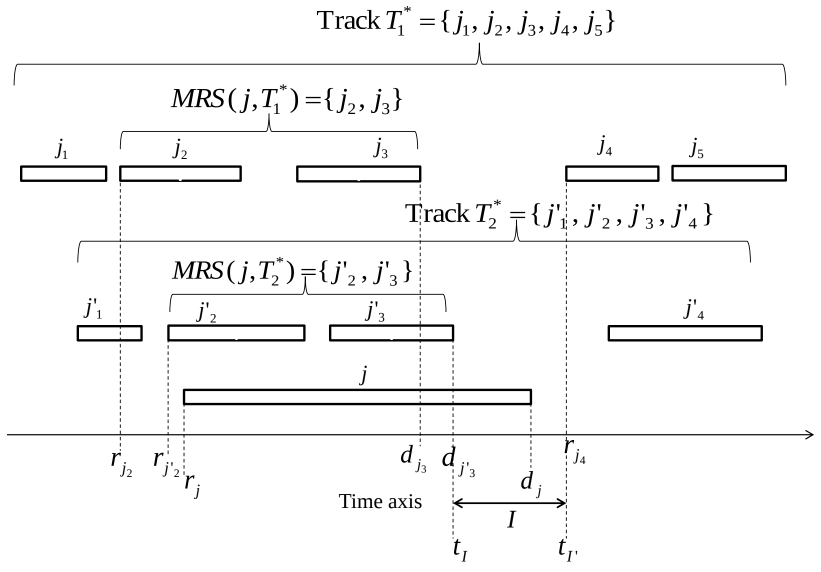


Figure 5.7: An example showing the minimum replaceable set of a job j, i.e., $MRS(j)$ with respect to a track T.

Now we proceed with the proof.

Case 3. *There exists a job j in $\mathcal{J}'_i \setminus \{\mathcal{T}_1^* \cup \mathcal{T}_2^*\}$, such that $r_j < t_I$ and $t_I < d_j < t'_I$.*

Consider $MRS(j, \mathcal{T}_1^*)$ and $MRS(j, \mathcal{T}_2^*)$. Without loss of generality, they cannot be empty, as otherwise, by adding j to the corresponding track, we could have increased $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. Let j_e be the job with the earliest release time r_e in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$, and without loss of generality, suppose it belongs to $\mathcal{T}_1^*$. Replacing $MRS(j, \mathcal{T}_2^*)$ with j will increase $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. This is because, whereas $Sp(MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)) < [r_e, t_I)$, $Sp(j) \geq [d_e, t_I]$, and hence $Sp(MRS(j, \mathcal{T}_1^*) \cup j) > [r_e, t_I]$. See Figure 5.8 for an example.



Track $T_1^* = \{j_1, j_2, j_3, j_4, j_5\}$
$MRS(j, T_1^*) = \{j_2, j_3\}$
j_1
j_2
j_3
j_4
j_5
Track $T_2^* = \{j'_1, j'_2, j'_3, j'_4\}$
$MRS(j, T_2^*) = \{j'_2, j'_3\}$
j'_1
j'_2
j'_3
j'_4
j
r_{j_2}
$r_{j'_2}$
r_j
d_{j_3}
$d_{j'_3}$
d_j
r_{j_4}
Time axis
I
t_I
$t_{I'}$

Figure 5.8: An example for Case 3 of Lemma 16. Here replacing $MRS(j, \mathcal{T}_2^*)$ by j will increase the span of the union of the tracks: $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$.

Hence, this case is not possible.

Case 4. *There exists a job j in $\mathcal{J}'_i \setminus \{\mathcal{T}_1^* \cup \mathcal{T}_2^*\}$, such that $t_I < r_j < t'_I$ and $d_j > t'_I$.*

Consider $MRS(j, \mathcal{T}_1^*)$ and $MRS(j, \mathcal{T}_2^*)$. Without loss of generality, they cannot be empty sets as argued in Case 1. Let the job j_ℓ have the latest deadline d_ℓ in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$. Without loss of generality, suppose j_ℓ belongs to $\mathcal{T}_1^*$. Then we can replace $MRS(j, \mathcal{T}_2^*)$ with j, thereby increasing $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$. This is because, $Sp(MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)) \leq [t'_I, d_\ell)$, whereas $Sp(MRS(j, \mathcal{T}_1^*) \cup j) > [t'_I, d_\ell)$, since $Sp(j) > [t'_I, d_j)$.

Hence, this case is also not possible.

Case 5. *There exists a job j, such that $[r_j, d_j) \supset [t_I, t'_I)$.*

Let the earliest release time of any job in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$ be r_e (the corresponding job is j_e) and the latest deadline of any job in $MRS(j, \mathcal{T}_1^*) \cup MRS(j, \mathcal{T}_2^*)$ be d_ℓ (the corresponding job is j_ℓ). Once again we assume WLOG, that these sets are not empty. If both the jobs j_e and j_ℓ belong to the same track, say, $\mathcal{T}_1^*$, we can replace $MRS(j, \mathcal{T}_2^*)$ with j in $\mathcal{T}_2^*$ and increase the union of the span of $\mathcal{T}_1^* \cup \mathcal{T}_2^*$. This is because, $Sp(j) \geq [d_e, r_\ell)$ and includes $I = [t_I, t'_I)$, whereas $Sp(MRS(j, \mathcal{T}_2^*) \setminus MRS(j, \mathcal{T}_1^*))$ is at most $[d_e, r_\ell) \setminus [t_I, t'_I)$. Therefore, j_e and j_ℓ must belong to different tracks.

Without loss of generality, let $j_e \in \mathcal{T}_1^*$ and $j_\ell \in \mathcal{T}_2^*$. Let us replace $MRS(j, \mathcal{T}_2^*)$ with j. Next, we put j_ℓ in $\mathcal{T}_1^*$ replacing $MRS(j_\ell, \mathcal{T}_1^*)$. Note that $d_e \leq t_I$, $r_\ell \geq t'_I$, and $t'_I - t_I > 0$ by our assumptions. Therefore, $j_e \notin MRS(j_\ell, \mathcal{T}_1^*)$. In fact, none of the jobs in $\mathcal{T}_1^*$ with release time $< t'_I$ are included in $MRS(j_\ell, \mathcal{T}_1^*)$, and hence none of them are discarded. Therefore, the loss of coverage by $\mathcal{T}_1^*$ after putting j_ℓ in place of $MRS(j_\ell, \mathcal{T}_1^*)$ is at most the interval $[t'_I, r_\ell)$. However, we have added j to

$\mathcal{T}_2^*$, and not only does j span $[t_I, t'_I)$, but also the interval $[t'_I, r_\ell]$, since $d_j \geq r_\ell$ for j_ℓ to be originally a part of $MRS(j, \mathcal{T}_2^*)$. Hence, we would increase $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)$, which is a contradiction. Therefore, this case is also not possible,

Since no job window in $\mathcal{J}'_i$ can intersect I, there exists no such I in $Sp(\mathcal{J}'_i)$. Therefore, $Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*) = Sp(\mathcal{J}'_i)$. Furthermore, $|Sp(\mathcal{T}_1^* \cup \mathcal{T}_2^*)| \leq |Sp(\mathcal{T}_1^*)| + |Sp(\mathcal{T}_2^*)|$, in other words, the longer of $\mathcal{T}_1^*$ and $\mathcal{T}_2^*$ is $\geq \frac{|Sp(\mathcal{J}'_i)|}{2}$. Since, $\mathcal{T}_i$ is the longest track in $\mathcal{J}'_i$, therefore, $|Sp(\mathcal{J}'_i)| \leq 2|Sp(\mathcal{T}_i)|$. □

We next prove that our algorithm generates a solution within 3 times the cost of an optimal solution via the following lemmas.

Lemma 17. *For any $i > 1$, the span of bundle $\mathcal{B}_i$ can be bounded by the mass of the bundle $\mathcal{B}_{i-1}$ as follows: $|Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{B}_{i-1})}{g}$.*

Proof. Let $\mathcal{T}_i^1$ denote the first track of the bundle $\mathcal{B}_i$. From Lemma 16, it follows that $|Sp(\mathcal{B}_i)| \leq 2|Sp(\mathcal{T}_i^1)|$. The jobs in $\mathcal{T}_i^1$ are disjoint by definition of a track, hence $|Sp(\mathcal{T}_i^1)| = \ell(\mathcal{T}_i^1)$, and $|Sp(\mathcal{B}_i)| \leq 2\ell(\mathcal{T}_i^1)$. Since $\mathcal{T}_i^1$ started the ith bundle, bundle $\mathcal{B}_{i-1}$ must already have had g tracks in it. Furthermore, the lengths of these tracks are longer than that of $\mathcal{T}_i^1$ since GreedyTracking chooses tracks in non-increasing order of length. Therefore, $\ell(\mathcal{B}_{i-1}) = \sum_{p=1}^{g} \ell(\mathcal{T}_{i-1}^p) \geq g \cdot \ell(\mathcal{T}_i^1)$. It follows that $|Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{B}_{i-1})}{g}$.

□

Lemma 18. *The total busy time of all the bundles except the first one is at most twice that of an optimal solution for the entire instance. Specifically, $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2OPT(\mathcal{J}')$.*

Proof. This proof follows from Lemma 17. For any $i > 1$, $|Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{B}_{i-1})}{g}$. Summing over all $i > 1$, we get the following: $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2\frac{\sum_{i>1} \ell(\mathcal{B}_{i-1})}{g} = 2\frac{\sum_{i>1}\sum_{j\in\mathcal{B}_{i-1}} \ell(j)}{g}$. Therefore, $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2\frac{\ell(\mathcal{J})}{g}$. Note that $\ell(\mathcal{J}) = \sum_{j\in\mathcal{J}'} p_j$, where $\mathcal{J}'$ is the original flexible interval job instance. This is true because the dynamic program converting a flexible instance to an interval instance, does not reduce the processing length of any job. Hence, from Observation 1, $OPT(\mathcal{J}') \geq \frac{\ell(\mathcal{J}')}{g}$. It follows that $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2OPT(\mathcal{J}')$. □

Theorem 14. *The cost of the algorithm is at most* 3 *times the cost of an optimal solution. Specifically,* $\sum_i |Sp(\mathcal{B}_i)| \leq 3OPT(\mathcal{J})$.

Proof. From Lemma 18, $\sum_{i>1} |Sp(\mathcal{B}_i)| \leq 2OPT(\mathcal{J}')$. Furthermore, $|Sp(\mathcal{B}_1)| \leq OPT_\infty(\mathcal{J}')$. From Observation 2, $OPT_\infty(\mathcal{J}') \leq OPT(\mathcal{J}')$. Therefore, $\sum_i |Sp(\mathcal{B}_i)| \leq 3OPT(\mathcal{J}')$. □

Figure 5.9 shows that the approximation factor of 3 achieved by GREEDY-TRACKING is tight. In the instance shown, a gadget of $2g$ interval jobs is repeated g times. In this gadget, there are g identical unit length interval jobs which overlap for ϵ amount with another g identical unit length interval jobs. The g gadgets are disjoint from one another, which means, there is no overlap among the jobs of any two gadgets. There are $2g$ flexible jobs, whose windows span the windows of all the g gadgets. These jobs are of length $1 - \frac{\epsilon}{2}$. An optimal packing would pack each set of g identical jobs of each gadget in one bundle, and the flexible jobs in 2 bundles, giving a total busy time of $2g + 2 - \epsilon$. However, the dynamic program minimizing the span does not take capacity into consideration, hence in a possible output, the flexible jobs may be packed 2 each with each of the g gadgets, in a manner such that they intersect with all of the jobs of the gadget. Hence, the flexible jobs cannot be considered in the same track as any unit interval job in the gadget it is packed with. Due to the greedy nature of GreedyTracking, the tracks selected would not consider the flexible jobs in the beginning, and the interval jobs may also get split up as in Figure 5.10, giving a total busy time of $4(1-\epsilon)g + (2 - o(\epsilon))g = (6 - o(\epsilon))g$, hence it approaches a factor 3 asymptotically.

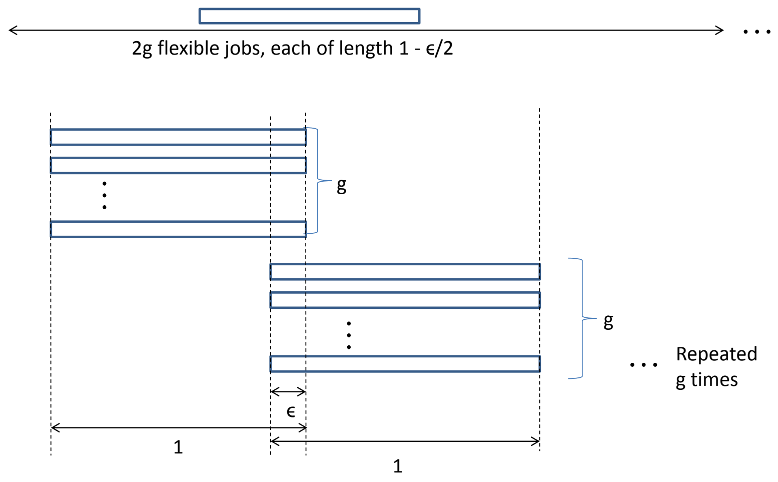


Figure 5.9: Gadget for factor 3 for GreedyTracking

5.7.3 Preemptive Model

In this section, we remove the restriction, a job needs to be assigned to a single machine. A job j needs to be assigned a total of p_j time units within the interval $[r_j, d_j)$ and at most one machine may be working on it at any given time.

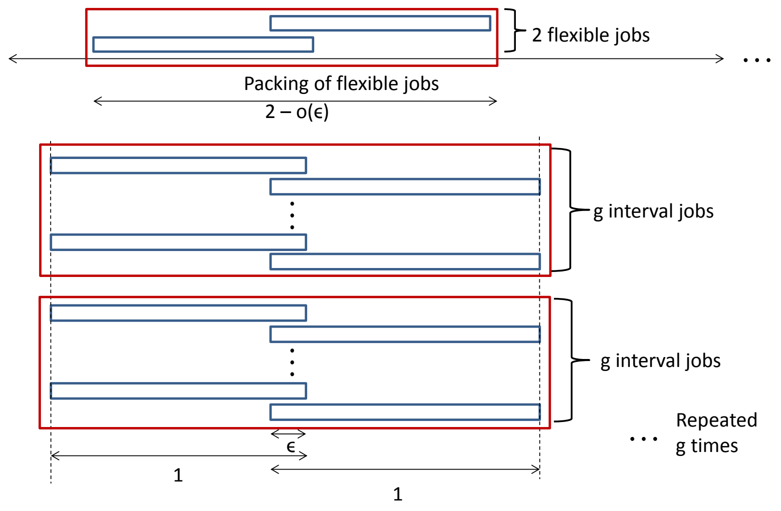


Figure 5.10: Possible packing by GreedyTracking

Theorem 15. *For unbounded g and preemptive jobs, there is an exact algorithm to minimize busy time.*

Proof. The algorithm is a simple greedy one. Let $\mathcal{J}_1$ be the set of jobs of earliest deadline d_1 and let the longest job $j_{max,1}$ in $\mathcal{J}_1$ have length $\ell_{max,1}$. We open the interval $[d_1 - \ell_{max,1}, d_1)$, and for every the job $j \in \mathcal{J}$ such that $[r_j, d_j) \cap [d_1 - \ell_{max,1}, d_1) \neq \emptyset$, we schedule it up to $d_1 - r_j$ in the interval $[r_j, d_1)$. Then we shrink the interval $[d_1 - \ell_{max,1}, d_1)$ and adjust the windows and remaining processing lengths of the jobs in $\mathcal{J}$ and then repeat till all jobs in $\mathcal{J}$ have been completely scheduled.

In the first iteration, without loss of generality, the optimal solution will also open the interval $[d_1 - \ell_{max,1}, d_1)$; $j_{max,1}$ has to be scheduled completely d_1 and since d_1 is the earliest deadline, opening this length of interval as late as possible ensures that we can schedule the maximum length of any job in the instance $\mathcal{J}$ with $j_{max,1}$.

The correctness follows by induction on the remaining iterations. □

As a consequence, one can approximate preemptive busy time scheduling for bounded g. First, solve the instance under the assumption that g is unbounded; denote by $\mathcal{S}_\infty$ this (possibly infeasible) solution. The busy time of $\mathcal{S}_\infty$ is $OPT_\infty(\mathcal{J})$, and is a lower bound on the optimal solution for bounded g. The algorithm for bounded g will commit to working on job j precisely in the time intervals where $\mathcal{S}_\infty$ had scheduled it. Partition the busy time of $\mathcal{S}_\infty$ into the set of interesting intervals $\{I_1, \ldots, I_k\}$, where $k = \theta(n)$.

For every interesting interval I_i, assign the jobs scheduled in I_i to $\lceil \frac{n(I_i)}{g} \rceil$ machines in arbitrary order, filling the machines greedily such that there is at most one machine with strictly less than g jobs.

For each I_i, at most one machine contains less than g jobs, which we charge to $OPT_\infty(\mathcal{J})$ All other machines are at capacity, i.e., have exactly g jobs and hence we charge them to $\frac{\ell(\mathcal{J})}{g}$. This implies an approximation of 2.

Theorem 16. *There is a preemptive algorithm whose busy time is at most twice that of the optimal preemptive solution, for bounded g.*

Chapter 6: Active Time

In this chapter, we consider the active time problem, which was introduced by Chang et al. [7]. The notion of *active time*, similar to busy time, is motivated by the total amount of time that a machine is actively working. The key difference between busy time and active time is, as outlined in Chapter 1, while the busy time model can open up an unbounded number of machines if necessary, the active time model assumes access to a single machine. The input for both models is the same: a set of jobs, each of which has an associated feasible time window, where it needs to be scheduled; however, the machine capacity is limited with respect to the number of simultaneous jobs that can be processed at any instant of time. An instance which may be feasible in the busy time model (in fact, every instance is feasible in the busy time model), may become *infeasible* in the active time model. For example, consider an instance with $g + 1$ unit jobs of window $[0, 1)$, and the machine capacity is g. This is a perfectly feasible instance in the busy time model, which will open up 2 machines of in the window $[0, 1)$, but becomes infeasible in the active time model.

6.1 Problem Definition

The input consists of a set of jobs $\mathcal{J}$, where each job j has a release time r_j, a deadline d_j, and a length p_j. This means that p_j units of job j must be scheduled within time slots $[r_j, d_j)$, which we sometimes refer to as the *window* of j. We have access to a single machine, which is either active ('on') at any instant or not. The machine can process only g jobs at any time instant. Since there is a single machine, we simply refer to the time axis henceforth in place of the machine. We consider time to be slotted, and hence the release times and deadlines of jobs are integral. Consequently, the jobs are integral in length, and additionally, we allow preemption at integer boundaries. In other words, we consider each job j of length p_j to be a *chain* of p_j unit jobs, with identical windows $[r_j, d_j)$ and the restriction that in any time slot, at most one of these unit jobs can be scheduled. Hence, in this model, $\sum_{j \in \mathcal{J}} p_j$ is polynomial. Let us denote by T the length of the time window, spanning the union of the windows of the entire job instance. In other words, $T = |\bigcup_{j \in \mathcal{J}} [r_j, d_j)|$. We assume without loss of generality that the earliest release time of any job $j \in \mathcal{J}$ is 0 and the latest deadline of any job in $j \in \mathcal{J}$ is T. Sometimes for ease of notation, we will be using t to denote any slot $[t-1, t)$. In this notation, let $\mathcal{T}$ denote the set of time slots $[1, \ldots, T]$. Figure 6.1 shows a collection of jobs and the corresponding optimal schedule minimizing the active time, when machine capacity $g = 3$.

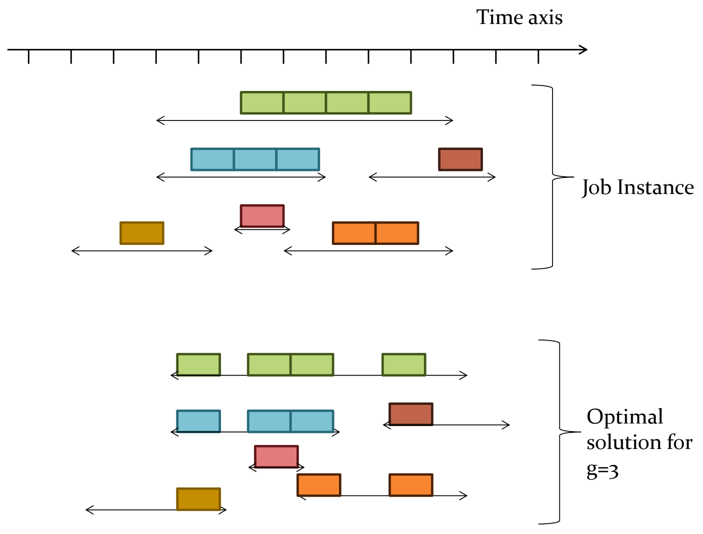


Figure 6.1: Here we show the optimal solution for the active time problem with integral preemption, for an instance of 6 jobs and $g = 3$.

6.1.1 Hardness of the problem

When preemption is not allowed, it becomes strongly NP-hard (by a reduction from 3-PARTITION) to determine whether there exists a feasible solution for non-unit length jobs, even for the special case when the windows of all the jobs are identical [6]. The complexity of the model allowing preemption at integral boundaries has still not been resolved and is an open question.

6.2 Prior Work and Our Contributions

Chang et al. [7] consider the problem of unit length jobs under the slotted time model (in other words, the release times and deadlines are integral), for which they present a fast linear time greedy algorithm. When the release times and deadlines are allowed to be real numbers, they give an $O(n^7)$ dynamic program to solve it; this result has since been improved to an $O(n^3)$-time algorithm in the work of Koehler and Khuller [41]. Chang et al. [7] also consider generalizations to the case where jobs can be scheduled in a union of time intervals (in contrast to the usual single release time and deadline). Under this generalization, once the capacity constraints exceeds two, minimizing active time becomes NP-hard via a reduction from 3-EXACT-COVER. However, for a capacity of two, they provide a polynomial-time solution based on finding maximum degree-constrained subgraphs; this result extends to non-unit length jobs that can be preempted only at integral time points. The active time problem for non-unit length jobs with integral preemption was considered by Chang [6], who showed that any minimal feasible solution is 5 approximate.

We show that in fact any minimal feasible solution for non-unit length jobs with integral preemption, is at most 3 times the cost of an optimal solution and also show that the factor 3 is tight. We then present a 2 approximation algorithm for the problem via an LP rounding scheme and also show that the integrality gap of the LP is 2.

6.3 3 approximation for active time scheduling of chains

First we define the following notation for ease of presenting the analysis.

Definition 9. *A job j is said to be* live *at t if $t \in [r_j, d_j)$*

Definition 10. *A slot is* active *if at least one job is scheduled in it. It is* inactive *otherwise.*

Definition 11. *A slot is* full *if there are g jobs assigned to it. It is* non-full *otherwise.*

A feasible solution σ is specified by a set of active time slots $\mathcal{A} \subseteq \mathcal{T}$, and a mapping or assignment of jobs to the time slots in $\mathcal{A}$, such that at most g jobs are scheduled in any slot in $\mathcal{A}$, at most one unit of any job j is scheduled in any time slot in $\mathcal{A}$ and every job j has been assigned to p_j active slots within its window $[r_j, d_j)$. Once the set $\mathcal{A}$ of active slots has been determined, a feasible integral assignment can be found via a max-flow computation. We talk about this in detail in the following section in the context of the 2 approximation.

The cost of a feasible solution σ is the number of active slots in the solution, denoted by $|\mathcal{A}|$. Let $\mathcal{A}_f$ denote the set of active slots which are full, and $\mathcal{A}_n$ denote the set of active slots which are non-full. Therefore, $|\mathcal{A}| = |\mathcal{A}_f| + |\mathcal{A}_n|$.

Definition 12. *A minimal feasible solution is one in which no active slot can be made inactive, and still feasibly satisfy the entire job set.*

Given a feasible solution, one can easily find a minimal feasible solution.

Definition 13. *A non-full-rigid job is one which is scheduled one unit in every non-full slot where it is live.*

Lemma 19. *For any minimal feasible solution σ, there exists another solution σ' of same cost, where every slot that is non-full, that is, has less than g jobs, has at least one non-full-rigid job scheduled in it.*

Proof. This is a proof by construction. Consider any non-full slot in the minimal feasible solution σ, which does not have any non-full-rigid job scheduled in it. Move any job in that slot to any other (non-full, active) slot that it may be scheduled in, and where it is not already scheduled. There must at least one such slot, otherwise this would be a non-full-rigid job. Continue this process for as long as possible. Note that in moving these jobs, we are not increasing the cost of the solution, as we are only moving jobs to already active slots. If we can do this till there are no jobs scheduled in this slot, then we would have found a smaller cost solution, violating our assumption of minimal feasibility. Otherwise, there must be at least one job left in that slot, which cannot be moved to any other active slots. This can only happen if all the slots in the window of this job are either full, or inactive, or non-full where already one unit of this job has been scheduled, thus making this a non-full rigid job.

Continue this process till in all the non-full slots, there is at least one non-full-rigid job scheduled. □

Corollary 1. *There exists a set of jobs $\mathcal{J}^*$ consisting of non-full-rigid jobs, such that at least one of these jobs is scheduled in every non-full slot.*

We say that such a set $\mathcal{J}^*$ covers all the non-full slots.

Lemma 20. *There exists a set $\mathcal{J}^*$ of non-full-rigid jobs covering all the non-full slots, such that no job window is completely contained within the window of another job.*

Proof. Let us consider a set $\mathcal{J}^*$ of non-full-rigid jobs which is covering all the non-full slots. Suppose it consists of a pair of non-full-rigid jobs j and j', such that the $[r_j, d_j) \subseteq [r_{j'}, d_{j'})$. One unit of j' must be scheduled in every non-full slot in the window of j'. However, this also includes the non-full slots in the window of j, hence we can discard j from $\mathcal{J}^*$ without any loss.

We repeat this with every pair of non-full-rigid jobs in $\mathcal{J}^*$, such that the window of one is contained within the window of another, till there exists no such pair. □

Let us call such a set $\mathcal{J}^*$ of non-full-rigid jobs whose windows are not contained within each other, and which covers all the non-full slots, as a minimal set $\mathcal{J}^*$.

Now, we prove that there exists a minimal set $\mathcal{J}^*$ such that at every time slot, at most two of the jobs in the set $\mathcal{J}^*$ are live. We will be charging the cost of the non-full slots to the set $\mathcal{J}^*$. The full slots can obviously be charged to the mass bound, which is a lower bound on the optimal solution.

Lemma 21. *There exists a minimal set $\mathcal{J}^*$ of non-full-rigid jobs such that at least one of these jobs is scheduled in every non-full slot, and at every time slot, at most two of the jobs in the set $\mathcal{J}^*$ are live.*

Proof. Consider the first time slot t where 3 or more jobs of $\mathcal{J}^*$ are live. Let these jobs be numbered according to their deadline $(j_1, j_2, j_3, \dots . j_\ell,\ \ell \geq 3)$. By definition, the deadline of all of these jobs must be $\geq t$ since they are all live at t. Moreover, they are all non-full-rigid, being a part of $\mathcal{J}^*$, which means they are scheduled one unit in every non-full active slot in their window. Since the set $\mathcal{J}^*$ is minimal, no job window is contained within another, hence none of the jobs $j_2, \dots, j_\ell$ have release time earlier than that of j_1. Therefore, all non-full slots before the deadline of j_1 must be charging either j_1 or some other job with an earlier release time. Consequently, discarding any of the jobs $j_2, \ldots, j_\ell$ will not affect the charging of these slots.

Let t' be the first non-full active slot after the deadline of j_1. t' therefore needs to charge one of $j_2, j_3, \ldots, j_\ell$. Note that if there exists no such t', then all the jobs $j_2, \ldots, j_\ell$ can be discarded from the minimal set $\mathcal{J}^*$ without any loss since no non-full slot needs to charge them. Hence, let us assume that such a t' exists. Among the jobs $j_2, \ldots, j_\ell$, all the jobs which have a deadline earlier than t', can be discarded from the minimal set $\mathcal{J}^*$, without any loss, since no non-full slot needs to charge it. Hence, let us assume that all of these jobs $j_2, j_3, \ldots, j_\ell$ are live at t'. However, all of them being non-full-rigid, and t' being non-full and active, all of them must have one unit scheduled in t'. Therefore, if we discard all of the jobs $j_2, \ldots, j_{\ell-1}$ and keep j_ℓ alone, that would be enough since it can be charged all the non-full slots between t' and its deadline d_ℓ. Hence, after discarding these intermediate jobs from the minimal set $\mathcal{J}^*$, there would be only two jobs j_1 and j_ℓ left which overlap at t.

Repeat this for the next slot t'' where 3 or more jobs of the minimal set $\mathcal{J}^*$ are live, till there are no such time slots left. □

The cost of the non-full slots of the minimal feasible solution σ' is $|\mathcal{A}_n| \leq \sum_{j \in \mathcal{J}^*} p_j$.

Theorem 17. *The cost of any minimal feasible solution is at most* 3 *times that of an optimal solution.*

Proof. It follows from Lemma 21 that $\mathcal{J}^*$ can be partitioned into two job sets $\mathcal{J}_1$ and $\mathcal{J}_2$, such that the jobs in each set have windows disjoint from one another. Therefore the sum of the processing times of the jobs in each such partition is a lower bound on the cost of any optimal solution. Let us denote the cost of the optimal solution as OPT. Hence, the cost of of the non-full slots is $|\mathcal{A}_n| \leq \sum_{j \in \mathcal{J}^*} p_j \leq \sum_{j \in \mathcal{J}_1} p_j + \sum_{j' \in \mathcal{J}_2} p_{j'} \leq 2OPT$. Furthermore, the full slots charge once to OPT, since they have a mass of g scheduled in them, which is a lower bound on OPT. $|\mathcal{A}_f| \leq \frac{\sum_{j \in \mathcal{J}} p_j}{g} \leq OPT$. Therefore, in total the cost of any minimal feasible solution $cost(\sigma) = cost(\sigma') = |\mathcal{A}| = |\mathcal{A}_f| + |\mathcal{A}_n| \leq 3OPT$. This proves the theorem. □

The above bound is asymptotically tight as proved by the following example of a minimal feasible solution.

There are 2 jobs each of length g, one has a window $[0, 2g)$ and the other one has a window $[g, 3g)$. There are $g-2$ rigid jobs, each of length $g-2$, with windows: $[g+1, 2g-1)$. There are $g-2$ unit jobs with window $[g+1, 2g)$ and another $g-2$ unit jobs with window $[g, 2g-1)$. The optimal solution would have scheduled the two longest jobs from $[g, 2g)$, and one set of $g-2$ unit jobs on time slot g, and the other set of unit jobs on time slot $2g-1$. The total cost of the solution is g. However, a minimal feasible solution may schedule the two sets of $g-2$ unit jobs in the window $[g+1, 2g-1)$, with the rigid jobs of length $g-2$. Now, the two longest jobs cannot fit anywhere in the window $[g+1, 2g-1)$, since these slots have become full slots. Hence, it has to pay the cost of the g length jobs additionally. So, one feasible way to pack all the jobs would be to pack one of the longest jobs from

$[1, g+1)$ and the other one from $[2g-1, 3g-1)$. The total cost would therefore be $3g-2$, which tends to 3 times the optimal solution as $g \to \infty$.

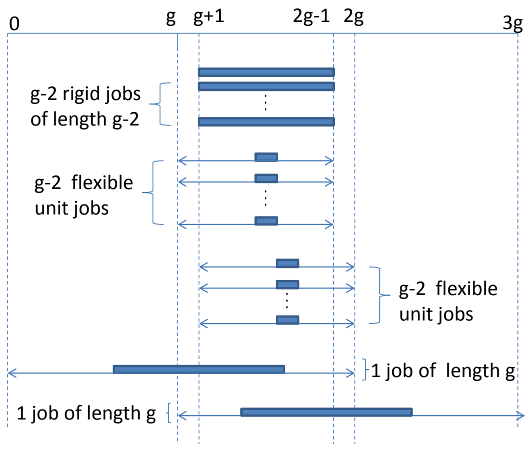


Figure 6.2: Example of an instance where the minimal feasible solution is almost 3 times the optimal solution.

6.4 A 2 approximation algorithm based on LP rounding

Here we use LP-rounding to give a 2 approximation for the active time problem on non-unit length jobs with preemption allowed at integral boundaries. In this section onwards, we will be using t to denote any slot $[t-1, t)$ for ease of notation. Let us first write an LP for the problem. Let y_t denote the indicator variable for every time slot $t \in \mathcal{T}$. Let $x_{t,j}$ denote the indicator variable for unit job $j \in \mathcal{J}$ and every $t \in \mathcal{T}$. The LP is as follows:

$$
\begin{aligned}
&\min \sum_{t \in \mathcal{T}} y_t \\
\text{s.t. } & x_{t,j} \leq y_t \ \forall t \in \mathcal{T}, j \in \mathcal{J} \\
& \sum_{j \in \mathcal{J}} x_{t,j} \leq g y_t \ \forall t \in \mathcal{T} \\
& \sum_{t \in \mathcal{T}} x_{t,j} \geq p_j \ \forall j \in \mathcal{J} \\
& 0 \leq y_t \leq 1 \ \forall t \in \mathcal{T} \\
& x_{t,j} \geq 0 \ \forall t \in \mathcal{T}, j \in \mathcal{J} \\
& x_{t,j} = 0 \ \forall t \notin [r_j, \ldots, d_j]
\end{aligned}
$$

We first solve the LP to optimality. Since any integral optimal solution is a feasible LP solution, the optimal LP solution is a lower bound on the cost of any optimal solution. Our goal is to round it to get a feasible integral solution within twice the cost of the optimal LP solution. However, before we do the rounding, we pre-process the optimal LP solution, to get a certain structure without increasing the cost of the solution. We show that there exists a feasible fractional assignments of the jobs to this pre-processed LP solution and outline the procedure to find one. Then we round this solution to get a feasible integral solution.

In the rounding, our goal will be to find a set of slots to open integrally, such that there exists a feasible fractional assignment for the jobs in the integrally open slots. An integral assignment can be found at the end of the procedure, when we have determined the integrally open slots, via a max-flow computation. We do not try to integrally assign jobs at any intermediate steps. In the max-flow construction,

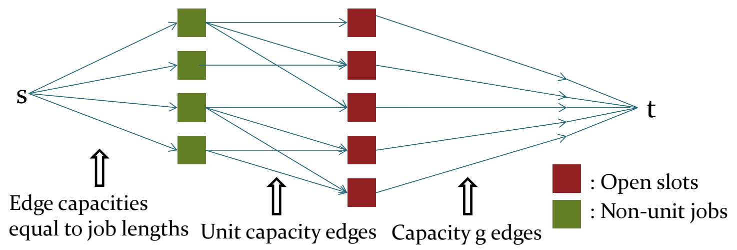


Figure 6.3: Here we show the flow network construction for finding an integral, feasible assignment of jobs in integrally open slots.

we create a node for each job and a node for each integrally open slot. We add edges from the jobs to the integrally open time slots at which they are feasible. These edges have unit capacity, since only one unit of a job can be done on any time slot. We create a source node s, and add edges from s to the jobs, such that an edge from s to j will have capacity p_j. We also create a sink node t and edges from the integrally open slots to t, each of capacity g. Since there exists a feasible fractional assignment of the jobs in the slots, there exists a flow of $\sum_j p_j$ in the network. Hence, any max-flow algorithm will find a flow of this value, and since all capacities and sizes are integral, the flow will be integral, and hence we can get an integral, feasible assignment of the jobs in the integrally open slots. See Figure 6.3 for an example.

6.4.1 Pre-Processing

We sort the deadlines of the jobs to get the set of distinct deadlines in increasing order and then process the LP solution sequentially according to these deadlines.

Let the set of distinct deadlines be $\mathcal{D} = [d_1, d_2, \ldots, d_\ell]$. We denote the set of jobs with deadline d_i as $\mathcal{J}_i$.

We define Y_1 as the sum of the y values for all time slots $\leq d_1$, that is, $Y_1 = \sum_{t \leq d_1} y_t$. Then we modify the optimal LP solution as follows. We open the slots $[d_1 - \lfloor Y_1 \rfloor + 1, \ldots, d_1]$ integrally and the slot $d_1 - \lfloor Y_1 \rfloor$ fractionally up to $Y_1 - \lfloor Y_1 \rfloor$, and close all the earlier slots.

Similarly, for the i^{th} deadline, define $Y_i = \sum_{d_{i-1} < t \leq d_i} y_t$. We want to preprocess the optimal solution to have the right-shifted structure, where for every deadline d_i, $\lfloor Y_i \rfloor$ slots are open integrally backwards from d_i, in other words, the slots $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ are fully open, and the slot $d_i - \lfloor Y_i \rfloor$ is open up to $Y_i - \lfloor Y_i \rfloor$.

By definition, the LP solution can be written as: $\sum_{t \in \mathcal{T}} y_t = \sum_{i \in [1, \ldots, \ell]} Y_i$.

For every job $j \in \mathcal{J}$, let us define $a_{j,1} = \sum_{t \leq d_1} x_{t,j}$, where $a_{j,1}$ the total assignment any job j is getting from slots $\leq d_1$ in the optimal LP solution. Similarly, for any deadline d_i, $i > 1$, we define $a_{j,i} = \sum_{d_{i-1} < t \leq d_i} x_{t,j}$, where $a_{j,i}$ is the total assignment any job j is getting from the slots $[d_{i-1} + 1, \ldots, d_i]$.

The following lemma proves that there exists an optimal feasible LP solution in the modified instance up to d_1.

Lemma 22. *There exists an assignment of the jobs in the modified LP solution up to d_1 such that every job j in $\mathcal{J}$ can be accommodated up to $a_{j,1}$.*

Proof. This is a proof by construction. Let t be the latest slot $\leq d_1$ for which $y_t > 0$ in the optimal LP solution. Note that, without loss of generality, we can make $y_{d_1} = y_t$ and $y_t = 0$, and move all the job assignments intact from t to d_1. This is because, all jobs which are feasible at t are also feasible at d_1 by definition of d_1. Now, let t' be the latest slot $\leq d_1$ for which $y_{t'} > 0$. If $y_{d_1} + y_{t'} \leq 1$, we merge $y_{t'}$ with y_{d_1}, in other words, set $y_{d_1} = y_{d_1} + y_{t'}$ and transfer all the job assignments from t' to d_1, without violating feasibility or increasing the cost. Otherwise, we set $y_{d_1 - 1} = y_{d_1} + y_{t'} - 1$ and $y_{d_1} = 1$. However, we need to maintain the constraint $x_{t,j} \leq y_t$ to get a feasible LP solution. Suppose $J_{t'}$ is the set of jobs that were originally assigned to the t'. Set $\delta = g \cdot y_{d_1 - 1}$ initially. As long as there exists some job in $j \in J_{t'}$, we assign j up to $x_{d_1-1,j} = \min(x_{t',j}, y_{d_1-1}, \delta)$ to the slot $d_1 - 1$, and decrement both δ and $x_{t',j}$ by $x_{d_1-1,j}$. If $x_{t',j}$ becomes 0, then we remove that job from $J_{t'}$. Once δ goes to 0 on adding some job to slot $d_1 - 1$, we assign all the remaining jobs (portions) in $J_{t'}$ to d_1, without violating feasibility, since 1) $y_{d_1} = 1$, 2) we have assigned up to $g \cdot y_{d_1-1}$ in $d_1 - 1$, there is space in d_1 up to g, and we know $y_{d_1} + y_{t'} = 1 + y_{d_1-1}$, hence there is enough space in d_1 to accommodate the remaining jobs. If δ is not 0, but $J_{t'}$ is not empty, then every job $j \in J_{t'}$ have been assigned either up to $x_{t',j}$ to $d_1 - 1$, or up to y_{d_1-1}, and the remaining jobs (portions) are assigned to d_1. This is also feasible because of the following reasons. The space available in d_1 before merging with t' was $g \cdot y_{d_1}$. Let the space occupied by jobs with $x_{t',j} \leq y_{t'}$ be δ' in $d_1 - 1$. The jobs with $x_{t',j} > y_{d_1-1}$ must have all been assigned up to y_{d_1-1}, since δ is still not 0. Therefore, the number of such jobs is $< \frac{g \cdot y_{d_1-1} - \delta'}{y_{d_1-1}}$. These jobs could have been assigned to at most $y_{t'}$ in t' before merging. Hence, the remaining portion of these jobs is $< \frac{g \cdot y_{d_1-1} - \delta'}{y_{d_1-1}} \cdot (y_{t'} - y_{d_1-1})$. The space available for accommodating them in d_1 is $(y_{t'} - y_{d_1-1}) \cdot g$. However,

$\frac{g \cdot y_{d_1-1} - \delta'}{y_{d_1-1}} \cdot (y_{t'} - y_{d_1-1}) \leq (y_{t'} - y_{d_1-1}) \cdot g$, therefore, the remaining portions of the jobs in $J_{t'}$ can be accommodated in d_1, without violating LP feasibility. If δ does not go to 0, while there are no jobs left in $J_{t'}$, then we have feasibly assigned all the jobs in $y_{t'}$, respecting all the constraints.

Now, we repeat the above procedure with the latest slot $t'' < d_1 - 1$, merging $y_{t''}$ with y_{d_1-1}, if $y_{d_1-1} > 0$, otherwise with y_{d_1}. If there exists no such t'' we stop. At the end, we have a feasible LP solution where all jobs $j \in \mathcal{J}$ have been assigned up to $a_{j,1}$ in the right shifted structure. $\square$

Let us assume by induction hypothesis that the above holds for all deadlines d_k, where $k \leq i-1$. The next lemma proves that the property holds for the deadline d_i, assuming it holds for all earlier deadlines.

Lemma 23. *There exists an assignment of the jobs in the modified LP solution up to d_i ($i \geq 1$) such that every job j in $\mathcal{J}$ can be accommodated up to $\sum_{1 \leq k \leq i} a_{j,i}$.*

Proof. We prove this by induction. The base case is proved in Lemma 22. We assume by induction hypothesis, that the claim holds for all iterations $1 \leq k < i$ and now do the same construction as in Lemma 22 for proving the claim for iteration i. Now, we only consider slots $d_{i-1} + 1, \ldots d_i$, starting with the closest slot t to d_i with $y_t > 0$ and repeat the merging and reassignment procedure as described in Lemma 22. Again the key observation is that any job j which is feasible at some $t \leq d_i$, is also feasible in all slots $[t, t+1, \ldots, d_i]$. $\square$

Theorem 18. *There exists a feasible LP solution of the same cost as any optimal LP solution, which possesses the right-shifted structure.*

Proof. As already noted, the LP solution can be written as the sum of the Y_i values, as $\sum_{t \in \mathcal{T}} y_t = \sum_{i \in [1,\ldots,\ell]} Y_i$. Therefore, the right shifted solution has the same cost as the optimal solution. The theorem follows from the Lemmas 22 and 23 by induction. $\square$

We can use the construction process highlighted in the Lemmas 22 and 23 or we can set the y values of all the slots as per the right-shifted structure, and solve the following feasibility LP to get a feasible fractional assignment for all the jobs.

$$x_{t,j} \leq y_t \ \forall t \in \mathcal{T}, j \in \mathcal{J}$$

$$\sum_{j \in \mathcal{J}} x_{t,j} \leq g y_t \ \forall t \in \mathcal{T}$$

$$\sum_{t \in \mathcal{T}} x_{t,j} \geq p_j \ \forall j \in \mathcal{J}$$

$$x_{t,j} \geq 0 \ \forall t \in \mathcal{T}, j \in \mathcal{J}$$

$$x_{t,j} = 0 \ \forall t \notin [r_j, \ldots, d_j]$$

Henceforth, we work with this feasible, right-shifted optimal LP solution. See Figure 6.4 for an example of a right-shifted LP solution.

6.4.2 Overview of Rounding

We process the deadlines one after another in at most ℓ iterations, where we process the d_i in iteration i. At the end of every iteration i, we have a set of integrally open slots $\mathcal{O}_i$. We maintain the invariant that at the end of the i^{th} iteration, the number of integrally open slots up to d_i is $|\mathcal{O}_i| \leq 2 \sum_{j \leq i} Y_j$, and there exists a

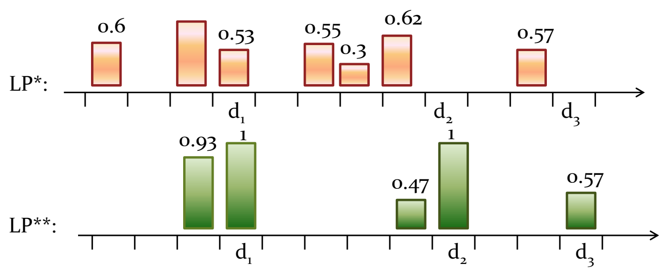


Figure 6.4: LP* is an optimal LP solution, and LP** is the rightshifted solution of the same cost.

feasible fractional assignment of $\mathcal{J}_i$ in $\mathcal{O}_i$. This gives us the 2 approximation by the end of the ℓ^{th} iteration, when we have processed the last deadline.

We refer to a slot t with $y_t = 1$ as "fully open", that with $1 > y_t \geq \frac{1}{2}$ as "half-open", that with $0 < y_t < \frac{1}{2}$ as "barely open" and that with $y_t = 0$ as "closed".

Obviously, fully open slots do not charge anything extra to the LP solution. Half-open slots can be opened at a cost of at most 2, charging themselves. For opening a barely open slot, we need to charge it to an fully open slot. We say that a barely open slot is "dependent" on the fully open slot that it charges to. In this case, the y value of the barely open slot is not charged at all. We will sometimes allow two barely open slots on either side of a fully open slot to open up along with a fully open slot, when the total sum of the y values of the barely open slot and the fully open slot is $\geq \frac{3}{2}$. We will refer to such slots as a "trio". Note that here we *do* charge the y value of the barely open slot.

We will additionally maintain the invariant that at every iteration, every barely open slot that we have opened is either a dependent on a fully open slot, or is part of a trio, and every fully open slot has at most one dependent or it is part of at most one trio. Half open slots charge themselves. This will ensure that we have charged the LP solution at most twice.

Every time we open a barely open slot as a dependent, we make it a dependent on the earliest fully open slot that does not have a dependent or is not part of a trio. See Figure 6.5 for an example.

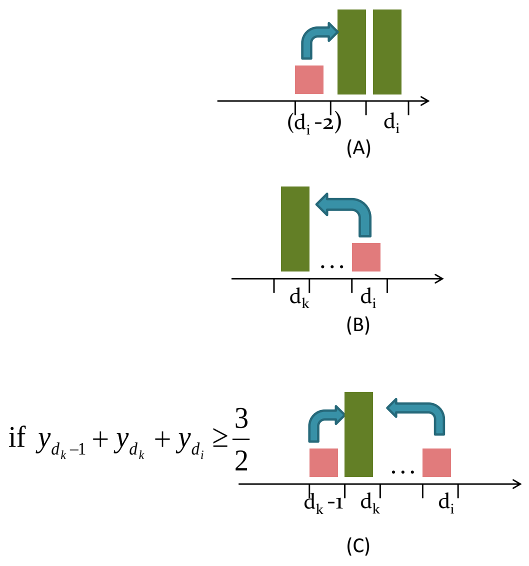


Figure 6.5: Here we show the three possible ways in which we charge a barely open slot when max-flow cannot close it.

Sometimes, in the rounding process we close a barely open slot $d_i - t$, $(t \geq 0)$, while processing deadline d_i. However, if jobs of later deadline were assigned by the LP in this barely open slot, then we need to accommodate them. We make sure that when we close a slot, we are not charging its y value at all. Hence we create a proxy copy of the slot that we closed, and carry it over to the next iteration. The y value of this proxy slot is set to be the y value of the slot we have just closed, without any double counting. The proxy also carries a pointer to the actual slot that it is a proxy for. In any iteration i, when we have a proxy, we treat it as a regular fractionally open slot (though there may be no actual slot at that point). If this slot remains closed after the rounding, the proxy gets carried over to the next iteration, whereas if it does get opened by the rounding, the actual slot which it points to gets opened. However, now the cost of opening it will be accounted for by the current solution. This is outline in details in Section 6.4.4.1. There can be at most one proxy slot at any iteration.

6.4.3 Processing d_1

Lemma 24. $Y_1 \geq 1$

Proof. This is obvious as otherwise the LP solution is not feasible: none of the chains with deadline d_1 or otherwise, could have been assigned one unit before the first deadline. □

Slots $[d_1 - \lfloor Y_1 \rfloor + 1, \ldots, d_1]$ are fully open. In the following, we outline how we deal with the slot $d_1 - \lfloor Y_1 \rfloor$.

Case 6. $Y_1 - \lfloor Y_1 \rfloor \geq \frac{1}{2}$.

We open $d_1 - \lfloor Y_1 \rfloor$ as half open and charge it to itself.

Case 7. $Y_1 - \lfloor Y_1 \rfloor < \frac{1}{2}$.

We first try to close $d_1 - \lfloor Y_1 \rfloor$ and find a feasible assignment of $\mathcal{J}_1$ in $\lfloor Y_1 \rfloor$ slots fully opened, using max-flow. If successful, then we keep it closed, and move to the next deadline, after passing over a proxy slot with y value $Y_1 - \lfloor Y_1 \rfloor$ to iteration 2. The proxy slot has the same job assignments as in the right-shifted solution LP solution, except those in $\mathcal{J}_1$, which are removed from the proxy slot. If we do not find a feasible assignment for $\mathcal{J}_1$ by closing $d_1 - \lfloor Y_1 \rfloor$, we keep it barely opened and make it dependent on $d_1 - \lfloor Y_1 \rfloor + 1$. We are guaranteed to find a fully open slot on which to make it dependent since $Y_1 > 1$.

6.4.4 Processing deadline d_i, $i > 1$

Now, we proceed to the next deadline d_i.

6.4.4.1 Dealing with a proxy slot

While processing a deadline d_i, suppose there is a proxy of value y_p carried over from iteration $(i - 1)$. Note that we are working with a right shifted solution. and the slots $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ are fully open. Before we do any rounding in the iteration i, we do the following.

Case 8. $y_p + Y_i - \lfloor Y_i \rfloor \leq 1$.

In this case, we merge the proxy with the slot $d_i - \lfloor Y_i \rfloor$. Specifically, we add y_p to $y_{d_i - \lfloor Y_i \rfloor}$, and transfer all the job assignments from the proxy slot to the slot $(d_i - \lfloor Y_i \rfloor)$ without violating the LP solution feasibility. (This maintains all the LP constraints). Now, we proceed with the rounding, treating $d_i - \lfloor Y_i \rfloor$ as a regular fractional slot in the right shifted solution (in other words, consider Y_i to be $Y_i + y_p$). Note that the jobs originally assigned to the proxy must have deadlines $\geq d_i$, since they were passed over from iteration $i-1$. Hence, the pointer to the proxy can now be safely changed to $d_i - \lfloor Y_i \rfloor$. If this gets opened, then no proxy is carried over. Otherwise, the new proxy carried over will be of value $y_p + Y_i - \lfloor Y_i \rfloor$ and the pointer will be to $d_i - \lfloor Y_i \rfloor$. If however, $d_{i-1} = d_i - \lfloor Y_i \rfloor$, then $\lfloor Y_i \rfloor = Y_i$, and in this case, we consider an imaginary slot $d_{i-1} = d_i - \lfloor Y_i \rfloor$ in between d_{i-1} and $d_i - \lfloor Y_i \rfloor + 1$, to which we assign y_p, consider Y_i to be $Y_i + y_p$ and process it for the time being as a regular fractional slot. If this remains closed, we pass over a proxy of value y_p with the original pointer, otherwise, we open up the actual slot to which the proxy points.

Case 9. $y_p + Y_i - \lfloor Y_i \rfloor > 1$.

In this case, let $y_p{}' = y_p + Y_i - \lfloor Y_i \rfloor - 1$. We make $d_i - \lfloor Y_i \rfloor$ fully open and create a new proxy with y value equal to $y_p{}'$. If $d_{i-1} < d_i - \lfloor Y_i \rfloor - 1$, we set y of the slot $d_i - \lfloor Y_i \rfloor - 1$ to $y_p{}'$, since all jobs in the proxy are feasible here, and treat it as a regular fractional slot. In other words, assume Y_i to be $Y_i + y_p$, and process it. If this fractional slot gets opened, then we are fine, otherwise, this new proxy of value $y_p{}'$ gets carried over to iteration $i+1$, with the pointer changed to $d_i - \lfloor Y_i \rfloor - 1$. Otherwise, $d_{i-1} = d_i - \lfloor Y_i \rfloor - 1$; in this case, consider an imaginary slot $d_{i-1} = d_i - \lfloor Y_i \rfloor - 1$ in between d_{i-1} and $d_i - \lfloor Y_i \rfloor$, to which we assign $y_p{}'$, consider Y_i to be $Y_i + y_p$ and process it for the time being as a regular fractional slot. If this imaginably slot remains closed, we pass over a proxy slot of value $y_p{}'$ with the original pointer, otherwise, we open up the actual slot to which the proxy points. For the case $y_p + Y_i - \lfloor Y_i \rfloor > 1$, since we create a new proxy $y_p{}'$, we need to specify how to change the assignments of the actual jobs, in order to create a new LP feasible solution. The procedure is essentially the same as in Lemma 22. For completeness, it is outlined below.

We can assign a mass up to $g{y_p}'$ in the new proxy slot. Let $\delta = g \cdot {y_p}'$ initially. As long as there exists some job in $j \in J_p$, we assign j up to a $x_{p',j} = \min(x_{p,j}, {y_p}', \delta)$, where $x_{p,j}$ was the original assignment of j to y_p. We decrement both δ and $x_{p,j}$ by $x_{p',j}$. If $x_{p,j}$ becomes 0, then we remove that job from J_p. Once δ goes to 0 on adding some job to the proxy slot we assign all the remaining jobs (portions) in J_p to $d_i - \lfloor Y_i \rfloor$, without violating any feasibility. The slot $d_i - \lfloor Y_i \rfloor$ is now fully open, hence constraint $x_{t,j} \leq y_t$ won't be violated. Moreover, ${y_p}'$ being fully utilized, the remaining mass to be accommodated must be $\leq (y_p - {y_p}') \cdot g$, hence there is enough space in $d_i - \lfloor Y_i \rfloor$ to accommodate the remaining jobs. If δ is not 0, but J_p is not empty, then every job $j \in J_p$ have been assigned either up to $x_{p,j}$ to the proxy slot, or up to ${y_p}'$, and the remaining jobs (portions) are assigned to $d_i - \lfloor Y_i \rfloor$. This is feasible because of the following reasons. The space available in $d_i - \lfloor Y_i \rfloor - 1$ originally was $g{y_p}'$. Let the space occupied by jobs with $x_{p,j} \leq {y_p}'$ be δ'. The jobs with $x_{p,j} > {y_p}'$ must have all been assigned up to ${y_p}'$, since δ is still not 0. Therefore, the number of such jobs is $< \frac{g{y_p}' - \delta'}{{y_p}'}$. These jobs could have been assigned to at most y_p in the proxy slot originally. Hence, the remaining portion of these jobs is $< \frac{g{y_p}' - \delta'}{{y_p}'} \cdot (y_p - {y_p}')$. The space available for accommodating them in $d_i - \lfloor Y_i \rfloor$ is $(y_p - {y_p}') \cdot g$. However, $\frac{g{y_p}' - \delta'}{{y_p}'} \cdot (y_p - {y_p}') \leq (y_p - {y_p}') \cdot g$, therefore, the remaining portions of the jobs in J_p can be accommodated in $d_i - \lfloor Y_i \rfloor$, without violating LP feasibility. If δ does not go to 0, while there are no jobs left in J_p, then we have feasibly assigned all the jobs in y_p, respecting all the constraints.

Clearly by the above procedure, there can be at most one proxy in an iteration. Note that when we pass over a proxy from an iteration i, all job assignments of jobs in $\bigcup_{1 \leq k \leq i} \mathcal{J}_k$ are removed from the proxy (since they have already been accounted for), without violating any feasibility.

6.4.4.2 Processing Y_i

In the following discussion, we assume Y_i already takes into account any proxy from iteration $i-1$ and the processing outlined above for a proxy slot has already been done.

Case 10. $1 > Y_i \geq \frac{1}{2}$.

We open d_i and charge it to itself as half-open.

Case 11. $Y_i > 1$ *and* $Y_i - \lfloor Y_i \rfloor \geq \frac{1}{2}$.

We open $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ as fully open and $d_i - \lfloor Y_i \rfloor$ as half open and charge it to itself.

Case 12. $Y_i > 1$ *and* $Y_i - \lfloor Y_i \rfloor < \frac{1}{2}$.

We open $[d_i - \lfloor Y_i \rfloor + 1, \ldots, d_i]$ as fully open. $d_i - \lfloor Y_i \rfloor$ is barely open. We first close $d_i - \lfloor Y_i \rfloor$, and try to find a feasible assignment of all jobs in $\bigcup_{j \leq i} \mathcal{J}_j$, using max-flow. If successful, then we keep it closed and move on to the next deadline. If d_i is not the last deadline, we pass over a proxy for $d_i - \lfloor Y_i \rfloor$ to iteration $i+1$. This proxy has the y value of $Y_i - \lfloor Y_i \rfloor$, and all the job assignments from $(d_i - \lfloor Y_i \rfloor)$

except those in $\mathcal{J}_i$. Moreover, we have already adjusted for any proxy from iteration $i-1$, as outlined in the previous section.

Otherwise, we need to open $d_i - \lfloor Y_i \rfloor$, and have to account for its cost. We charge $d_i - \lfloor Y_i \rfloor$ as a dependent on the earliest fully open slot that has no dependents. If all the fully open slots $< d_i - \lfloor Y_i \rfloor$ have dependents, we charge it to $d_i - \lfloor Y_i \rfloor + 1$, which is fully open since $Y_i > 1$.

Case 13. $Y_i < \frac{1}{2}$.

We will first try to close d_i. Suppose the closest deadline open before d_i is d_k. Since d_i is barely open, all jobs in $\mathcal{J}_i$ must have release time $\leq d_k$, for a feasible LP solution. We try to find a feasible assignment of all jobs in $\bigcup_{j \leq i} \mathcal{J}_j$, using max-flow in the open slots earlier than d_i that are already accounted for. If successful, then we keep d_i closed and move on to the next deadline. If d_i is not the last deadline, we create a proxy for d_i which we pass over to iteration $(i+1)$, assign it y value of Y_i, transfer all the job assignments from d_i to the proxy, except those of $\mathcal{J}_i$. If there was a proxy already in iteration i, carried over from the previous iteration, then note that this new proxy value already adjusts for the previous proxy value as per the rounding outlined in Section 6.4.4.1 and since all jobs in $\bigcup_{j \leq i} \mathcal{J}_j$ are already accounted for, without loss of generality we change the pointer of the proxy to d_i.

Suppose closing d_i and finding a feasible assignment of jobs in $\bigcup_{j \leq i} \mathcal{J}_j$ was not successful. Let the earliest open deadline before d_i be d_k. If d_k was barely open, then flow would have certainly been successful, because all jobs feasible at d_i must be feasible at d_k, and both d_k and d_i being barely open with all intermediate deadlines closed in a feasible LP solution, there is space to accommodate all the jobs of $\bigcup_{k\leq x\leq i}\mathcal{J}_x$. Hence, d_k must be half open or fully open. If half-open, we first try to merge d_k and d_i. If $y_{d_k}+y_{d_i}\leq 1$, then flow would have found a feasible assignment. Therefore, $y_{d_k}+y_{d_i}>1$. In this case, we make d_k fully open (now barely open slots can be made dependent on it) and set the new $Y_i=y_{d_k}+y_{d_i}-1$. The job assignments are altered to make the resultant solution LP feasible in the same manner as outlined earlier for modifying assignments for proxy slot in Section 6.4.4.1. The new Y_i, which is still barely open, is then processed similar to the previous Y_i until we have either closed d_i locally, or opened it as part of a trio or as a dependent on a fully open slot. Once we have done either of that, we can move to d_{i+1}.

The final possibility is that d_k is fully open. We then try to charge d_i as a dependent on the earliest fully open slot that has no dependents. If successful, we move on to d_{i+1}. Otherwise, all of the fully open slots up to time slot d_k have dependents. We then try to charge d_i as a trio with the fully open d_k and its dependent. If we are successful, we move on to iteration $(i+1)$.

We will next argue that we will always be able to charge a barely open slot d_i that the rounding needs to open.

Lemma 25. *If we need to open a barely open slot d_i in an iteration, then we will always find a fully open slot to charge it as a dependent or as a trio.*

In order to prove the above, let us first assume that it is not true, and we are in the situation where we could not close d_i which is barely open, the closest open slot to d_i is fully open, and we could not charge d_i as a trio or as a dependent on any of the fully open earlier slots. That necessarily means all the fully open slots before d_i must have dependents.

The following lemma argues that in case all the fully open slots before d_i have dependents, then the structure of the solution must be of the following form. There must be a deadline d_z, with the closest open slot $< d_z$ being d_w (there may be no d_w if d_z is the first fully open deadline), such that either 1) there are $\geq g+1$ jobs in $\mathcal{J}_z$ with release time later than d_w; or, 2) the sum of the number of rigid (unit) jobs with release time and d_z and the length 2 jobs with release time $\geq d_w$ is $\geq g+1$. Let us call such a d_z a *stopping deadline.* In other words, any integral solution would have to open at least one slot in $[d_w+1, \ldots, d_z-1]$, along with d_z, since there are $g+1$ job units to be scheduled between d_w+1 and d_z. Between stopping deadline d_z and d_i, the fully open slots will be a subset of the set of deadlines, and for each deadline d_x that is fully open, the slot $d_x - 1$ is barely open and dependent on d_x. (Note that $d_x - 1$ can be another deadline itself). Let us call this structure an *alternating* structure.

Lemma 26. *If the closest open slot to deadline d_i is fully open, and all the fully open slots before d_i have dependents, then without loss of generality, there is a stopping deadline d_z and between d_z and d_i, the structure of the solution must be alternating.*

Proof. Let d_k be the closest open slot to d_i, and d_k is fully open. Given the structure of the solution that we start out with and the rounding process, this must be a deadline. All the fully open slots before d_i have dependents. Since no slot between

d_k and d_i are open, the dependent on d_k has to be some earlier slot. Either it is a barely open slot t such that $d_{k-1}+1 \leq t \leq d_k - 1$. Or it must be a proxy slot carried over from an earlier deadline than d_k. Only one barely open slot is opened in any iteration by the rounding process. Therefore, if we open a proxy slot in an iteration, then there can be no other local barely open slots open in this iteration. Any barely open slot is dependent on the earliest fully open slot without a dependent. If $Y_k \geq 2$, d_k would not have got charged in iteration k, and hence would have no dependents even at iteration i. Therefore, $Y_k < 2$ and only d_k is fully open in the slots after d_{k-1}.

If d_k is not charged by a proxy slot, then necessarily $d_k - 1$ is barely open. However, even if d_k is charged by a proxy slot, we show that any such proxy slot can be considered to be a local barely open slot without any loss of generality. Let us suppose that the dependent on d_k is a proxy slot. In that case, the alternating structure may not hold when we open the actual slot for the proxy. That means, the actual slot must be occurring in at some t', $d_{j-1} \leq t' \leq d_j$, where $j < k$. No barely open or half open slots could have opened between j and k as otherwise it would have accounted for the proxy slot. If there is a fully open slot between d_j (inclusive of d_j) and d_k then the proxy can charge this slot as it must have been uncharged so far. Since the proxy is a dependent on d_k, d_k must be the first fully open slot from iteration j onwards. Moreover, all the jobs in $\bigcup_{x<k} J_x$ do not need the proxy value for a feasible assignment. Hence, we can change the pointer of the proxy slot to $d_k - 1$ without any loss of generality and consider $d_k - 1$ as dependent on d_k. Note that $d_k - 1$ may also be equal to d_j. Therefore, even if d_k is charged by

proxy slot, we can convert it to a local barely open slot $d_k - 1$.

Now, consider the rounding process in the iteration k. We must have first tried to close $d_k - 1$ and find a feasible assignment using max flow. Clearly that must have failed. Also, no job in $\mathcal{J}$ with release time $> d_{k-1}$ can be of length > 1, because $Y_k < 2$. Therefore, one reason can be that there are $\geq g + 1$ unit jobs in $\mathcal{J}_k$ with release time $\geq d_{k-1}$. In that case, d_k is the stopping deadline, and we have the alternating structure trivially.

If that is not the case, then that necessarily means the closest earlier open slot, say d_p,was half-open or fully open. The closest open slot cannot be barely open in a feasible LP solution, otherwise max flow would have been able to find a feasible assignment of the jobs in $\bigcup_{p \leq x \leq k} \mathcal{J}_k$ even after closing the barely open slot $d_k - 1$. If half-open, then clearly, $y_{d_p} + y_{d_k - 1} > 1$, otherwise, an assignment could be found by max-flow. However, in this case, the rounding would have made d_p fully open, and charged the new $y_{d_k-1} = y_{d_p} + y_{d_k-1} - 1$ as a dependent to it, if no other fully open slots were available for charging. Therefore, the only possibility is that d_p is fully open, and has a dependent already. Then the same argument can be be repeated for d_p and $d_p - 1$. We repeat this argument for next closest open slot (which must be fully open with a dependent) till we come to a stopping deadline. We are guaranteed to find a stopping deadline because, if we ultimately come d_1, then that must also have a dependent $d_1 - 1$, (which means no jobs in $\mathcal{J}_1$ can be of length > 1) and we know from our rounding rule for d_1, that $d_1 - 1$ is opened only when max flow failed, which implies there are $\geq g + 1$ unit jobs in $\mathcal{J}_1$. Hence, without loss we can convert our LP solution to the alternating form between d_i and the stopping deadline d_z.

$\square$

Lemma 27. *Suppose d_z is the latest stopping deadline, in the alternating structure going backwards from d_i. Then for every intermediate fully open deadline $d_x \notin d_z, d_i$, at least $\geq 2g + 1$ job units in $\mathcal{J}_u \cup \mathcal{J}_x$ must have release time $\geq d_u$, where d_u is the latest open deadline before d_x.*

Proof. We shall prove this by induction. Let the closest open slot before d_z be d_w. There are $\geq g+1$ job units in d_z which need to be scheduled in slots $[d_w + 1, \ldots, d_z]$ due to release time constraints. This follows from the definition of a stopping deadline. Let the next fully open deadline after d_z in the alternating structure be d_a ($d_a > d_z$). Note that the total mass scheduled by the LP in $[d_z - 1, d_z, d_a - 1, d_a]$ is $\leq \frac{5g}{2}$, by the definition of the alternating structure. (The barely open slots could not form trio with each other). We want to prove that there are $2g + 1$ job units in $\mathcal{J}_z \cup \mathcal{J}_a$ with release time $\geq d_z$. Let us assume there are $\leq 2g$ units of jobs in $\mathcal{J}_z \cup \mathcal{J}_a$ with release time $\geq d_z$ for contradiction. No job in $\mathcal{J}_a$ can be ≥ 2 in length for a feasible LP solution since $Y_a < 2$. Let n_z denote the rigid jobs in $\mathcal{J}_x$ (those releasing at d_z), n'_z denote the flexible jobs in $\mathcal{J}_z$ which need to be assigned before d_w (if there is any d_w), $n_{a,2}$ denote the number of length 2 jobs in $\mathcal{J}_a$, $n_{a,1}$ denote the unit length jobs in $\mathcal{J}_a$ with release time d_a and $n'_{a,1}$ denote the unit length jobs in $\mathcal{J}_a$ with release time $\geq d_z$. We know that $n_z + 2n_{a,2} + n_{a,1} + n'_{a,1} \leq 2g$ by assumption, $n_z + n'_z \geq g + 1$ by definition of d_z, and since d_z is the latest stopping deadline, $n_{a,1} + n_{a,2} \leq g$. Since max flow failed, the only possibility is that $n_z + n_{a,2} \geq g+1$. For LP feasibility, $n_z + \frac{n'_z}{2} + \frac{n_{a,2}}{2} \leq g$. However, $n_z + n'_z + n_z + n_{a,2} > 2g$. Hence we get a contradiction. Therefore, there must be $\geq 2g+1$ job units $\mathcal{J}_z \cup \mathcal{J}_a$ with release time $\geq d_z$.

For ease of notation, without loss of generality, assume that the deadlines are consecutive. Therefore, all the deadlines $[d_z, d_{z+1}, \ldots, d_k]$ are fully open (here, $d_a = d_{z+1}$). Now, assume by induction hypothesis, that the claim is true for all deadlines up to d_k in the alternating structure, and the next fully open slot is d_{k+1}. For any deadline d_p which is fully open in the alternating structure between d_z and d_{k+1}, let us denote by $n_{p,1}$ the unit length rigid jobs in $\mathcal{J}_p$, $n_{p,2}$ the length 2 jobs of release time $\geq d_{p-1}$, and $n'_{p,1}$ the unit length jobs of release time $\geq d_{p-1}$ in $\mathcal{J}_p$. By induction hypothesis, for any two adjacent open deadlines d_{p-1} and d_p, where $p \geq 2$, in the alternating structure, there are $\geq 2g+1$ job units in $\mathcal{J}_{p-1} \cup \mathcal{J}_p$ with release time $\geq d_{p-1}$, i.e., $n_{(p-1),1} + n_{p,1} + n'_{p,1} + 2n_{p,2} \geq 2g+1$. As in the base case, assume for contradiction, that there are $\leq 2g$ job units in $\mathcal{J}_k \cup \mathcal{J}_{k+1}$ with release time $\geq d_k$. Therefore, $n_{k,1} + n_{(k+1),1} + n'_{(k+1),1} + 2n_{(k+1),2} \leq 2g$. Since the latest stopping deadline $d_z < d_{k+1}$, it also holds that $n_{(k+1),1} + n_{(k+1),2} \leq g$.

Claim 6. *If there are $\leq 2g$ job units in $\mathcal{J}_k \cup \mathcal{J}_{k+1}$ with release time $\geq d_k$, and d_{k+1} is not a stopping deadline, as well as $n_{k,1} + n_{k+1,2} \leq g$, then max-flow will find a feasible assignment for all jobs in $\bigcup_{1 \leq x \leq k+1} \mathcal{J}_x$ in the set of slots opened integrally up to d_k and including d_{k+1}.*

Proof. Suppose this is not true. We first assign all the jobs in $\mathcal{J}_k \cup \mathcal{J}_{k+1}$ with release times $\geq d_k$. Now we try to find a max-flow in the accounted for integrally open slots, with the capacities in d_k and d_{k+1} adjusted after the above assignment. We define a chain of jobs as follows: a chain of jobs $j_1, j_2, \ldots, j_q$ denotes a set of jobs scheduled by max-flow in full slots, such that j_1 is feasible in a full slot that j_2 is scheduled in, j_2 is feasible in another full slot j_3 is scheduled in and so on, till j_{q-1} is feasible in another full slot that j_q is scheduled in and j_q is feasible in a non-full slot.(A full slot has g jobs assigned, and a non-full slot has $< g$ jobs.) By our assumption therefore, there exists at least one job j with $r_j \leq d_k - 1$, such that one unit of j cannot be scheduled anywhere, due to our assignment of the jobs in $\mathcal{J}_{k+1}$ in d_k. Without loss of generality, this job has deadline d_k and of course by definition, $r_j < d_k$ (we have already assigned rigid jobs with deadline d_k).

Therefore, one unit of j must be scheduled in d_k. In other words, in all the (accounted for) integrally open slots in $[r_j, \ldots d_k - 1]$, j must be already scheduled one unit in all the non-full slots, and there is no possible chain in the full-slots. Let the number of full slots in this window be c_f. Therefore, the number of (accounted for) integrally open slot given to max-flow in this window must be $c_f + p_j - 1$. However, in a feasible LP solution, at most one unit of the job could have come from d_k. Hence, $p_j - 1$ units must come from the remaining slots in $[r_j, \ldots, d_k - 1]$. Since there is no possible chain with the jobs in the full slots in this window, therefore, LP must have also scheduled them in the full slots, and hence j could not get any assignment from these slots in the LP as well. Therefore, in $[r_j, \ldots, d_k - 1]$, the sum of y values in the LP must be $\geq p_j - 1 + c_f$. However, given the alternating structure, it must be true that at most $\frac{p_j - 1 + c_f}{2} + 1$ slots in $[r_j, \ldots, d_k - 1]$ can be fully open and the rest are barely open, such that no two barely open slots could even form a trio. Hence, the sum of the y values in this window is $< \frac{p_j-1+c_f}{2} + 1 + \frac{p_j-1+c_f}{8}$. Hence, if $p_j - 1 + c_f > \frac{8}{3}$, then this is not possible. Since everything is integral, let us consider $p_j - 1 + c_f = 2$. However, any two consecutive slots which are not closed in the LP together has a y value < 1.5 in the barely open structure, which is less than 2. Hence, this case is also not possible. Finally, we consider $p_j - 1 + c_f = 1$, which means the number of integrally open accounted slot within the window of this job j is 1. That means, the release time of this job $r_j = d_k - 1$, since one unit has to be scheduled in d_k. However $d_k - 1$ is originally barely open with y value < 0.5, hence it is not possible that none of the jobs assigned in $d_k - 1$ can be moved elsewhere, for a feasible LP solution to exist. Therefore, no such job j exists, and max-flow will find a feasible assignment if the conditions of the claim hold. □

Proof of Lemma 27 continued:

From the above claim, for max-flow to fail, since $n_{k,1} + n_{k+1,1} + 2n_{k+1,2} + n'_{k+1,1} \leq 2g$ and d_{k+1} is not a stopping deadline, therefore it must be that $n_{k,1} + n_{k+1,2} \geq g + 1$. For LP feasibility, the $\sum_{z \leq p \leq k} n_{p,1} + \sum_{z \leq p \leq k} \frac{n'_{p,1}}{2} + \sum_{z \leq p \leq k} \frac{3n'_{p,2}}{2} + \frac{n_{(k+1),2}}{2} \leq g(k - z + 1)$. However, from the induction hypothesis, $2\sum_{z \leq p \leq k} n_{p,1} + \sum_{z \leq p \leq k} n'_{p,1} + 2\sum_{z \leq p \leq k} n'_{p,2} + n_{(k+1),2} > 2g(k - z + 1)$. Hence this case is also not possible. Therefore, max-flow can fail only if there are $\geq 2g + 1$ job units (including flexible, unit length and non-unit length) jobs in $\mathcal{J}_k \cup \mathcal{J}_{k+1}$ with release time $\geq d_k$.

Therefore, we have proved the claim by induction. □

Proof of Lemma 25 continued:

When all the fully open slots before d_i have dependents, we try making d_i a trio with the closest fully open slot d_k and its dependent (which must necessarily be $d_k - 1$ according to the above lemmas). If we can, then we move to d_{i+1}. Suppose we cannot. Therefore, the total mass scheduled by the LP feasible solution in slots d_i, d_k and $d_k - 1$ is $< \frac{3g}{2}$, whereas opening d_k and $d_k - 1$ would give a space of $2g$. However, we cannot close d_i since flow did not find a feasible assignment. Therefore, there must be $\geq g + 1$ jobs with release time d_k in $\mathcal{J}_i \cup \mathcal{J}_k$ and these must be unit length jobs for LP feasibility.

Similar to Lemma 27, we assume the deadlines are open in consecutive order. The alternating structure consists of $[d_z, d_{z+1}, \ldots, d_i]$. We assume the same notation for the jobs as in Lemma 27. For d_z, $n'_z + n_z \geq g + 1$, where d_z is the stopping deadline, where n'_z denotes the number of length 2 jobs plus the flexible unit length jobs which must be scheduled before the closest earlier open deadline. For any $d_i > d_p >$, $n_{p,1}$ denotes the number of unit length rigid jobs with release time d_p, $n_{p,2}$ denotes the number of length 2 jobs with release time $\geq d_{p-1}$ and $n'_{p,1}$ denotes the number of flexible unit length jobs with deadline $\geq d_{p-1}$. By Lemma 27, for any $d_z < d_p < d_i$, $n_{p-1} + n_{p,1} + 2n_{p,2} + n'_{p,1} \geq 2g + 1$. For d_i, we have argued above that $n_{i-1,1} + n'_{i,1} \geq g + 1$, since for d_i, $n_{i,1} = n_{i,2} = 0$ for LP feasibility.

Adding the above, $\sum_{z \leq p \leq i} n'_{p,1} + 2\sum_{z \leq p \leq i} n_{p,1} + 2\sum_{z \leq p \leq i} n_{p,2} > 2(i - z)g$.

However, for LP feasibility, jobs can be assigned to an extent $< \frac{1}{2}$ in the barely open slots. Hence, $\sum_{z \leq p \leq i} \frac{n'_{p,1}}{2} + \sum_{z \leq p \leq i} \frac{3n_{p,2}}{2} + \sum_{z \leq p \leq i} n_{p,1} \leq (i - z)g$, which is a contradiction. Therefore we will always be able to charge a barely open slot which cannot be closed by max-flow assignment.

The proof of Lemma 25 follows from the above discussion.

Theorem 19. *There exists a polynomial time algorithm which gives a solution of cost at most twice that of any optimal solution to the active time problem on non-unit length jobs with integral preemption.*

Proof. From the above discussion, it follows, that at the end of every iteration i, we have a set of integrally open slots $\mathcal{O}_i$, such that there is a LP feasible fractional assignment of jobs in $\bigcup_{x \le i} \mathcal{J}_x$ in $\mathcal{O}_i$. Furthermore, $|\mathcal{O}_i| \le 2 \sum_{1 \le k \le i} Y_k$. We do this till the last deadline d_ℓ. At the end, we are assured of an integral feasible assignment on the set of opened slots via max-flow, while the number of open slots is at most twice the optimal LP objective function value. Hence, we get a 2 approximation. □

6.4.5 LP Integrality gap

We show here that the natural LP for this problem has an integrality gap of 2. Hence, a 2 approximation is the best possible using LP rounding. Consider, g pairs of adjacent slots. In each pair, there are $g+1$ jobs which can only be assigned to that pair of slots. An integral optimal solution will have cost $2g$, where as LP optimal solution, will open each such pair up to 1 and $\frac{1}{g}$, assign all the $g+1$ jobs up to $\frac{g}{g+1}$ to the fully open slot, and up to $\frac{1}{g+1}$, to the barely open slot, thus maintaining all the constraints. Therefore, optimal LP solution has cost $g+1$ and $\frac{g+1}{2g} \to 2$ as $g \to \infty$.

Chapter 7: The Percentile Problem

The previous chapters have dealt with problems of resource optimization for data centers, that arise due to energy issues. However, energy usage is not the only source of the huge operating costs of data centers. Massive amounts of data are moved daily between data centers, for which data centers have to pay the Internet Service Providers a huge amount of money. As outlined in Chapter 1, cost accounting for network bandwidth usage is performed either using the percentile rule or the peak bandwidth usage rule. In this chapter, we formally define the percentile problem, and also give an overview of the related work in this area and present our results on this problem.

7.1 The Percentile Rule and Overview of Results

For network bandwidth cost, a widely used rule for charging is the 95th percentile rule. Over each billing cycle (say a day), the cycle is broken into "slots" (for example one minute period of time) and the bandwidth usage per slot is sampled. In other words, the billing is based on a vector $\mathbf{x}$ where x_i is the volume of traffic sent over the i^{th} slot. In the peak bandwidth charging scheme, one is charged for the maximum bandwidth utilization over any slot in one billing cycle. In the 95th percentile charging scheme, one is not charged on the maximum, but on the 95th percentile of this vector for billing.

The performance requirements of data transfers typically include transmission delay, as the data chunks have to be delivered within some time period. Often, such delay requirements are not stringent. Our goal is to “adjust” the traffic, so that no packet is delayed by more than its allowed delay and the required percentile is minimized.

We can view data chunks as jobs, and links (or senders) as servers. Then, the problem can be viewed as a scheduling problem aimed as reducing the 95th percentile (or any other percentile) usage. The arrival time of jobs can be considered to be their release time, and the allowed delays for jobs determine their deadlines.

Consider the following simple example: Suppose there is a server that serves jobs of the same size. Jobs arrive at the server in each time slot. The server has a capacity of serving up to 5 jobs in one slot. Over an accounting period of 5 time slots, the number of jobs arriving at the server is $[3, 3, 3, 3, 0]$. Our goal is to minimize the 80th percentile (second largest in this case) of the number of jobs served in one time slot. If all jobs are served within the time slot in which they arrive, with no delay, then the service sequence is also $[3, 3, 3, 3, 0]$. Note that according to the 80th percentile rule, the billed output bandwidth usage is also 3. However by delaying each job by at most one time slot, the server can instead choose the service sequence $[2, 4, 2, 2, 2]$, thus reducing the billed bandwidth usage to 2, since the slot with the maximum load is not counted. Note that the trade-off here is that four jobs are delayed. One is delayed from slot 1 to slot 2, one from slot 3 to slot 4, and two from slot 4 to slot 5.

The above example looks straightforward. However, in reality, packets can have different delay requirements. For example, web traffic is more important with stringent delivery requirements, but bulk file transfers could be less time sensitive, and the same is true with transfer of video files or other large data sets. As a result, we have different classes of traffic with different delay requirements. Furthermore, the data may have dependencies among each other, requiring all the data of a single type to be transmitted together. Furthermore, the links have a certain fixed capacity which cannot be exceeded by the data transmission in any time slot.

In this work we study the relevant and realistic problem of multi-class traffic. In this setting, packets can have different delay requirements, between 0 and D. First we consider the offline problem, where where all packet arrivals are known in advance and the 95^{th} percentile of the bandwidth utilization needs to be minimized. In most cases network traffic exhibits some pattern, and bulk data that can be delayed is already known. Hence algorithms for the offline version of the problem are quite relevant. We give an optimal polynomial algorithm for the case of multi-class traffic.

We then consider the problem where data packets have dependencies among them and certain groups need to be be transmitted together. While prior work shows that this problem is weakly NP-hard, no algorithm was provided. We first show strong NP-hardness when the delay D is large, even for the uniform delay problem. Then we provide a fully polynomial time approximation scheme for the case of $D = 1$.

7.2 Related Work

While scheduling problems have been investigated for decades [61], metrics considered in prior work have more to do with minimizing the maximum load, or scheduling a given fraction of jobs, while minimizing maximum load [32]. Our model represents a new class of problems, and is based on the charging mechanism in use.

The bounded delay buffer management problem, first introduced by Kesselman et al. [38] is a somewhat similar problem that has been widely studied in literature. This is an online problem, where jobs have weights and deadlines, and the goal is to maximize the weight of jobs served within their deadline, where the processor can only process one job at a time. Though the problem seems related to the problems we consider in this work, it is not obvious that their techniques can be directly applied to our problem.

A few papers do study the impact of the 95th percentile. For example, Dimitropoulos et al. [15] study the impact of time slot size through modeling and measurement study. Further, this accounting property is leveraged to reduce the cost of bulk data transfer by Laoutaris et al. [44]. Goldenberg et al. [27] design smart routing algorithms for data streams on multiple ISP's with the consideration of the 95th percentile model. In contrast, the main problems we consider are the following: minimize the 95th percentile through job level scheduling, and exploit the 95th percentile model.

In the broadcast scheduling literature, a related problem was studied by Charikar and Khuller [9]. They studied the problem of minimizing the response time for a fixed fraction of the total requests, ignoring the response time of a few requests, which is allowed to be large.

7.3 Problem Description

Consider a system with a single server, where time is slotted. Let N be the total number of time slots in an accounting cycle. A certain number of jobs arrive in time slot i. The jobs can be of unit size or variable size. Furthermore, each job is allowed some delay. Some number of them are served by the end of that time slot. Jobs that are served in the time slot in which they arrived incur no delay. Unserved jobs are carried over for service in future time slots; each time a job is carried over, it incurs an additional delay of one time slot.

The goal is to choose a service schedule, such that the 95th percentile of maximum of the data transfer volume over all slots in the billing cycle is minimized. The server has a capacity of C. Thus, at most C amount of jobs can be served at any time slot. No job should be delayed by more than its allowed delay.

7.4 Prior Work and Our Contributions

A special case of this problem was first considered by Golubchik et al. [28]. In that problem, all jobs have identical delay D. An optimal polynomial time algorithm for the offline problem was presented. Here we generalize the above result, by extending it to multi-class traffic, where jobs can have any delay between 0 and D. We then generalize the problem even further by extending our analysis to consider non-unit sized jobs. For this problem, we show strong NP-hardness for the general case, and give a fully polynomial time approximation scheme for the special case where jobs can have delays in $0, 1$.

We do not consider the online variant of this problem, as it was shown by Golubchik et al. [28] that there can be no deterministic online algorithm with bounded competitive ratio for this problem, even for the special case where all jobs are allowed unit delay.

7.5 Offline Problem: Minimizing Percentile Cost with varying Delays

We provide an optimal offline algorithm for the minimization of percentile cost when the jobs are unit sized, but the delays can vary from 0 to D. Let us denote the job arrival in time slot i as a vector $\boldsymbol{A(i)}$ of dimension $D+1$, where the element $A_d(i)$ denotes the jobs arrived in time slot i which can be delayed by d time slots, $d \in \{0, \ldots, D\}$.

According to the 95th percentile accounting rule, the number of jobs served in the top 5% of the total time slots are not included in accounting. These time slots can be viewed as "free" because the jobs served in these time slots do not change the 95th percentile result. Let $T = 5\% \times N$ be the number of "free" time slots (we assume T is an integer). Assume that in "free" time slots, up to C jobs can be served, whereas in the other $N - T$ time slots, at most $H (H \leq C)$ jobs can be served. In this way, H serves as the upper bound of the 95th percentile of the number of jobs served in one time slot. Our algorithm works for any choice of percentile however. The goal is to find the smallest H, so that there is a schedule to serve jobs, such that all but T slots serve at most H jobs. We guess H, and then verify if our guess leads to a feasible solution. We can do a binary search to find the smallest value of H.

The jobs left unfinished by time slot i consist of jobs that have been delayed for $\{1, 2, ..., D\}$ time slots. Intuitively, the service priority for these jobs should be different, as should be the service priority for the jobs which have arrived in i with different possible delays. In fact, the key observation is that there is no difference in priority between a job that has arrived in i with allowed delay 0, and a job that arrived earlier, exhausted all the delay allowed, and is still unserved. So, similar to the arrival vector, we define an unserved vector U, where $U_d(i)$ denotes the jobs which can still be delayed by d more slots. In general, let $\boldsymbol{U(i)} = (U_0(i), ..., U_D(i))$ be the vector of unfinished jobs at *the end* of time slot i. Note that for a feasible solution, $U_0(i) = 0$ for all i. Let us define $\boldsymbol{\tilde{U}(i)}$ as the vector of unserved jobs coming into the time slot i from $i-1$. $\tilde{U}_D(i) = 0$, since the maximum allowed delay for any job is D, and the jobs in the vector $\boldsymbol{\tilde{U}(i)}$ have all arrived in earlier time slots.

Let us define an augmented vector $\boldsymbol{U^A(i)} = (\boldsymbol{A(i)} + \boldsymbol{\tilde{U}(i)})$. At a time slot i, $U_d^A(i)$, (where $d \in \{0, \ldots, D\}$) represents the total amount of pending jobs arrived at i or earlier, which can be delayed for d more slots. Let $\boldsymbol{S(i)} = (S_0(i), ..., S_D(i))$ be the service vector. The service vector $\boldsymbol{S(i)}$ represents the amount of work done in time slot i. Specifically, $S_d(i)$ represents the quantity of jobs $U_d^A(i)$ served in time slot i, hence, $0 \leq S_d(i) \leq U_d^A(i)$. For a solution to be feasible, $S_0(i) = U_0^A(i)$ for all i. In this setting, a choice of H being feasible means that $\sum_{d=0}^{D} S_d(i) \leq H$ for an "accounted" slot and $\sum_{d=0}^{D} S_d(i) \leq C$ for a "free" slot.

One might ask if a given instance is at all feasible, i.e., if there exists any value $H \leq C$, for which all jobs can be served respecting their release times and allowed delays. An easy way of testing feasibility of a given instance is via a max-flow computation. The jobs are unit in length and each job has a maximum delay constraint. Time is slotted, and the maximum number of packets sent in a single time slot can be at most C. Hence, one can set up a flow network, and compute the maximum flow to check if the instance is feasible.

We choose the service vector $\boldsymbol{S(i)}$ in a greedy manner. Specifically, jobs with lower allowed delay will be served before those with longer allowed delay.

Let us define $OPT(i,t), \{i = 1,2,3,...,n; t = 0,1,2,...,T\}$ as the minimum number of total jobs left unfinished by the end of time slot i, using t "free" time slots. We show how to compute $OPT(i,t)$ in the following.

For every (i,t), we have associated vectors $\boldsymbol{U(i,t)}$ and $\boldsymbol{S(i,t)}$, where the former is the vector of unserved jobs at the end of time slot i when t free slots have been used, and the latter is the vector of served jobs in time slot i when t free slots have been used. These vectors are stored along with the $OPT(i,t)$ entry.

$OPT(i,t)$ is chosen to be one of the following values $OPT_1(i,t)$ and $OPT_2(i,t)$, depending on whether i is a free slot or an accounted slot. $OPT_1(i,t)$ denotes the number (or cost) of pending jobs after time slot i if in slot i we serve up to C jobs and we have used up $t-1$ free slots before i. $OPT_2(i,t)$ denotes the number (or cost) of pending jobs after time slot i if in slot i we serve up to H jobs and we have used up t free slots before i. If in either case, jobs with allowed delay 0 at i cannot be fully served in i, the cost of pending jobs is set to ∞, which means the current choice of H is infeasible.

If i is a free slot, then we can transmit up to C jobs. This is distributed among the unserved jobs as follows. First, we define the augmented vector $\boldsymbol{U^A(i,t)} = \boldsymbol{A(i)} + \boldsymbol{\tilde{U}(i,t)}$. Here, $\boldsymbol{\tilde{U}(i,t)}$ is equal to the vector $\boldsymbol{U(i-1,t-1)}$ with all its elements shifted down by one position, and $\tilde{U}_D(i,t) = 0$.

Let k be the smallest index such that $\sum_{j=0}^{k} U_j^A(i,t) \leq C$. If there exists no such k, then the guess of H is infeasible. Set $OPT_1(i,t) = \infty$ in this case.

Let $\hat{U}_{k+1}^A(i,t) = C - \sum_{j=0}^{k} U_j^A(i,t)$. Then
$\boldsymbol{S(i,t)} = (U_0^A(i,t), U_1^A(i,t), \ldots, U_k^A(i,t), \hat{U}_{k+1}^A(i,t), 0, \ldots, 0)$. We define $\boldsymbol{U^C(i,t)}$ to be $\boldsymbol{U^A(i,t)} - \boldsymbol{S(i,t)}$. Note that $U_0^C(i,t) = 0$, because k is now well-defined.

In this case, the total number of unserved jobs after (i,t) is $OPT_1(i,t) = \sum_{d=0}^{D} U_d^C(i,t)$.

If i is an accounted slot, then we can transmit up to H jobs. This is again distributed among the unserved jobs as follows. First, we define the augmented vector for this case as $\boldsymbol{U^A(i,t)} = \boldsymbol{A(i)} + \boldsymbol{\tilde{U}(i,t)}$. Here, $\boldsymbol{\tilde{U}(i,t)}$ is equal to the vector $\boldsymbol{U(i-1,t)}$ with all its elements shifted down by one position and $\tilde{U}_D(i,t) = 0$.

Let k be the smallest index such that $\sum_{j=0}^{k} U_j^A(i,t) \leq H$.

If there exists no such k, that means $U_0^A(i,t) > H$, hence this value of H is infeasible. In this case, set $OPT_2(i,t) = \infty$.

Otherwise, let $\hat{U}_{k+1}^A(i,t) = H - \sum_{j=0}^{k} U_j^A(i,t)$. Then
$\boldsymbol{S(i,t)} = (U_0^A(i,t), U_1^A(i,t), \ldots, U_k^A(i,t), \hat{U}_{k+1}^A(i,t), 0, \ldots, 0)$. Now, $\boldsymbol{U^H(i,t)}$ is defined to be $\boldsymbol{U^A(i,t)} - \boldsymbol{S(i,t)}$. Note that $U_0^H(i,t) = 0$, because k is now well-defined.

In this case, the total number of unserved jobs after (i, t) is $OPT_2(i, t) = \sum_{d=0}^{D} U_d^H(i, t)$.

$OPT(i, t)$ is chosen to be the minimum of $OPT_1(i, t)$ and $OPT_2(i, t)$. The corresponding vector $\boldsymbol{U^C(i, t)}$ or $\boldsymbol{U^H(i, t)}$ is retained as vector $\boldsymbol{U(i, t)}$.

Therefore, for every (i, t), $OPT(i, t) = \sum_{d=0}^{D} U_d(i, t)$ by definition. Hence, we can now define $OPT(i, t)$ more compactly as follows.

$$OPT(1, 0) = \max(\sum_{d=0}^{D} A_d(1) - H, 0)$$
$$OPT(1, 1) = \max(\sum_{d=0}^{D} A_d(1) - C, 0)$$
$$OPT(i, t) = \min \begin{cases} \max(OPT(i-1, t-1) + \sum_{d=0}^{D} A_d(i) - C, 0) \\ \text{ if } (U_1(i-1, t-1) + A_0(i)) \leq C \\ \max(OPT(i-1, t) + \sum_{d=0}^{D} A_d(i) - H, 0) \\ \text{ if } (U_1(i-1, t) + A_0(i)) \leq H \\ \infty \end{cases}$$

We guess the values of H by using binary search over the interval $[1, \ldots, C]$ to find the minimum value of H for which $OPT(N, T) = 0$. This takes time $O(N \cdot T \cdot D \cdot \log_2 C)$.

If $OPT(N, T) = 0$ for a given H, then that value of H is certainly feasible, by definition of our dynamic program and algorithm, since we do not violate any of the required conditions. We claim that if a feasible solution exists for a given $H(\leq C)$, then for that H, $OPT(N, T) = 0$. The following lemmas establish this claim. We refer to jobs with allowed delay 1 which remain unserved at the end of the current time slot as "urgent" jobs. These are jobs that must be completed in the next time slot by any feasible solution.

Lemma 28. *For a given value of H for which a feasible solution exists, among all such solutions, $OPT(i,t)$ minimizes the* number of urgent jobs *at the end of any time slot i, assuming that at most t of the time slots $[1,\ldots,i]$ can serve up to C jobs and all other slots can serve at most H jobs.*

Proof. We prove this by induction. The base cases, $i = 1$ and $t = 0$ and $t = 1$ are true since we favor jobs with lower allowed delay, and serve higher delayed jobs only if the lower delay jobs are exhausted and there is still unused serving capacity.

Suppose by induction hypothesis, the claim is true for all $i < k$ and $t \leq i$. We now wish to prove this for $i = k$ and all $t \leq k$. Consider the optimal solution $\mathcal{O}$ which minimizes the number of urgent jobs at the end of time slot k such that at most $t \leq k$ time slots out of slots $[1,\ldots,k]$ are allowed to serve up to C, and all others H.

Suppose in $\mathcal{O}$ in time slot k, the number of urgent jobs coming from slot $k-1$ and the current arrivals with delay 0, $A_k(0)$ together sum up to more than H. Therefore, $\mathcal{O}$ must make the slot i a free slot. Let us denote the total number of urgent jobs in $\mathcal{O}$ coming from slot $k-1$, (assuming at most $t-1$ slots out of $[1,\ldots,k-1]$ in $\mathcal{O}$ were free), as $U_1^{\mathcal{O}}(k-1,t-1)$. By induction hypothesis, is $U_1^{\mathcal{O}}(k-1,t-1) \geq U_1(k-1,t-1)$. The quantity $A_k(0)$ is the same for all algorithms. Since we prioritize the jobs with lower delay, the number of unserved jobs with delay

1 that remains at the end of time slot k after serving C jobs in our dynamic program is only $\leq$ that for $\mathcal{O}$. Hence, the induction hypothesis is proved for this case.

On the other hand, if $\mathcal{O}$ in time slot k serves up to H jobs, the number of urgent jobs coming from slot $k-1$ and the current arrivals with delay 0, $A_k(0)$ together sum up to $\leq H$. Let us denote the total number of urgent jobs coming from slot $k-1$ (assuming at most t slots out of $[1,\dots,k-1]$ in $\mathcal{O}$ were free) as $U_1^{\mathcal{O}}(k-1,t)$. By induction hypothesis, is $U_1^{\mathcal{O}}(k-1,t) \geq U_1(k-1,t)$. Since $A_k(0)$ is the same for all algorithms, and we prioritize the jobs with lower delay, the induction hypothesis is proved for this case as well. $\square$

Lemma 29. *For a given H for which a feasible solution exists, among all such solutions, $OPT(i,t)$ minimizes the* number of unserved jobs *in the first i slots, assuming that at most t slots can have load up to C, and all the other slots are* required *to have load at most H.*

Proof. The proof is by induction on (i,t). Given the base cases defined in our dynamic program, we assume that we have proven the claim for $(i-1,j)$ for all $j \leq i-1$ and wish to prove it now for (i,t) for all $t \leq i$. From Lemma 28, we know that $U_1(i,t)$ is minimized by OPT for all i and $t \leq i$.

Let us consider the optimal solution $\mathcal{O}(i,t)$ which minimizes the total number of unserved jobs at the end of slot i using at most t free slots. Suppose in $\mathcal{O}$, the number of jobs arrived with delay 0 plus the number of urgent jobs from slot $i-1$ exceeds H. Therefore, this must be a free slot (of capacity C) in $\mathcal{O}$, since the number of urgent jobs is greater than H. Let the unserved job vector from $i-1$ using at most $t-1$ free slots in $\mathcal{O}$ be denoted as $U_1^{\mathcal{O}}(i-1,t-1)$. We know from Lemma 28 that $U_1(i-1,t-1) \leq U_1^{\mathcal{O}}(i-1,t-1)$. The job arrivals $A_0(i)$ are the same for any algorithm. Hence, we can also handle *feasibly* all these packets by using a capacity C. We serve jobs till we have exhausted the capacity C, hence the total number of unserved jobs passed on to $i+1$ is $\leq$ that passed on by $\mathcal{O}$. Hence the induction hypothesis is proved for this case.

On the other hand, suppose $\mathcal{O}$ serves $\leq H$ jobs in time slot k. Therefore, the number of urgent jobs in slot i in $\mathcal{O}$ is $\leq H$. These jobs must include those unserved with delay 1 at the end of time slot $k-1$ in $\mathcal{O}$, where at most t out of slots $[1 \ldots k-1]$ could serve up to C jobs. Let us denote this as $U_1^{\mathcal{O}}(k-1,t)$. From Lemma 28, we know that $U_1(k-1,t) \leq U_1^{\mathcal{O}}(k-1,t)$. The newly arrived jobs is the same irrespective of the algorithm used. Hence, we can also handle feasibly all the newly arrived packets of allowed delay 0 by using a capacity H. Furthermore, we serve jobs till we have exhausted the capacity H, hence the total number of unserved jobs passed on to $i+1$ is $\leq$ that passed on by $\mathcal{O}$. This proves the induction hypothesis is proved for this case. $\square$

Theorem 20. *There exists a polynomial algorithm which determines whether a given instance of min-percentile problem with variable job delays is feasible, and if so, finds an optimal feasible schedule minimizing the value of percentile cost.*

The theorem follows from Lemma 28 and Lemma 29.

7.6 Offline Problem: Minimizing Percentile Cost with Non-unit Sizes

In this section we consider the problem of minimizing percentile cost when the jobs are not unit-size any more, but may require arbitrary units of processing.

7.6.1 Hardness of Approximation

We first prove that the offline problem of minimizing the percentile cost when the input can have variable sizes is strongly NP hard, for large D, even when all jobs are allowed uniform delay D.

Let us refer to this problem as $MinPerc$.

Theorem 21. *$MinPerc$ is strongly NP-hard.*

Proof. We are given an instance of 3-Partition with $3n$ elements a_i, such that $\sum_{i=1}^{3n} a_i = nB$. We want to know if there exist n partitions of the elements such that each partition sums up to B. From this problem, we create an instance of $MinPerc$ such that a feasible schedule exists for percentile cost $H = B$ if and only if a valid 3 partition exists. We set $D = n - 1$ and $C > B$. Let us minimize the x percentile for a total of N time slots in an accounting cycle. This means, we are allowed to drop $T = x\%$ of N time slots. We create T jobs of size C, released at time $t = [1 \ldots T]$. We create $3n$ jobs, all released at time $T + 1$, of sizes corresponding to the elements a_i in the instance of 3-Partition. A feasible schedule would send all the jobs integrally, respecting their deadline constraints. Any feasible solution for percentile cost $H = B < C$ to this instance of $MinPerc$ would have to schedule the first T jobs of size C each, in the time slots 1 to T and would have to drop these time slots. The rest of the jobs released at $T+1$ all need to be integrally scheduled within the next $D+1$ time slots, and for percentile cost to be at most B, sum of the job sizes scheduled in any time slot $t \in [T+1, \ldots, T+D+1]$ should be at most B. Since the total sum of job sizes is nB, and $D+1 = n$, each slot should serve exactly B size of jobs. This schedule would also give a valid 3-partition. Conversely, if a valid 3-partition exists, the partitions give a feasible schedule for $B = H$. □

7.6.2 FPTAS for $MinPerc$ for $D = 1$

In this section, we describe an FPTAS to $MinPerc$, where jobs can have variable sizes, and the allowable delay for every job is 1.

Theorem 22. *There exists a polynomial time algorithm that returns a feasible schedule to the $MinPerc$ problem with variable job sizes and uniform delay $D = 1$, such that the minimum value of H it returns is $\leq OPT(1+\epsilon)$ for a fixed $\epsilon > 0$, where OPT is the optimal solution.*

Our approach in determining a feasible value of H will involve guessing a value of H. We perform binary search over $[1 \ldots C]$ in time $O(\log_2 C)$ since we assume job sizes are integral. For each guessed value of H, we will try to find a feasible schedule by a dynamic program. The lowest value of H for which a feasible schedule is found will be returned by the algorithm. First, we limit the number of different job sizes to be a function of C and ϵ. The minimum job size is 1 and the maximum job size is C (otherwise the instance is not feasible). We round down the object sizes, such that a job with size s, which is $(1+\epsilon)^i < s \leq (1+\epsilon)^{i+1}$, is rounded down to $(1+\epsilon)^i$. The total number of different job sizes now becomes $O(\log_{1+\epsilon} C)$. Let this modified instance be I'. We also set the new capacity to be $C' = \frac{C}{1+\epsilon}$. Note that feasibility of the instance is not affected by this, since any job which is $> C'$ in size, after rounding down, becomes at most C'.

For each guess H, we then solve the following dynamic program. Let $Perc(i,t)$ denote the amount of work not completed after i time slots, out of which at most t slots could have served up to C' and the remaining could have served at most H. Let $\boldsymbol{A(i)}$ denote the set of jobs arriving at time slot t (which need to be completed by $t+1$) and $A(i)$ denote their total size. We denote the set of jobs unserved in time slot t as $\boldsymbol{U(i)}$ and their total size as $U(i)$. $U(i) \leq A(i)$. The set of jobs served in time slot t is denoted as $\boldsymbol{S(i)}$ and their total size as $S(i)$. For a feasible solution, if i is a free slot, then $S(i) \leq C'$, otherwise $S(i) \leq H$.

We will set $Perc(i,t)$ to be the minimum of two quantities $Perc_1(i,t)$ and $Perc_2(i,t)$, where the former corresponds to i being a free slot and the latter correspond to i being a paid slot. If the slot i is a free slot, the unserved amount of jobs from the previous time slot to be served would be $U(i-1,t-1)$. If $U(i-1,t-1) > C'$, set $Perc_1(t) = \infty$. Otherwise, define $cap(i,t) = C' - U(i-1,t-1)$. Now we solve a knapsack problem, where the jobs in $\boldsymbol{A(i)}$ correspond to the items to be packed in the knapsack, the profit of every job being equal to its size and the capacity of the knapsack is $cap(i,t)$. The knapsack FPTAS maximizes the profit of items packed in the knapsack when the number of distinct profits is fixed. In our case, since profits equal sizes, this will maximize the total size of jobs which can be served in the time slot without violating the capacity $cap(i,t)$. Let the set of items (jobs) successfully packed in the knapsack by the knapsack FPTAS be $\boldsymbol{S'(i,t)}$ with total size $S'(i,t)$. Therefore, the set of served jobs in this case is $\boldsymbol{S(i,t)} = \boldsymbol{S'(i,t)} \cup \boldsymbol{U(i-1,t-1)}$ and $S(i,t) = S'(i,t) + U(i-1,t-1) \leq C'$. Set $Perc_1(i,t) = U(i,t) = \max(A(i,t) - S'(i,t), 0)$ and $\boldsymbol{U(i,t)} = \boldsymbol{A(i,t)} \setminus \boldsymbol{S(i,t)}$.

If slot i is a paid slot, the set of unserved jobs to be served in i is $\boldsymbol{U(i-1,t)}$. If $U(i-1,t) > H$, set $Perc_2(i,t) = \infty$. Otherwise, set $cap(i,t) = H - U(i-1,t)$, and pack the knapsack as in the previous case. Let the set of jobs packed in the knapsack maximizing the space utilization be $\boldsymbol{S'(i,t)}$. The set of served jobs in this case is $\boldsymbol{S(i,t)} = \boldsymbol{S'(i,t)} \cup \boldsymbol{U(i-1,t)}$ and $S(i,t) = S'(i,t) + U(i-1,t) \leq H$. Set $Perc_2(i,t) = U(i,t) = \max(A(i,t) - S'(i,t), 0)$ and $\boldsymbol{U(i,t)} = \boldsymbol{A(i,t)} \setminus \boldsymbol{S(i,t)}$.

We set $Perc(i,t) = \min\{Perc_1(i,t), Perc_2(i,t)\}$ and store the corresponding vectors $\boldsymbol{S(i,t)}$ and $\boldsymbol{U(i,t)}$.

The lowest value of H for which $Perc(N,T) = 0$, is returned by the algorithm and the corresponding service vectors in each step can be obtained by backtracking in the main dynamic programming table (that is, noninclusive of the intermediate knapsack dynamic programs for each slot). For the DP table, define $U(0,t) = 0$ for all t.

Lemma 30. *For the instance I', $Perc(i,t)$ minimizes the total size of unserved jobs after time slot i, such that at most t of the i time slots can serve up to a size of C' of jobs, and the remaining slots can served up to a size H.*

Proof. We will prove this via induction. Let us consider the base cases $Perc(1,0)$

and $Perc(1,1)$. Since we use the knapsack dynamic program to compute the set of jobs which maximize the capacity utilization (given a fixed number of sizes), the base cases are true.

Suppose by induction hypothesis, the claim is true for all slots $< i$ and for all $t \leq i-1$. We wish to prove it for i and all $t \leq i$. In the optimal solution which minimizes the amount of unserved jobs in slot i with at most t free slots, if the number of jobs at i which are pending from $i-1$ is $> H$, then i must be a free slot. We use an optimal solution for the sub-problem $(i-1, t-1)$, hence if the optimal solution passes on at most p size of unserved jobs, we also pass on at most p. Therefore, if a feasible solution exists, that is, $p \leq C'$, we will also be able to serve all the pending jobs. Further, we will pass on minimum size of unserved jobs, by the optimality of the knapsack dynamic program for fixed number of profits (here: sizes). Similarly, if the optimal solution serves at most H size of jobs in slot i, then the number of pending jobs from previous slot in our case is at most that for the optimal solution, since by induction hypothesis, we use an optimal solution for the sub-problem $(i-1, t)$. Furthermore, we pass on the minimum size of unserved jobs, by the optimality of knapsack dynamic program. Hence the induction hypothesis is proved. $\square$

Since we have rounded down the object sizes, the optimal solution for I is $OPT \geq H$, where H is the minimum feasible value returned by $Perc$. On replacing the true sizes of the jobs, the free slots increase in load size to at most $(1+\epsilon)C' = C$. The accounted slots increase in load size to at most $(1+\epsilon)H \leq (1+\epsilon)OPT$. Hence we get a $(1 + \epsilon)$ feasible (not violating C) approximation to the optimal offline solution in time $O(N^2 \cdot T \cdot \log_2 C \cdot \log_{1+\epsilon} C)$.

Chapter 8: The Min-Max Problem

In this chapter we consider the online problem of peak bandwidth usage minimization for multi class traffic. Data chunks or jobs arrive at the sender or server, with specified allowed delays. All jobs need to be served within their allowed delays. Our goal is to minimize the maximum amount of job served (or data transferred) in any time slot over the billing cycle. This problem can also be viewed as the 100 percentile problem. Interestingly, the min-max problem has connections to the energy minimization problem, as outlined in Section 8.1.

8.1 Related Work

The min-max problem can be viewed as dual to the *bounded delay buffer management problem*, first introduced by Kesselman et al. [38], that has since been widely studied in literature, eg. [17,36,37,46,47]. This is an online problem, where jobs have weights and deadlines, and the goal is to maximize the weight of jobs served within their deadline, where the processor can only process one job at a time. The jobs that are not served before their deadlines are dropped. Both deterministic and randomized variants have been studied in literature. In another variant of this problem, finite capacity buffers were considered by Li [45]. In contrast, in our problem, all the packets have to be transmitted, and we wish to minimize the maximum number of packets sent in a single time slot (for the 100 percentile problem).

In the CPU scheduling literature, Yao, Demers and Shenker [75] provide an optimal offline algorithm for minimizing the maximum energy consumed by a processor when the power consumed is a convex function of the speed. Their algorithm also minimizes the maximum speed of the processor at any point in time, subject to the constraint that all jobs are completed by their deadline. The algorithm to compute the offline optimum for the min-max problem is essentially the same algorithm as theirs. Bansal, Kimbrel and Pruhs [2] gave an optimal algorithm for the online version of the energy minimization problem. This algorithm is e-competitive for the online problem of minimizing the maximum speed at any instant such that all jobs are completed by their deadline. They also show that the lower bound of e is tight for the problem of minimizing the maximum speed, when time is considered to be continuous, and jobs may be allowed arbitrarily long delays.

8.2 Prior Work and Our Contributions

A special case of this problem was first considered by Golubchik et al. [28], where all the jobs are allowed the same delay D. A lower bound of 1.53 was presented for the competitive ration of any online deterministic algorithm for this problem, and this bound was achieved when $D \to \infty$. An online algorithm, Equal Split, with competitive ratio of 2 was also presented in this work. Here, we improve the lower bounds for several values of D from prior work, when all data can be delayed by D. Significantly, we show new bounds for small D, which exceed the best bound known earlier, that was achieved by allowing D to be arbitrarily large. Then we provide new lower bounds for the problem of multi-class traffic, where delay D can be variable. We show that a natural extension of the online algorithm Equal Split ([28]) can perform very badly when traffic can have variable delay. Finally, we give a tight online algorithm for $D \in \{0, 1\}$ and a 2-competitive algorithm for $D \in \{0, 1, 2\}$.

As in prior work [28], we consider the jobs to be arbitrarily splittable. Given the large volumes of data that need to be transmitted per time slot, this is an entirely reasonable assumption. In fact for low volumes of data, when packets need to be considered integrally, it is easy to show better lower bounds. Consider the case of $D = 1$. In the first time slot, the adversary sends two unit size packets. If the algorithm sends both, the adversary stops, having forced a competitive ratio of 2. Otherwise, the algorithm sends over 1 packet to the next time slot, when the adversary sends 4 unit size packets. Now, the algorithm will be forced to send at least 3 packets in 1 time slot for feasibility, whereas the optimum is 2, forcing a competitive ratio of 1.5. Henceforth, we consider the work to be arbitrarily splittable. In the following sections, we improve lower bounds for uniform D, as well as provide new lower bounds for variable D, and furthermore, analyze online algorithms for variable D. The lower bounds are for deterministic algorithms in an adversarial setting. For notational ease, we consider time to be starting from 0 in the rest of the chapter.

8.3 Lower Bounds for uniform delay D

In this section we improve the lower bounds for a single class of traffic, where all data are allowed to be delayed by D. First we show an improved bound for $D = 1$ for ease of exposition. We then generalize this to show improved lower bound for $D \in \{1, 2, 3, 4\}$, which exceeds the best known lower bound for any D that was achieved in prior work by $D \to \infty$.

8.3.1 Lower Bound for $D = 1$

In this section, we improve the lower bound of $\frac{18}{13} \approx 1.3846$ for $D = 1$ presented in earlier work [28] to ≈ 1.3982. The upper bound for $D = 1$ is 1.5.

Theorem 23. *For uniform delay $D = 1$, there exists a sequence for which the competitive ratio of any online algorithm is at least* 1.3982*.*

Proof. Suppose that the target competitive ratio of the adversary is α. The adversary will generate some sequence of jobs. At every time slot t, $OPT(t)$ denotes the value of the offline optimal solution for the sequence from time 0 to t. If at any time slot t, the online algorithm serves $\geq \alpha \cdot OPT(t)$ amount of job, then the adversary stops generating any more jobs, since it has forced the competitive ratio of α. Otherwise, it continues till time $k + 3$, where k is a very large integer. Now, we describe the sequence generated by adversary.

Let $x = \frac{\sqrt{13}-1}{2}$. The sequence generated by the adversary is $1, x, x^2, \ldots, x^k, \frac{3}{2}x^k, \frac{3}{2}x^k, 3x^k$. The total amount of job arrived by time slot $k + 3$ is

$\frac{x^{k+1}-1}{x-1} + 6x^k$. The total amount of jobs served by time $k+3$ is $< \alpha \sum_{i=0}^{k+3} OPT(i)$. Obviously $OPT(0) = \frac{1}{2}$.

Proposition 1. *For the given choice of x, the following hold.*

1. $\frac{1+x}{3} > \frac{x}{2}$.
2. $\frac{x^2+x}{3} = \frac{x^2+x+1}{4}$.
3. $\frac{x^i+x^{i-1}+...+x^j}{i-j+2} \geq \frac{x^i+x^{i-1}+...+x^j+x^{j-1}}{i-j+3}$, *for* $i-j \geq 2$.
4. $\frac{\frac{5x}{2}+1}{4} < \frac{5x}{6}$

Lemma 31. *The value of the optimal offline solution at any time slot t is given below:*

1. *For* $0 < t < k+1$, $OPT(t) = \frac{x^t+x^{t-1}}{3}$.
2. $OPT(k+1) = \frac{5x^k}{6}$.
3. $OPT(k+2) = x^k$.
4. $OPT(k+3) = \frac{3}{2}x^k$.

Lemma 31 follows from Proposition 1.

Proof of Theorem 23 continued: We have mentioned already that the work done by the online algorithm has to be $\leq \alpha OPT(t)$ at any time slot t for the competitive ratio to remain below α. However, for feasibility all the work that has arrived needs to be done by their deadline. The total work that remains unserved at the start of time slot $k+4$ is $U(k+4)$, given below.

$$U(k+4) > \frac{x^{k+1}-1}{x-1} + 6x^k - \\ \alpha\left[\frac{1}{6} + \frac{2}{3}\left(\frac{x^{k+1}-1}{x-1}\right) + \frac{3}{2}\frac{x^k}{3} + x^k + \frac{3x^k}{2}\right]$$

We need, $U(k+4) < \alpha\frac{3}{2}x^k$. Hence, $\frac{x^{k+1}-1}{x-1} + 6x^k - \alpha\left[\frac{1}{6} + \frac{2}{3}\left(\frac{x^{k+1}-1}{x-1} + 3x^k\right) + x^k\right] < \alpha\frac{3}{2}x^k$. Dividing by x^k, where $k \to \infty$, we get, $\alpha > \frac{42x-36}{31x-27} > 1.3982$.

This completes the proof of Theorem 23. $\square$

8.3.2 Lower bound for $1 \leq D \leq 4$

Here we generalize the above to hold for any $D \in \{1,2,3,4\}$. The lower bound achieved for $D = 3$ itself exceeds the best known earlier bound for any D from earlier work, that was achieved by $D \to \infty$.

Theorem 24. *For $1 \leq D \leq 4$, there exists a sequence for which the competitive ratio of any online algorithm is at least $F(D)$ given below:*

$$F(D) \geq \frac{(8D^2+10D+3)x - (8D^2+8D+2)}{(5D^2+7.5D+3)x - (5D^2+6.5D+2)}$$

where x is the root of the equation: $x^{D+1} + x^D + \ldots + x = 2D+1$.

Proof. We choose x to be the root of the equation: $x^{D+1}+x^D+\ldots+x = 2D+1$. The sequence generated is as follows: $1, x, x^2, \ldots, x^k, \frac{2D+1}{D+1}x^k(D+1 \text{ times}), (2D+1)x^k$.

The following inequalities are true for our choice of x.

1. For any $p \leq D$, $x^p + x^{p-1} + \ldots x \leq D + p$.

2. For any $p > D$, $x^p + x^{p-1} + \ldots x \geq p + D + 1$.

The first inequality is obvious as otherwise, $x^{D+1} + x^D + \ldots + x > 2D + 1$. The second inequality follows from the choice of x and the fact that $x^i \geq 1$ for any $i \geq 0$.

We now prove something stronger than the first inequality above.

Proposition 2. *For our choice of x, for any $1 < p \leq D + 1$, $x^p + x^{p-1} + \ldots + x \leq 2p - 1$.*

Proof. It is obvious that the claim holds for $p = D + 1$. Let us now examine the case of $p < D + 1$. Suppose for the sake of contradiction, that the claim is not true. In other words, $x^p + x^{p-1} + \ldots + x > 2p - 1$. Since there are p terms in this summation, on average each term is therefore $> (2 - \frac{1}{p})$. Therefore, $x^p > (2 - \frac{1}{p})$, or $x > (2 - \frac{1}{p})^{1/p}$. Now let us consider the value of the polynomial: $x^{D+1} + x^D + \ldots x^{p+1} + x^p + \ldots + x$. Each term x^{p+i} for $1 \leq i \leq (D - p + 1)$ must be greater than $\geq (2 - \frac{1}{p})^{(1+\frac{1}{p})}$. Now, for $p \geq 4$, $(2 - \frac{1}{p})^{(1+\frac{1}{p})} \geq 2$. For $p = 3$, if $x^3 + x^2 + x > 5$, then $x > 1.278$, and $x^{p+i} > 1.278^3 > 2$. For $p = 2$, if $x^2 + x > 3$, then $x > 1.302$, and $x^{p+i} \geq x^3 > 2$. Therefore the polynomial in all cases evaluates to: $> 2(D - p + 1) + 2p - 1$, i.e., $> 2D + 1$ This is a contradiction, as according to our choice of x, $x^{D+1} + \ldots + x = 2D + 1$. This completes the proof for all $1 < p \leq D + 1$. □

Proposition 3. *For our choice of x, for any $1 \leq p \leq D + 1$, the following holds: $x^p + x^{p-1} + \ldots + x \leq p \cdot (\frac{2D+1}{D+1})$.*

Proof. The function $\frac{2y+1}{y+1}$ is an increasing function of y. Hence, $p \cdot \frac{2D+1}{D+1} \geq p \cdot \frac{2p-1}{p} = 2p - 1$. From Proposition 2, we know that for $1 < p \leq (D+1)$, $x^p + x^{p-1} + \ldots + x \leq 2p - 1$. Therefore, for $1 < p \leq (D + 1)$, $x^p + x^{p-1} + \ldots + x \leq p \cdot (\frac{2D+1}{D+1})$.

For $p = 1$, we claim $x \leq \frac{2D+1}{D+1}$. Suppose this is not true for the sake of contradiction, and $x > \frac{2D+1}{D+1}$. Since $\frac{2y+1}{y+1}$ is an increasing function of y, and $D \geq 1$, $x > \frac{3}{2}$. However, from Proposition 2, we know that $x^2 + x \leq 3$, hence $x \leq 1.302$. This is a contradiction. □

Proposition 4. *For our choice of x, for $1 \leq p < D$*

$p \cdot \left(\frac{2D+1}{D+1}\right) x^{D-p} + x^{D-p} + x^{D-p-1} + \ldots + x < 2D.$

The above proposition also implies, $p\frac{2D+1}{D+1}x^{D-p} + x^{D-p} + \ldots x^q < 2D - q$ for any $q \geq 0$.

Lemma 32. *The optimal offline solution at time t, $OPT(t)$ is as follows:*

1. $\frac{x^t + x^{t-1} + \ldots + x^{t-D}}{2D+1}$, *for $D \leq t \leq k$.*
2. $\frac{i \cdot \frac{2D+1}{D+1} x^k + x^k + \ldots x^{k-D+i}}{2D+1}$, *for $t = k + i$, $0 < i \leq D$.*
3. x^k *at $t = k + D + 1$.*
4. $\frac{2D+1}{D+1}x^k$, *for $k + D + 1 < t \leq k + 2D + 1$*

Proof. The optimal offline solution at any time t, is the highest density of work over all times 0 to t as follows from the work of Yao et al. [75]. Hence the above lemma follows from the inequalities presented earlier and Propositions 2, 3 and 4. □

Proof of Theorem 24 continued: Continuing the analysis as before and using Lemma 32, the largest competitive ratio that can be forced by the adversary is α

given below.

$$\begin{aligned}
\alpha &\geq \frac{\frac{x^{k+1}-1}{x-1} + 2(2D+1)x^k}{\frac{D+1}{2D+1}(\frac{x^{k+1}-1}{x-1}) + \frac{2D+1}{D+1}x^k \frac{\sum_{i=1}^{D+1} i}{2D+1} + (2D+1)x^k} \\
&\geq \frac{(2D+1)\left(\frac{x}{x-1} + 2(2D+1)\right)}{(D+1)\frac{x}{x-1} + \frac{(D+2)(2D+1)}{2} + (2D+1)^2} \\
&= \frac{(8D^2+10D+3)x - (8D^2+8D+2)}{(5D^2+7.5D+3)x - (5D^2+6.5D+2)}
\end{aligned}$$

This improves the lower bound known in prior work for any value of $1 \leq D \leq 4$. More importantly, the lower bound for $D = 3$, is already greater than the best known prior lower bound of 1.53 for any D. The table below lists the new lower bounds. □

Table 8.1: New Lower Bounds

D value	New Lower Bound
1	1.3982
2	1.4958
3	1.5397
4	1.5646

8.4 Lower Bound for variable delay

In this section, we prove lower bounds on the minimum competitive ratio any online algorithm can achieve in an adversarial setting for the case where different jobs are allowed different amounts of delay. We first give specific lower bounds for small values of maximum delay D_{max} for any packet, specifically, for $D_{max} \in [1, \ldots, 7]$. Then we give a general lower bound for any D.

8.4.1 Lower Bound for $D \in \{0, 1\}$

Theorem 25. *For $D \in \{0, 1\}$, there exists a sequence for which the competitive ratio of any online algorithm is at least $\frac{3}{2}$.*

Proof. Let us assume for the sake of contradiction that there exists an online algorithm with competitive ratio $\frac{3}{2}(1-\epsilon)$. Let the adversary generate a sequence from time slot 0 to k: $1, 2, 2^2, \ldots, 2^k$, with every job having $D = 1$. At time slot $k+1$, it sends 2^k, this time with $D = 0$. Since the algorithm has a competitive ratio $\frac{3}{2}(1-\epsilon)$, at every time slot i, the total work done by the algorithm is $\leq \frac{3}{2}(1-\epsilon)OPT(i)$ where $OPT(i)$ is the value of the optimal offline solution for the sequence from time slot 0 to i. It can be easily seen that at any time slot i, $OPT(i) = 2^{i-1}$. The total amount of work that has arrived since time slot k (inclusive) is $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$. Hence, the total work that remains unserved at the end of time slot k is $U(k+1) \geq 2^{k+1} - 1 - \frac{3}{2}(1-\epsilon)\sum_{j=0}^{k-1} 2^j = 2^{k+1} - \frac{3}{2}(1-\epsilon)(2^k - 1)$. Therefore $U(k+1) \geq 2^{k-1}(1+3\epsilon) + \frac{3}{2}(1-\epsilon)$.

At time slot $k+1$, $A(k+1) = 2^k$ with delay $D = 0$. Hence, the optimum solution at $k+1 = 2^k$. However, the algorithm needs to do total work $S(k+1) = U(k+1) + 2^k$. $S(k+1) \geq 2^{k-1}(1+3\epsilon) + \frac{3}{2}(1-\epsilon) + 2^k$. The competitive ratio is

$$\frac{S(k+1)}{OPT(k+1)} \geq \frac{1+3\epsilon}{2} + \frac{\frac{3}{2}(1-\epsilon)}{2^k} + 1 \geq \frac{3}{2}(1+\epsilon)$$

This proves the theorem. $\square$

8.4.2 Lower Bound for $D_{max} \in \{2, 3, \ldots, 7\}$.

Theorem 26. *When the delay allowed can vary from $\{0, \ldots, D_{max}\}$, where $2 \leq D_{max} \leq 7$, there exists a sequence for which the competitive ratio of any online algorithm is given below (where $d = D_{max}$ for notational ease)*

$$\alpha \geq \frac{\frac{x^d}{x-1} + dx^{d-1}}{\left(\frac{x^d}{x-1}\right) \cdot \left(\frac{d+1}{2d+1}\right) - \frac{\sum_{i=1}^{d} ix^{i-1}}{2d+1} + \sum_{i=1}^{d} \left(\frac{ix^{d-1} + \sum_{j=i}^{d} x^{j-1}}{2d+1-i}\right)}$$

where $x = 1 + \frac{1}{d}$.

Proof. The sequence generated by the adversary is as follows (provided the online algorithm does $\leq \alpha$ times the optimal offline solution at any time slot t): $1, x, x^2, \ldots, x^k$ from time $t = 0$ to $t = k$. All of these jobs have maximum delay allowed d. For every time slot $t = k + i$, where, $1 \leq i \leq d$, the adversary sends a jobs x^k, with delay allowed $d - i$.

For the choice of x and d, the following facts are true.

1. For any $1 \leq p < d + 1$, $x^p + x^{p-1} + \ldots x \leq d + p$.
2. $x^{d+1} + x^d + \ldots x > 2d + 1$.
3. For $i \in [1 \ldots d]$, $x^{d-i}(d + i + 1) - 3d - 1 + i \leq 0$.
4. For $i \in [1 \ldots d]$, $x^{d-i+1}(d + i + 1) - 3d - 2 + i \geq 0$.

The above facts can be easily verified numerically.

Lemma 33. *The optimal offline solution at any time t is given below.*

1. For $d \leq t \leq k$, $OPT(t) = \frac{x^t + x^{t-1} + \ldots x^{t-d}}{2d+1}$.

2. For $t = k + i$, where $1 \leq i \leq d$,

$OPT(t) = x^{k-d+i} \cdot \left(\frac{ix^{d-i} + \sum_{p=0}^{d-i} x^p}{2d+1-i} \right)$.

Proof. The proof follows from the numerical facts stated earlier for our choice of x and constraints on d. The first two facts imply $OPT(t)$ for $t \leq k$. For $t > k$, the specifically, the following are true, as implied by the facts above, and these in turn imply $OPT(t)$ for $t > k$.

1. $\frac{ix^{d-i+1} + x^{d-i+1} + \ldots + x}{2d+1-i} \geq \frac{ix^{d-i+1} + x^{d-i+1} + \ldots + x + 1}{2d+2-i}$.

2. $\frac{ix^{d-i} + x^{d-i} + \ldots + x}{2d-i} \leq \frac{ix^{d-i} + x^{d-i} + \ldots + x + 1}{2d-i+1}$.

□

Proof of Theorem 26 continued: The total work arrived by the start of time $t = (k + d)$ is $W = \frac{x^{k+1}-1}{x-1} + d \cdot x^k$. For competitive ratio to remain $\leq \alpha$, any online algorithm must serve $\leq \alpha \cdot OPT(t)$ at any time slot t. However, for feasibility, all work needs to be served. Let the total work served by the end of time $t = (k + d)$ be $S \leq \sum_{t=0}^{k+d} \alpha OPT(t)$.

Therefore the competitive ratio that can be forced by the adversary is $\alpha \geq \frac{W}{S}$:

$$\alpha \geq \frac{\frac{x^d}{x-1} + dx^{d-1}}{\left(\frac{x^d}{x-1}\right) \cdot \left(\frac{d+1}{2d+1}\right) - \frac{\sum_{i=1}^{d} ix^{i-1}}{2d+1} + \sum_{i=1}^{d} \left(\frac{ix^{d-1} + \sum_{j=i}^{d} x^{j-1}}{2d+1-i} \right)}$$

□

The specific values for $d \in \{1, \ldots, 7\}$ are tabulated below.

Table 8.2: Lower Bounds for Variable $D \in \{0, \dots, d\}, 1 \leq d \leq 7$

d value	Lower Bound
1	1.5
2	1.7045
3	1.8113
4	1.8766
5	1.9206
6	1.9523
7	1.9761

8.4.3 Lower Bound for $D \in \{0, 1, \dots, d\}$.

Theorem 27. *For $D \in \{0, 1, \dots d\}$, there exists a sequence for which the competitive ratio of any online algorithm is at least $2\left(1 - \frac{d^2+1}{d^3+3d^2+2d+2}\right)$. For large d, this approaches 2.*

Proof. Set $x = d + 1$. Let the sequence be a $1, x, x^2, \dots, x^k$, all with D values $= d$. This is followed by d more jobs of size x^k, with their D values progressively decreasing, specifically the job arriving at $t = k + i$, where $1 \leq i \leq d$, has delay $d - i$. The total work arrived till time $t = k + d$ is $W = \frac{x^{k+1-1}}{x-1} + dx^k$. The total work served by the online algorithm at any time t is $< \alpha OPT(t)$. From time 0 to k, $OPT(t) = \frac{x^t}{d+1}$. From time $k + 1$ to $k + d$, $OPT(k + j) = \frac{jx^k}{d+1}$. hence the total work S served by the end of time $k+d$ is

$$\begin{aligned} S &\leq \alpha \left(\frac{x^{k+1}-1}{x-1} \cdot \frac{1}{d+1} + \sum_{i=2}^{d+1} \frac{i \cdot x^k}{d+1} \right) \\ &\leq \frac{\alpha}{d+1} \left(\frac{x^{k+1}-1}{x-1} + \frac{d^2+3d}{2} \right) \end{aligned}$$

Hence, it can be seen that the competitive ratio α can be enforced by the adversary to be as large as

$$\alpha > \frac{W}{S} \geq \frac{2d^3+4d^2+4d+2}{d^3+3d^2+2d+2} = 2\left(1 - \frac{d^2+1}{d^3+3d^2+2d+2}\right)$$

For large d this approaches 2. Note however, $N \gg d$. $\square$

When time is not slotted such that the release times of jobs can be arbitrary, as well as the delays allowed can be arbitrarily large, specifically $d = N-1$, a lower bound of e is implied by the work of Bansal, Kimbrel and Pruhs [2]. They also give a tight e competitive algorithm for this problem.

8.5 Online Algorithms for variable D

In this section we describe and analyze online algorithms for multi-class traffic, where the delay allowed for different data packets may be different. We first show that the natural extension of the Equal Split algorithm suggested in prior work [28] can perform very badly.

8.5.1 Performance of Equal Split when $D \in \{0, 1, \ldots, D_{max}\}$.

We first describe the natural extension of the Equal Split algorithm for this case. When D is uniform for all jobs, the algorithm simply splits every job equally over its allowed duration $D+1$. Let $A(i)$ denote the job arrival at any time slot i. At time slot t, the total work done by the algorithm is $\frac{\sum_{i=t-D}^{t} A(i)}{D+1}$. This achieves a competitive ratio of $\frac{2D+1}{D+1}$ for uniform D.

For variable D, an extension of the Equal Split algorithm would again split every job over its allowed duration, except that this duration is now job dependent. Let the job arrivals at time slot i be denoted by the vector $\boldsymbol{A}(i)$ where $A_d(i)$ denotes the jobs with allowed delay d, $d \in \{0, \ldots, D_{max}\}$. Each job $A_d(i)$ would be split equally over its duration $d+1$ time slots. Therefore the total work done by the algorithm at a time t is

$ES(t) = A_0(t) + \frac{A_1(t)+A_1(t-1)}{2} + \ldots + \frac{\sum_{i=t-D_{max}}^{t} A_{D_{max}}(i)}{D_{max}+1}$.

Let us consider a job arrival sequence as follows: at time slot i, $i \in \{0, \ldots, D_{max}-1\}$, a job of size 1 arrives with allowed delay $D_{max}-i$. At time slot D_{max}, $\boldsymbol{A}(D_{max}) = (1, 1, \ldots, 1)$. In other words, $D_{max}+1$ jobs arrive at time D_{max}, each of size 1 and every job has a different delay in $\{0, \ldots, D_{max}\}$. Clearly, the optimal offline solution is to finish every job that arrived before D_{max} in its arrival slot, and then each of the jobs that arrived in D_{max} one by one in the intervals D_{max} to $2D_{max}$. Hence the value of the optimal offline solution is 1. However, in the time slot D_{max}, Equal Split would end up doing $\frac{1}{D_{max}+1} + \frac{1}{D_{max}} + \ldots + \frac{1}{2} + 1 + \frac{1}{2} + \ldots + \frac{1}{D_{max}+1}$ amount of work. Therefore, the competitive ratio of Equal Split is: $\sum_{i=1}^{D_{max}} \frac{2}{1+i} + 1 = 2H_{D_{max}+1} - 1$, where H_n denotes the n^{th} Harmonic number. For $D_{max} = 1$, the competitive ratio of Equal Split is 2. For large D_{max}, the competitive ratio of Equal Split approaches $2 \ln (D_{max} + 1) - 1$.

8.5.2 Tight Online Algorithm for $D \in \{0, 1\}$

We have already shown that the competitive ratio of any online algorithm in an adversarial setting must be at least $\frac{3}{2}$, when $D \in \{0, 1\}$. Here we give an algorithm (Algorithm 7) with competitive ratio exactly $\frac{3}{2}$.

Algorithm 7 Algorithm for $D \in \{0, 1\}$.

```
1: At time slot $t$, let $\boldsymbol{A}(t)$ be the newly arrived job vector and $p(t)$ is the pending job.
2: Compute $OPT'(t)$ as max $(A_0(t), \frac{A_0(t)+A_1(t)}{2})$.
3: Set $S(t) = p(t) + OPT'(t)$.
4: Serve up to $S(t)$ jobs with earliest deadline first.
```

Lemma 34. *The solution returned by Algorithm 7 is feasible.*

Proof. This is obvious, since at any time slot, we serve all of the pending jobs and all jobs with delay $D = 0$. □

Lemma 35. *At any time slot t, the amount of pending job $p(t) \leq \frac{A_0(t-1)+A_1(t-1)}{2}$.*

Proof. This follows since at time slot $t-1$, we serve at least $\frac{A_0(t-1)+A_1(t-1)}{2}$ of $A_0(t-1) + A_1(t-1)$. □

Theorem 28. *Algorithm 7 is $\frac{3}{2}$ competitive in an adversarial setting with respect to any optimal offline solution for any sequence of job arrivals I.*

Proof. For a given sequence I, let $OPT(t)$ be the optimal solution for the sequence up to time t. It is obvious that $OPT(t)$ is a non-decreasing function of t. We will prove that for any time slot t, the amount of jobs that Algorithm 7 serves $S(t) \leq \frac{3}{2}OPT(t)$. We know from Lemma 35 that $p(t) \leq \frac{A_0(t-1)+A_1(t-1)}{2}$. Therefore, $S(t) \leq \frac{A_0(t-1)+A_1(t-1)}{2} + OPT'(t)$.

Case 1: $OPT'(t) = A_0(t)$.

In this case, $S(t) = \frac{A_0(t-1)+A_1(t-1)}{2} + A_0(t)$.

Case 1.1: $A_0(t) < A_0(t-1) + A_1(t-1)$. In this case it can be seen that $\frac{A_0(t-1)+A_1(t-1)+A_0(t)}{2} > A_0(t)$. Hence, $OPT(t) \geq \frac{A_0(t-1)+A_1(t-1)+A_0(t)}{2}$. Therefore $\frac{S(t)}{OPT(t)} \leq \frac{A_0(t-1)+A_1(t-1)+2A_0(t)}{A_0(t-1)+A_1(t-1)+A_0(t)}$.

$$\frac{S(t)}{OPT(t)} \leq 1 + \frac{1}{1 + \frac{A_0(t-1)+A_1(t-1)}{A_0(t)}} \qquad \leq \frac{3}{2}$$

Case 1.2: $A_0(t) \geq A_0(t-1) + A_1(t-1)$.

In this case it can be seen that $\frac{A_0(t-1)+A_1(t-1)+A_0(t)}{2} \leq A_0(t)$. Hence, $OPT(t) \geq A_0(t)$.

$$\frac{S(t)}{OPT(t)} \leq \frac{\frac{A_0(t-1)+A_1(t-1)}{2} + A_0(t)}{A_0(t)} \qquad \leq \frac{3}{2}$$

Case 2: $OPT(t') = \frac{A_0(t)+A_1(t)}{2}$.

Now, we know, $OPT(t) \geq \frac{A_0(t-1)+A_1(t-1)+A_0(t)+A_1(t)}{3}$. $S(t) \leq \frac{A_0(t-1)+A_1(t-1)}{2} + \frac{A_0(t)+A_1(t)}{2}$.

$$\frac{S(t)}{OPT(t)} \leq \frac{\frac{A_0(t-1)+A_1(t-1)+A_0(t)+A_1(t)}{2}}{\frac{A_0(t-1)+A_1(t-1)+A_0(t)+A_1(t)}{3}} \qquad = \frac{3}{2}$$

Hence, we have proved that at for any sequence I, at any time slot t, the amount of job served by Algorithm 7 is at most $\frac{3}{2}$ of the optimum solution. This proves the theorem. □

8.5.3 Online Algorithm for $D \in \{0, 1, 2\}$

In this section, we give an online algorithm with competitive ratio 2 for $D \in \{0, 1, 2\}$. Equal Split on the other hand has a competitive ratio that is at least as bad as $2\frac{2}{3} = \frac{8}{3}$. The algorithm is given in Algorithm 8.

Algorithm 8 Algorithm for $D \in \{0, 1, 2\}$.

```
1: At time slot $t$, let $\boldsymbol{A}(t)$ be the newly arrived job vector.
2: Compute $OPT(t)$ as the offline optimal solution for the arrivals up to time $t$.
3: Set $S(t) = 2 \cdot OPT(t)$.
4: Serve up to $S(t)$ jobs with earliest deadline first.
```

Theorem 29. *Algorithm 8 achieves a competitive ratio of* 2 *with respect to any optimal offline solution.*

Proof. The competitive ratio of Algorithm 8 is at least 2 since it does twice of the work the optimal offline solution does at any time slot. We will show that the solution returned by Algorithm 8 is always feasible, in other words, all the work gets completed before their allowed delay expires. At any time i, the optimal offline solution is $OPT(i) \geq \frac{A_0(i-1)+A_1(i-1)+A_2(i-1)+A_0(i)+A_1(i)+A_2(i)}{4}$. Therefore, by the end of time slot $t-1$, the total work completed by the algorithm is $2\sum_{i=0}^{t-1} OPT(i) \geq \left(\sum_{i-1}^{t-2}\sum_{d=0}^{2} A_d(i)\right) + \left(\frac{A_0(t-1)+A_1(t-1)+A_2(t-1)}{2}\right)$. We need to complete $\sum_{i-1}^{t-2}\sum_{d=0}^{2}(A_d(i)) + (A_0(t-1) + A_1(t-1)) + A_0(t)$ by time t. The optimal solution at time t, $OPT(t) \geq \frac{A_0(t-1)+A_1(t-1)+A_0(t)}{2}$. Therefore, by doing $2 \cdot OPT(t)$ we can finish all the work that needs to be completed by time slot t. Since this is true for any time slot, we have proved that the competitive ratio of Algorithm 8 is at most 2. □

Note that this is strictly better than Equal Split as well as the e competitive online algorithm for unslotted time slots that was given by Bansal et al. [2].

Chapter 9: Conclusion

In this dissertation, we have studied problems inspired by the needs to reduce energy costs and network bandwidth billing costs in data centers. As already outlined in Chapter 1, energy and network bandwidth are two resources which account for a major fraction of the huge operating costs of data centers. Reducing energy consumption is an important problem as it will be directly beneficial to the environment. Intelligent optimization practices which can reduce the consumption of resources without violating the service level agreements of the data centers with their clients, and at the same time, do not require a complete overhaul of the existing systems, may help tremendously in reducing the costs.

In this dissertation we have tried doing the above, by modeling these problems as resource allocation problems and providing optimization algorithms with provable guarantees on these models. To recapitulate, the specific problems that we study are the following: (1) optimizing the energy consumption due to cooling machines and racks in data centers assuming a standard raised floor cooling technology, (2) reducing the energy costs due to equipment powering of the servers or machines by effective batching of jobs respecting job requirements and machine capacities, and finally, (3) reducing the network bandwidth billing costs under the percentile model and the peak bandwidth model incurred by data centers due to the large volume of data transmitted daily. Traditional models do not accurately capture the essential complexities of these resource allocation problems arising from the new technology and practices deployed in today's world. Hence, studying them required us to study new scheduling and packing problems that generalize classical scheduling and packing problems. It is not obvious how to extend existing techniques on classical problems trivially to these new problems.

In our work, we have shown how some of the models and problems we study generalize classical problems. Our models capture effectively, with some degree of abstraction, the observations of empirical and physical studies conducted in systems and networking literature. For these new models, we have designed and analyzed algorithms with worst case asymptotic performance guarantees. Some of the other problems we consider have already been studied in the scheduling theory and algorithms literature. For such problems, we have improved the existing results and given new results on further generalizations of the problems.

Our results include design and analysis of offline algorithms, both approximate and exact, and online algorithms along with new and improved lower bounds for the problems we consider. We have used discrete optimization techniques like combinatorial optimization, LP rounding and competitive analysis in our work. However, in the dissertation, we have only studied deterministic algorithms and analyzed their asymptotic worst case performance. Use of randomization may help in improving the performance guarantees significantly. Furthermore, given the huge volume of historical data available, one can derive reasonable stochastic assumptions on the data packets or the incoming job characteristics. Knowing the stochastic nature of the input may help in designing customized algorithms with provably better performance guarantees, either in expectation or with a high probability. In the near future, we would like to extend this work in the above directions.

The models we study generalize the traditional models elegantly, and in a manner which is more relevant to modern day resource allocation problems. These models can perhaps be generalized even further to model the intricacies of the real world problems. For example, one can consider assignment restrictions being specified as a part of the job characteristics in the thermal scheduling model. This would generalize the Generalized Assignment Problem, in the presence of spatial cross-interference. There can be several other generalizations of our models based on the issues and limitations faced by real world systems. For example, sometimes the job characteristics cannot be estimated completely before assigning them, or perhaps, the spatial cross interference factor needs to take into account the actual distance between two neighboring machines, instead of considering it to be unit.

It would be interesting to see if our techniques can extend smoothly to further generalizations of our models, or if new techniques would be needed to solve them. It is not clear if such generalizations would even be tractable, in other words, admit any polynomial time approximation algorithms with constant approximation ratio. Specifically, it would be interesting not only to provide upper bounds on these generalizations, but also to analyze the lower bounds on the performance guarantees.

We have considered some natural optimization questions on the models we study. However, one can ask several other optimization questions on these models.

For example, even if we know that no deterministic online algorithm can achieve bounded competitive ratio in the percentile model (as shown by Golubchik et al. [28]), it would be interesting to study bicriteria optimization problems. A natural bicriteria optimization can be defined on a partial version of the problem. Suppose we need to send only a certain fraction of the data packets, say, 90% of the data. Can we minimize the percentile cost compared to an optimal solution? In fact, both the offline and online versions of this problem would be interesting. We may also consider resource augmentation for achieving bounded competitive ratio for online algorithms in the percentile model.

Studying any of these open questions would be interesting and practically relevant. Moreover, in some cases we have shown gaps exist between the lower bounds and the upper bounds that either we have provided, or that is known from existing results in the literature. Closing these gaps would be very interesting as well. We would like to study some of the open problems and close some gaps that we have highlighted and we hope that the open questions raised by our work will inspire even more interesting algorithmic work in the near future.

Bibliography

[1] Mansoor Alicherry and Randeep Bhatia. Line system design and a generalized coloring problem. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA)*, pages 19–30, 2003.

[2] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1):3.1–3.39, 2007.

[3] Nikhil Bansal and Kirk Pruhs. Speed scaling to manage temperature. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 460–471, 2005.

[4] Philippe Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2(6):245–252, 1999.

[5] Christian L. Belady. In the data center, power and cooling costs more than the it equipment it supports. *Electronics cooling*, 13(1):24, 2007.

[6] Jessica Chang. Energy-aware batch scheduling. *Ph.D. Thesis, Computer Science and Engineering, University of Washington*, 2013.

[7] Jessica Chang, Harold Gabow, and Samir Khuller. A model for minimizing active processor time. In *Proceedings of the 20th European Symposium on Algorithms (ESA)*, pages 289–300, 2012.

[8] Jessica Chang, Samir Khuller, and Koyel Mukherjee. Lp rounding and combinatorial algorithms for minimizing active and busy time, 2013. submitted.

[9] Moses Charikar and Samir Khuller. A robust maximum completion time measure for scheduling. In *Proceedings of the 17th ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 324–333, 2006.

[10] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

[11] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Henrdrik Hamann, Alan Weger, and Pradip Bose. Thermal-aware task scheduling at the system software level. In *Proceedings of the International symposium on Low power electronics and design*, pages 213–218, 2007.

[12] Marek Chrobak, Christoph Durr, Mathilde Hurand, and Julien Robert. Algorithms for temperature-aware task scheduling in microprocessor systems. *Sustainable Computing: Informatics and Systems*, 1(3):241 – 247, 2011.

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2001.

[14] Ayse Kivilcim Coskun, Jose L. Ayala, David Atienza, Tajana Simunic Rosing, and Yusuf Leblebici. Dynamic thermal management in 3d multicore architectures. In *Proceedings of the Design, Automation and Test in Europe (DATE)*, pages 1410–1415, 2009.

[15] Xenofontas Dimitropoulos, Paul Hurley, Andreas Kind, and Marc Ph. Stoecklin. On the 95-percentile billing method. In *Proceedings of the 10th Passive and Active Network Measurement conference (PAM)*, pages 207–216, 2009.

[16] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system power analysis and modeling for server environments. In *Proceedings of the Workshop on Modeling, Benchmarking, and Simulation*, pages 70–77, 2006.

[17] Matthias Englert and Matthias Westermann. Considering suppressed packets improves buffer management in qos switches. In *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 209–218, 2007.

[18] Ulrich Faigle, R. Garbe, and Walter Kern. Randomized online algorithms for maximizing busy time interval scheduling. *Computing*, 56(2):95–104, 1996.

[19] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News*, 35(2):13–23, 2007.

[20] U.S. Department of Energy Federal Energy Management Program. Data center energy consumption trends. `http://www1.eere.energy.gov/femp/program/dc_energy_consumption.html`, 2013. Last accessed October 24, 2013.

[21] Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Thermal-aware global real-time scheduling on multicore systems. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 131–140, 2009.

[22] Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Hadas Shachnai, Mordechai Shalom, Tami Tamir, and Shmuel Zaks. Minimizing total busy

time in parallel scheduling with application to optical networks. In *Proceedings of the IEEE 23rd International Parallel and Distributed Processing Symposium*, pages 1–12, 2009.

[23] Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Approximating the traffic grooming problem with respect to adms and oadms. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, pages 920–929, 2008.

[24] Michele Flammini, Gianpiero Monaco, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Optimizing regenerator cost in traffic grooming. *Theoretical Computer Science*, 412(52):7109–7121, 2011.

[25] Michele Flammini, Luca Moscardelli, Mordechai Shalom, and Shmuel Zaks. Approximating the traffic grooming problem. In *Proceedings of the 16th international conference on Algorithms and Computation (ISAAC)*, pages 915–924, 2005.

[26] Yang Ge, Parth Malani, and Qinru Qiu. Distributed task migration for thermal management in many-core systems. In *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC)*, pages 579–584, 2010.

[27] David Goldenberg, Lili Qiuy, Haiyong Xie, Yang Richard Yang, and Yin Zhang. Optimizing cost and performance for multihoming. *ACM SIGCOMM Computer Communication Review*, 34(4):79–92, 2004.

[28] Leana Golubchik, Samir Khuller, Koyel Mukherjee, and Yuan Yao. To send or not to send: Reducing the cost of data transmission. In *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM)*, pages 2472–2478, 2013.

[29] Mohamed Gomaa, Michael D. Powell, and T.N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. *ACM SIGARCH Computer Architecture News*, 32(5):260–270, 2004.

[30] Ronald Graham. Bounds for certain multiprocessing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

[31] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2009.

[32] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Danny Segev. Scheduling with outliers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX / RANDOM)*, pages 149–162, 2009.

[33] Magnus K. Herrlin and Christian L. Belady. Gravity-assisted air mixing in data centers and how it affects the rack cooling effectiveness. In *Proceedings of the 10th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronics Systems (ITHERM)*, pages 434–438, 2006.

[34] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms (TALG)*, 3(4):41.1–41.23, 2007.

[35] Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 39(4):1392–1412, 2009.

[36] Lukasz Jez. A 4/3-competitive randomised algorithm for online packet scheduling with agreeable deadlines. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 489–500, 2010.

[37] Lukasz Jez. One to rule them all: A general randomized algorithm for buffer management with bounded delay. In *Proceedings of the 19th European Symposium on Algorithms (ESA)*, pages 239–250, 2011.

[38] Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in qos switches. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, pages 520–529, 2001.

[39] Rohit Khandekar, Baruch Schieber, Hadas Shachnai, and Tami Tamir. Minimizing busy time in multiple machine real-time scheduling. In *Proceedings of the 30th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 169 – 180, 2010.

[40] Samir Khuller and Koyel Mukherjee. Buffer management for multi-class traffic, 2013. submitted.

[41] Frederic Koehler and Samir Khuller. Optimal batch schedules for parallel machines. In *Proceedings of the 13th Algorithms and Data Structures Symposium (WADS)*, 2013.

[42] Vijay Kumar and Atri Rudra. Approximation algorithms for wavelength assignment. In *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 152–163, 2005.

[43] Eren Kursun, Chen-Yong Cher, Alper Buyuktosunoglu, and Pradip Bose. Investigating the effects of task scheduling on thermal behavior. In *Third Workshop on Temperature-Aware Computer Systems (TACS)*, 2006.

[44] Nikolaos Laoutaris, Georgios Smaragdakis, Pablo Rodriguez, and Ravi Sundaram. Delay tolerant bulk data transfers on the internet. In *Proceedings of the 11th international joint conference on Measurement and modeling of computer systems*, pages 229–238, 2009.

[45] Fei Li. Competitive scheduling of packets with hard deadlines in a finite capacity queue. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1062–1070, 2009.

[46] Fei Li, Jay Sethuraman, and Clifford Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proceedings of the 16th annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 801–802, 2005.

[47] Fei Li, Jay Sethuraman, and Clifford Stein. Better online buffer management. In *Proceedings of the 18th ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 199–208, 2007.

[48] Jiayin Li, Meikang Qiu, Jingtong Hu, and Edwin H.-M Sha. Thermal-aware rotation scheduling for 3d multi-core with timing constraint. In *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS)*, pages 323–326, 2010.

[49] Shaobo Liu, Jingyi Zhang, Qing Wu, and Qinru Qiu. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *Proceedings of the 11th International Symposium on Quality Electronic Design (ISQED)*, pages 390–398, 2010.

[50] Gabriel H Loh, Yuan Xie, and Bryan Black. Processor design in 3d die-stacking technologies. *IEEE Micro*, 27(3):31–48, 2007.

[51] George B. Mertzios, Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, and Shmuel Zaks. Optimizing busy time on parallel machines. In *Proceedings of the IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 238–248, 2012.

[52] Microsoft. The pattern of overbuilding: Server utilization and energy productivity for green it. `http://www.triplepundit.com/2011/07/overbuilding-server-utilization-and-energy-productivity`, 2011. Last accessed October 24, 2013.

[53] Rich Miller. How many data centers? emerson says 500,000. `http://www.datacenterknowledge.com/archives/2011/12/14/how-many-data-centers-emerson-says-500000`, 2011. Last accessed October 24, 2013.

[54] Justin D. Moore, Jeffrey S. Chase, Parthasarathy Ranganathan, and Ratnesh K. Sharma. Making scheduling cool: Temperature-aware workload placement in data centers. In *Proceedings of the USENIX annual technical conference, General Track*, pages 61–75, 2005.

[55] Koyel Mukherjee, Samir Khuller, and Amol Deshpande. Saving on cooling: the thermal scheduling problem. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):397–398, 2012.

[56] Koyel Mukherjee, Samir Khuller, and Amol Deshpande. Algorithms for the thermal scheduling problem. In *Proceedings of the IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 949–960, 2013.

[57] Tridib Mukherjee, Ayan Banerjee, Georgios Varsamopoulos, Sandeep K.S. Gupta, and Sanjay Rungta. Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers. *Computer Networks*, 53(17):2888–2904, 2009.

[58] Thyaga Nandagopal and Krishna P. N. Puttaswamy. Lowering inter-datacenter bandwidth costs via bulk data scheduling. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 244–251, 2012.

[59] Emerson network power. State of the data center 2011. `http://www.emersonnetworkpower.com/en-US/About/NewsRoom/pages/2011DataCenterState.aspx`, 2011. Last accessed October 24, 2013.

[60] Ehsan Pakbaznia and Massoud Pedram. Minimizing data center cooling and server power costs. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 145–150, 2009.

[61] Michael Pinedo. *Scheduling: Theory, Algorithms and Systems.* Prentice Hall, second edition, 2006.

[62] U.S. Department of Energy Prepared by the National Renewable Energy Laboratory (NREL) for the Federal Energy Management Program. Best practices guide for energy-efficient data center design. `www1.eere.energy.gov/femp/pdfs/eedatacenterbestpractices.pdf`, 2011. Last accessed October 24, 2013.

[63] Roger Schmidt and Ethan Cruz. Raised floor computer data center: effect on rack inlet temperatures of chilled air exiting both the hot and cold aisles. In *Proceedings of the 8th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM)*, pages 580–594, 2002.

[64] Mordechai Shalom, Ariella Voloshin, Prudence W.H. Wong, Fencol C.C. Yung, and Shmuel Zaks. Online optimization of busy time on parallel machines. *Theory and Applications of Models of Computation. Lecture notes in Computer Science*, 7287:448–460, 2012.

[65] Bing Shi and Ankur Srivastava. Thermal and power-aware task scheduling for hadoop based storage centric datacenters. In *Proceedings of the International Green Computing Conference (greencomp)*, pages 73–83, 2010.

[66] Kevin Skadron, Tarek Abdelzaher, and Mircea R. Stan. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal

management. In *Proceedings of the 8th International Symposium on High Performance Computer Architecture (HPCA)*, pages 17–28, 2002.

[67] Qinghui Tang, Sandeep K.S. Gupta, Daniel Stanzione, and Phil Cayton. Thermal-aware task scheduling to minimize energy usage of blade server based datacenters. In *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 195–202, 2006.

[68] Qinghui Tang, Sandeep K.S. Gupta, and Georgios Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster)*, pages 129–138, 2007.

[69] Qinghui Tang, Sandeep K.S. Gupta, and Georgios Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Transactions on Parallel and Distributed Systems*, 19(11):1458–1472, 2008.

[70] Qinghui Tang, Tridib Mukherjee, Sandeep K.S. Gupta, and Phil Cayton. Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters. In *Proceedings of the 4th International Conference on Intelligent Sensing and Information Processing (ICISIP)*, pages 203–208, 2006.

[71] Patrick Thibodeau. Data centers, under strain, expand at furious pace. `http://www.computerworld.com/s/article/9216841/Data_centers_under_strain_expand_at_furious_pace_`, 2011. Last accessed October 24, 2013.

[72] ENERGY STAR Program United States (U.S.) Environmental Protection Agency (EPA). Report to congress on server and data center energy efficiency, public law 109-431. `http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf`, 2007. Last accessed: October 24, 2013.

[73] Georgios Varsamopoulos, Ayan Banerjee, and Sandeep K.S. Gupta. Energy efficiency of thermal-aware job scheduling algorithms under various cooling models. *Contemporary Computing*, 20:568–580, 2009.

[74] Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 830 – 831, 2003.

[75] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

[76] Yuan Yao. Qos-aware algorithm design for distributed systems. *Ph.D. Thesis, Electrical Engineering, University of Southern California*, 2012.

[77] Sushu Zhang and Karam S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *Proceedings of the IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 281–288, 2007.

[78] Xiuyi Zhou, Jun Yang, Yi Xu, Youtao Zhang, and Jianhua Zhao. Thermal-aware task scheduling for 3d multicore processors. *IEEE Transactions on Parallel and Distributed Systems*, 21(1):60–71, 2010.