DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Algorithms for Dynamic Right-Sizing in Data Centers

Kevin Kappelmann

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Algorithms for Dynamic Right-Sizing in Data Centers

Algorithmen für dynamische Kapazitätsanpassungen in Datenzentren

Author:	Kevin Kappelmann
Supervisor:	Prof. Dr. Susanne Albers
Advisor:	Jens Quedenfeld
Submission Date:	July 27, 2017

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, July 27, 2017 Kevin Kappelmann

Acknowledgments

I would like to thank my supervisor Prof. Dr. Albers for giving me the chance to write my thesis at her chair and supervising me open-heartedly. Moreover, I want to thank my advisor Jens Quedenfeld for generously taking his time to answer my questions, checking my proofs, and giving me great suggestions for improvements.

I would also like to thank my family: my sister, Janet Kappelmann, and my parents, Erika and Hans-Jürgen Kappelmann, for their unconditional love, trust, and support. Thank you for believing in me.

Finally, since this thesis constitutes the finishing line of my life in Munich for the time being, I want to express my gratitude towards my friends who made the last three years not only an important part of my academic development but also of my personal life. Thank you for your shared time and happiness. I wish you all the best, and I am going to miss you.

Abstract

A significant fraction of modern data centers' energy costs is due to their high energy consumption during periods of low load. One possible solution to deal with this waste of energy is to dynamically adjust the number of active servers and to efficiently distribute the varying workloads, that is to dynamically right-size the data center. In this thesis, we derive algorithms capable of finding approximative as well as optimal solutions for this dynamic right-sizing process by reducing the data centers' scheduling problem to the shortest path problem of acyclic graphs. All developed algorithms are proved for correctness and made accessible by using pseudo-code that can be easily transferred to arbitrary programming languages.

Contents

1 Introduction

1.1 Motivation

Data centers are a centerpiece to deal with the IT industry's fast-growing demand for data collection, processing, and storage [3] and hence become an ever-increasing part of modern companies' infrastructure and thereby budget. One major and increasingly more important part of data centers' budget are energy costs [5]. The question of how to reduce these costs is thus of great interest.

One large fraction of data centers' energy costs is due to modern servers' disproportionate energy consumption to varying utilization levels, i.e. assigned loads. Even energy-efficient servers still consume almost half their full power when nearly idle [2], wasting valuable energy while doing virtually nothing. One possible solution for this waste of energy, which we will address in this thesis, is to dynamically right-size a data center, that is dynamically adjusting the number of active servers and distributing the current workload by software. Algorithms dealing with this dynamic right-sizing thus face a scheduling problem: They have to decide when to power on machines or move machines to a power-saving mode (e.g. sleep or shut down) during periods of high or low load, respectively; and further, they need to efficiently distribute the workload to the selected number of active servers.

It is common to consider two different variants of such optimization problems. The first variant, called *offline optimization*, assumes that the whole problem data is given from the very beginning. The second variant, called *online optimization*, processes its input piece-by-piece in a serial fashion, without having knowledge about the entire problem data from the very start. Algorithms dealing with these variants are consequently called offline and online algorithms, respectively.

We are not the first to address the described scheduling issue. Progress has been made in terms of offline as well as online variants for data center models with a homogeneous collection of servers. Lin et al. [7] showed that the scheduling problem for this homogeneous case can be modeled as a convex optimization problem, and that the optimal offline solution satisfies a backward recurrence relation. This observation is used to develop a 3-competitive online algorithm based on an analogous forward recurrence relation for which progressively larger convex programs have to be solved at each time step. Bansal et al. [1] provided a different online approach, described in fairly mathematical terminology, which improved the upper bound on the optimal competitive ratio from 3 to 2 while also using a more general model, thereby improving applicability.

All named solutions, however, give rise to two concerns: Firstly, they do not enforce the number of active servers to be integer, which means that their calculated solution does not directly translate to a real-world schedule. Secondly, both works do not provide procedures that can be conveniently implemented as a computer algorithm, but they settle for abstract mathematical procedures[1].

The goal of this thesis is thus to establish and verify new procedures that guarantee integer solutions and can be conveniently implemented as a computer algorithm. Our main results, presented in Chapter 4 and Chapter 5, are an optimal offline algorithm with pseudo-linear time complexity and a $(1 + \varepsilon)$-optimal offline algorithm with linear time complexity.

1.2 Thesis Outline

Before we start with our work, we want to give an overview on the structure and content of this thesis. In Chapter 2, we give a formal definition of the regarded data center model and its associated scheduling problem. The definitions will be given in a fairly general form such that they apply to a majority of real-world data centers. We then proceed to examine the described model in Chapter 3 in order to find useful patterns and properties, which will be used to greatly simplify the problem statement, thereby facilitating the subsequent main work of the thesis.

The main work is divided into two chapters: one dealing with the calculation of optimal solutions, and one focusing on approximative methods. The former, Chapter 4, begins with a reduction of the data centers' scheduling problem to the shortest path problem of an acyclic graph. This shortest path problem can then be solved in pseudo-polynomial-time, and its solution can be used to extract an optimal scheduling strategy for the data center. This first approach is further refined to derive an improved algorithm with pseudo-linear time complexity capable of calculating optimal scheduling strategies.

To further minimize the algorithm's time complexity, we consider approximative solution strategies in Chapter 5. For this, we modify the optimal algorithm's underlying graph such that its shortest path can be calculated in linear time. As a trade-off, the extracted schedule will be an approximation rather than an optimal solution. The first section of Chapter 5 deals with the general idea of this approximative approach and results in a 2-optimal algorithm, that is an algorithm whose calculated solution's cost is at most twice as much as an optimal solution's cost. The second section then generalizes this initial

[1] Lin et al. conducted experiments using their online algorithm, but they did not elaborate on how they implemented their algorithm and solved the occurring convex programs.

approach to derive a $(1+\varepsilon)$-optimal algorithm with linear time complexity that is able to approximate optimal solutions with arbitrary precision.

The thesis finishes with a brief summary of the achieved results as well as some final thoughts about possible future developments and applications of the derived algorithms.

2 Model and Problem Formulation

2.1 Model Description

We examine a scheduling problem that commonly arises in data centers. More specifically, we consider a model consisting of a fixed number of homogeneous servers, denoted by $m \in \mathbb{N}$, and a fixed number of time slots, denoted by $T \in \mathbb{N}$. Each server possesses two power states, to wit: a machine is either powered on (active) or shut down. For notational convenience, given a natural number $n \in \mathbb{N}$, we define the sets $[n]$ and $[n]_0$ as

$$[n] \coloneqq \{1, \dots, n\} \subset \mathbb{N}$$
$$[n]_0 \coloneqq \{0, \dots, n\} \subset \mathbb{N}_0$$

For any time slot $t \in [T]$, we have a *mean arrival rate*, denoted by λ_t, that is the expected load to be processed in time slot t. We assume that each server can handle a load between 0 and 1 in any time slot. The assigned load for server i in time slot t is denoted by $\lambda_{i,t} \in [0, 1]$. Consequently, for any time slot t, we expect an arrival rate between 0 and m, that is $\lambda_t \in [0, m]$; otherwise, the servers would not be able to process the given load in the allotted time.

Naturally, a machine incurs *operating costs* when processing its assigned load, which we specify by $f : [0, 1] \to \mathbb{R}$. The operating cost function f may not exclusively account for energy costs. For example, f may also allow for costs incurred by delays, such as lost revenue caused by users waiting for their responses [6, 7]. We assume that a server does not create any operating costs if it is shut down. Note that $f(0)$ denotes the costs incurred by a server that is idle, not shut down; in other words, $f(0)$ may be non-zero. Further, we expect f to be a *convex* function. We call a function $f : D \to \mathbb{R}$ convex if its domain D (in our case $D = [0, 1]$) is a convex set and f satisfies

$$\forall \lambda_1, \lambda_2 \in D, \mu \in [0, 1] : f(\mu\lambda_1 + (1 - \mu)\lambda_2) \leq \mu f(\lambda_1) + (1 - \mu)f(\lambda_2) \tag{2.1}$$

This might at first seem like a notable restriction, but it still allows to capture the behavior of modern server models [1, 7].

For convenience, we assume that all machines are shut down at time $t = 0$ and also force all machines to shut down after the scheduling process, i.e. at times $t > T$. Consequently, we expect that there are no loads at times $t \notin [T]$, that is $\lambda_t = \lambda_{i,t} = 0$ for $t \notin [T]$. As another consequence, we know that any server must power down exactly as many times as it powers on.

A machine also incurs *switching costs* when changing its power state. In general, we can distinguish between power-up costs $\beta_\uparrow \in \mathbb{R}_{\geq 0}$ and power-down costs $\beta_\downarrow \in \mathbb{R}_{\geq 0}$. However, since we know that any server must shut down exactly as many times as it powers on, we can model both costs as being incurred when powering up a server; more precisely, a model with power-up costs $\beta_\uparrow$ and power-down costs $\beta_\downarrow$ can simply be transferred to a model without power-down costs by setting $\beta'_\uparrow \coloneqq \beta_\uparrow + \beta_\downarrow$ and $\beta'_\downarrow \coloneqq 0$.[1] In our work, we will always implicitly conduct this transformation as a first pre-processing step and denote the combined switching costs by $\beta \coloneqq \beta_\uparrow + \beta_\downarrow$. Like our operating cost function f, our switching costs β may not exclusively account for energy costs but also allow for delay costs, wear and tear costs, and the like [7]. Finally, since we are dealing with homogeneous servers, we note that f and β are the same for all machines.

2.2 Problem Statement

Using the above definitions, we can define the input of our model by setting

$$\mathcal{I} \coloneqq (m, T, \Lambda, \beta, f)$$

where $\Lambda = (\lambda_1, \ldots, \lambda_T)$ is the sequence of arrival rates. We will subsequently identify a problem instance by its input $\mathcal{I}$. Naturally, given a problem instance $\mathcal{I}$, we want to schedule our servers to minimize the sum of incurred costs while ensuring that the servers process the given loads in time. To do this, consider for each server $i \in [m]$ the sequence of its states S_i and the sequence of its assigned loads L_i:

$$\begin{aligned} S_i &\coloneqq (s_{i,1}, \ldots, s_{i,T}) \in \{0,1\}^T \\ L_i &\coloneqq (\lambda_{i,1}, \ldots, \lambda_{i,T}) \in [0,1]^T \end{aligned}$$

where $s_{i,t} \in \{0,1\}$ denotes whether server i at time t is shut down (0) or active (1). Recall that we assume that all machines are shut down at times $t \notin [T]$; thus, for $t \notin [T]$ and $i \in [m]$, we have $s_{i,t} = 0$. The sequence of all state changes and the sequence of all assigned loads are then defined by

$$\begin{aligned} \mathcal{S} &\coloneqq (S_1, \ldots, S_m) \\ \mathcal{L} &\coloneqq (L_1, \ldots, L_m) \end{aligned}$$

[1]Note that we could also choose to incur both costs when shutting down a server.

We will call a pair $\Sigma := (\mathcal{S}, \mathcal{L})$ a *schedule*. Finally, we are ready to define our problem statement. Given an input $\mathcal{I}$, our goal is to find a schedule Σ that satisfies the following optimization:

$$\underset{\Sigma}{\text{minimize}} \qquad c(\Sigma) := \underbrace{\sum_{t=1}^{T}\sum_{i=1}^{m}\big(f(\lambda_{i,t})s_{i,t}\big)}_{\text{operating costs}} + \underbrace{\beta\sum_{t=1}^{T}\sum_{i=1}^{m}\max\{0, s_{i,t} - s_{i,t-1}\}}_{\text{switching costs}} \tag{2.2}$$

$$\text{subject to} \qquad \sum_{i=1}^{m}(\lambda_{i,t}s_{i,t}) = \lambda_t, \quad \forall t \in [T] \tag{2.3}$$

We call a schedule *feasible* if it satisfies (2.3), and *optimal* if it satisfies (2.2) and (2.3).

3 Preliminaries

In this chapter, we conduct the preparatory work that will lay the foundations for our algorithms. For this, we will analyze the structure of feasible schedules to find characteristics of optimal strategies. These characteristics will then allow us to greatly simplify our optimization conditions.

We begin by examining the state sequences of feasible schedules. As we consider homogeneous servers, we do not care which exact machines process the given work loads. Rather, we only care about the number of active servers and the distribution of loads between them. It is in particular unreasonable to shut down a machine and to power on a different one in return; we could just keep the first server powered on and thereby save switching costs. This investigation is captured by our first proposition.

Proposition 3.1 (Reasonable switching). *Given a problem instance $\mathcal{I}$ and a feasible schedule Σ, there exists a feasible schedule Σ' such that*

(i) $c(\Sigma') \leq c(\Sigma)$ *and*

(ii) *Σ' never powers on and shuts down servers at the same time slot. More formally, Σ' satisfies the formula $\forall t \in [T] : F$ where F is defined as*

$$F \coloneqq \big(\forall i \in [m] : s_{i,t} - s_{i,t-1} \geq 0\big) \vee \big(\forall i \in [m] : s_{i,t} - s_{i,t-1} \leq 0\big) \tag{3.2}$$

Proof. Let $\Sigma = (\mathcal{S}, \mathcal{L})$ be a feasible schedule for $\mathcal{I}$. We give a procedure that repeatedly modifies Σ such that it satisfies condition (ii) and reduces or retains its cost.

Let $t \in [T]$ be the first time slot that falsifies (3.2). If there does not exist such a time slot, we are finished. Otherwise, we can obtain machines $i, j \in [m]$ such that $s_{i,t} - s_{i,t-1} = 1$ and $s_{j,t} - s_{j,t-1} = -1$, that is server i powers on at time t and server j shuts down. Without loss of generality, we may assume $i < j$. Since all servers are shut down at time $t = 0$, we have

$$s_{k,1} - s_{k,0} = s_{k,1} - 0 = s_{k,1} \geq 0, \quad \forall k \in [m]$$

which shows that formula (3.2) is satisfied for $t = 1$. Thus, we further assume $t > 1$. Now consider the state sequences of server i and j:

$$S_i = (s_{i,1}, \ldots, s_{i,t-1} = 0, s_{i,t} = 1, \ldots, s_{i,T})$$
$$S_j = (s_{j,1}, \ldots, s_{j,t-1} = 1, s_{j,t} = 0, \ldots, s_{j,T})$$

We modify S_i and S_j by swapping their states for time slots $\geq t$, i.e. we set

$$S'_i \coloneqq (s_{i,1}, \dots, s_{i,t-1} = 0, s_{j,t} = 0, \dots, s_{j,T})$$
$$S'_j \coloneqq (s_{j,1}, \dots, s_{j,t-1} = 1, s_{i,t} = 1, \dots, s_{i,T})$$

Similarly, we need to swap the assigned loads for server i and j:

$$L'_i \coloneqq (\lambda_{i,1}, \dots, \lambda_{i,t-1}, \lambda_{j,t}, \dots, \lambda_{j,T})$$
$$L'_j \coloneqq (\lambda_{j,1}, \dots, \lambda_{j,t-1}, \lambda_{i,t}, \dots, \lambda_{i,T})$$

Finally, we construct a new schedule $\Sigma' \coloneqq (\mathcal{S}', \mathcal{L}')$ by setting

$$\mathcal{S}' \coloneqq \big(S_1, \dots, S_{i-1}, S'_i, S_{i+1}, \dots, S_{j-1}, S'_j, S_{j+1}, \dots, S_T\big)$$
$$\mathcal{L}' \coloneqq \big(L_1, \dots, L_{i-1}, L'_i, L_{i+1}, \dots, L_{j-1}, L'_j, L_{j+1}, \dots, L_T\big)$$

We now want to verify that Σ' is a feasible schedule, that is Σ' satisfies (2.3). For time slots $< t$, the schedules Σ' and Σ still coincide. For time slots $\geq t$, we only changed the order of summation in (2.3). Thus, Σ' is feasible.

Further, Σ and Σ' coincide in their operating costs; however, by exchanging $\mathcal{S}$ with $\mathcal{S}'$, we reduced the number of servers powering up at time t. As we assume $\beta \geq 0$, we conclude $c(\Sigma') \leq c(\Sigma)$. Moreover, we decreased the number of servers violating (3.2) at time t. Hence, by repeating the described process on Σ', we obtain a terminating procedure that returns a schedule satisfying the conditions. □

Our next proposition – which will pose a cornerstone of our subsequent works – requires the use of Jensen's inequality, a well-known and frequently used analytic result found by Johan Jensen in 1906. It generalizes the idea that the secant line of a convex function lies above the graph of the function; more specifically, it states that the value of a convex function at a finite convex-combination of sampling points is less than or equal to the convex-combination of the function values at the sampling points.

Lemma 3.3 (Jensen's inequality). *Let $f : D \to \mathbb{R}$ be a convex function, $n \in \mathbb{N}$, $\lambda_1, \dots, \lambda_n \in D$, and $\mu_1, \dots, \mu_n \in [0,1]$ satisfying $\sum\limits_{i=1}^{n} \mu_i = 1$. Then the following inequality holds:*

$$f\left(\sum_{i=1}^{n} (\mu_i \lambda_i)\right) \leq \sum_{i=1}^{n} \big(\mu_i f(\lambda_i)\big)$$

Proof. We proof the claim by induction on $n \in \mathbb{N}$.

- <u>Basis:</u> For $n = 1$, we have $\mu_1 = 1$ and thus

$$f\left(\sum_{i=1}^{1} (\mu_i \lambda_i)\right) = f(\lambda_1) = \sum_{i=1}^{1} \big(\mu_i f(\lambda_i)\big)$$

- Step: Let $n \in \mathbb{N}$ be arbitrary and fixed.
  - I.H.: The assertion holds for n.
  - Claim: The assertion holds for $n+1$.
  - Proof: Since $\sum_{i=1}^{n+1} \mu_i = 1$, $\mu_i \in [0,1]$, and $n+1 \geq 2$, at least one μ_i must be smaller than 1. Without loss of generality, we may assume $\mu_1 < 1$.

    $$f\left(\sum_{i=1}^{n+1}(\mu_i\lambda_i)\right) = f\left(\mu_1\lambda_1 + \sum_{i=2}^{n+1}(\mu_i\lambda_i)\right) \overset{\mu_1 \neq 1}{=} f\left(\mu_1\lambda_1 + (1-\mu_1)\sum_{i=2}^{n+1}\frac{\mu_i\lambda_i}{1-\mu_1}\right)$$

    As f is convex, we have

    $$f\left(\mu_1\lambda_1 + (1-\mu_1)\sum_{i=2}^{n+1}\frac{\mu_i\lambda_i}{1-\mu_1}\right) \overset{f \text{ convex } (2.1)}{\leq} \mu_1 f(\lambda_1) + (1-\mu_1) f\left(\sum_{i=2}^{n+1}\frac{\mu_i\lambda_i}{1-\mu_1}\right)$$

    Since $\sum_{i=2}^{n+1} \frac{\mu_i}{1-\mu_1} = 1$, we can apply our induction hypothesis.

    $$\mu_1 f(\lambda_1) + (1-\mu_1) f\left(\sum_{i=2}^{n+1}\frac{\mu_i\lambda_i}{1-\mu_1}\right) \overset{\text{I.H.}}{\leq} \mu_1 f(\lambda_1) + (1-\mu_1)\sum_{i=2}^{n+1}\left(\frac{\mu_i}{1-\mu_1} f(\lambda_i)\right)$$

    We combine our steps and obtain

    $$\begin{aligned} f\left(\sum_{i=1}^{n+1}(\mu_i\lambda_i)\right) &\leq \mu_1 f(\lambda_1) + (1-\mu_1)\sum_{i=2}^{n+1}\left(\frac{\mu_i}{1-\mu_1} f(\lambda_i)\right) \\ &= \mu_1 f(\lambda_1) + \sum_{i=2}^{n+1}\big(\mu_i f(\lambda_i)\big) \\ &= \sum_{i=1}^{n+1}\big(\mu_i f(\lambda_i)\big) \end{aligned}$$

Thus, the assertion holds for any natural number n. □

Next, we want to consider the sequence of the number of active servers $\mathcal{X}$ defined as

$$\mathcal{X} := (x_1, \ldots, x_T) \in [m]_0^T \qquad \text{where} \qquad x_t := \left(\sum_{i=1}^{m} s_{i,t}\right) \in [m]_0$$

As we assume that all machines are shut down at times $t \notin [T]$, we have $x_t = 0$ for $t \notin [T]$. We now want to establish an optimal scheduling strategy given a fixed number of active servers. It turns out that an even load distribution seems a very desirable strategy.

Proposition 3.4 (Even load distribution)**.** *Given $x_t \in \mathbb{N}$ active servers in time slot t, an arrival rate $\lambda_t \in [0, x_t]$, and a convex operating cost function f, a most cost-efficient and feasible scheduling strategy is to assign each active server a load of λ_t / x_t.*

Proof. Let Σ be an arbitrary feasible schedule using x_t servers in time slot t, and let A be its set of active servers in time slot t, that is $A := \{i \in [m] \mid s_{i,t} = 1\}$. Consider the operating costs of Σ at time t given by

$$\sum_{i=1}^{m} \big(f(\lambda_{i,t}) \cdot s_{i,t}\big) = \sum_{i \in A} \big(f(\lambda_{i,t}) \cdot 1\big) + \sum_{i \in [m] \setminus A} \big(f(\lambda_{i,t}) \cdot 0\big) = \sum_{i \in A} f(\lambda_{i,t})$$

Since Σ is feasible (see constraint (2.3)), we have

$$\sum_{i \in A} \lambda_{i,t} = \lambda_t$$

Hence, we can obtain weights $\mu_1, \dots, \mu_{x_t} \in [0, 1]$ that relate λ_t to $\lambda_{i,t}$ for $i \in A$ such that

$$\sum_{j=1}^{x_t} \mu_j = 1 \qquad \text{and} \qquad \sum_{i \in A} f(\lambda_{i,t}) = \sum_{j=1}^{x_t} f(\mu_j \lambda_t) \tag{3.5}$$

In particular, we have

$$\sum_{j=1}^{x_t} (\mu_j \lambda_t) = \lambda_t \tag{3.6}$$

Using these weights, we now consider the operating costs of a schedule Σ^* that evenly distributes λ_t to its x_t active servers:

$$\sum_{j=1}^{x_t} f\left(\frac{\lambda_t}{x_t}\right) = x_t f\left(\frac{\lambda_t}{x_t}\right) \overset{(3.6)}{=} x_t f\left(\sum_{j=1}^{x_t} \frac{\mu_j \lambda_t}{x_t}\right)$$

With the fact that $\sum_{j=1}^{x_t} (1/x_t) = 1$ and the use of Jensen's inequality (Lemma 3.3), we can give an upper bound for the costs:

$$x_t f\left(\frac{\lambda_t}{x_t}\right) \overset{\text{Lemma 3.3}}{\leq} x_t \sum_{j=1}^{x_t} \left(\frac{1}{x_t} f(\mu_j \lambda_t)\right) = \frac{x_t}{x_t} \sum_{j=1}^{x_t} f(\mu_j \lambda_t) = \sum_{j=1}^{x_t} f(\mu_j \lambda_t) \overset{(3.5)}{=} \sum_{i \in A} f(\lambda_{i,t})$$

Thus, the operating costs of Σ^* give a lower bound for the operating costs of Σ, and the claim follows. □

As a special case, we can apply our just derived proposition to optimal schedules.

Corollary 3.7. *Given a problem instance $\mathcal{I}$, there exists an optimal schedule Σ^* that evenly distributes its arrival rates to its active servers in each time slot.*

Proof. Let $\Sigma = (\mathcal{S}, \mathcal{L})$ be an optimal schedule for $\mathcal{I}$. We exchange $\mathcal{L}$ with a new strategy $\mathcal{L}^*$ that evenly distributes the arrival rates to all active servers of Σ in each time slot, that is we set $\Sigma^* := (\mathcal{S}, \mathcal{L}^*)$. By Proposition 3.4, we have $c(\Sigma^*) \leq c(\Sigma)$ and, since Σ is optimal, $c(\Sigma) \leq c(\Sigma^*)$. Thus, we conclude $c(\Sigma^*) = c(\Sigma)$, i.e. Σ^* is optimal, which finishes the proof. □

As a result of Corollary 3.7, we can restrict ourselves to finding an optimal schedule that evenly distributes its arrival rates to its active servers. We now combine our results to derive the main theorem of our preliminary work.

Theorem 3.8. *Given a problem instance $\mathcal{I}$, there exists an optimal schedule Σ^* that evenly distributes its arrival rates and never powers on and shuts down servers at the same time slot.*

Proof. By Corollary 3.7, we obtain an optimal schedule Σ that evenly distributes its arrival rates to its active servers. Applying the procedure given in Proposition 3.1 to Σ yields Σ^* that further satisfies (3.2) for all $t \in [T]$ and $c(\Sigma^*) \leq c(\Sigma)$. Since Σ is optimal, we conclude $c(\Sigma^*) = c(\Sigma)$, and the claim follows. □

Theorem 3.8 allows us to identify an optimal schedule by its sequence of the number of active servers $\mathcal{X}$ and thereby to simplify our optimization conditions (2.2) and (2.3). For this, given a problem instance $\mathcal{I}$, we define the operating cost function $c_{op}(x, \lambda)$, which describes the costs incurred by evenly distributing λ on x active servers using f:

$$c_{op} : [m]_0 \times [0, m] \to \mathbb{R} \cup \{\infty\}, \quad c_{op}(x, \lambda) = \begin{cases} \infty, & \text{if } \lambda > x \\ x f(\lambda/x), & \text{if } x \neq 0 \land \lambda \leq x \\ 0, & \text{if } x = \lambda = 0 \end{cases} \tag{3.9}$$

We assign infinite costs in case $\lambda > x$ as there would be too few active servers to process the arrival rate, i.e. the schedule would not be feasible. Naturally, this definition can be lifted to whole schedules by setting

$$c_{op}(\mathcal{X}) := \sum_{i=1}^{T} c_{op}(x_i, \lambda_i) \tag{3.10}$$

which will come in handy in later sections. Next, we define the switching costs function $c_{sw}(x_{t-1}, x_t)$, which describes the incurred costs when changing the number of active server from x_{t-1} to x_t:

$$c_{sw}(x_{t-1}, x_t) := \beta \max\{0, x_t - x_{t-1}\} \tag{3.11}$$

Lastly, we can define the cost function $c(x_{t-1}, x_t, \lambda_t)$, which describes the incurring costs for a single time step using an even distribution of loads:

$$c(x_{t-1}, x_t, \lambda_t) \coloneqq c_{op}(x_t, \lambda_t) + c_{sw}(x_{t-1}, x_t) \tag{3.12}$$

The optimization conditions for a schedule now simplify to one single minimization:

$$\underset{\mathcal{X}}{\text{minimize}} \quad c(\mathcal{X}) \coloneqq \sum_{t=1}^{T} c(x_{t-1}, x_t, \lambda_t) \tag{3.13}$$

We subsequently call a schedule $\mathcal{X}$ *optimal* if it satisfies (3.13). Having greatly simplified our optimization conditions, we next approach the first main goal of our work: the development of an optimal offline algorithm.

4 Optimal Offline Scheduling

Now, as we have finished our preliminary work, we are able to derive our first two algorithms; more precisely, we will derive two optimal offline algorithms in this chapter. To begin, we will reduce our problem instance $\mathcal{I}$ to a shortest path problem of a weighted directed acyclic graph G. Then we proceed to find a shortest path in G and thereby an optimal schedule for $\mathcal{I}$ in pseudo-polynomial time $\Theta(Tm^2)$. After that, we refine our initial approach to derive an improved algorithm with pseudo-linear time complexity $\Theta(Tm)$.

4.1 A Pseudo-Polynomial-Time Algorithm

Let $\mathcal{I}$ be a problem instance. Thanks to our preliminary work, we know that there exists an optimal schedule which is identifiable by its sequence of the number of active servers $\mathcal{X}$. In order to find this sequence, we consider the weighted directed acyclic graph G defined as follows:

$$\begin{aligned}
&V \coloneqq \big\{v_{x,t} \mid x \in [m]_0, t \in [T]\big\} \cup \{v_{0,0}, v_{0,T+1}\}\\
&E \coloneqq \big\{(v_{x,t}, v_{x',t+1}) \mid x, x' \in [m]_0, t \in [T]_0, v_{x,t}, v_{x',t+1} \in V\big\}\\
&c_G(v_{x,t}, v_{x',t+1}) \coloneqq c(x, x', \lambda_{t+1})\\
&G \coloneqq (V, E, c_G)
\end{aligned}$$

For any possible number of active servers x and any time slot t, we add a node $v_{x,t}$. Moreover, we add a start node $v_{0,0}$ as well as an end node $v_{0,T+1}$. Next, we connect all nodes to their successors with respect to time. Semantically, $v_{x,t}$ denotes the state of distributing the arrival rate λ_t evenly to x servers in time slot t. For any edge connecting $v_{x,t}$ with $v_{x',t+1}$, we assign costs $c(x, x', \lambda_{t+1})$, that is the edge's cost corresponds to switching from x to x' machines and processing the load λ_{t+1} with x' machines. A graphical representation can be found in the following figure.

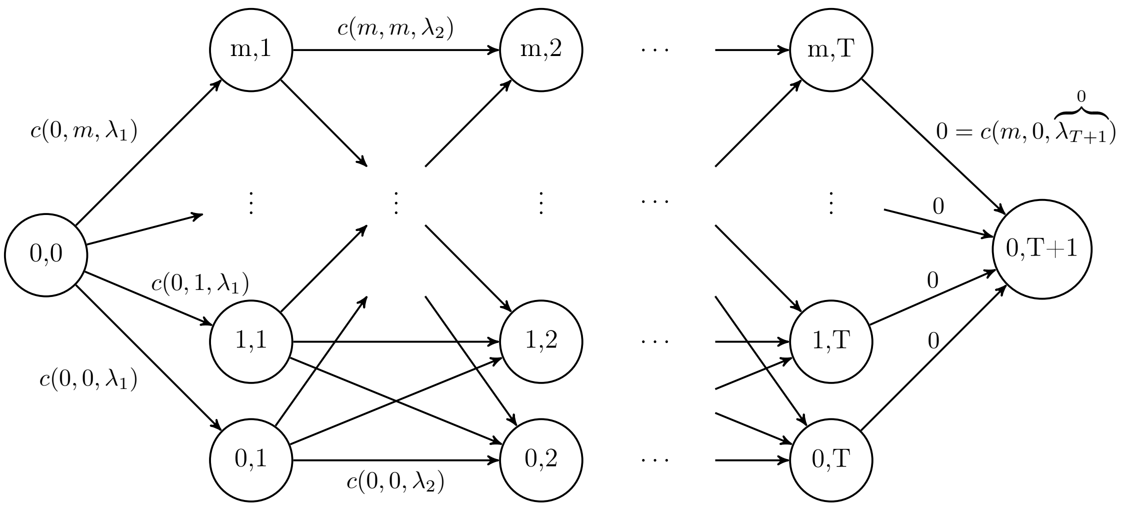


Figure 4.1: Weighted directed acyclic graph for a pseudo-polynomial-time optimal offline algorithm

As we can see in Figure 4.1, the cost of a path $P = (v_{0,0}, v_{x_1,1}, \dots, v_{x_T,T}, v_{0,T+1})$ from our start node $v_{0,0}$ to our end node $v_{0,T+1}$ is given by

$$c(P) = c(0, x_1, \lambda_1) + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) + \overbrace{c(x_T, 0, 0)}^{0} = c(0, x_1, \lambda_1) + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) \quad (4.1)$$

Note that the cost of such a path directly corresponds to that of a schedule $\mathcal{X}$ (see (3.13)). Any shortest path from $v_{0,0}$ to $v_{0,T+1}$ is thus forced to minimize the cost of the corresponding schedule. Needless to say, this demands for a proof of correctness.

Lemma 4.2. *Let $\boldsymbol{\mathcal{X}}$ be the set of all schedules $\mathcal{X}$ for $\mathcal{I}$, and let $\boldsymbol{\mathcal{P}}$ the set of all paths from $v_{0,0}$ to $v_{0,T+1}$. The map*

$$\Phi : \boldsymbol{\mathcal{P}} \to \boldsymbol{\mathcal{X}}, \quad (v_{0,0}, v_{x_1,1}, v_{x_2,2}, \dots, v_{x_T,T}, v_{0,T+1}) \mapsto (x_1, \dots, x_T)$$

is a bijection with inverse map

$$\Psi : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{P}}, \quad (x_1, \dots, x_T) \mapsto (v_{0,0}, v_{x_1,1}, v_{x_2,2}, \dots, v_{x_T,T}, v_{0,T+1})$$

satisfying $c(\mathcal{X}) = c\big(\Psi(\mathcal{X})\big)$.

Proof. It is easy to check that $\Psi \circ \Phi = id_{\boldsymbol{\mathcal{P}}}$ and $\Phi \circ \Psi = id_{\boldsymbol{\mathcal{X}}}$ (the functions merely extract and embed the states x_t). Thus, Φ is indeed bijective with inverse map Ψ. Next, let $\mathcal{X} = (x_1, \dots, x_T)$ be a schedule for $\mathcal{I}$. We have

$$P \coloneqq \Psi(\mathcal{X}) = (v_{0,0}, v_{x_1,1}, v_{x_2,2}, \dots, v_{x_T,T}, v_{0,T+1})$$

We examine the cost of $\mathcal{X}$ and conclude

$$c(\mathcal{X}) \overset{(3.13)}{=} \sum_{t=1}^{T} c(x_{t-1}, x_t, \lambda_t) = \underbrace{c(x_0, x_1, \lambda_1)}_{=c(0,x_1,\lambda_1)} + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) \overset{(4.1)}{=} c(P)$$

which shows that Ψ and, as a consequence, Φ are cost-preserving maps. □

We can now use this bijection to obtain our desired result: the correspondence between optimal schedules and shortest paths.

Theorem 4.3. *There exists a cost-preserving bijection between optimal schedules $\mathcal{X}^*$ for $\mathcal{I}$ and shortest paths from $v_{0,0}$ to $v_{0,T+1}$.*

Proof. By Lemma 4.2, we have a bijection Ψ between schedules $\mathcal{X}$ and paths from $v_{0,0}$ to $v_{0,T+1}$ obeying $c(\mathcal{X}) = c\big(\Psi(\mathcal{X})\big)$. Thus, we have

$$c(\mathcal{X}) \text{ minimal } \iff c\big(\Psi(\mathcal{X})\big) \text{ minimal}$$

and the claim follows. □

In the following, we give an algorithm based on our just verified construction. We split our procedure into two subroutines.

SHORTEST_PATHS calculates the minimum costs of the graph's nodes, layer by layer; that is, it calculates the shortest paths following the graph's topological sorting. It returns the minimum costs to all nodes, stored in C, as well as the predecessor of any node with respect to its shortest path, stored in P.

EXTRACT_SCHEDULE uses the predecessors calculated by SHORTEST_PATHS in order to obtain the sequence of nodes describing a shortest path; thereby, it calculates an optimal schedule for our problem instance.

The correctness of Algorithm 1 directly follows from Theorem 4.3 and the correctness of the shortest path calculation for directed acyclic graphs (for a proof see [4, Section 24.2]).

Naturally, we are interested in our algorithm's time and memory complexity. For this, we first need to consider the size of our input $\mathcal{I} = (m, T, \Lambda, \beta, f)$. In theory, our function $f : [0,1] \to \mathbb{R}$ may be easily defined by saying, for example, $f(\lambda) := \lambda^2$; however, practically, it is difficult to answer how such a function may be specified and what the size of such a function as part of the input may be. For simplicity, we consider the size of f negligible in comparison with the remaining input variables' size. Further, we assume that a non-negative real number r requires $\Theta\big(\log_2(r)\big)$ bits for its encoding; if r turns out to be smaller

Algorithm 1 Pseudo-polynomial-time optimal offline scheduling

```
1: function OPTIMAL_OFFLINE_SCHEDULING($m, T, \Lambda, \beta, f$)
2:     $(C, P) \leftarrow$ SHORTEST_PATHS($m, T, \Lambda, \beta, f$)
3:     $\mathcal{X} \leftarrow$ EXTRACT_SCHEDULE($C, P, T$)
4:     return $\mathcal{X}$

5: function SHORTEST_PATHS($m, T, \Lambda, \beta, f$)
6:     let $C[0 \dots m, 1 \dots T]$ and $P[0 \dots m, 1 \dots T]$ be new tables
                                          ▷ Allocate cost and predecessor tables
7:     for $x \leftarrow 0$ to $m$ do                        ▷ Initialize first layer
8:         $P[x, 1] \leftarrow 0$ and $C[x, 1] \leftarrow c(0, x, \lambda_1)$
9:     for $t \leftarrow 2$ to $T$ do      ▷ Iteratively calculate costs and predecessors
10:        for $x' \leftarrow 0$ to $m$ do
11:            $P[x', t] \leftarrow \underset{x \in [m]_0}{\arg\min} \big\{ C[x, t-1] + c(x, x', \lambda_t) \big\}$   ▷ Get best preceding choice
12:            $C[x', t] \leftarrow C\big[P[x', t], t-1\big] + c\big(P[x', t], x', \lambda_t\big)$   ▷ Set the node's cost
13:    return $(C, P)$

14: function EXTRACT_SCHEDULE($C, P, T$)
15:    let $\mathcal{X}[1 \dots T]$ be a new array             ▷ Allocate the schedule array
16:    $\mathcal{X}[T] \leftarrow \underset{x \in [m]_0}{\arg\min} \big\{ C[x, T] \big\}$         ▷ Find best choice for last time slot
17:    for $t \leftarrow T - 1$ to $1$ do   ▷ Iteratively obtain schedule from predecessor table
18:        $\mathcal{X}[t] \leftarrow P\big[\mathcal{X}[t+1], t+1\big]$
19:    return $\mathcal{X}$
```

than 1, we assume some minor constant encoding size. The size of our input is then given by

$$\begin{aligned}
\mathit{size}(\mathcal{I}) &= \mathit{size}(m) + \mathit{size}(T) + \mathit{size}(\Lambda) + \mathit{size}(\beta) \\
&= \Theta\big(\log_2(m)\big) + \Theta\big(\log_2(T)\big) + \sum_{i=1}^{T} \Theta\big(\log_2(\lambda_i)\big) + \Theta\big(\log_2(\beta)\big) \\
&= \Theta\big(\log_2(m) + \log_2(T) + \log_2(\beta)\big) + \sum_{i=1}^{T} \Theta\big(\log_2(\overbrace{\lambda_i}^{\leq m})\big) \\
&\leq \Theta\big(\log_2(m) + \log_2(T) + \log_2(\beta)\big) + \mathcal{O}\big(T \log_2(m)\big) \\
&\leq \mathcal{O}\big(T \log_2(m) + \log_2(\beta)\big) \qquad (4.4)
\end{aligned}$$

For our runtime analysis, we assume that calling the operating cost function $f(\cdot)$, and as a consequence also $c(\cdot,\cdot,\cdot)$, incurs constant cost. Subroutine SHORTEST_PATHS requires $\Theta(m)$ steps for its initialization and $\Theta(Tm^2)$ steps for the iterative calculation of the nodes' predecessors and costs. In addition, EXTRACT_SCHEDULE needs $\Theta(m)$ steps for its initial minimization search and $\Theta(T)$ iterations for its schedule retrieval. Thus, Algorithm 1 has a time complexity of

$$\Theta(m + Tm^2 + m + T) = \Theta(Tm^2)$$

The runtime is polynomial in the numeric value of m and T; however, it is exponential in the size of the input since we only need $\log_2(m)$ bits to encode m, as we saw in (4.4). Hence, the algorithm is pseudo-polynomial.

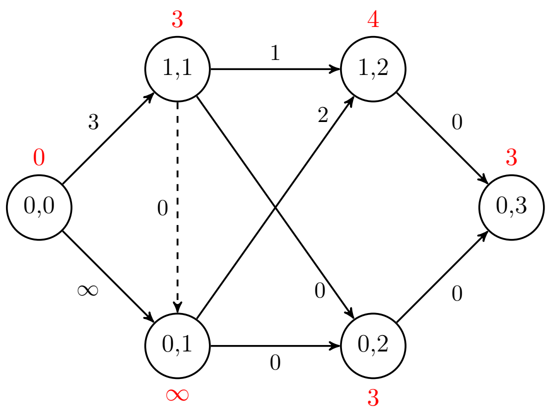
Our memory demand is determined by the size of the array $\mathcal{X}$ and the size of the tables C and P. The former is of size $\Theta(T)$, and the latter are of size $\Theta(Tm)$. Thus, Algorithm 1 has a memory complexity of $\Theta(T + 2Tm) = \Theta(Tm)$.

4.2 A Pseudo-Linear-Time Algorithm

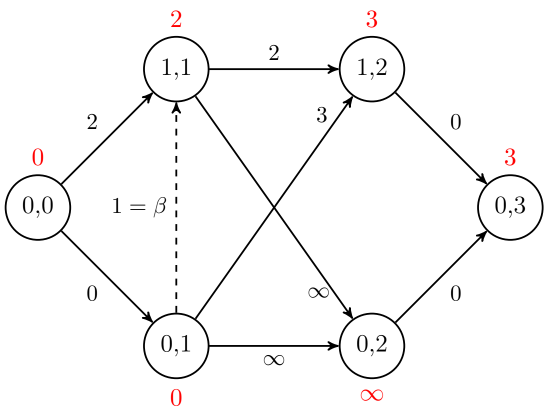
The algorithm developed in Section 4.1 is of a quite simple nature. Its underlying graph G is able to represent any possible schedule $\mathcal{X}$ since we simply add an edge for any possible scheduling choice at any possible time slot. This approach seems rather intuitive and readily verifiable, but this convenience comes with a cost: The density of G causes a quadratic runtime in the number of servers m. In order to improve the runtime to pseudo-linear complexity, we need to "thin out" our graph.

Let us revise our initial approach. Our graph consists of nodes $v_{x,t}$, any of which represents the state of distributing the arrival rate λ_t evenly to x servers in time slot t. The algorithm calculates the minimum costs to all those nodes. Thus, for any node $v_{x,t}$, it returns the lowest achievable cost up to time slot t of all schedules $\mathcal{X}$ that assign x servers at time t to process the arrival rate λ_t. In particular, the cost of the end node $v_{0,T+1}$ tells us the minimum cost of all schedules. This approach, however, does not consider the possibility to schedule $y \neq x$ servers in time slot t and to switch to x servers just at the very last moment of t when calculating the cost of $v_{x,t}$. Consider the following example:

Example 4.5. Let $\mathcal{I}_i = (m = 1, T = 2, \Lambda_i, \beta = 1, f)$ be the inputs for two problem instances where $f(\lambda) = \lambda^2 + 1$, $\Lambda_1 = (1, 0)$, and $\Lambda_2 = (0, 1)$. The graphs of the two problem instances and the corresponding calculations done by Algorithm 1 are illustrated in the following figure.



Problem $\mathcal{I}_1$: State $v_{0,1}$ could be reached with cost 3 by moving down from node $v_{1,1}$.



Problem $\mathcal{I}_2$: State $v_{1,1}$ could be reached with cost 1 by moving up from $v_{0,1}$.

Figure 4.2: Two examples depicting a shortcoming of our initial approach. The calculated costs are highlighted in red. Dashed edges are not part of Algorithm 1.

Although our algorithm delivers the correct end results, its immediate steps are somewhat unsatisfying. We want our states to capture a more general notion than given in Section 4.1; preferably, we would like a node $v_{x,t}$ to denote the state of having x active servers at the end of time slot t. In practice, we may reach such a state $v_{x,t}$ by moving down from a state $v_{y^\downarrow,t}$ where $y^\downarrow > x$ with cost 0 or by moving up from a state $v_{y^\uparrow,t}$ where $y^\uparrow < x$ with cost $\beta(x - y^\uparrow)$. In order to allow for these new possibilities, given a problem instance $\mathcal{I}$, we define a weighted directed acyclic graph as follows:

$$
\begin{aligned}
V &\coloneqq \left\{ v_{x,t\downarrow} \mid x \in [m]_0, t \in [T] \right\} \cup \left\{ v_{x,t\uparrow} \mid x \in [m]_0, t \in [T-1] \right\} \cup \{v_{0,0}\} \\
E_s &\coloneqq \left\{ (v_{0,0}, v_{x,1\downarrow}) \mid x \in [m]_0 \right\} \\
E_\downarrow &\coloneqq \left\{ (v_{x,t\downarrow}, v_{x-1,t\downarrow}) \mid x \in [m], t \in [T] \right\} \\
E_\uparrow &\coloneqq \left\{ (v_{x-1,t\uparrow}, v_{x,t\uparrow}) \mid x \in [m], t \in [T-1] \right\} \\
E_{\downarrow\uparrow} &\coloneqq \left\{ (v_{x,t\downarrow}, v_{x,t\uparrow}) \mid x \in [m]_0, t \in [T-1] \right\} \\
E_{\uparrow\downarrow} &\coloneqq \left\{ (v_{x,t\uparrow}, v_{x,t+1\downarrow}) \mid x \in [m]_0, t \in [T-1] \right\} \\
E &\coloneqq E_s \cup E_\downarrow \cup E_\uparrow \cup E_{\downarrow\uparrow} \cup E_{\uparrow\downarrow} \\
c_G(e) &\coloneqq \begin{cases} c(0, x, \lambda_1), & \text{if } e = (v_{0,0}, v_{x,1\downarrow}) \in E_s \\ c_{op}(x, \lambda_{t+1}), & \text{if } e = (v_{x,t\uparrow}, v_{x,t+1\downarrow}) \in E_{\uparrow\downarrow} \\ \beta, & \text{if } e \in E_\uparrow \\ 0, & \text{if } e \in (E_\downarrow \cup E_{\downarrow\uparrow}) \end{cases} \\
G &\coloneqq (V, E, c_G)
\end{aligned}
$$

A more appealing, graphical representation can be found in the following figure.

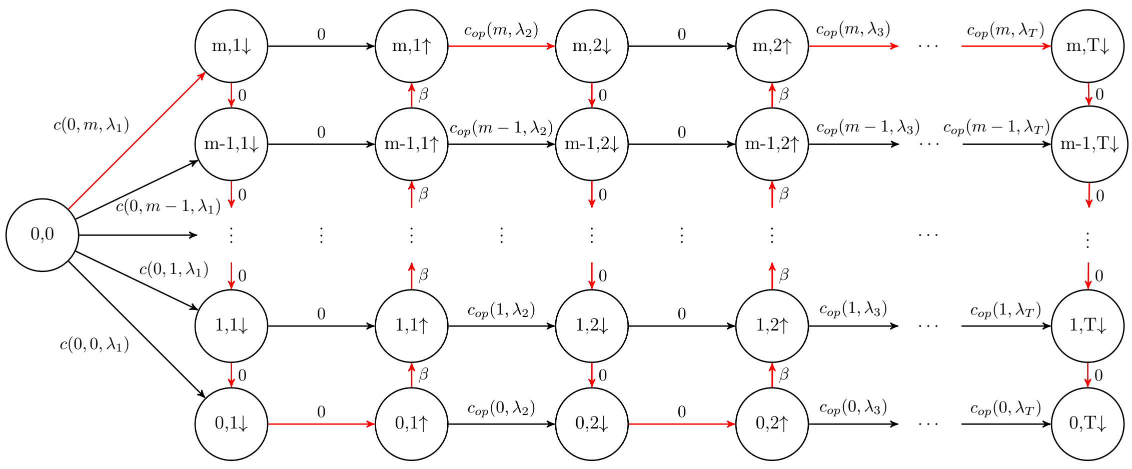


Figure 4.3: Graph for a pseudo-linear-time optimal offline algorithm; the path of the topological sorting is highlighted in red.

For any possible number of active servers x and any time slot t, we add a node $v_{x,t\downarrow}$. Semantically, the cost of $v_{x,t\downarrow}$ will denote the minimum cost up to time slot t when processing λ_t with x or more servers. Further, for any time slot $t \in [T-1]$, we add a node $v_{x,t\uparrow}$. The cost of $v_{x,t\uparrow}$ will denote the minimum cost of having x active servers at the end of time slot t. Moreover, we add a start node $v_{0,0}$.

The set of edges E_s denotes the start initialization step. An edge $(v_{x,t\uparrow}, v_{x,t+1\downarrow}) \in E_{\uparrow\downarrow}$ accounts for the operating costs that incur when processing the arrival rate λ_{t+1} with x active servers.

After any time step from $t-1$ to t, we have a minimization step in our graph. For this, we first move down the chain $v_{m,t\downarrow}, v_{m-1,t\downarrow}, \dots, v_{0,t\downarrow}$ using edges from $E_\downarrow$ with cost 0. Then we proceed to move to the right from $v_{0,t\downarrow}$ to $v_{0,t\uparrow}$. Lastly, we move up the chain $v_{0,t\uparrow}, v_{1,t\uparrow}, \dots, v_{m,t\uparrow}$ using edges from $E_\uparrow$ with cost β. In order to have the possibility to keep the calculated cost of $v_{x,t\downarrow}$ while moving up, we add edges $(v_{x,t\downarrow}, v_{x,t\uparrow}) \in E_{\downarrow\uparrow}$ with cost 0. This minimization step is the key to our runtime improvement. It facilitates the determination of the best predecessor of a state $v_{x,t\downarrow}$ since we already know that the minimum cost of having x servers at the end of time slot $t-1$ is stored in $v_{x,t-1\uparrow}$. Thus, the cheapest possibility to process the next arrival rate λ_t using x servers can simply be calculated by adding $c_{op}(x, \lambda_t)$ to the cost of $v_{x,t-1\uparrow}$. Consequently, the cost of $v_{x,t\downarrow}$ is given by the minimum of $v_{x,t-1\uparrow} + c_{op}(x, \lambda_t)$ and $v_{x+1,t\downarrow}$.

As one can see in Figure 4.3, we stretched our graph but at the same time also greatly reduced the number of edges compared to our initial approach in Section 4.1. By following the colored path of the topological sorting, we can work our way through the graph to calculate the nodes' minimum costs, ultimately reaching the destination $v_{0,T\downarrow}$. The cost of our destination $v_{0,T\downarrow}$ will denote the minimum cost up to time slot T when processing λ_T with 0 or more servers. Hence, it will contain our desired end result – the minimum cost of all possible schedules.

Our next task shall be the verification of our new construction. We first examine the possible paths from $v_{0,0}$ to $v_{0,T\downarrow}$ in our graph. In contrast to our approach in Section 4.1, our new graph contains paths that do not directly correspond to a schedule $\mathcal{X}$. Consider, for example, the following path:

$$P \coloneqq (v_{0,0}, v_{1,1\downarrow}, v_{0,1\downarrow}, v_{0,1\uparrow}, v_{1,1\uparrow}, v_{2,1\uparrow}, v_{2,2\downarrow}, \ldots, v_{0,T\downarrow})$$

A schedule corresponding to P would use one active server in its first time slot, then power down this server, and subsequently turn on two servers to process the next arrival rate. This seems unreasonable: We could just keep the initial server on to save switching costs (note the correspondence to Proposition 3.1). In fact, this behavior cannot even be represented using our schedule notation $\mathcal{X}$ and optimization condition (3.13). We can, however, modify P to represent a more reasonable sequence by setting

$$P' \coloneqq (v_{0,0}, v_{1,1\downarrow}, v_{1,1\uparrow}, v_{2,1\uparrow}, v_{2,2\downarrow}, \ldots, v_{0,T\downarrow})$$

This revised path pleasantly translates to a schedule $\mathcal{X}$, in this case $\mathcal{X} = (1, 2, \ldots)$. We now want to give a more formal definition of our observation.

Definition 4.6 (Reasonable paths)**.** Let P be a path from $v_{0,0}$ to $v_{0,T\downarrow}$. For any time slot $t \in [T]$, let $E^t_\downarrow$ be the set of edges $(v_{x,t\downarrow}, v_{x',t\downarrow}) \in E_\downarrow$ used by P at time t. Similarly, let $E^t_\uparrow$ be the set of edges $(v_{x,t\uparrow}, v_{x',t\uparrow}) \in E_\uparrow$ used by P at time $t \in [T-1]$. The path P is called *reasonable* if for any time slot $t \in [T-1]$, the path does not shut down and power on servers simultaneously at t. More formally, P must satisfy the formula $\forall t \in [T-1] : F$ where F is defined as

$$F \coloneqq \left(E^t_\downarrow = \emptyset\right) \lor \left(E^t_\uparrow = \emptyset\right) \tag{4.7}$$

The next proposition justifies that our reasonable paths indeed deserve to be called *reasonable*.

Proposition 4.8. *Any given path P from $v_{0,0}$ to $v_{0,T\downarrow}$ can be transformed to a reasonable path P' with $c(P') \leq c(P)$.*

Proof. Let P be a path from $v_{0,0}$ to $v_{0,T\downarrow}$. We give a procedure that repeatedly modifies P such that it satisfies (4.7) and reduces or retains its cost.

Let $t \in [T-1]$ be the first time slot that falsifies (4.7). If there does not exist such a time slot, we are finished. Otherwise, let $E^t_\downarrow$ and $E^t_\uparrow$ be its sets of edges as defined in

Definition 4.6. Since P falsifies (4.7) at time t, both $E^t_\downarrow$ and $E^t_\uparrow$ must be non-empty. Thus, we can obtain the "maximum" nodes involved in these sets.

$$\begin{aligned} x_s &:= \max\bigl\{x \in [m] \mid (v_{x,t\downarrow}, v_{x-1,t\downarrow}) \in E^t_\downarrow\bigr\} \\ x_e &:= \max\bigl\{x \in [m] \mid (v_{x-1,t\uparrow}, v_{x,t\uparrow}) \in E^t_\uparrow\bigr\} \end{aligned}$$

Next, consider the subpath $S = (v_{x_s,t\downarrow}, \dots, v_{x_e,t\uparrow})$ of P. Note that the subpath in particular uses all edges from $E^t_\downarrow$ and $E^t_\uparrow$. Evidently, a most cost-efficient path S' from $v_{x_s,t\downarrow}$ to $v_{x_e,t\uparrow}$ minimizes the number of edges $(v_{x,t\uparrow}, v_{x+1,t\uparrow})$ since each of these edges incurs cost $\beta \geq 0$. For the construction of S', we observe that we must not shut down servers if $x_s \leq x_e$, and that we must not power on servers if $x_s \geq x_e$; we thus consider three cases for S':

$$S' := \begin{cases} (v_{x_s,t\downarrow}, v_{x_s,t\uparrow}, v_{x_s+1,t\uparrow}, \dots, v_{x_e,t\uparrow}), & \text{if } x_s < x_e \\ (v_{x_s,t\downarrow}, v_{x_s-1,t\downarrow}, \dots, v_{x_e,t\downarrow}, v_{x_e,t\uparrow}), & \text{if } x_s > x_e \\ (v_{x_s,t\downarrow}, v_{x_e,t\uparrow}), & \text{if } x_s = x_e \end{cases}$$

In each case, S' uses edges from at most one of the sets $E^t_\downarrow$ and $E^t_\uparrow$. Thus, by replacing the subpath S of P with S', we obtain a new path P' that satisfies (4.7) up to and including time slot t. Moreover, the paths P and P' coincide in their costs before visiting $v_{x_s,t\downarrow}$ and after visiting $v_{x_e,t\uparrow}$; however, they differ in that there are less switching costs β at time t using P'. As we assume $\beta \geq 0$, we conclude $c(P') \leq c(P)$.

Hence, by repeating described process on P', we obtain a terminating procedure that returns a path satisfying the conditions. □

As a result of Proposition 4.8, we shall focus our attention on reasonable paths. Our goal is to establish the connection between reasonable paths P of our graph and schedules $\mathcal{X}$. Intuitively, one can see that such a path P is uniquely determined by the "maximum" nodes $v_{x,t\downarrow}$ taken by P at any time slot t; these nodes represent the state of processing λ_t with x servers. Consider the following example:

Example 4.9. Let $\mathcal{I} = \bigl(m = 3, T = 3, \Lambda = (3,0,1), \beta, f\bigr)$ be the input for a problem instance. Then one example of a reasonable path is given by

$$P := (v_{0,0}, v_{3,1\downarrow}, v_{2,1\downarrow}, v_{1,1\downarrow}, v_{0,1\downarrow}, v_{0,1\uparrow}, v_{0,2\downarrow}, v_{0,2\uparrow}, v_{1,2\uparrow}, v_{1,3\downarrow}, v_{0,3\downarrow})$$

A schedule corresponding to P would use three active server in its first time slot, then shut down all servers for the second time slot, and ultimately power on one server to process the last arrival rate; that is, the corresponding schedule of P is $\mathcal{X} = (3,0,1)$. As can be seen in Figure 4.4, this sequence also corresponds to the sequence of "maximum" nodes $v_{x,t\downarrow}$ taken by P. Conversely, given the schedule $\mathcal{X} = (3,0,1)$, it can easily be seen that there exists only one reasonable path corresponding to $\mathcal{X}$, namely P.

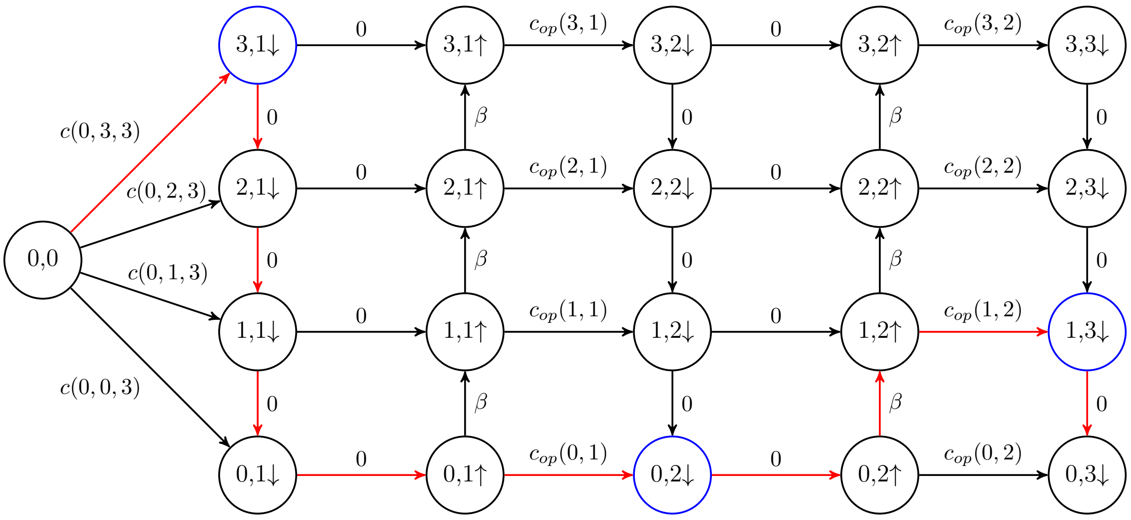


Figure 4.4: Illustration of the path P. The path is highlighted in red. The "maximum" nodes $v_{x,t\downarrow}$ taken by P are highlighted in blue.

Before we approach our next lemma, which formalizes these ideas, we need to give a precise definition of what we understand as *"maximum" nodes* $v_{x,t\downarrow}$ used by a reasonable path.

Definition 4.10. Let P be a reasonable path. For any $t \in [T]$, let $V_{\downarrow}^{t}$ be the set of nodes $v_{x,t\downarrow}$ visited by P. The "maximum" node $v_{x,t\downarrow}$ at each time slot $t \in [T]$ can then be identified by

$$x_t := \max\{x \mid v_{x,t\downarrow} \in V_{\downarrow}^{t}\}$$

Remember that a schedule $\mathcal{X} = (x_1, \ldots, x_T)$ also uses the notation x_t to identify the number of active severs at time t. Needless to say, this clash of names is intentional and justified by the next lemma.

Lemma 4.11. *Let $\boldsymbol{\mathcal{X}}$ be the set of all schedules $\mathcal{X}$ for $\mathcal{I}$, and let $\boldsymbol{\mathcal{P}}$ be the set of all reasonable paths. The map*

$$\Phi : \boldsymbol{\mathcal{P}} \to \boldsymbol{\mathcal{X}}, \quad P \mapsto (x_1, \ldots, x_T)$$

is a bijection satisfying $c(P) = c\big(\Phi(P)\big)$.

Proof. First, we check that Φ is bijective, i.e. Φ is injective and surjective.

Let P and P' be reasonable paths with $\Phi(P) = \Phi(P')$. Then P as well as P' must start with the edge $(v_{0,0}, v_{x_1,1\downarrow})$. As P and P' are reasonable, they both satisfy $E_{\downarrow}^{t} = \emptyset \vee E_{\uparrow}^{t} = \emptyset$

for each time slot t. Due to this restriction, their paths between $v_{x_t,t\downarrow}$ and $v_{x_{t+1},t+1\downarrow}$ must coincide for $t \in [T-1]$. Further, the path from $v_{x_T,T}$ to $v_{0,T}$ is unique in our graph. Thus, we conclude $P = P'$, which shows the injectivity of Φ.

Next, let $\mathcal{X} = (x_1, \ldots, x_T) \in \boldsymbol{\mathcal{X}}$ be a schedule for $\mathcal{I}$. For any $t \in [T-1]$, we set

$$S_t \coloneqq \begin{cases} (v_{x_t,t\downarrow}, v_{x_t,t\uparrow}, v_{x_t+1,t\uparrow}, \ldots, v_{x_{t+1},t\uparrow}), & \text{if } x_t < x_{t+1} \\ (v_{x_t,t\downarrow}, v_{x_t-1,t\downarrow}, \ldots, v_{x_{t+1},t\downarrow}, v_{x_{t+1},t\uparrow}), & \text{if } x_t > x_{t+1} \\ (v_{x_t,t\downarrow}, v_{x_{t+1},t\uparrow}), & \text{if } x_t = x_{t+1} \end{cases}$$

We then concatenate these subpaths to construct a reasonable path

$$P \coloneqq (v_{0,0}, S_1, S_2, \ldots, S_{T-1}, v_{x_T,T\downarrow}, v_{x_T-1,T\downarrow}, \ldots, v_{0,T\downarrow})$$

Evidently, the constructed path satisfies $\Phi(P) = \mathcal{X}$, which shows the surjectivity of Φ.

It remains to show that Φ is cost-preserving. For this, let P be a reasonable path and let $\mathcal{X} \coloneqq \Phi(P) = (x_1, \ldots, x_T)$ be its image. As P is reasonable, we have $E^t_\downarrow = \emptyset \vee E^t_\uparrow = \emptyset$ for any time slot t. If $E^t_\uparrow$ is empty, we have $x_{t-1} \geq x_t$. The cost between the nodes $v_{x_{t-1},t-1\downarrow}$ and $v_{x_t,t\downarrow}$ is then given by $c_{op}(x_t, \lambda_t)$. If $E^t_\uparrow$ is non-empty, we have $x_{t-1} < x_t$. In this case, the cost is given by $\beta(x_t - x_{t-1}) + c_{op}(x_t, \lambda_t)$. Using these observations, the cost of P can be calculated by

$$\begin{aligned} c(P) &= c(0, x_1, \lambda_1) + \sum_{t=2}^{T} \Big(\overbrace{\beta \max\{0, x_t - x_{t-1}\}}^{c_{sw}(x_{t-1}, x_t)} + c_{op}(x_t, \lambda_t) \Big) \\ &\overset{(3.11)}{=} c(0, x_1, \lambda_1) + \sum_{t=2}^{T} \overbrace{\Big(c_{sw}(x_{t-1}, x_t) + c_{op}(x_t, \lambda_t) \Big)}^{c(x_{t-1}, x_t, \lambda_t)} \\ &\overset{(3.12)}{=} c(0, x_1, \lambda_1) + \sum_{t=2}^{T} c(x_{t-1}, x_t, \lambda_t) \\ &= \sum_{t=1}^{T} c(x_{t-1}, x_t, \lambda_t) \\ &\overset{(3.13)}{=} c(\mathcal{X}) \end{aligned}$$

which shows that Φ is a cost-preserving map. □

Again, we can use the established bijection to obtain our desired result: the correspondence between optimal schedules and shortest reasonable paths.

Theorem 4.12. *There exists a cost-preserving bijection between optimal schedules $\mathcal{X}^*$ for $\mathcal{I}$ and shortest reasonable paths.*

Proof. By Lemma 4.11, we have a bijection Φ between reasonable paths P and schedules $\mathcal{X}$ obeying $c(P) = c\big(\Phi(P)\big)$. Thus, we have

$$c(P) \text{ minimal} \iff c\big(\Phi(P)\big) \text{ minimal}$$

and the claim follows. □

Naturally, common shortest path algorithms do not have any knowledge about our "reasonable" paths. What if we obtain a shortest path that coincidentally is not reasonable? This shall not turn out to be a problem, as the next corollary shows us.

Corollary 4.13. *Any shortest path P from $v_{0,0}$ to $v_{0,T\downarrow}$ can be transformed to an optimal schedule $\mathcal{X}^*$ for $\mathcal{I}$ with $c(\mathcal{X}^*) = c(P)$.*

Proof. Let P be a shortest path from $v_{0,0}$ to $v_{0,T\downarrow}$. By Proposition 4.8, the path P can be transformed to a reasonable path P' with $c(P') \leq c(P)$. In turn, P' corresponds to an optimal schedule $\mathcal{X}^*$ with $c(\mathcal{X}^*) = c(P')$ by Theorem 4.12. Since P is a shortest path, we know that $c(P) \leq c(P')$ and thus conclude $c(P') = c(P)$. Hence, we have $c(\mathcal{X}^*) = c(P)$, and the claim follows. □

Like in Section 4.1, we next give an algorithm based on our just verified construction. Although we could search for an arbitrary shortest path in our graph and transform it to an optimal schedule, as shown in Corollary 4.13, our algorithm only considers reasonable paths. Again, we split our procedure into two subroutines.

SHORTEST_PATHS calculates the minimum costs of the graph's nodes by following the graph's topological sorting. The calculated costs are stored in table C. When calculating a node's minimum cost at time t, it further keeps track of the selection that has to be made at time t to obtain the node's minimum cost. This selection is then stored in table S. The procedure merges the costs of the nodes $v_{x,t\downarrow}$ and $v_{x,t\uparrow}$ in each time slot $t \in [T-1]$ and ultimately only keeps the relevant information for each node $v_{x,t\uparrow}$, namely the node's cost and best scheduling selection at time t. The information that would tell us the path between $v_{x,t\uparrow}$ and its best scheduling selection at time t in our graph is lost; however, this information is of no concern since we only want to consider reasonable paths, which are already uniquely identified by their scheduling choices (see Lemma 4.11). The restriction to reasonable paths additionally improves the algorithm's (practical) memory demand as we can reduce the sizes of the tables C and S from $2mT$ to mT. The function returns the minimum costs to all nodes $v_{x,T\downarrow}$ and to all nodes $v_{x,t\uparrow}$ for $t \in [T-1]$, stored in C, as well as the selections that have to be made to obtain the nodes' minimum costs, stored in S. Note that the selection table S differs from the predecessor table P from Algorithm 1. While S stores the selection that has to be made at time t to obtain a node's minimum cost, P stores a node's predecessor at time $t-1$.

EXTRACT_SCHEDULE uses the selections calculated by SHORTEST_PATHS in order to obtain the sequence of nodes describing a shortest path; thereby, it calculates an optimal schedule for our problem instance.

Algorithm 2 Pseudo-linear-time optimal offline scheduling

```
1: function OPTIMAL_OFFLINE_SCHEDULING($(m, T, \Lambda, \beta, f)$)
2:     $(C, S) \leftarrow$ SHORTEST_PATHS$(m, T, \Lambda, \beta, f)$
3:     $\mathcal{X} \leftarrow$ EXTRACT_SCHEDULE$(S, T)$
4:     return $\mathcal{X}$

5: function SHORTEST_PATHS($(m, T, \Lambda, \beta, f)$)
6:     let $C[0 \dots m, 1 \dots T]$ and $S[0 \dots m, 1 \dots T]$ be new tables
                                                  ▷ Allocate cost and selection tables
7:     $S[m, 1] \leftarrow m$ and $C[m, 1] \leftarrow c(0, m, \lambda_1)$        ▷ Initialize first node in first layer
8:     for $x \leftarrow m - 1$ to $0$ do      ▷ Initialize first layer (downward minimization step)
9:         if $C[x + 1, 1] < c(0, x, \lambda_1)$ then
10:            $S[x, 1] \leftarrow S[x + 1, 1]$ and $C[x, 1] \leftarrow C[x + 1, 1]$
11:        else
12:            $S[x, 1] \leftarrow x$ and $C[x, 1] \leftarrow c(0, x, \lambda_1)$
13:    for $t \leftarrow 1$ to $T - 1$ do                     ▷ Iteratively calculate costs and selections
14:        for $x \leftarrow 1$ to $m$ do                                 ▷ Upward minimization step
15:            if $C[x - 1, t] + \beta < C[x, t]$ then
16:                $S[x, t] \leftarrow S[x - 1, t]$ and $C[x, t] \leftarrow C[x - 1, t] + \beta$
17:        $S[m, t + 1] \leftarrow m$                 ▷ Move to next time slot and initialize first node
18:        $C[m, t + 1] \leftarrow C[m, t] + c_{op}(m, \lambda_{t+1})$
19:        for $x \leftarrow m - 1$ to $0$ do                           ▷ Downward minimization step
20:            if $C[x + 1, t + 1] < C[x, t] + c_{op}(x, \lambda_{t+1})$ then
21:                $S[x, t + 1] \leftarrow S[x + 1, t + 1]$ and $C[x, t + 1] \leftarrow C[x + 1, t + 1]$
22:            else
23:                $S[x, t + 1] \leftarrow x$ and $C[x, t + 1] \leftarrow C[x, t] + c_{op}(x, \lambda_{t+1})$
24:    return $(C, S)$

25: function EXTRACT_SCHEDULE($(S, T)$)
26:    let $\mathcal{X}[1 \dots T]$ be a new array
27:    $\mathcal{X}[T] \leftarrow S[0, T]$                                ▷ Get best selection for last time slot
28:    for $t \leftarrow T - 1$ to $1$ do          ▷ Iteratively obtain schedule from selection table
29:        $\mathcal{X}[t] \leftarrow S\big[\mathcal{X}[t + 1], t\big]$
30:    return $\mathcal{X}$
```

The correctness of Algorithm 2 directly follows from Theorem 4.12 (and the correctness of the shortest path calculation for directed acyclic graphs).

For our runtime analysis, we take the same assumptions as done for Algorithm 1. Subroutine SHORTEST_PATHS needs $\Theta(m)$ steps for its initialization and $\Theta(2Tm)$ steps for the iterative calculation of the selections and costs. In addition, EXTRACT_SCHEDULE requires $\Theta(T)$ iterations for its schedule retrieval. Thus, Algorithm 2 has a time complexity of

$$\Theta(m + 2Tm + T) = \Theta(Tm)$$

The runtime is linear in the numeric value of m and T; however, it is still exponential in the size of the input. The algorithm is thus pseudo-linear, which is an improvement over the pseudo-polynomial complexity $\Theta(Tm^2)$ of Algorithm 1.

Our memory demand is determined by the size of the array $\mathcal{X}$ and the size of the tables C and S. The former is of size $\Theta(T)$, and the latter are of size $\Theta(Tm)$. Thus, Algorithm 2 has a memory complexity of $\Theta(T + 2Tm) = \Theta(Tm)$, which shows that the memory complexity does not change in comparison to Algorithm 1.

Although we have achieved a notable improvement compared to our initial approach, we shall not stop here. Our runtime is, strictly speaking, still exponential; hence, a large value of m may cause undesired long execution times. Our next goal is therefore the reduction of our runtime to sub-exponential complexity.

5 Approximative Offline Scheduling

Heretofore, we have derived two optimal offline algorithms for our scheduling problem. Unfortunately, the algorithms' time complexities are exponential in the input size of the number of servers m. Needless to say, we want to reduce this exponential runtime. For this, we must slightly loosen our aspirations, that is we move to approximative methods. Further, in the course of the next section, we will see that we need to assume that our convex operating cost function f is non-negative and monotonically increasing; however, this restriction is of no great significance in practice, as will be discussed later.

In this chapter, we will first modify our algorithm derived in Section 4.2 to obtain a 2-optimal offline algorithm with linear time complexity. To clarify, given a number $y \in \mathbb{R}$, we say that a scheduling algorithm is *y-optimal* if its calculated solution incurs at most y times as much cost as an optimal solution does. Similarly, we say that a schedule/operation is *y-approximative* if its cost is at most y times as much as much as the original schedule's/operation's cost. As a final step of this thesis, we will generalize the 2-optimal algorithm to derive an $(1+\varepsilon)$-optimal algorithm with linear time complexity.

5.1 A 2-Optimal Linear-Time Algorithm

Recall our algorithm and its corresponding graph G derived in Section 4.2. The algorithm's time complexity of $\Theta(Tm)$ is determined by the number of nodes and edges of G. Since we desire to reduce our runtime complexity, we need to reduce the number of nodes and edges in G. In particular, we must get rid of the factor m. This factor is a consequence of the "height" of our graph, i.e. the number of nodes in each layer. Therefore, we have to "thin out" G by reducing its number of nodes in each layer.

As we saw in Equation (4.4), the size of our input $\mathcal{I}$ is given by $\mathcal{O}\big(T\log_2(m)+\log_2(\beta)\big)$. Consequently, in order to obtain a linear time complexity, we want to reduce the graph's height from $m+1$ to a logarithmic height of $\mathcal{O}\big(\log(m)\big)$. Given this observation, it seems natural for a computer scientist to choose a logarithmic scale for the number of servers in each layer, to wit, instead of adding a node for each possible number of active servers (i.e. $0,1,\dots,m$), we only add nodes for logarithmic choices (i.e. $0,2^0,2^1,\dots,2^{\lfloor\log_2(m)\rfloor},m$).

More formally, given a problem instance $\mathcal{I}$, we set

$$b \coloneqq \lfloor \log_2(m) \rfloor \qquad \text{and} \qquad B \coloneqq \{0, 2^0, 2^1, \dots, 2^b, m\}$$

where B will subsequently represent the set of possible scheduling choices at each time step. Using this set of possible choices, we can then consider the following adaption of our former graph:

$$\begin{aligned}
V &\coloneqq \{v_{0,0}\} \mathbin{\dot\cup} \big\{v_{x,t\downarrow} \mid x \in B, t \in [T]\big\} \mathbin{\dot\cup} \big\{v_{x,t\uparrow} \mid x \in B, t \in [T-1]\big\} \\
E_s &\coloneqq \big\{(v_{0,0}, v_{x,1\downarrow}) \mid x \in B\big\} \\
E_\downarrow &\coloneqq \big\{(v_{m,t\downarrow}, v_{2^b,t\downarrow}) \mid t \in [T]\big\} \mathbin{\dot\cup} \big\{(v_{2^i,t\downarrow}, v_{2^{i-1},t\downarrow}) \mid i \in [b], t \in [T]\big\} \\
&\quad \mathbin{\dot\cup} \big\{(v_{2^0,t\downarrow}, v_{0,t\downarrow}) \mid t \in [T]\big\} \\
E_\uparrow &\coloneqq \big\{(v_{0,t\uparrow}, v_{2^0,t\uparrow}) \mid t \in [T-1]\big\} \mathbin{\dot\cup} \big\{(v_{2^{i-1},t\uparrow}, v_{2^i,t\uparrow}) \mid i \in [b], t \in [T-1]\big\} \\
&\quad \mathbin{\dot\cup} \big\{(v_{2^b,t\uparrow}, v_{m,t\uparrow}) \mid t \in [T-1]\big\} \\
E_{\downarrow\uparrow} &\coloneqq \big\{(v_{x,t\downarrow}, v_{x,t\uparrow}) \mid x \in B, t \in [T-1]\big\} \\
E_{\uparrow\downarrow} &\coloneqq \big\{(v_{x,t\uparrow}, v_{x,t+1\downarrow}) \mid x \in B, t \in [T-1]\big\} \\
E &\coloneqq E_s \mathbin{\dot\cup} E_\downarrow \mathbin{\dot\cup} E_\uparrow \mathbin{\dot\cup} E_{\downarrow\uparrow} \mathbin{\dot\cup} E_{\uparrow\downarrow} \\
c_G(e) &\coloneqq \begin{cases} c(0, x, \lambda_1), & \text{if } e = (v_{0,0}, v_{x,1\downarrow}) \in E_s \\ c_{op}(x, \lambda_{t+1}), & \text{if } e = (v_{x,t\uparrow}, v_{x,t+1\downarrow}) \in E_{\uparrow\downarrow} \\ \beta(x' - x), & \text{if } e = (v_{x,t\uparrow}, v_{x',t\uparrow}) \in E_\uparrow \\ 0, & \text{if } e \in (E_\downarrow \mathbin{\dot\cup} E_{\downarrow\uparrow}) \end{cases} \\
G &\coloneqq (V, E, c_G)
\end{aligned}$$

If m is a power of two, i.e. $m = 2^b$, unnecessary loops with cost 0 are added in $E_\downarrow$ and $E_\uparrow$, which we can simply ignore for our following work. A graphical representation of G can be found in Figure 5.1. The nodes' and edges' semantical meaning and the graph's working principle stays similar to that given in Section 4.2. Again, by following the topological sorting, we can work our way through the graph to calculate the shortest paths, ultimately reaching the destination $v_{0,T\downarrow}$. However, since some possible scheduling choices are not representable in this new graph, we may just obtain approximative costs for our nodes. Thus, the shortest path in our graph might not correspond to an optimal schedule, but it will at least correspond to an approximative one. Before we start to establish the graph's approximation guarantee, we first have to conduct some observations. We start by making a convenient definition that helps us to identify schedules that are representable in our graph.

Definition 5.1 (Restricted schedules). Given an input $\mathcal{I}$ and a set $A \subseteq [m]_0$, we say that a schedule $\mathcal{X} = (x_1, \dots, x_T)$ is *A-restricted* if $\mathcal{X}$ only uses scheduling choices contained in A, that is $\mathcal{X}$ satisfies the formula $\forall t \in [T] : x_t \in A$.

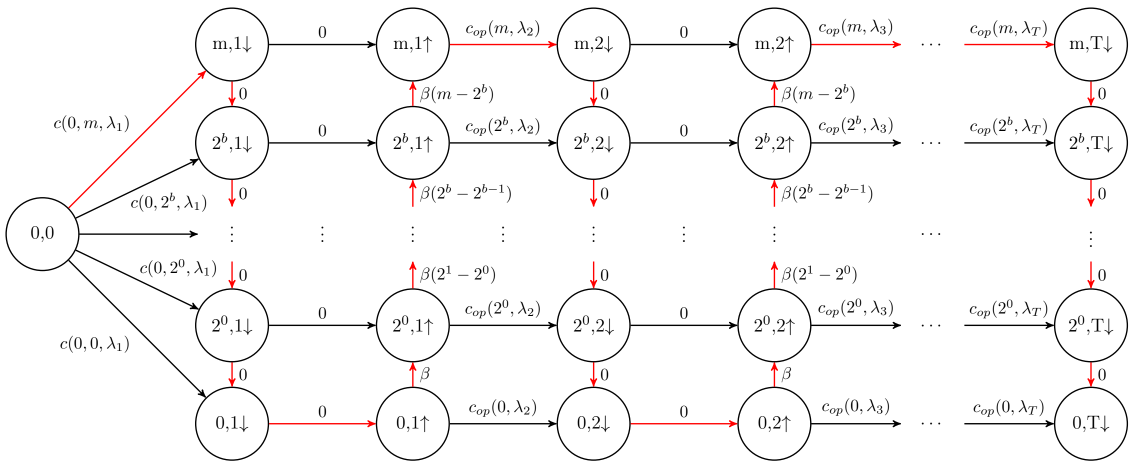


Figure 5.1: Graph for a 2-optimal linear-time offline algorithm; the path of the topological sorting is highlighted in red.

Evidently, our graph is able to represent every B-restricted schedule. We now examine the incurring operating costs of such a B-restricted schedule $\mathcal{X}'$. Since we are forced to schedule a number of servers contained in B, we might not be able to choose an optimal scheduling choice that minimizes the schedule's operating costs. Instead, we may choose the nearest scheduling choice which is contained in B. For instance, if the optimal scheduling strategy at some time slot t would be to choose $x_t = 3$ servers (which is not a power of two), we may instead have to choose $x'_t = 4 \in B$ servers for $\mathcal{X}'$. One might suspect that this strategy would incur at most twice as much operating costs as an optimal schedule does. This, however, is sadly not the case, as one can see in the following example.

Example 5.2. Let $\mathcal{I} = \big(m = 4, T = 5, \Lambda = (3,3,3,3,3), \beta = 0, f\big)$ be the input for a problem instance where $f(\lambda) = (\lambda - 1)^2$. Since we need at least 3 active servers at any time slot, any B-restricted schedule $\mathcal{X}' = (x'_1, \dots, x'_5)$ forces us to constantly use $x'_t = 4$ active machines. An optimal schedule $\mathcal{X} = (x_1, \dots, x_5)$, on the other hand, is able to minimize its cost by constantly scheduling $x_t = 3$ servers. Let us compare the costs between $\mathcal{X}'$ and $\mathcal{X}$. The schedules' costs are given by

$$c(\mathcal{X}) \overset{(3.13)}{=} \sum_{t=1}^{5} \Big(\overbrace{x_t f(\lambda_t/x_t)}^{3(3/3-1)^2} + \overbrace{c_{sw}(x_{t-1}, x_t)}^{0\max\{\cdots\}}\Big) = 5 \cdot 3 \left(\frac{3}{3} - 1\right)^2 = 0$$

$$c(\mathcal{X}') \overset{(3.13)}{=} \sum_{t=1}^{5} \Big(\overbrace{x'_t f(\lambda_t/x'_t)}^{4(3/4-1)^2} + \overbrace{c_{sw}(x'_{t-1}, x'_t)}^{0\max\{\cdots\}}\Big) = 5 \cdot 4 \left(\frac{3}{4} - 1\right)^2 = \frac{20}{16}$$

Albeit our approximative schedule uses only one server in addition, the schedule's cost is already inestimably higher than that of an optimal schedule $\mathcal{X}$, preventing any sensible approximation estimation. Naturally, we may ask ourselves how this explosion of costs is even possible. Evidently, the switching costs are not the root of this explosion since $\beta = 0$. Thus, we shall take a closer look on the used operating cost function. The optimal schedule $\mathcal{X}$ evenly distributes every load $\lambda_t = 3$ to $x_t = 3$ servers. Hence, every active server has to process a load of $\lambda_t/x_t = 1$ at every time step, incurring costs of $f(1) = 0$. On the other hand, the approximative schedule $\mathcal{X}'$ is able to distribute every load to 4 active machines. Thus, every machine incurs costs of $f(3/4) = \frac{1}{16}$. This observation seems rather surprising: Although every server has to process a smaller load using $\mathcal{X}'$, the incurring operating costs of each server turn out to be higher. Intuitively, however, we would expect that a less stressed machine would incur less costs. This surprising behavior is due to the fact that our operating cost function f is not monotonically increasing, as one can see in the following figure.

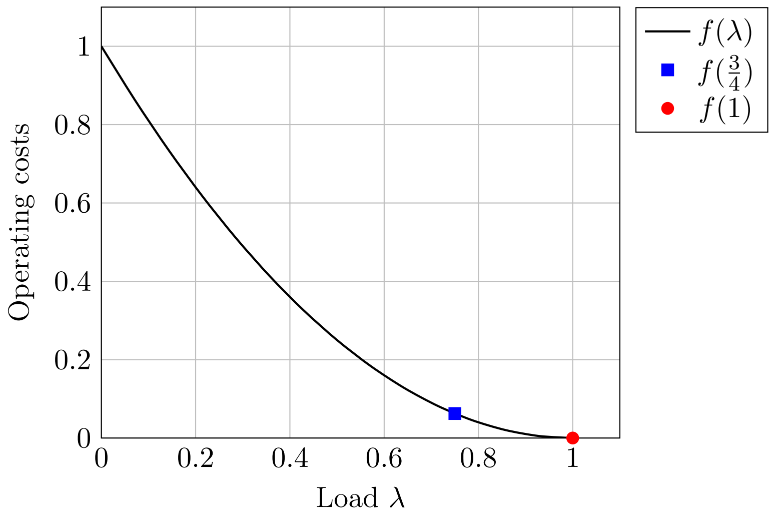


Figure 5.2: Example of a non monotonically increasing operating cost function $f(\lambda) = (\lambda - 1)^2$, where smaller loads incur higher costs.

The above example shows us that our graph may not able to deliver a sensible approximation when dealing with general convex operating cost functions f. Luckily, this inconvenience can be solved by additionally assuming that f is non-negative and monotonically increasing. To see this, assume that at some time slot t the scheduling choice x_t minimizes the operating costs to process the load λ_t. Then let $x'_t \in B$ the next scheduling choice representable in G. Since x_t minimizes the operating costs at time slot t, we have

$$c_{op}(x_t, \lambda_t) \leq c_{op}(x'_t, \lambda_t) \overset{(3.9)}{=} x'_t f(\lambda_t/x'_t)$$

Further, since B contains all powers of two up to m, we have $x_t \le x'_t \le 2x_t$. If we additionally assume that f is non-negative, we can infer that

$$x'_t f(\lambda_t / x'_t) \le 2x_t f(\lambda_t / x'_t)$$

Now, using the fact that $x_t \le x'_t$ and assuming that f is monotonically increasing, we can see that

$$2x_t f(\lambda_t / x'_t) \le 2x_t f(\lambda_t / x_t) \overset{(3.9)}{=} 2c_{op}(x_t, \lambda_t)$$

Ultimately, we can combine our observations and conclude

$$c_{op}(x_t, \lambda_t) \le c_{op}(x'_t, \lambda_t) \le 2c_{op}(x_t, \lambda_t)$$

which shows us that our approximative scheduling choice incurs at most twice as much operating costs as an optimal scheduling strategy does. We thus subsequently restrict ourselves to non-negative, monotonically increasing convex cost functions. This evidently reduces the theoretical generality of our initial approach, but it does not interfere with practical applicability. On the one hand, negative cost functions would semantically allow to "generate profit by consuming energy", which seems unreasonable in practice. On the other hand, if we have a non monotonically increasing convex cost function, we can simply add artificial loads to our machines to reduce our costs in given circumstances. For instance, in our previous example, we could assign every machine a load of 1 instead of $\frac{3}{4}$ to reduce the approximative schedule's cost. This trick, which is exemplarily outlined in Figure 5.3, allows us to transform any arbitrary convex cost function to a monotonically increasing one.

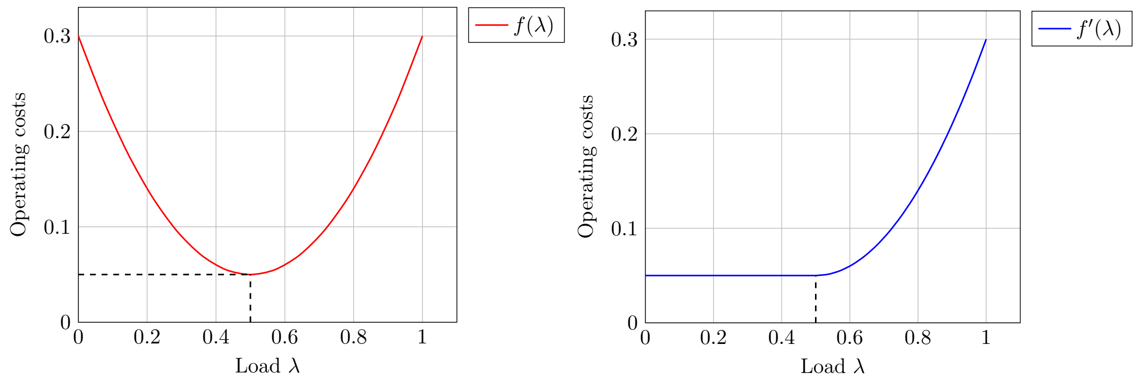


Figure 5.3: The non monotonically increasing convex function f can be transformed to the monotonically increasing convex function f' by adding artificial loads λ^+ to assignments $\lambda \in [0, 0.5)$ such that we obtain a new assignment $\lambda' \coloneqq \lambda + \lambda^+ = 0.5$.

Next, we examine the incurring switching costs of a B-restricted schedule $\mathcal{X}'$. Again, given an optimal scheduling choice x_t, we use the idea to choose the nearest scheduling choice x'_t

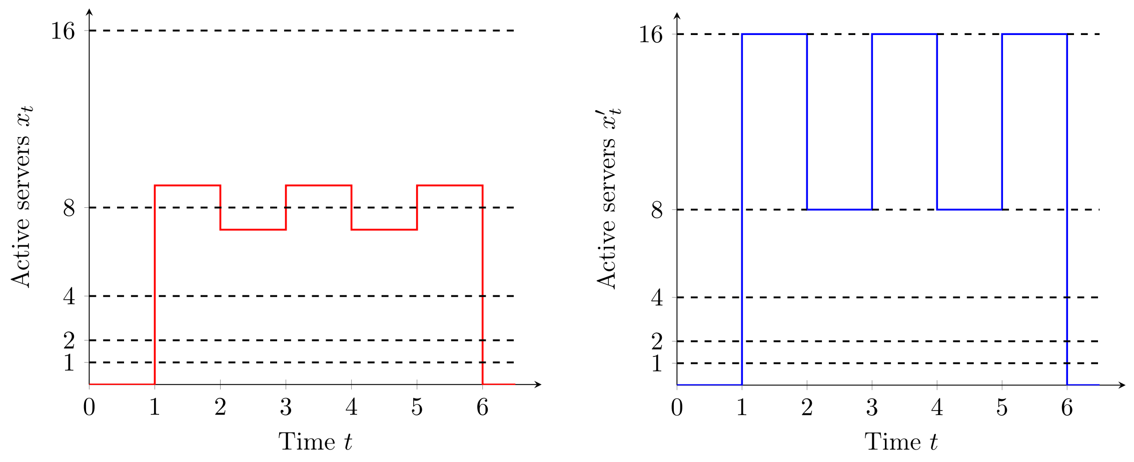
that is contained in B to construct $\mathcal{X}'$. Once more, one might hope that $\mathcal{X}'$ would incur at most twice as much switching costs as an optimal schedule. Needless to say, the next example dashes this hope.

Example 5.3. Let $\mathcal{I} = \big(m = 16, T = 5, \Lambda = (9,7,9,7,9), \beta = 1, f\big)$ be the input for a problem instance where $f(\lambda) = 0$. Our possible scheduling choices are then given by $B = \{0,1,2,4,8,16\}$, and one optimal schedule is given by $\mathcal{X} = (9,7,9,7,9)$. The B-restricted schedule corresponding to $\mathcal{X}$ is then given by $\mathcal{X}' = (16,8,16,8,16)$. The schedules' costs amount to

$$c(\mathcal{X}) \overset{(3.13)}{=} \sum_{t=1}^{5} \Big(\overbrace{x_t f(\lambda_t/x_t)}^{x_t \cdot 0} + \overbrace{c_{sw}(x_{t-1}, x_t)}^{\max\{0, x_t - x_{t-1}\}} \Big) = 9 + 0 + 2 + 0 + 2 = 13$$

$$c(\mathcal{X}') \overset{(3.13)}{=} \sum_{t=1}^{5} \Big(\overbrace{x'_t f(\lambda_t/x'_t)}^{x'_t \cdot 0} + \overbrace{c_{sw}(x'_{t-1}, x'_t)}^{\max\{0, x'_t - x'_{t-1}\}} \Big) = 16 + 0 + 8 + 0 + 8 = 32$$

which shows that $\mathcal{X}'$ is not 2-approximative. Of course, we are again curious about how this cost explosion is possible. Since we set $f(\lambda) = 0$, our servers do not incur operating costs, which means that the cost explosion must be due to the increased switching costs of $\mathcal{X}'$. The problem in this case is the oscillating behavior of $\mathcal{X}$ around a power of two (namely $8 = 2^3$), as one can see in the following figure.



Optimal schedule $\mathcal{X}$: Note how the schedule oscillates around 8 (a power of two).

Approximative schedule $\mathcal{X}'$: The approximative schedule is restricted to powers of two.

Figure 5.4: Comparison between an optimal schedule and its approximative counterpart. The approximative schedule incurs more than twice as much switching costs.

So, is this the end of our hunt for a 2-optimal algorithm? No, certainly not! Although our naive approach was to no avail, there is indeed a better B-restricted schedule for our example. Instead of following the optimal schedule's oscillation, we can simply use a schedule that stays put during these oscillating steps, namely the schedule $\mathcal{X}' = (16, 16, 16, 16, 16)$ with cost $c(\mathcal{X}') = 16$. Obviously, this seems like a rather trivial example since we set $f(\lambda) = 0$, and hence we do not need to worry about the new schedule's operating costs. However, it indeed turns out that making the right choice between following the optimal schedule's oscillation and staying put will always allow us to acquire a 2-optimal solution. This observation will be a key part of the next lemma's proof.

In the proof, we are going to divide a schedule $\mathcal{X}$ into periods at which its plot crosses a power of two. We then show that every such period can be transformed to a period in a B-restricted schedule $\mathcal{X}'$ such that the transformed period incurs at most twice as much costs. In order to formalize how to exactly split our schedules, we make a handy definition.

Definition 5.4 (2-state changes)**.** Let $\mathcal{X} = (x_1, \dots, x_T)$ be a schedule and $t \in [T+1]$. We say that $\mathcal{X}$ changes its *2-state* at time t if $\mathcal{X}$ satisfies the formula

$$x_{t-1} \neq x_t \wedge \left(x_{t-1} = 0 \vee x_t \notin \left[2^{\lfloor \log_2(x_{t-1}) \rfloor}, 2^{\lfloor \log_2(x_{t-1}) \rfloor + 1}\right)\right)$$

For example, the schedule $\mathcal{X} = (5, 4, 8, 7, 8, 15, 10, 16)$ changes its 2-state at times $t \in \{1, 3, 4, 5, 8, 9\}$. As one can see in Figure 5.5, we can say that $\mathcal{X}$ changes its 2-state if its plot ascends and touches a next higher power of two or descends and leaves the current pair of powers of two. Using this notion of 2-state changes, we are geared up to deal with the next lemma – the main work of this section.

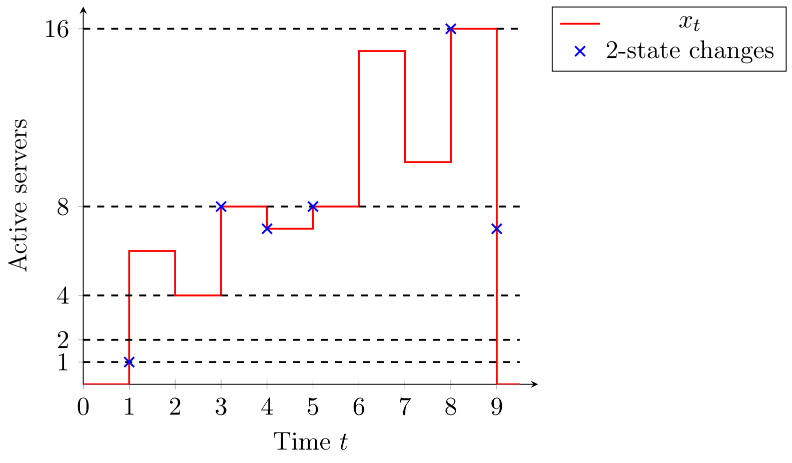


Figure 5.5: Plot of the schedule $\mathcal{X} = (5, 4, 8, 7, 8, 15, 10, 16)$ and its 2-state changes

Lemma 5.5. *Let $\mathcal{X}$ be a schedule for $\mathcal{I}$. There exists a B-restricted schedule $\mathcal{X}'$ satisfying $c(\mathcal{X}') \leq 2c(\mathcal{X})$.*

Proof. Let $\mathcal{X} = (x_1, \dots, x_T)$ be a schedule for $\mathcal{I}$. We need to construct a B-restricted schedule $\mathcal{X}' = (x'_1, \dots, x'_T)$ such that $c(\mathcal{X}') \leq 2c(\mathcal{X})$. First, if $\mathcal{X}$ is not feasible, we have $c(\mathcal{X}) = \infty$, and thus any arbitrary B-restricted schedule $\mathcal{X}'$ satisfies $c(\mathcal{X}') \leq 2c(\mathcal{X})$; hence, assume that $\mathcal{X}$ is feasible. Next, we notice that if $\mathcal{X}$ shuts down all servers at some time slot $t \in [T]$ (i.e. $x_t = 0$), we can split $\mathcal{X}$ into two subschedules $\mathcal{X}_1 \coloneqq (x_1, \dots, x_{t-1})$ and $\mathcal{X}_2 \coloneqq (x_{t+1}, \dots, x_T)$. It then suffices to prove the claim for $\mathcal{X}_1$ and $\mathcal{X}_2$, since we can then construct the 2-approximative schedule by setting $x'_t \coloneqq 0$ and $\mathcal{X}' \coloneqq (\mathcal{X}'_1, x'_t, \mathcal{X}'_2)$. Thus, by recursively applying this method, we can reduce our proof to a list of subschedules $\mathcal{X}_1, \dots, \mathcal{X}_N$ that never shut down all servers. Consequently, without loss of generality, we subsequently assume that $\mathcal{X}$ never powers down all servers, that is $x_t > 0$ for all $t \in [T]$.

Next, we show that we can iteratively construct $\mathcal{X}'$ by transforming every period between two 2-state changes of $\mathcal{X}$ to a 2-approximative period in $\mathcal{X}'$. To prove that our transformations will be 2-approximative, we have to conduct an amortized analysis for the switching costs of $\mathcal{X}'$ using the *accounting method.* The basic idea of the accounting method is to overcharge some operations and to save the excess charge as a *credit*, which can be used to compensate for subsequent, more expensive operations. An introduction about the accounting method can be found in [4, Section 17.2]. To see the necessity of such an amortized analysis, and to get a basic idea about its working principle, consider the schedule $\mathcal{X} = (7, 9)$ and its approximative counterpart $\mathcal{X}' = (8, 16)$. Although the total switching costs of $\mathcal{X}'$ are 2-approximative, the individual switching steps of $\mathcal{X}'$ are not, since $\mathcal{X}'$ has to turn on 8 machines at time $t = 2$ while $\mathcal{X}$ only turns on 2 machines. However, at time $t = 1$, $\mathcal{X}'$ only turns on 8 servers while it would be allowed to turn on $2 \cdot 7 = 14$. We can thus overcharge the first switching operation and use the excess charge $14 - 8 = 6$ as a credit to compensate for the second switching step, which misses $8 - 2 \cdot 2 = 4$ credit points.

For our iterative construction, let i and $j + 1$ with $i, j \in [T]$ and $i \leq j$ be the first unprocessed time slots at which $\mathcal{X}$ changes its 2-state. To conveniently refer to the schedules' periods between i and j, we define the subschedules $\mathcal{X}_{i,j} \coloneqq (x_i, \dots, x_j)$ and $\mathcal{X}'_{i,j} \coloneqq (x'_i, \dots, x'_j)$. We then have to show that $\mathcal{X}_{i,j}$ can be transformed to $\mathcal{X}'_{i,j}$ with 2-approximative costs. Note that the periods are consecutive, i.e. after processing the period $[i, j]$, the next pair of indices i', j' will be chosen such that $j + 1 = i'$. To conduct the transformations, we will need to refer to the lower and upper bound of $\mathcal{X}_{i,j}$, namely

$$l \coloneqq 2^{\lfloor \log_2(x_i) \rfloor} \qquad \text{and} \qquad u \coloneqq \min\{2l, m\}$$

as well as to the lower and upper bound of $\mathcal{X}$ at time $j+1$:

$$l' := \begin{cases} 2^{\lfloor \log_2(x_{j+1}) \rfloor}, & \text{if } x_{j+1} \neq 0 \\ 0, & \text{if } x_{j+1} = 0 \end{cases} \qquad \text{and} \qquad u' := \min\{2l', m\}$$

Note that $u, u' \in B$ and that $x_t \leq u \leq 2x_t$ holds for any $i \leq t \leq j$ since $\mathcal{X}$ does not change its 2-state between i and j. More precisely, we know that $x_t \in [l, u)$ if $u = 2l$, and $x_t \in [l, u]$ if $u = m < 2l$, for $i \leq t \leq j$. To conveniently combine both cases at some points, given two boundaries $a, b \in \mathbb{N}$, we define the conditional interval

$$[a, b)^m := \begin{cases} [a, b), & \text{if } b = 2a \\ [a, b], & \text{if } b = m < 2a \end{cases}$$

As an *invariant* of the following transformations, we are going to ensure that $\mathcal{X}'$ can potentially move to u at time i in a 2-approximative manner, i.e. we ensure that $\mathcal{X}'$ incurs at most twice as much switching costs as $\mathcal{X}$ if we set $x'_i := u$, whether this step will be taken in the end or not. Further, we are going to ensure that $x'_t \geq u$ holds for any $i \leq t \leq j$ after every transformation step. This guarantees that $\mathcal{X}'$ will be feasible, since $x_t \leq u$. Moreover, we show that the credit of our amortized analysis will be greater than or equal to $\beta(2x_{j+1} - u')$ if $u' \neq m$ and $\mathcal{X}$ exits $[l, u)$ by ascending to $[l', u')$ at time $j+1$. Since the periods are consecutive, this equivalently means that the credit will be greater than or equal to $\beta(2x_i - u)$ if $u \neq m$ and $\mathcal{X}$ enters $[l, u)$ at time i from below.

First, we have to check that our invariant initially holds. For the initial start-up process (i.e. $i = 1$), we can simply move to the next power of 2 larger than x_1 contained in B, that is we set $x'_1 := u \in B$. Since we know that $u \leq 2x_1$, we can conclude that $\beta x'_1 \leq \beta 2x_1$, which shows that $\mathcal{X}'$ has 2-approximative start-up switching costs. Moreover, we can use the difference of switching costs $\beta(2x_1 - x'_1) = \beta(2x_1 - u)$ for the credit of our amortized analysis. Thus, the invariant initially holds.

Now, let us have a closer look on the possible behaviors of $\mathcal{X}$ between its 2-state changes. The behaviors can be classified based on how $\mathcal{X}$ enters and leaves the interval $[l, u)^m$. We have to consider four different cases:

(a) $\mathcal{X}$ enters $[l, u)^m$ from below and then descends to $[l', u')$.

(b) $\mathcal{X}$ enters $[l, u)$ from above and then descends to $[l', u')$.

(c) $\mathcal{X}$ enters $[l, u)$ from below and then ascends to $[l', u')^m$.

(d) $\mathcal{X}$ enters $[l, u)$ from above and then ascends to $[l', u')^m$.

To finish the proof, we show that any case can be transformed to a 2-approximative period in $\mathcal{X}'$ while ensuring that our invariant will not be violated. Additionally, we have to verify that the credit of our amortized analysis stays non-negative.

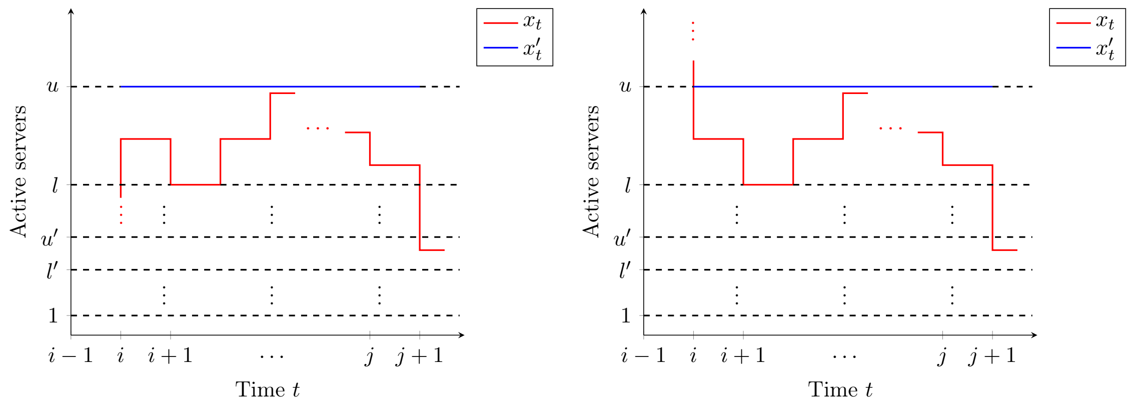


Figure 5.6: The original schedule comes from below (case (a)) or above (case (b)), stays between $[l, u)^m$, and then descends to $[l', u')$. The approximative schedule stays put at u for time slots $i \le t \le j$.

We begin with cases (a) and (b), which are both depicted in Figure 5.6. Due to our invariant, we know that we can set $x'_i := u$ with 2-approximative switching costs. Consequently, setting $x'_i := x'_{i+1} := \cdots := x'_j := u \in B$ gives us a strategy with 2-approximative switching costs between i and j. Further, since $x_t \le u \le 2x_t$ for any $i \le t \le j$, and f is non-negative and monotonically increasing, the operating costs of $\mathcal{X}'_{i,j}$ can be estimated by

$$c_{op}\left(\mathcal{X}'_{i,j}\right) \overset{(3.10)}{=} \sum_{t=i}^{j} \left(uf\left(\frac{\lambda_t}{u}\right)\right) \le \sum_{t=i}^{j} \left(2x_t f\left(\frac{\lambda_t}{u}\right)\right) \le 2 \sum_{t=i}^{j} \left(x_t f\left(\frac{\lambda_t}{x_t}\right)\right) \overset{(3.10)}{=} 2c_{op}\left(\mathcal{X}_{i,j}\right)$$

Thus, the operating costs of $\mathcal{X}'_{i,j}$ are 2-approximative. Further, since $u' < u$, we do not need to account for additional switching costs to satisfy our invariant at time $j+1$; however, we want to stress that one cannot tell at this step if we should indeed set $x_{j+1} := u'$ (cf. Figure 5.8 and its related case). Lastly, we note that the credit of our amortized analysis stays untouched in both cases, i.e. the credit stays non-negative.

Next, we consider case (c), which is illustrated in Figure 5.7. Again, due to our invariant, we know that setting $x'_i := x'_{i+1} := \cdots := x'_j := u \in B$ gives us a strategy with 2-approximative switching costs. Moreover, as in the previous case, the operating costs of $\mathcal{X}'_{i,j}$ are 2-approximative. To verify the invariant at time $j+1$, we have to show that $\mathcal{X}'$ can potentially move to u' at time $j+1$ with 2-approximative costs, and, since $\mathcal{X}$ ascends at time $j+1$, we also need to account for the new credit required by the invariant. To compensate for these costs, we notice that $\mathcal{X}$ has to power on at least $x_{j+1} - x_i$ servers between i and $j+1$. Further, as $\mathcal{X}$ ascends at time i, we can use the old credit guaranteed

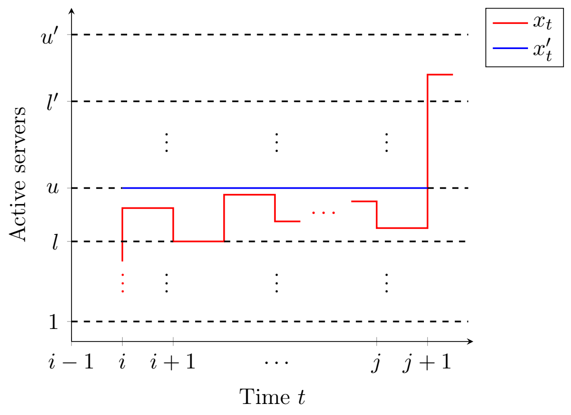


Figure 5.7: The original schedule (case (c)) comes from below, stays between $[l, u)$, and then ascends to $[l', u')^m$. The approximative schedule stays put at u for time slots $i \le t \le j$.

by our invariant to compensate for the costs. Putting it all together, we have

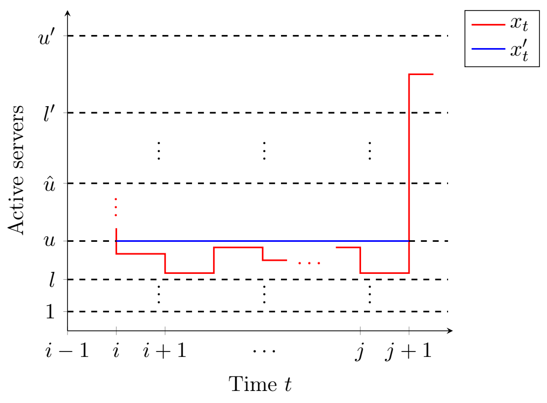
$$\begin{aligned}
&2\,\overbrace{\beta(x_{j+1}-x_i)}^{\text{sw. costs }\mathcal{X}} + \overbrace{\beta(2x_i-u)}^{\text{old credit}} - \overbrace{\beta(u'-u)}^{\text{sw. costs }\mathcal{X}'} - \overbrace{\beta(2x_{j+1}-u')}^{\text{new credit}} \\
= \;& \beta(2x_{j+1} - 2x_i + 2x_i - u - u' + u - 2x_{j+1} + u') \\
= \;& 0
\end{aligned}$$

that is, the cost difference is non-negative, and thus our invariant at time $j+1$ is satisfied.

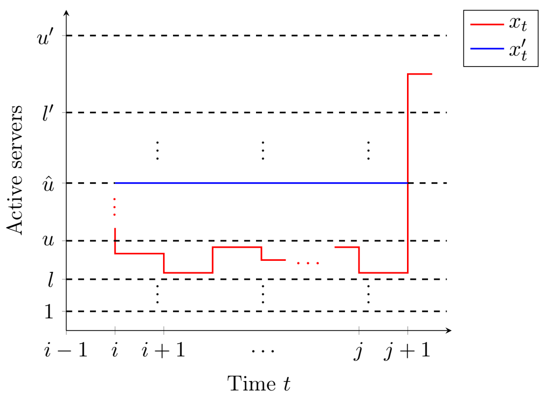
Finally, we have to consider case (d). As we have seen in Example 5.3, this case turns out to be slightly more complicated, since simply following an oscillating behavior of $\mathcal{X}$ can lead to a cost explosion for $\mathcal{X}'$. Nevertheless, we can solve this issue by considering two different strategies for $\mathcal{X}'_{i,j}$, namely

$$\begin{aligned}
&\mathcal{X}^u_{i,j} := \left(x^u_i, \ldots, x^u_j\right), &&\text{where} && x^u_t := u, \text{ for } i \le t \le j \\
\text{and}\quad &\mathcal{X}^{\hat{u}}_{i,j} := \left(x^{\hat{u}}_i, \ldots, x^{\hat{u}}_j\right), &&\text{where} && x^{\hat{u}}_t := \hat{u}, \text{ for } i \le t \le j
\end{aligned}$$

with $\hat{u} := \min\{2u, m\} \in B$. Both strategies are illustrated in Figure 5.8. Due to our invariant and the fact that $\mathcal{X}$ descends at time i, we know that $x'_{i-1} \ge \hat{u} \ge u$, and thus both strategies do no incur switching costs up to time j (we can simply move down from x'_{i-1} to $\hat{u}$ or u). We are now going to prove that either the cost of $\mathcal{X}^u_{i,j}$ or of $\mathcal{X}^{\hat{u}}_{i,j}$, including costs to satisfy our invariant at time $j+1$, must be 2-approximative. First, we examine the switching costs of $\mathcal{X}$ from i up to $j+1$. To do so, we set $d := \min\{x_t \mid i \le t \le j\}$ to refer to the smallest number of active servers used by $\mathcal{X}_{i,j}$. Then, since $\mathcal{X}$ has to power on at



Strategy $\mathcal{X}^{u}_{i,j}$ stays put at u for $i \leq t \leq j$.



Strategy $\mathcal{X}^{\hat{u}}_{i,j}$ stays put at $\hat{u}$ for $i \leq t \leq j$.

Figure 5.8: The original schedule (case (d)) comes from above, stays between $[l, u)$, and then rises to $[l', u')^m$. The approximative schedule has two different possibilities.

least $x_{j+1} - d$ servers between i and $j + 1$, we can give a lower bound for its switching costs:

$$\sum_{t=i+1}^{j+1} c_{sw}(x_{t-1}, x_t) \geq \beta(x_{j+1} - d)$$

The next idea is to split $\mathcal{X}_{i,j}$ and its associated switching costs at the final switching step into two parts; more precisely, we split $\beta(x_{j+1} - d)$ into $\beta(x_{j+1} - u)$ and $\beta(u - d)$. It then suffices to show that either $\mathcal{X}^{u}_{i,j}$ or $\mathcal{X}^{\hat{u}}_{i,j}$ can process the loads $\lambda_i, \ldots, \lambda_j$ and eventually switch to $\hat{u}$ with 2-approximative costs under the assumption that $\mathcal{X}$ only moves to u as a first step. This is because the remaining switching costs $\beta(x_{j+1} - u)$ of $\mathcal{X}$ already suffice to compensate for the remaining switching costs $\beta(u' - \hat{u})$ of $\mathcal{X}'$ and the costs to satisfy the invariant:

$$\begin{aligned} 2\beta(x_{j+1} - u) - \beta(u' - \hat{u}) - \overbrace{\beta(2x_{j+1} - u')}^{\text{new credit}} &= \beta(2x_{j+1} - 2u - u' + \hat{u} - 2x_{j+1} + u') \\ &= \beta(\hat{u} - 2u) \\ &= 0 \end{aligned}$$

where we assumed that $\hat{u} = 2u$ holds; otherwise, we have $u' = \hat{u} = m$ and thus do not need to account for the new credit to satisfy our invariant (recall that we require the credit only if $u \neq m$), which allows us to infer that

$$2\beta(x_{j+1} - u) - \beta(u' - \hat{u}) = 2\beta(x_{j+1} - u) - \beta(m - m) = 2\beta(x_{j+1} - u) \geq 0$$

To proceed, we notice that if $\hat{u} - u \leq 2(u - d)$ holds, strategy $\mathcal{X}^{u}_{i,j}$ has 2-approximative switching costs:

$$\beta(\hat{u} - u) \leq 2\beta(u - d)$$

Further, as in the previous cases, the operating costs of $\mathcal{X}^u_{i,j}$ are 2-approximative, and thus $\mathcal{X}^u_{i,j}$ is 2-approximative if $\hat{u} - u \leq 2(u-d)$ holds. Hence, we subsequently assume that

$$\hat{u} - u > 2(u-d) \qquad \text{or equivalently} \qquad \hat{u} - 3u + 2d > 0 \tag{5.6}$$

Next, by the law of excluded middle, the schedule $\mathcal{X}^u_{i,j}$ is either 2-approximative, or it is not 2-approximative. If it is 2-approximative, we are done; hence, from now on, we assume that $\mathcal{X}^u_{i,j}$ is not 2-approximative, that is

$$c_{op}\big(\mathcal{X}^u_{i,j}\big) + \beta(\hat{u} - u) > 2\left(c_{op}\big(\mathcal{X}_{i,j}\big) + \beta(u-d)\right)$$

We then have to show that $\mathcal{X}^{\hat{u}}_{i,j}$ is 2-approximative. First note that $\mathcal{X}^u_{i,j}$ uses at most twice as many active servers as $\mathcal{X}_{i,j}$, and therefore $\mathcal{X}^u_{i,j}$ incurs at most twice as much operating costs as $\mathcal{X}_{i,j}$. Consequently, the switching costs of $\mathcal{X}^u_{i,j}$ must significantly outweigh those of $\mathcal{X}_{i,j}$. Hence, it seems interesting to get an estimation for β. Rearranging the previous inequality gives us

$$\begin{aligned} & & c_{op}\big(\mathcal{X}^u_{i,j}\big) + \beta(\hat{u} - u) &> 2\left(c_{op}\big(\mathcal{X}_{i,j}\big) + \beta(u-d)\right) \\ &\iff & \beta(\hat{u} - 3u + 2d) &> 2c_{op}\big(\mathcal{X}_{i,j}\big) - c_{op}\big(\mathcal{X}^u_{i,j}\big) \\ &\overset{\hat{u}-3u+2d>0}{\iff} & \beta &> \frac{2c_{op}\big(\mathcal{X}_{i,j}\big) - c_{op}\big(\mathcal{X}^u_{i,j}\big)}{\hat{u} - 3u + 2d} \end{aligned}$$

where the last step is justified by Assumption (5.6). This allows us to find a lower bound for the cost of $\mathcal{X}_{i,j}$:

$$\begin{aligned} c_{op}\big(\mathcal{X}_{i,j}\big) + \beta(u-d) &\geq c_{op}\big(\mathcal{X}_{i,j}\big) + \frac{2c_{op}\big(\mathcal{X}_{i,j}\big) - c_{op}\big(\mathcal{X}^u_{i,j}\big)}{\hat{u} - 3u + 2d}(u-d) \\ &= \frac{(\hat{u} - 3u + 2d)c_{op}\big(\mathcal{X}_{i,j}\big) + 2(u-d)c_{op}\big(\mathcal{X}_{i,j}\big) - (u-d)c_{op}\big(\mathcal{X}^u_{i,j}\big)}{\hat{u} - 3u + 2d} \\ &= \frac{(\hat{u} - u)c_{op}\big(\mathcal{X}_{i,j}\big) - (u-d)c_{op}\big(\mathcal{X}^u_{i,j}\big)}{\hat{u} - 3u + 2d} \end{aligned}$$

Next, since we want to prove that $\mathcal{X}^{\hat{u}}_{i,j}$ is 2-approximative, we have to show that the following cost difference is non-negative:

$$2\left(c_{op}\big(\mathcal{X}_{i,j}\big) + \beta(u-d)\right) - c_{op}\big(\mathcal{X}^{\hat{u}}_{i,j}\big)$$

Recall that $\mathcal{X}^{\hat{u}}_{i,j}$ does not incur switching costs to move to $\hat{u}$, and thus we only have to consider its operating costs. We can now use our just derived lower bound for $c_{op}\big(\mathcal{X}_{i,j}\big) + \beta(u-d)$ to estimate the difference:

$$2\left(c_{op}\big(\mathcal{X}_{i,j}\big) + \beta(u-d)\right) - c_{op}\big(\mathcal{X}^{\hat{u}}_{i,j}\big) \geq 2\,\frac{(\hat{u} - u)c_{op}\big(\mathcal{X}_{i,j}\big) - (u-d)c_{op}\big(\mathcal{X}^u_{i,j}\big)}{\hat{u} - 3u + 2d} - c_{op}\big(\mathcal{X}^{\hat{u}}_{i,j}\big)$$

Thus, it suffices to show that

$$2\,\frac{(\hat{u}-u)c_{op}\big(\mathcal{X}_{i,j}\big)-(u-d)c_{op}\big(\mathcal{X}^{u}_{i,j}\big)}{\hat{u}-3u+2d}-c_{op}\big(\mathcal{X}^{\hat{u}}_{i,j}\big)\geq 0$$

which, using Assumption (5.6), that is $\hat{u}-3u+2d>0$, can be simplified to

$$2(\hat{u}-u)c_{op}\big(\mathcal{X}_{i,j}\big)-2(u-d)c_{op}\big(\mathcal{X}^{u}_{i,j}\big)-(\hat{u}-3u+2d)c_{op}\big(\mathcal{X}^{\hat{u}}_{i,j}\big)\geq 0$$

To show this inequality, we have to take a closer look on the schedules' operating costs:

$$\begin{aligned}&2(\hat{u}-u)c_{op}\big(\mathcal{X}_{i,j}\big)-2(u-d)c_{op}\big(\mathcal{X}^{u}_{i,j}\big)-(\hat{u}-3u+2d)c_{op}\big(\mathcal{X}^{\hat{u}}_{i,j}\big)\\ \overset{(3.10)}{=}\quad&\sum_{t=i}^{j}\left(2(\hat{u}-u)x_tf\left(\frac{\lambda_t}{x_t}\right)-2(u-d)uf\left(\frac{\lambda_t}{u}\right)-(\hat{u}-3u+2d)\hat{u}f\left(\frac{\lambda_t}{\hat{u}}\right)\right)\end{aligned}$$

First, we notice that $u\leq\hat{u}$ and $x_t<u$ for $i\leq t\leq j$. Together with the fact that f is monotonically increasing, we infer that

$$\begin{aligned}&\sum_{t=i}^{j}\left(2(\hat{u}-u)x_tf\left(\frac{\lambda_t}{x_t}\right)-2(u-d)uf\left(\frac{\lambda_t}{u}\right)-(\hat{u}-3u+2d)\hat{u}f\left(\frac{\lambda_t}{\hat{u}}\right)\right)\\ \geq\quad&\sum_{t=i}^{j}\left(2(\hat{u}-u)x_tf\left(\frac{\lambda_t}{u}\right)-2(u-d)uf\left(\frac{\lambda_t}{u}\right)-(\hat{u}-3u+2d)\hat{u}f\left(\frac{\lambda_t}{u}\right)\right)\end{aligned}$$

Since d is the smallest number of active servers scheduled by $\mathcal{X}$, and f is non-negative, we can conclude that

$$\begin{aligned}&\sum_{t=i}^{j}\left(2(\hat{u}-u)x_tf\left(\frac{\lambda_t}{u}\right)-2(u-d)uf\left(\frac{\lambda_t}{u}\right)-(\hat{u}-3u+2d)\hat{u}f\left(\frac{\lambda_t}{u}\right)\right)\\ \geq\quad&\sum_{t=i}^{j}\left(2(\hat{u}-u)df\left(\frac{\lambda_t}{u}\right)-2(u-d)uf\left(\frac{\lambda_t}{u}\right)-(\hat{u}-3u+2d)\hat{u}f\left(\frac{\lambda_t}{u}\right)\right)\end{aligned}$$

Hence, to finish the case, it suffices to show that

$$\sum_{t=i}^{j}\left(2(\hat{u}-u)df\left(\frac{\lambda_t}{u}\right)-2(u-d)uf\left(\frac{\lambda_t}{u}\right)-(\hat{u}-3u+2d)\hat{u}f\left(\frac{\lambda_t}{u}\right)\right)\geq 0$$

Further rearranging the left hand side gives us

$$
\begin{aligned}
& \sum_{t=i}^{j} \left(2(\hat{u}-u)df\left(\frac{\lambda_t}{u}\right) - 2(u-d)uf\left(\frac{\lambda_t}{u}\right) - (\hat{u}-3u+2d)\hat{u}f\left(\frac{\lambda_t}{u}\right) \right) \\
= \quad & \sum_{t=i}^{j} \left(\Big(2(\hat{u}-u)d - 2(u-d)u - (\hat{u}-3u+2d)\hat{u} \Big) f\left(\frac{\lambda_t}{u}\right) \right) \\
= \quad & \sum_{t=i}^{j} \left(\Big(2d\hat{u} - 2du - 2u^2 + 2du - \hat{u}^2 + 3u\hat{u} - 2d\hat{u} \Big) f\left(\frac{\lambda_t}{u}\right) \right) \\
= \quad & \sum_{t=i}^{j} \left(\left(-\hat{u}^2 + 3u\hat{u} - 2u^2 \right) f\left(\frac{\lambda_t}{u}\right) \right) = \sum_{t=i}^{j} \left(\underbrace{-(u-\hat{u})(2u-\hat{u})}_{g(\hat{u})} f\left(\frac{\lambda_t}{u}\right) \right)
\end{aligned}
$$

Now let $g(\hat{u}) := -(u-\hat{u})(2u-\hat{u})$. Since f is non-negative, it suffices to show that $g(\hat{u}) \geq 0$ for all possible values of $\hat{u}$, namely $\hat{u} \in [u, 2u]$. We notice that $g(\cdot)$ is an inverted parabola with roots u and $2u$. Thus, we know that $g(u) = g(2u) = 0$ and $g(\hat{u}) > 0$ for $u < \hat{u} < 2u$, which finishes the case.

By iteratively applying above procedure to $\mathcal{X}$, starting with $i = 1$ and stopping with $j = T + 1$, we obtain a B-restricted schedule $\mathcal{X}'$ that satisfies $c(\mathcal{X}') \leq 2c(\mathcal{X})$. □

Safe in the knowledge that there always exists a 2-approximative B-restricted schedule $\mathcal{X}'$ for any schedule $\mathcal{X}$, we are just left with the verification of our modified graph. Since we did not change the graph's general construction idea, this verification turns out to be very similar to that done in Section 4.2. For the proof of the next lemma, which establishes the bijection between B-restricted schedules and reasonable paths, we need to recall our notion of "maximum" nodes x_t in reasonable paths P, as described in Definition 4.10.

Lemma 5.7. *Let $\boldsymbol{\mathcal{X}}$ be the set of all B-restricted schedules for $\mathcal{I}$, and let $\boldsymbol{\mathcal{P}}$ be the set of all reasonable paths. The map*

$$\Phi : \boldsymbol{\mathcal{P}} \to \boldsymbol{\mathcal{X}}, \quad P \mapsto (x_1, \ldots, x_T)$$

is a bijection satisfying $c(P) = c\big(\Phi(P)\big)$.

Proof. The proof is analogous to the proof of Lemma 4.11 considering that we conduct logarithmic instead of incremental switching steps. □

Finally, we arrive at the main result of this section.

Theorem 5.8. *Any shortest reasonable path P corresponds to a 2-optimal, B-restricted schedule $\mathcal{X}$ for $\mathcal{I}$ with $c(P) = c(\mathcal{X})$.*

Proof. By Lemma 5.7, we have a bijection Φ between reasonable paths P and B-restricted schedules obeying $c(P) = c\big(\Phi(P)\big)$. Thus, we have

$$c(P) \text{ minimal } \iff c\big(\Phi(P)\big) \text{ minimal}$$

Now let P be a shortest reasonable path, and let $\mathcal{X}^*$ be an optimal schedule for $\mathcal{I}$. We have to verify that $c\big(\Phi(P)\big) \leq 2c(\mathcal{X}^*)$. By Lemma 5.5, we know that there exists a B-restricted schedule $\mathcal{X}'$ such that $c(\mathcal{X}') \leq 2c(\mathcal{X}^*)$. Since Φ is a bijection, and P is a shortest reasonable path, we know that $c\big(\Phi(P)\big) \leq c(\mathcal{X}')$. Thus, we conclude that

$$c(P) = c\big(\Phi(P)\big) \leq c(\mathcal{X}') \leq 2c(\mathcal{X}^*)$$

and the claim follows. □

Again, it is not difficult to show that also shortest paths which are not reasonable can be transformed to a desired 2-optimal schedule.

Corollary 5.9. *Any shortest path P from $v_{0,0}$ to $v_{0,T\downarrow}$ can be transformed to a 2-optimal, B-restricted schedule $\mathcal{X}$ for $\mathcal{I}$ with $c(P) = c(\mathcal{X})$.*

Proof. Let P be a shortest path from $v_{0,0}$ to $v_{0,T\downarrow}$. By using a small adaption of Proposition 4.8 that uses logarithmic instead of incremental switching steps, we can transform P to a reasonable path P' with $c(P') = c(P)$. In turn, P' corresponds to a 2-optimal, B-restricted schedule $\mathcal{X}$ with $c(\mathcal{X}) = c(P') = c(P)$ by Theorem 5.8, which finishes the proof. □

To finish this section, we give an algorithm based on our verified constructions. Naturally, since we did not change the graph's overall structure and working principle, a small adaption of Algorithm 2 will serve its purpose. To account for the logarithmic steps in our graph, we introduce an auxiliary function NODES, which calculates the number of servers associated to an given index i in our tables C and S. As a matter of implementation convenience, the variable b is defined as $\lceil \log_2(m) \rceil$ instead of $\lfloor \log_2(m) \rfloor$ in Algorithm 3. The correctness of the algorithm directly follows from Theorem 5.8.

For our runtime analysis, we again take the same assumptions as done for Algorithm 1. Subroutine SHORTEST_PATHS requires $\Theta\big(\log_2(m)\big)$ steps for its initialization and $\Theta\big(2T\log_2(m)\big)$ steps for the iterative calculation of the selections and costs. Further, EXTRACT_SCHEDULE needs $\Theta(T)$ iterations for its schedule retrieval. Hence, we receive a time complexity of

$$\Theta\big(\log_2(m) + 2T\log_2(m) + T\big) = \Theta(T\log_2(m))$$

To our great joy, the runtime is linear in the size of the input. Similarly, our memory demand is reduced to linear space $\Theta\big(T\log_2(m)\big)$ since the size of the tables C and S shrank to $\Theta\big(T\log_2(m)\big)$.

We saw in this section that our reduction to logarithmic steps allows us to derive a 2-optimal linear-time algorithm. In our approach, we chose the base two logarithm for the graph's step sizes. It seems like an interesting question whether this approach can be generalized to arbitrary bases, allowing for more precise approximations. The answer of this question shall be final task of this thesis.

Algorithm 3 2-optimal linear-time offline scheduling

```
1: function 2_OPTIMAL_OFFLINE_SCHEDULING($m, T, \Lambda, \beta, f$)
2:     $(C, S) \leftarrow$ SHORTEST_PATHS($m, T, \Lambda, \beta, f$)
3:     $\mathcal{X} \leftarrow$ EXTRACT_SCHEDULE($S, T, m$)
4:     return $\mathcal{X}$

5: function NODE($i, m$)
6:     return $\min\{m, \lfloor 2^{i-1} \rfloor\}$

7: function SHORTEST_PATHS($m, T, \Lambda, \beta, f$)
8:     $b \leftarrow \lceil \log_2(m) \rceil$
9:     let $C[0 \ldots b+1, 1 \ldots T]$ and $S[0 \ldots b+1, 1 \ldots T]$ be new tables
                                                  ▷ Allocate cost and selection tables
10:    $S[b+1, 1] \leftarrow b+1$ and $C[b+1, 1] \leftarrow c(0, m, \lambda_1)$ ▷ Initialize first node in first layer
11:    for $i \leftarrow b$ to $0$ do          ▷ Initialize first layer (downward minimization step)
12:        if $C[i+1, 1] < c(0, \text{NODE}(i, m), \lambda_1)$ then
13:            $S[i, 1] \leftarrow S[i+1, 1]$ and $C[i, 1] \leftarrow C[i+1, 1]$
14:        else
15:            $S[i, 1] \leftarrow i$ and $C[i, 1] \leftarrow c(0, \text{NODE}(i, m), \lambda_1)$
16:    for $t \leftarrow 1$ to $T-1$ do          ▷ Iterative calculate costs and selections
17:        for $i \leftarrow 1$ to $b+1$ do          ▷ Upward minimization step
18:            if $C[i-1, t] + \beta(\text{NODE}(i, m) - \text{NODE}(i-1, m)) < C[i, t]$ then
19:                $S[i, t] \leftarrow S[i-1, t]$
20:                $C[i, t] \leftarrow C[i-1, t] + \beta(\text{NODE}(i, m) - \text{NODE}(i-1, m))$
21:        $S[b+1, t+1] \leftarrow b+1$ and $C[b+1, t+1] \leftarrow C[b+1, t] + c_{op}(m, \lambda_{t+1})$
22:        for $i \leftarrow b$ to $0$ do          ▷ Downward minimization step
23:            if $C[i+1, t+1] < C[i, t] + c_{op}(\text{NODE}(i, m), \lambda_{t+1})$ then
24:                $S[i, t+1] \leftarrow S[i+1, t+1]$ and $C[i, t+1] \leftarrow C[i+1, t+1]$
25:            else
26:                $S[i, t+1] \leftarrow i$ and $C[i, t+1] \leftarrow C[i, t] + c_{op}(\text{NODE}(i, m), \lambda_{t+1})$
27:    return $(C, S)$

28: function EXTRACT_SCHEDULE($S, T, m$)
29:     let $\mathcal{X}[1 \ldots T]$ be a new array
30:     $i \leftarrow S[0, T]$          ▷ Get index of best selection for last time slot
31:     $\mathcal{X}[T] \leftarrow$ NODE($i, m$)          ▷ Calculate best selection for last time slot
32:     for $t \leftarrow T-1$ to $1$ do          ▷ Iteratively obtain schedule from selection table
33:         $i \leftarrow S[i, t]$
34:         $\mathcal{X}[t] \leftarrow$ NODE($i, m$)
35:     return $\mathcal{X}$
```

5.2 A $(1+\varepsilon)$-Optimal Linear-Time Algorithm

In this section, we will generalize our algorithm from Section 5.1 such that it will be able to approximate an optimal solution with arbitrary precision. More precisely, for any user-provided positive number $\varepsilon \in \mathbb{R}_{>0}$, our final algorithm will be able to find a $(1+\varepsilon)$-optimal schedule in linear time. For the rest of this section, we denote the algorithm's desired precision as y, that is we set

$$y \coloneqq 1+\varepsilon \in \mathbb{R}_{>1}$$

Previously, we chose the base two logarithm for the step sizes of our graph and derived an 2-optimal algorithm. Consequently, one might conjecture that choosing the base y for the graph's logarithmic step sizes will result in a y-optimal algorithm. But to make this conjecture a theorem, it requires some more work than merely changing the base from 2 to y.

Recall the graph's structure from Section 5.1. For any power of two smaller than m, we added a node in each layer of our graph. As a result, the set of possible scheduling choices was given by $B = \{0, 2^0, 2^1, \ldots, 2^{\lfloor \log_2(m) \rfloor}, m\}$. Sadly, we cannot simply pursue the same strategy for arbitrary bases y, but we have to slightly adapt our set of scheduling choices B, as one can see in the following example:

Example 5.10. Let $m = 7, y = 1.5$, and $b = \lfloor \log_y(m) \rfloor = 4$. The set B is then given by

$$B = \{0, y^0, y^1, y^2, y^3, y^b, 7\} = \{0, 1, 1.5, 2.25, 3.375, 5.0625, 7\}$$

The first apparent issue is that B contains non-integer numbers, which we cannot schedule in practice. We thus try to adapt the new set by rounding the contained numbers:

$$B' = \left\{0, y^0, \lfloor y^1 \rfloor, \lfloor y^2 \rfloor, \lfloor y^3 \rfloor, \lfloor y^b \rfloor, 7\right\} = \{0, 1, 2, 3, 5, 7\}$$

This adaption may seem promising at first glance, but there is still another issue with this approach: The distance between two possible scheduling choices may become too large. For instance, the ratio $5/3 = 1.666\ldots$ is greater than $y = 1.5$, and thus the distance is not logarithmic with base y. As a consequence, the approximated schedule's switching costs may become too large. This issue, however, can be solved by additionally adding the rounded up powers of y:

$$B'' = \left\{0, y^0, \lfloor y^1 \rfloor, \lceil y^1 \rceil, \lfloor y^2 \rfloor, \lceil y^2 \rceil, \lfloor y^3 \rfloor, \lceil y^3 \rceil, \lfloor y^b \rfloor, \lceil y^b \rceil, 7\right\} = \{0, 1, 2, 3, 4, 5, 6, 7\}$$

In this example, B'' degenerates to the complete set of non-negative integers up to m, but this is just a special case and does not happen in general.

Given a fixed precision y, we take the ideas discussed in the previous example to define the set of possible scheduling choices by setting

$$\begin{aligned} b &:= \lfloor \log_y(m) \rfloor \\ \text{and} \qquad B &:= \Big\{0, 1, \lfloor y^1 \rfloor, \lceil y^1 \rceil, \lfloor y^2 \rfloor, \lceil y^2 \rceil, \ldots, \lfloor y^b \rfloor, \lceil y^b \rceil, m\Big\} \end{aligned}$$

Note that $|B| \in \Theta\big(\log_y(m)\big)$, i.e. the size of B is linear in the size of the input $\mathcal{I}$ for some fixed precision y. Using this new set of scheduling choices, we consider the following modification of our graph:

$$\begin{aligned}
V &:= \{v_{0,0}\} \mathbin{\dot\cup} \big\{v_{x,t\downarrow} \mid x \in B, t \in [T]\big\} \mathbin{\dot\cup} \big\{v_{x,t\uparrow} \mid x \in B, t \in [T-1]\big\} \\
E_s &:= \big\{(v_{0,0}, v_{x,1\downarrow}) \mid x \in B\big\} \\
E_\downarrow &:= \big\{(v_{m,t\downarrow}, v_{\lceil y^b \rceil,t\downarrow}) \mid t \in [T]\big\} \cup \big\{(v_{\lceil y^i \rceil,t\downarrow}, v_{\lfloor y^i \rfloor,t\downarrow}) \mid i \in [b], t \in [T]\big\} \\
&\quad \cup \big\{(v_{\lfloor y^i \rfloor,t\downarrow}, v_{\lceil y^{i-1} \rceil,t\downarrow}) \mid i \in [b], t \in [T]\big\} \cup \big\{(v_{y^0,t\downarrow}, v_{0,t\downarrow}) \mid t \in [T]\big\} \\
E_\uparrow &:= \big\{(v_{0,t\uparrow}, v_{y^0,t\uparrow}) \mid t \in [T-1]\big\} \cup \big\{(v_{\lceil y^{i-1} \rceil,t\uparrow}, v_{\lfloor y^i \rfloor,t\uparrow}) \mid i \in [b], t \in [T-1]\big\} \\
&\quad \cup \big\{(v_{\lfloor y^i \rfloor,t\uparrow}, v_{\lceil y^i \rceil,t\uparrow}) \mid i \in [b], t \in [T-1]\big\} \cup \big\{(v_{\lceil y^b \rceil,t\uparrow}, v_{m,t\uparrow}) \mid t \in [T-1]\big\} \\
E_{\downarrow\uparrow} &:= \big\{(v_{x,t\downarrow}, v_{x,t\uparrow}) \mid x \in B, t \in [T-1]\big\} \\
E_{\uparrow\downarrow} &:= \big\{(v_{x,t\uparrow}, v_{x,t+1\downarrow}) \mid x \in B, t \in [T-1]\big\} \\
E &:= E_s \mathbin{\dot\cup} E_\downarrow \mathbin{\dot\cup} E_\uparrow \mathbin{\dot\cup} E_{\downarrow\uparrow} \mathbin{\dot\cup} E_{\uparrow\downarrow} \\
c_G(e) &:= \begin{cases} c(0, x, \lambda_1), & \text{if } e = (v_{0,0}, v_{x,1\downarrow}) \in E_s \\ c_{op}(x, \lambda_{t+1}), & \text{if } e = (v_{x,t\uparrow}, v_{x,t+1\downarrow}) \in E_{\uparrow\downarrow} \\ \beta(x' - x), & \text{if } e = (v_{x,t\uparrow}, v_{x',t\uparrow}) \in E_\uparrow \\ 0, & \text{if } e \in (E_\downarrow \mathbin{\dot\cup} E_{\downarrow\uparrow}) \end{cases} \\
G &:= (V, E, c_G)
\end{aligned}$$

A graphical representation of G can be found in Figure 5.9. Note that unnecessary loops with cost 0 may be added in $E_\downarrow$ and $E_\uparrow$ if some rounded powers of y coincide. For example, if $y = 1.5$, the numbers $\lceil y^1 \rceil = 2$ and $\lfloor y^2 \rfloor = 2$ coincide. These loops can again simply be ignored for our following work. There is not much to say about the graph's working principle, since it stays the same as for our construction in Section 5.1. Instead, we dive right into the proving process. To begin, we need to adapt Lemma 5.5, for which we have to generalize our definition of 2-state changes.

Definition 5.11 (y-state changes)**.** Let $\mathcal{X} = (x_1, \ldots, x_T)$ be a schedule and $t \in [T+1]$. We say that $\mathcal{X}$ changes its *y-state* at time t if $\mathcal{X}$ satisfies the formula

$$x_{t-1} \neq x_t \wedge \left(x_{t-1} = 0 \vee x_t \notin \left[y^{\lfloor \log_y(x_{t-1}) \rfloor}, y^{\lfloor \log_y(x_{t-1}) \rfloor + 1}\right)\right)$$

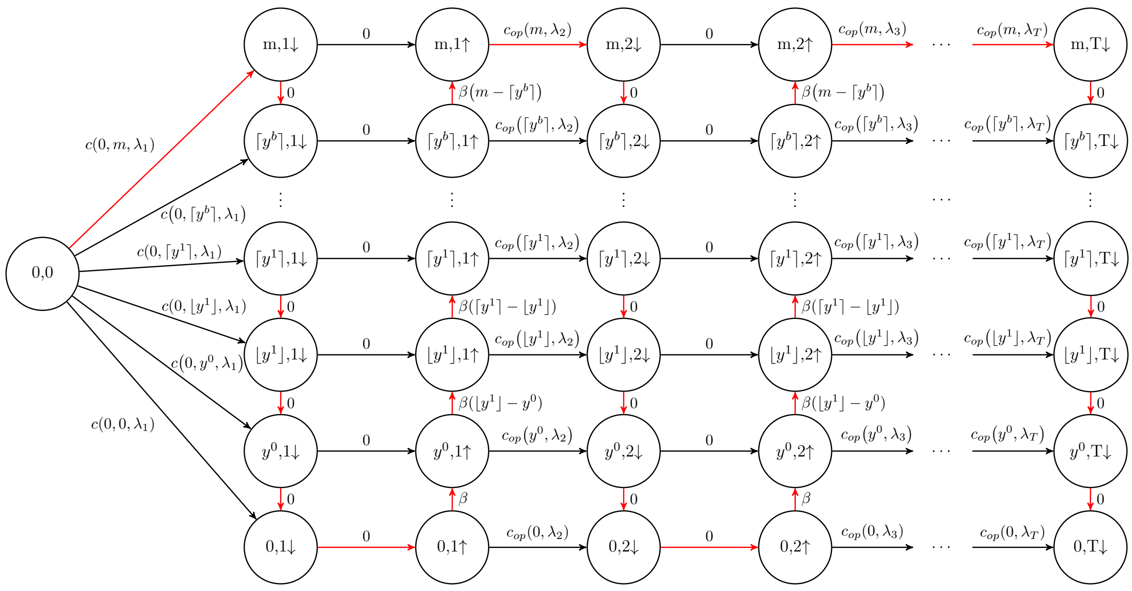


Figure 5.9: Graph for a y-optimal linear-time offline algorithm; the path of the topological sorting is highlighted in red.

The proof of the next lemma will to a great extent be very similar to that of Lemma 5.5. However, as the interval boundaries of our y-state changes are not integer anymore, we will have to proceed more cautious and consider some more cases.

Lemma 5.12. *Let $\mathcal{X}$ be a schedule for $\mathcal{I}$. There exists a B-restricted schedule $\mathcal{X}'$ satisfying $c(\mathcal{X}') \le yc(\mathcal{X})$.*

Proof. Let $\mathcal{X} = (x_1, \dots, x_T)$ be a schedule for $\mathcal{I}$. We need to construct a B-restricted schedule $\mathcal{X}' = (x'_1, \dots, x'_T)$ such that $c(\mathcal{X}') \le yc(\mathcal{X})$. By following the same reasoning as done for Lemma 5.5, we can again assume that $\mathcal{X}$ is feasible and never shuts down all servers. We then have to show that every period between two y-state changes of $\mathcal{X}$ can be iteratively transformed to a y-approximative period in $\mathcal{X}'$. The proof will again require an amortized analysis for the switching costs of $\mathcal{X}'$ using the accounting method. For our iterative transformation, let i and $j+1$ with $i, j \in [T]$ and $i \le j$ be the first unprocessed time slots at which $\mathcal{X}$ changes its y-state. We define the lower and upper bound of $\mathcal{X}_{i,j}$ as

$$l := y^{\lfloor \log_y(x_i) \rfloor} \qquad \text{and} \qquad u := \min\{yl, m\}$$

and the lower and upper bound of $\mathcal{X}$ at time $j+1$ as

$$l' := \begin{cases} y^{\lfloor \log_y(x_{j+1}) \rfloor}, & \text{if } x_{j+1} \neq 0 \\ 0, & \text{if } x_{j+1} = 0 \end{cases} \qquad \text{and} \qquad u' := \min\{yl', m\}$$

Further, given two boundaries $a, b \in \mathbb{R}$, we again define the conditional interval

$$[a, b)^m := \begin{cases} [a, b), & \text{if } b = ya \\ [a, b], & \text{if } b = m < ya \end{cases}$$

In the proof of Lemma 5.5, we knew that, for example, u is contained in B, and thus we were able to simply use u servers for $\mathcal{X}'$. This time, however, we have to appropriately round the upper bound, e.g. use $\lfloor u \rfloor \in B$ or $\lceil u \rceil \in B$ instead of u servers. Note that $\mathcal{X}$ does not change its y-state between i and j, which means that $l \leq x_t \leq u$ holds for $i \leq t \leq j$. Since x_t is integer, $\lceil l \rceil$ is the smallest integer contained in $[l, u)^m$, and $\lfloor u \rfloor$ is the largest integer contained in $[l, u)^m$, we can even refine our bounds to $\lceil l \rceil \leq x_t \leq \lfloor u \rfloor$. Further, by using the fact that $u \leq yl$, we can infer that $\lfloor u \rfloor \leq yx_t$ holds for any $i \leq t \leq j$.

We adapt the invariant used in Lemma 5.5 in that we are going to ensure that $\mathcal{X}'$ can potentially move to $\lfloor u \rfloor$ at time i in a y-approximative manner, i.e. we ensure that $\mathcal{X}'$ incurs at most y times as much switching costs as $\mathcal{X}$ if we set $x'_i := \lfloor u \rfloor$. Further, we are going to ensure that $x'_t \geq \lfloor u \rfloor$ holds for any $i \leq t \leq j$ after every transformation step. Since $x_t \leq \lfloor u \rfloor$, this guarantees that $\mathcal{X}'$ will be feasible. Moreover, we show that our credit will be greater than or equal to $\beta(yx_{j+1} - \lfloor u' \rfloor)$ if $\mathcal{X}$ exits $[l, u)$ by ascending to $[l', u')^m$ at time $j + 1$. Since the periods are consecutive, this equivalently means that the credit will be greater than or equal to $\beta(yx_i - \lfloor u \rfloor)$ if $\mathcal{X}$ enters $[l, u)^m$ at time i from below.

For the initial start-up process (i.e. $i = 1$), we can simply set $x'_1 := \lfloor u \rfloor \in B$. Since we know that $x'_1 \leq yx_1$, we can conclude that $\beta x'_1 \leq \beta yx_1$. Further, we can use the difference of switching costs $\beta(yx_1 - x'_1) = \beta(yx_1 - \lfloor u \rfloor)$ for the credit of our amortized analysis. Thus, the invariant initially holds.

Now recall the possible behaviors of $\mathcal{X}$ between its y-state changes:

(a) $\mathcal{X}$ enters $[l, u)^m$ from below and then descends to $[l', u')$.

(b) $\mathcal{X}$ enters $[l, u)$ from above and then descends to $[l', u')$.

(c) $\mathcal{X}$ enters $[l, u)$ from below and then ascends to $[l', u')^m$.

(d) $\mathcal{X}$ enters $[l, u)$ from above and then ascends to $[l', u')^m$.

We need to show that any case can be transformed to a y-approximative period in $\mathcal{X}'$ while ensuring that our invariant will not be violated. Additionally, we have to verify that the credit of our amortized analysis stays non-negative.

We start with case (a). Due to our invariant, we know that we can set $x'_i := \lfloor u \rfloor$ with y-approximative switching costs. Consequently, setting $x'_i := x'_{i+1} := \cdots := x'_j := \lfloor u \rfloor \in B$ gives us a strategy with y-approximative switching costs between i and j. Further, since $x_t \leq \lfloor u \rfloor \leq yx_t$ for any $i \leq t \leq j$, and f is non-negative and monotonically increasing, the operating costs of $\mathcal{X}'_{i,j}$ can be estimated by

$$c_{op}(\mathcal{X}'_{i,j}) \overset{(3.10)}{=} \sum_{t=i}^{j} \left(\lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \leq \sum_{t=i}^{j} \left(y x_t f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \leq y \sum_{t=i}^{j} \left(x_t f\left(\frac{\lambda_t}{x_t}\right)\right)$$
$$\overset{(3.10)}{=} y c_{op}(\mathcal{X}_{i,j})$$

Thus, the operating costs of $\mathcal{X}'_{i,j}$ are y-approximative. Moreover, since $\lfloor u' \rfloor \leq \lfloor u \rfloor$, we do not need to account for additional switching costs to satisfy our invariant at time $j+1$. Lastly, the credit of our amortized analysis stays untouched, i.e. the credit stays at $\beta(yx_i - \lfloor u \rfloor)$. We will have to use this credit for case (d).

Next, we consider case (b). Since $\mathcal{X}$ enters $[l,u)$ from above, we have the possibility to choose $\lceil u \rceil$ or $\lfloor u \rfloor$ servers for $\mathcal{X}'_{i,t}$ at time i without incurring switching costs. We thus consider two different strategies for $\mathcal{X}'_{i,j}$:

$$\begin{aligned} \mathcal{X}^{\lfloor u \rfloor}_{i,j} &:= \left(x^{\lfloor u \rfloor}_i, \ldots, x^{\lfloor u \rfloor}_j\right), \qquad &\text{where} \qquad & x^{\lfloor u \rfloor}_t := \lfloor u \rfloor, \text{ for } i \leq t \leq j, \\ \text{and} \quad \mathcal{X}^{\lceil u \rceil}_{i,j} &:= \left(x^{\lceil u \rceil}_i, \ldots, x^{\lceil u \rceil}_j\right), \qquad &\text{where} \qquad & x^{\lceil u \rceil}_t := \lceil u \rceil, \text{ for } i \leq t \leq j \end{aligned}$$

Since $\lfloor u' \rfloor < \lfloor u \rfloor \leq \lceil u \rceil$, both strategies do not incur switching costs to satisfy the invariant at time $j+1$. We decide to choose strategy $\mathcal{X}^{\lceil u \rceil}_{i,j}$ if $y\lceil l \rceil > \lceil u \rceil$, and strategy $\mathcal{X}^{\lfloor u \rfloor}_{i,j}$ otherwise. Let us check that these choices are indeed y-approximative. If $y\lceil l \rceil > \lceil u \rceil$, we know that $yx_t > \lceil u \rceil$ for $i \leq t \leq j$, which allows us to estimate the cost of $\mathcal{X}^{\lceil u \rceil}_{i,j}$ by

$$c_{op}(\mathcal{X}^{\lceil u \rceil}_{i,j}) \overset{(3.10)}{=} \sum_{t=i}^{j} \left(\lceil u \rceil f\left(\frac{\lambda_t}{\lceil u \rceil}\right)\right) \leq \sum_{t=i}^{j} \left(y x_t f\left(\frac{\lambda_t}{\lceil u \rceil}\right)\right) \leq y \sum_{t=i}^{j} \left(x_t f\left(\frac{\lambda_t}{x_t}\right)\right)$$
$$\overset{(3.10)}{=} y c_{op}(\mathcal{X}_{i,j})$$

which shows that $\mathcal{X}^{\lceil u \rceil}_{i,j}$ is y-approximative. Next, if $y\lceil l \rceil \leq \lceil u \rceil$, we choose strategy $\mathcal{X}^{\lfloor u \rfloor}_{i,j}$, whose cost, as we have already seen in case (a), is y-approximative. Note that case (b) could also be solved by always using $\mathcal{X}^{\lfloor u \rfloor}_{i,j}$; the case distinction, however, will be of importance for case (d).

For case (c), i.e. $\mathcal{X}$ enters $[l,u)$ from below and then ascends to $[l',u')^m$, we know that setting $x'_i := x'_{i+1} := \cdots := x'_j := \lfloor u \rfloor \in B$ gives us a strategy with y-approximative switching costs due to our invariant. Moreover, as in the previous cases, the operating costs of $\mathcal{X}'_{i,j}$ are y-approximative. To verify our invariant at time $j+1$, we have to show that we can account for the switching costs to move to u' and for the new credit required by the invariant. For this, we use the fact that $\mathcal{X}$ has to power on at least $x_{j+1} - x_i$ servers between i and $j+1$, and that we can use our old credit to compensate for our costs, since we entered $[l, u)$ from below:

$$\begin{aligned}
& y\,\overbrace{\beta(x_{j+1} - x_i)}^{\text{sw. costs } \mathcal{X}} + \overbrace{\beta(yx_i - \lfloor u \rfloor)}^{\text{old credit}} - \overbrace{\beta(\lfloor u' \rfloor - \lfloor u \rfloor)}^{\text{sw. costs } \mathcal{X}'} - \overbrace{\beta(yx_{j+1} - \lfloor u' \rfloor)}^{\text{new credit}} \\
= \;& \beta(yx_{j+1} - yx_i + yx_i - \lfloor u \rfloor - \lfloor u' \rfloor + \lfloor u \rfloor - yx_{j+1} + \lfloor u' \rfloor) \\
= \;& 0
\end{aligned}$$

that is, the cost difference is non-negative, and thus our invariant at time $j + 1$ is satisfied.

Finally, we have to study case (d), i.e. $\mathcal{X}$ enters $[l, u)$ from above and then ascends to $[l', u')^m$, for which we consider three different strategies, namely

$$\begin{aligned}
\mathcal{X}_{i,j}^{\lfloor u \rfloor} &:= \left(x_i^{\lfloor u \rfloor}, \ldots, x_j^{\lfloor u \rfloor}\right), && \text{where} && x_t^{\lfloor u \rfloor} := \lfloor u \rfloor, \text{ for } i \le t \le j, \\
\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor} &:= \left(x_i^{\lfloor \hat{u} \rfloor}, \ldots, x_j^{\lfloor \hat{u} \rfloor}\right), && \text{where} && x_t^{\lfloor \hat{u} \rfloor} := \lfloor \hat{u} \rfloor, \text{ for } i \le t \le j, \\
\text{and} \quad \mathcal{X}_{i,j}^{\lceil \hat{u} \rceil} &:= \left(x_i^{\lceil \hat{u} \rceil}, \ldots, x_j^{\lceil \hat{u} \rceil}\right), && \text{where} && x_t^{\lceil \hat{u} \rceil} := \lceil \hat{u} \rceil, \text{ for } i \le t \le j
\end{aligned}$$

with $\hat{u} := \min\{yu, m\}$. Notice that due to our invariant and the fact that $\mathcal{X}$ descends at time i, strategies $\mathcal{X}_{i,j}^{\lfloor u \rfloor}$ and $\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}$ do no incur switching costs up to time j, since $x'_{i-1} \ge \lfloor \hat{u} \rfloor \ge \lfloor u \rfloor$. We are now going to prove that the cost of at least one of the three schedules, including costs to satisfy our invariant at time $j + 1$, must be y-approximative. First, we define $d := \min\{x_t \mid i \le t \le j\}$ to be the smallest number of active servers used by $\mathcal{X}_{i,j}$. We can use d to give a lower bound for the switching costs of $\mathcal{X}$ between i and $j + 1$:

$$\sum_{t=i+1}^{j+1} c_{sw}(x_{t-1}, x_t) \ge \beta(x_{j+1} - d)$$

Next, we have to conduct another case distinction; namely

$$\text{(I) } yd \le \lfloor \hat{u} \rfloor \qquad \text{or} \qquad \text{(II) } yd > \lfloor \hat{u} \rfloor$$

We begin with case (I), for which we proceed analogous as in the proof of Lemma 5.5. We again split the switching costs of $\mathcal{X}_{i,j}$ (including its final switching step) into two parts. This time, we split $\beta(x_{j+1} - d)$ into $\beta\big(x_{j+1} - \lfloor \hat{u} \rfloor / y\big)$ and $\beta\big(\lfloor \hat{u} \rfloor / y - d\big)$. It now suffices to show that either $\mathcal{X}_{i,j}^{\lfloor u \rfloor}$ or $\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}$ – strategy $\mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}$ will not be needed in this case – can process the loads $\lambda_i, \ldots, \lambda_j$ and eventually switch to $\lfloor \hat{u} \rfloor$ with y-approximative costs under the assumption that $\mathcal{X}_{i,j}$ only incurs $\beta\big(\lfloor \hat{u} \rfloor / y - d\big)$ switching costs in a first step. This is because the remaining switching costs $\beta\big(x_{j+1} - \lfloor \hat{u} \rfloor / y\big)$ of $\mathcal{X}$ already suffice to compensate for the remaining switching costs $\beta(\lfloor u' \rfloor - \lfloor \hat{u} \rfloor)$ of $\mathcal{X}'$ and the costs to satisfy the invariant:

$$\begin{aligned} & y \overbrace{\beta(x_{j+1} - \lfloor \hat{u} \rfloor / y)}^{\text{remaining sw. costs } \mathcal{X}} - \overbrace{\beta(\lfloor u' \rfloor - \lfloor \hat{u} \rfloor)}^{\text{rem. sw. costs } \mathcal{X}'} - \overbrace{\beta(y x_{j+1} - \lfloor u' \rfloor)}^{\text{new credit}} \\ = \; & \beta(y x_{j+1} - \lfloor \hat{u} \rfloor - \lfloor u' \rfloor + \lfloor \hat{u} \rfloor - y x_{j+1} + \lfloor u' \rfloor) \\ = \; & 0 \end{aligned}$$

To proceed, we notice that if $yd = \lfloor u \rfloor$ holds, strategy $\mathcal{X}_{i,j}^{\lfloor u \rfloor}$ has y-approximative switching costs:

$$\beta(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) = \beta(\lfloor \hat{u} \rfloor - yd) = y\beta\left(\frac{\lfloor \hat{u} \rfloor}{y} - d\right)$$

Further, as in the previous cases, the operating costs of $\mathcal{X}_{i,j}^{\lfloor u \rfloor}$ are y-approximative, and thus $\mathcal{X}_{i,j}^{\lfloor u \rfloor}$ is y-approximative if $yd = \lfloor u \rfloor$. Further, since $l \le d \le u \le yl$, we know that $yd < \lfloor u \rfloor$ is impossible, and thus we can safely assume that $yd > \lfloor u \rfloor$. Moreover, we can assume that $\mathcal{X}_{i,j}^{\lfloor u \rfloor}$ is not y-approximative; otherwise, we are already done. Hence, suppose that

$$c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor}) + \beta(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) > y\left(c_{op}(\mathcal{X}_{i,j}) + \beta\left(\frac{\lfloor \hat{u} \rfloor}{y} - d\right)\right)$$

We then have to show that $\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}$ is y-approximative. Rearranging the previous inequality gives us an estimation for β:

$$\begin{aligned} & c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor}) + \beta(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) > y\left(c_{op}(\mathcal{X}_{i,j}) + \beta\left(\frac{\lfloor \hat{u} \rfloor}{y} - d\right)\right) \\ \iff \quad & \beta(yd - \lfloor u \rfloor) > y c_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor}) \\ \overset{yd - \lfloor u \rfloor > 0}{\iff} \quad & \beta > \frac{y c_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor})}{yd - \lfloor u \rfloor} \end{aligned}$$

where the last step is justified by assumption $yd > \lfloor u \rfloor$. Now, by additionally using our assumption $yd \le \lfloor \hat{u} \rfloor$, which equivalently means $\frac{\lfloor \hat{u} \rfloor}{y} - d \ge 0$, we can find a lower bound for the cost of $\mathcal{X}_{i,j}$:

$$c_{op}(\mathcal{X}_{i,j}) + \beta \underbrace{\left(\frac{\lfloor \hat{u} \rfloor}{y} - d\right)}_{\ge 0} \ge c_{op}(\mathcal{X}_{i,j}) + \frac{y c_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor})}{yd - \lfloor u \rfloor} \left(\frac{\lfloor \hat{u} \rfloor}{y} - d\right)$$

which can be simplified to

$$\begin{aligned} & c_{op}(\mathcal{X}_{i,j}) + \frac{y c_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor})}{yd - \lfloor u \rfloor} \left(\frac{\lfloor \hat{u} \rfloor}{y} - d \right) \\ = \quad & \frac{(yd - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}) + (\lfloor \hat{u} \rfloor - yd) c_{op}(\mathcal{X}_{i,j}) - \left(\frac{\lfloor \hat{u} \rfloor}{y} - d \right) c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor})}{yd - \lfloor u \rfloor} \\ = \quad & \frac{(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}) - \left(\frac{\lfloor \hat{u} \rfloor}{y} - d \right) c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor})}{yd - \lfloor u \rfloor} \end{aligned}$$

Next, since we want to prove that $c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor})$ is y-approximative, we have to show that the following cost difference is non-negative:

$$y \left(c_{op}(\mathcal{X}_{i,j}) + \beta \left(\frac{\lfloor \hat{u} \rfloor}{y} - d \right) \right) - c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor})$$

Recall that $\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}$ does not incur switching costs to move to $\lfloor \hat{u} \rfloor$. We can now use our just derived lower bound for $c_{op}(\mathcal{X}_{i,j}) + \beta(\lfloor \hat{u} \rfloor / y - d)$ to estimate the difference:

$$\begin{aligned} & y \left(c_{op}(\mathcal{X}_{i,j}) + \beta \left(\frac{\lfloor \hat{u} \rfloor}{y} - d \right) \right) - c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) \\ \geq \quad & y \, \frac{(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}) - \left(\frac{\lfloor \hat{u} \rfloor}{y} - d \right) c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor})}{yd - \lfloor u \rfloor} - c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) \end{aligned}$$

Thus, it suffices to show that

$$y \, \frac{(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}) - \left(\frac{\lfloor \hat{u} \rfloor}{y} - d \right) c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor})}{yd - \lfloor u \rfloor} - c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) \geq 0$$

which can be simplified to

$$y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}) - (\lfloor \hat{u} \rfloor - yd) c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor}) - (yd - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) \geq 0$$

To show this inequality, we have to take a closer look on the the schedules' operating costs:

$$\begin{aligned} & y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}) - (\lfloor \hat{u} \rfloor - yd) c_{op}(\mathcal{X}_{i,j}^{\lfloor u \rfloor}) - (yd - \lfloor u \rfloor) c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) \\ \overset{(3.10)}{=} \quad & \sum_{t=i}^{j} \left(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) x_t f\left(\frac{\lambda_t}{x_t} \right) - (\lfloor \hat{u} \rfloor - yd) \lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor} \right) - (yd - \lfloor u \rfloor) \lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor} \right) \right) \end{aligned}$$

First, we recall that $\lfloor u \rfloor \leq \lfloor \hat{u} \rfloor$ and $x_t \leq \lfloor u \rfloor$ for $i \leq t \leq j$. Together with the fact that f is monotonically increasing, we infer that

$$\begin{aligned}
&\sum_{t=i}^{j} \left(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) x_t f\left(\frac{\lambda_t}{x_t}\right) - (\lfloor \hat{u} \rfloor - yd)\lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (yd - \lfloor u \rfloor)\lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) \\
\geq\ &\sum_{t=i}^{j} \left(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) x_t f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (\lfloor \hat{u} \rfloor - yd)\lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (yd - \lfloor u \rfloor)\lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right)
\end{aligned}$$

Since d is the smallest number of active servers scheduled by $\mathcal{X}$, and f is non-negative, we can conclude that

$$\begin{aligned}
&\sum_{t=i}^{j} \left(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) x_t f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (\lfloor \hat{u} \rfloor - yd)\lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (yd - \lfloor u \rfloor)\lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \\
\geq\ &\sum_{t=i}^{j} \left(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) d f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (\lfloor \hat{u} \rfloor - yd)\lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (yd - \lfloor u \rfloor)\lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right)
\end{aligned}$$

Hence, to finish the case, it suffices to show that

$$\sum_{t=i}^{j} \left(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) d f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (\lfloor \hat{u} \rfloor - yd)\lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (yd - \lfloor u \rfloor)\lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \geq 0$$

Further rearranging the left hand side gives us

$$\begin{aligned}
&\sum_{t=i}^{j} \left(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) d f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (\lfloor \hat{u} \rfloor - yd)\lfloor u \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right) - (yd - \lfloor u \rfloor)\lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \\
=\ &\sum_{t=i}^{j} \left(\Big(y(\lfloor \hat{u} \rfloor - \lfloor u \rfloor) d - (\lfloor \hat{u} \rfloor - yd)\lfloor u \rfloor - (yd - \lfloor u \rfloor)\lfloor \hat{u} \rfloor \Big) f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \\
=\ &\sum_{t=i}^{j} \left(\Big(yd\lfloor \hat{u} \rfloor - yd\lfloor u \rfloor - \lfloor u \rfloor\lfloor \hat{u} \rfloor + yd\lfloor u \rfloor - yd\lfloor \hat{u} \rfloor + \lfloor u \rfloor\lfloor \hat{u} \rfloor \Big) f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \\
=\ &\sum_{t=i}^{j} \left(0 \cdot f\left(\frac{\lambda_t}{\lfloor u \rfloor}\right)\right) \\
=\ &0
\end{aligned}$$

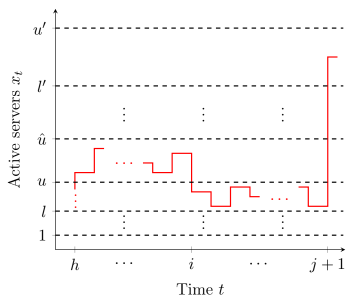
which finishes case (I). We are left with case (II), i.e. $yd > \lfloor \hat{u} \rfloor$. For this case, we have to analyze the behavior of $\mathcal{X}$ more precisely. To do so, let $h < i$ be the closest time before i at which $\mathcal{X}$ changes its y-state. There are three possible behaviors of $\mathcal{X}$ that can lead to case (d).

(i) $\mathcal{X}$ ascends to $[u, \hat{u})^m$ at time h and then descends to $[l, u)$ at time i.

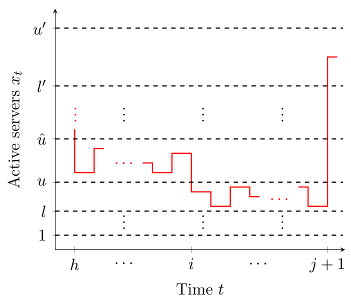
(ii) $\mathcal{X}$ descends to $[u, \hat{u})$ at time h and then descends to $[l, u)$ at time i.

(iii) $\mathcal{X}$ descends to $[l, u)$ at time i from a region above $[u, \hat{u})$.

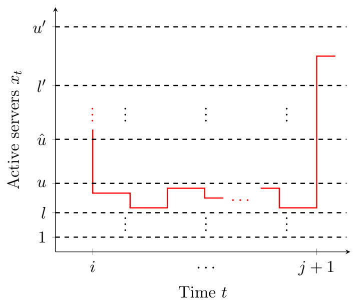
All three possibilities are illustrated in Figure 5.10. For each case, we have to show that at least one of our three strategies satisfies our conditions, to wit, it incurs y-approximative costs including the costs to satisfy the invariant at time $j+1$.



(i) $\mathcal{X}$ ascends to $[u, \hat{u})^m$ and then descends to $[l, u)$.



(ii) $\mathcal{X}$ descends to $[u, \hat{u})$ and then descends to $[l, u)$.



(iii) $\mathcal{X}$ descends to $[l, u)$ from a region above $[u, \hat{u})$.

Figure 5.10: The three different possible behaviors of $\mathcal{X}$ to reach case (d)

We begin with case (i). Since $\mathcal{X}$ ascends to $[u, \hat{u})^m$ at time h and descends to $[l, u)$ at time i, we know from case (a) that we have some remaining credit $\beta(yx_h - \lfloor \hat{u} \rfloor)$. Since $x_h \geq \lceil u \rceil$, this credit amounts to at least $\beta(y\lceil u \rceil - \lfloor \hat{u} \rfloor)$. We use this credit to show that strategy $\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}$ is y-approximative, that is we have to verify that the following cost difference is non-negative:

$$y\big(\overbrace{c_{op}(\mathcal{X}_{i,j}) + \beta(x_{j+1} - d)}^{\text{costs of } \mathcal{X}}\big) + \overbrace{\beta(y\lceil u \rceil - \lfloor \hat{u} \rfloor)}^{\text{old credit}} - \overbrace{\big(c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) + \beta(\lfloor u' \rfloor - \lfloor \hat{u} \rfloor)\big)}^{\text{costs of } \mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}}$$
$$- \underbrace{\beta(yx_{j+1} - \lfloor u' \rfloor)}_{\text{new credit}}$$

This difference can be simplified to

$$\begin{aligned} & yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) + \beta\big(yx_{j+1} - yd + y\lceil u \rceil - \lfloor \hat{u} \rfloor - \lfloor u' \rfloor + \lfloor \hat{u} \rfloor - yx_{j+1} + \lfloor u' \rfloor\big) \\ = \; & yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) + \underbrace{\beta\big(y\lceil u \rceil - yd\big)}_{\geq 0} \\ \geq \; & yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}) \end{aligned}$$

where $y\lceil u\rceil - yd \geq 0$ follows from $d \leq \lfloor u\rfloor \leq \lceil u\rceil$. Thus, it suffices to show that

$$yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}) \geq 0 \tag{5.13}$$

Further modifying the left hand side as done in previous cases reveals

$$\begin{aligned}
yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}) &\overset{(3.10)}{=} \sum_{t=i}^{j}\left(yx_t f\left(\frac{\lambda_t}{x_t}\right) - \lfloor\hat{u}\rfloor f\left(\frac{\lambda_t}{\lfloor\hat{u}\rfloor}\right)\right)\\
&\geq \sum_{t=i}^{j}\left(ydf\left(\frac{\lambda_t}{\lfloor\hat{u}\rfloor}\right) - \lfloor\hat{u}\rfloor f\left(\frac{\lambda_t}{\lfloor\hat{u}\rfloor}\right)\right)\\
&= \sum_{t=i}^{j}\left(\underbrace{(yd - \lfloor\hat{u}\rfloor)}_{>0} f\left(\frac{\lambda_t}{\lfloor\hat{u}\rfloor}\right)\right)\\
&\geq 0
\end{aligned}$$

where $yd - \lfloor\hat{u}\rfloor > 0$ follows from assumption $yd > \lfloor\hat{u}\rfloor$. Hence, strategy $\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}$ is y-approximative.

Lastly, we consider cases (ii) and (iii). Suppose for these cases that $\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}$ is not y-approximative, that is

$$y\overbrace{\left(c_{op}(\mathcal{X}_{i,j}) + \beta(x_{j+1} - d)\right)}^{\text{costs of } \mathcal{X}} - \overbrace{\left(c_{op}(\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}) + \beta(\lfloor u'\rfloor - \lfloor\hat{u}\rfloor)\right)}^{\text{costs of } \mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}} - \overbrace{\beta(yx_{j+1} - \lfloor u'\rfloor)}^{\text{new credit}} < 0$$

We already know from Inequality (5.13) that the operating costs of $\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}$ are y-approximative. However, as the total difference of costs is still negative, it seems interesting to get an estimation for β. For this, we rearrange the left hand side of the previous inequality:

$$\begin{aligned}
&yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}) + \beta(yx_{j+1} - yd - \lfloor u'\rfloor + \lfloor\hat{u}\rfloor - yx_{j+1} + \lfloor u'\rfloor)\\
=\;&yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lfloor\hat{u}\rfloor}) + \beta(\lfloor\hat{u}\rfloor - yd)
\end{aligned}$$

We plug in the definition of our operating cost function, and conduct our usual estimations to derive

$$\begin{aligned}
& y c_{op}\big(\mathcal{X}_{i,j}\big) - c_{op}\big(\mathcal{X}_{i,j}^{\lfloor \hat{u} \rfloor}\big) + \beta\big(\lfloor \hat{u} \rfloor - yd\big) \\
\overset{(3.10)}{=} \quad & \sum_{t=i}^{j} \left(y x_t f\left(\frac{\lambda_t}{x_t}\right) - \lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta\big(\lfloor \hat{u} \rfloor - yd\big) \\
\geq \quad & \sum_{t=i}^{j} \left(y d f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right) - \lfloor \hat{u} \rfloor f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta\big(\lfloor \hat{u} \rfloor - yd\big) \\
= \quad & \sum_{t=i}^{j} \left((yd - \lfloor \hat{u} \rfloor) f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta\big(\lfloor \hat{u} \rfloor - yd\big)
\end{aligned}$$

Thus, we know that

$$\sum_{t=i}^{j} \left((yd - \lfloor \hat{u} \rfloor) f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta\big(\lfloor \hat{u} \rfloor - yd\big) < 0$$

which we can rearrange to get our desired estimation for β:

$$\begin{aligned}
& \sum_{t=i}^{j} \left((yd - \lfloor \hat{u} \rfloor) f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta\big(\lfloor \hat{u} \rfloor - yd\big) < 0 \\
\Longleftrightarrow \quad & \sum_{t=i}^{j} \left((yd - \lfloor \hat{u} \rfloor) f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) < \beta(yd - \lfloor \hat{u} \rfloor) \\
\overset{yd > \lfloor \hat{u} \rfloor}{\Longleftrightarrow} \quad & \sum_{t=i}^{j} f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right) < \beta \qquad (5.14)
\end{aligned}$$

We are now going to prove that $\mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}$ is y-approximative. The just derived lower bound for β will be needed at the very end of this endeavor. First note that $\mathcal{X}$ descends at time h to $[u, \hat{u})$ in case (ii) and at time i in case (iii). Hence, we know that $\hat{u} = yu$; otherwise, we have $\hat{u} = m < yu$, and $\mathcal{X}$ would consequently not be able to descend at time h or i, respectively. Next, we show that $\mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}$ does not incur switching costs to move to $\lceil \hat{u} \rceil$ at time i. This is evidently true for case (iii) as $\mathcal{X}$ descends at time i from a region above $[u, \hat{u})$. Case (ii) requires some more effort. We first infer that $\hat{u} \neq \lfloor \hat{u} \rfloor$; in other words, $\hat{u}$ is not integer. If $\hat{u}$ would be integer, we could derive from our assumption that $yd > \lfloor \hat{u} \rfloor = \lfloor yu \rfloor = yu$, which leads to a contradiction since $d \leq \lfloor u \rfloor$. Further, we can can infer that $u \neq \lfloor u \rfloor$; otherwise, we know that $d \leq u - 1$ because d is integer and must be smaller than u (by our Definition 5.11 of y-state changes), which again leads to a contradiction:

$$\begin{aligned}
yd > \lfloor \hat{u} \rfloor \quad & \overset{u-1 \geq d}{\Longrightarrow} \quad y(u-1) > \lfloor \hat{u} \rfloor \quad \overset{\hat{u}=yu}{\Longleftrightarrow} \quad y(u-1) > \lfloor yu \rfloor \\
& \Longleftrightarrow \quad yu - \lfloor yu \rfloor > y \quad \lightning
\end{aligned}$$

The contradiction follows from $1 > yu - \lfloor yu \rfloor$ and $y > 1$. We can use the just made observations to derive

$$\begin{aligned} & yd > \lfloor \hat{u} \rfloor \quad \overset{\hat{u}\neq\lfloor\hat{u}\rfloor}{\Longleftrightarrow} \quad yd > \lceil \hat{u} \rceil - 1 \quad \overset{\lfloor u \rfloor \geq d}{\Longrightarrow} \quad y\lfloor u \rfloor > \lceil \hat{u} \rceil - 1 \\ \overset{u\neq\lfloor u\rfloor}{\Longleftrightarrow} \quad & y(\lceil u \rceil - 1) > \lceil \hat{u} \rceil - 1 \quad \Longleftrightarrow \quad y\lceil u \rceil > \lceil \hat{u} \rceil + y - 1 \quad \overset{y>1}{\Longrightarrow} \quad y\lceil u \rceil > \lceil \hat{u} \rceil \end{aligned}$$

Now recall that in case (ii) the schedule $\mathcal{X}$ descended at time h to $[u, \hat{u})$ and further descends at time i. This means that we considered case (b) for the transformation of $\mathcal{X}$ between h and $i-1$. But since u is the lower bound and $\hat{u}$ the upper bound of $\mathcal{X}$ between h and $i-1$, and we proved that $y\lceil u \rceil > \lceil \hat{u} \rceil$, we know from case (b) that $x'_{i-1} = \lceil \hat{u} \rceil$. Consequently, $\mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}$ does not require switching costs to move to $\lceil \hat{u} \rceil$ at time i. Thus, to finish the proof, it suffices to show that the following cost difference is non-negative:

$$\begin{aligned} & \overbrace{y\big(c_{op}(\mathcal{X}_{i,j}) + \beta(x_{j+1} - d)\big)}^{\text{costs of } \mathcal{X}} - \overbrace{\big(c_{op}(\mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}) + \beta(\lfloor u' \rfloor - \lceil \hat{u} \rceil)\big)}^{\text{costs of } \mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}} - \overbrace{\beta(yx_{j+1} - \lfloor u' \rfloor)}^{\text{new credit}} \\ = \quad & yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}) + \beta(yx_{j+1} - yd - \lfloor u' \rfloor + \lceil \hat{u} \rceil - yx_{j+1} + \lfloor u' \rfloor) \\ = \quad & yc_{op}(\mathcal{X}_{i,j}) - c_{op}(\mathcal{X}_{i,j}^{\lceil \hat{u} \rceil}) + \beta(\lceil \hat{u} \rceil - yd) \\ \overset{(3.10)}{=} \quad & \sum_{t=i}^{j} \left(yx_t f\left(\frac{\lambda_t}{x_t}\right) - \lceil \hat{u} \rceil f\left(\frac{\lambda_t}{\lceil \hat{u} \rceil}\right)\right) + \beta(\lceil \hat{u} \rceil - yd) \end{aligned}$$

We again apply the usual estimations and rewritings to derive

$$\begin{aligned} & \sum_{t=i}^{j} \left(yx_t f\left(\frac{\lambda_t}{x_t}\right) - \lceil \hat{u} \rceil f\left(\frac{\lambda_t}{\lceil \hat{u} \rceil}\right)\right) + \beta(\lceil \hat{u} \rceil - yd) \\ \geq \quad & \sum_{t=i}^{j} \left(yd f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right) - \lceil \hat{u} \rceil f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta(\lceil \hat{u} \rceil - yd) \\ = \quad & \sum_{t=i}^{j} \left((yd - \lceil \hat{u} \rceil) f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta(\lceil \hat{u} \rceil - yd) \\ = \quad & \sum_{t=i}^{j} \left(-(\lceil \hat{u} \rceil - yd) f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right) + \beta(\lceil \hat{u} \rceil - yd) \\ = \quad & \underbrace{\left(\beta - \sum_{t=i}^{j} f\left(\frac{\lambda_t}{\lfloor \hat{u} \rfloor}\right)\right)}_{>0} \underbrace{(\lceil \hat{u} \rceil - yd)}_{\geq 0} \geq 0 \end{aligned}$$

where we used Inequality (5.14) as well as $d \leq u$ and $\hat{u} = yu$ for the last step. This finishes case (d).

By iteratively applying above procedure to $\mathcal{X}$, starting with $i = 1$ and stopping with $j = T + 1$, we obtain a B-restricted schedule $\mathcal{X}'$ that satisfies $c(\mathcal{X}') \leq yc(\mathcal{X})$. □

The remaining required verification proofs for our graph are similar to those of Section 5.1.

Lemma 5.15. *Let $\boldsymbol{\mathcal{X}}$ be the set of all B-restricted schedules for $\mathcal{I}$, and let $\boldsymbol{\mathcal{P}}$ be the set of all reasonable paths. The map*

$$\Phi : \boldsymbol{\mathcal{P}} \to \boldsymbol{\mathcal{X}}, \quad P \mapsto (x_1, \dots, x_T)$$

is a bijection satisfying $c(P) = c\big(\Phi(P)\big)$.

Theorem 5.16. *Any shortest reasonable path P corresponds to a y-optimal, B-restricted schedule $\mathcal{X}$ for $\mathcal{I}$ with $c(P) = c(\mathcal{X})$.*

Corollary 5.17. *Any shortest path P from $v_{0,0}$ to $v_{0,T\downarrow}$ can be transformed to a y-optimal, B-restricted schedule $\mathcal{X}$ for $\mathcal{I}$ with $c(P) = c(\mathcal{X})$.*

Proof. The proofs are, once the necessary changes have been made (i.e. substituting 2 with y), the same as the ones of Lemma 5.7, Theorem 5.8, and Corollary 5.9. □

To no surprise, the final algorithm will be almost identical to the algorithm of our 2-approximation. The only major things that require a change are the auxiliary function NODE, which has to account for the new set of possible scheduling choices B, and the tables C and S, whose sizes have to be adapted. Since some rounded powers of y may coincide (cf. the definition of G), the function NODE may return the same number for two different indices $i \neq i'$. However, this is not an issue; the cost the corresponding number's node will merely be calculated multiple times. The correctness of Algorithm 4 directly follows from Theorem 5.16.

Our runtime analysis naturally also stays very similar. Subroutine SHORTEST_PATHS requires $\Theta\big(2\log_y(m)\big)$ steps for its initialization and $\Theta\big(2T \cdot 2\log_y(m)\big)$ steps for the iterative calculation of the selections and costs. Further, EXTRACT_SCHEDULE needs $\Theta(T)$ iterations for its schedule retrieval. Hence, we receive a time complexity of

$$\Theta\big(2\log_y(m) + 2T \cdot 2\log_y(m) + T\big) = \Theta(T\log_y(m)) = \Theta\left(T\,\frac{\log(m)}{\log(1+\varepsilon)}\right)$$

that is, the runtime is linear in the size of the input for fixed ε. Similarly, our memory demand amounts to $\Theta\big(T\log_y(m)\big)$ since the size of the tables C and S is given by $\Theta\big(2T\log_y(m)\big)$.

Algorithm 4 $(1+\varepsilon)$-optimal linear-time offline scheduling

```
function $(1+\varepsilon)$_OPTIMAL_OFFLINE_SCHEDULING$(m, T, \Lambda, \beta, f, \varepsilon)$
    $(C, S) \leftarrow$ SHORTEST_PATHS$(m, T, \Lambda, \beta, f, 1+\varepsilon)$
    $\mathcal{X} \leftarrow$ EXTRACT_SCHEDULE$(S, T, m, 1+\varepsilon)$
    return $\mathcal{X}$

function NODE$(i, m, y)$
    if $i = 0$ then return $0$
    else if $i \bmod 2 = 1$ then return $\min\{m, \lfloor y^{\lfloor \frac{i-1}{2} \rfloor} \rfloor\}$
    else return $\min\{m, \lceil y^{\lfloor \frac{i-1}{2} \rfloor} \rceil\}$

function SHORTEST_PATHS$(m, T, \Lambda, \beta, f, y)$
    $b \leftarrow \lfloor \log_y(m) \rfloor + 1$
    let $C[0 \dots 2b+1, 1 \dots T]$ and $S[0 \dots 2b+1, 1 \dots T]$ be new tables
    $S[2b+1, 1] \leftarrow 2b+1$ and $C[2b+1, 1] \leftarrow c(0, m, \lambda_1)$    ▷ Initialize first node
    for $i \leftarrow 2b$ to $0$ do    ▷ Initialize first layer (downward minimization step)
        if $C[i+1, 1] < c(0, \text{NODE}(i, m, y), \lambda_1)$ then
            $S[i, 1] \leftarrow S[i+1, 1]$ and $C[i, 1] \leftarrow C[i+1, 1]$
        else
            $S[i, 1] \leftarrow i$ and $C[i, 1] \leftarrow c(0, \text{NODE}(i, m, y), \lambda_1)$
    for $t \leftarrow 1$ to $T-1$ do    ▷ Iterative calculate costs and selections
        for $i \leftarrow 1$ to $2b+1$ do    ▷ Upward minimization step
            if $C[i-1, t] + \beta(\text{NODE}(i, m, y) - \text{NODE}(i-1, m, y)) < C[i, t]$ then
                $S[i, t] \leftarrow S[i-1, t]$
                $C[i, t] \leftarrow C[i-1, t] + \beta(\text{NODE}(i, m, y) - \text{NODE}(i-1, m, y))$
        $S[2b+1, t+1] \leftarrow 2b+1$ and $C[2b+1, t+1] \leftarrow C[2b+1, t] + c_{op}(m, \lambda_{t+1})$
        for $i \leftarrow 2b$ to $0$ do    ▷ Downward minimization step
            if $C[i+1, t+1] < C[i, t] + c_{op}(\text{NODE}(i, m, y), \lambda_{t+1})$ then
                $S[i, t+1] \leftarrow S[i+1, t+1]$ and $C[i, t+1] \leftarrow C[i+1, t+1]$
            else
                $S[i, t+1] \leftarrow i$ and $C[i, t+1] \leftarrow C[i, t] + c_{op}(\text{NODE}(i, m, y), \lambda_{t+1})$
    return $(C, S)$

function EXTRACT_SCHEDULE$(S, T, m, y)$
    let $\mathcal{X}[1 \dots T]$ be a new array
    $i \leftarrow S[0, T]$    ▷ Get index of best selection for last time slot
    $\mathcal{X}[T] \leftarrow$ NODE$(i, m, y)$    ▷ Calculate best selection for last time slot
    for $t \leftarrow T-1$ to $1$ do    ▷ Iteratively obtain schedule from selection table
        $i \leftarrow S[i, t]$
        $\mathcal{X}[t] \leftarrow$ NODE$(i, m, y)$
    return $\mathcal{X}$
```

6 Conclusion

6.1 Summary

Our work started with a fairly general data center model consisting of a homogeneous collection of servers with fixed switching costs and convex operating costs. The data center's load varies with time. Consequently, the number of active servers and the workload distribution have to be dynamically adapted to minimize the data center's incurring costs. To accomplish this task, we first transferred our model's optimization problem to a simpler form in Chapter 3 by exploiting discovered patterns and properties of optimal schedules.

We then observed in Chapter 4 that the data center's optimization problem can be reduced to the shortest path problem of an acyclic graph. This observation was then used to derive an optimal algorithm with pseudo-polynomial time complexity $\Theta(Tm^2)$, which in a second step was improved to pseudo-linear time $\Theta(Tm)$ by adapting the graph's structure.

Lastly, we sought to reduce the algorithm's time complexity to polynomial time. For this, we had to move to approximative methods and slightly adapt our assumptions concerning the servers' operating costs function f in Chapter 5. More specifically, we assumed f to be non-negative and monotonically increasing; these assumptions, however, do not interfere with practical applicability, as discussed in Section 5.1. We first derived a 2-optimal algorithm with linear time complexity $\Theta\big(T \log_2(m)\big)$ by modifying the previously used graph. We then proved that this approach can be generalized to a $(1+\varepsilon)$-optimal algorithm with linear time complexity $\Theta\left(T \frac{\log(m)}{\log(1+\varepsilon)}\right)$, able to approximate optimal solutions with arbitrary precision.

It is noteworthy that all our derived algorithms provide integer solutions and can be easily implemented as a computer algorithm. In fact, a pseudo-code implementation for each developed and verified algorithm is given at the end of each respective section.

6.2 Future Work

Throughout this thesis, we assumed the data centers' servers to be homogeneous. As a consequence, we were able to focus on the sheer number of active servers and not on the exact set of servers which are used for the scheduling process. Further, we only considered servers with two different power states (active or shut-down). Modern machines, however, contain more than these two states, that is they contain multiple sleep states with varying energy consumption. It is thus an interesting question whether our approach can be generalized to models with more than one homogeneous server collection or multiple sleep states. Moreover, since all our algorithms deal with the offline variant of the data centers' scheduling problem, it is natural to ask whether our approaches can in some way be modified to work as an online algorithm. Lastly, since we did not discover a polynomial-time optimal algorithm, it remains an open question whether there exists a polynomial-time solution for our problem setting or it is a **NP** problem.

One possible application of our procedures might be the online algorithm developed by Lin et al. [7]. Their procedure uses a forward recurrence relation for which progressively larger convex programs have to be solved at each time step. At some time t, these convex programs calculate the optimal offline solution up to time t for two different models: one which charges its switching costs β when powering up servers, and one which charges β when shutting down servers. While the former case can be simply calculated by our derived optimal offline algorithm, the latter merely requires a small adaption of the algorithm (cf. Section 2.1). These convex programs can thus be exchanged by two instances of our optimal offline algorithm if – and this has not been proven so far – Lin's approach and proofs still work with integer solutions. If this is indeed the case, the combination of Lin's online algorithm and our approximative offline solution might be another topic worth to examine.

List of Algorithms

Bibliography

[1] N. Bansal, A. Gupta, R. Krishnaswamy, K. Pruhs, K. Schewior, and C. Stein. “A 2-Competitive Algorithm For Online Convex Optimization With Switching Costs”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Ed. by N. Garg, K. Jansen, A. Rao, and J. D. P. Rolim. Vol. 40. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 96–109. ISBN: 978-3-939897-89-7. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2015.96.

[2] L. A. Barroso and U. Hölzle. “The Case for Energy-Proportional Computing”. In: *Computer* 40.12 (Dec. 2007), pp. 33–37. ISSN: 0018-9162. DOI: 10.1109/MC.2007.443.

[3] *Cisco Global Cloud Index: Forecast and Methodology, 2015–2020*. 2016. URL: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf (visited on 07/16/2017).

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009. ISBN: 9780262258104.

[5] J. Hamilton. *Cost of Power in Large-Scale Data Centers*. 2008. URL: http://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/ (visited on 07/16/2017).

[6] M. Lin, Z. Liu, A. Wierman, and L. L. H. Andrew. “Online Algorithms for Geographical Load Balancing”. In: *Proceedings of the 2012 International Green Computing Conference (IGCC)*. IGCC ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–10. ISBN: 978-1-4673-2155-6. DOI: 10.1109/IGCC.2012.6322266.

[7] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. “Dynamic Right-sizing for Power-proportional Data Centers”. In: *IEEE/ACM Trans. Netw.* 21.5 (Oct. 2013), pp. 1378–1391. ISSN: 1063-6692. DOI: 10.1109/TNET.2012.2226216.

Notation

Input

$m \in \mathbb{N}$	Number of homogeneous servers
$T \in \mathbb{N}$	Number of time slots
$\lambda_1, \dots, \lambda_T \in [0, m]$	Arrival rates
$\Lambda := (\lambda_1, \dots, \lambda_T)$	Sequence of arrival rates
$\beta \in \mathbb{R}_{\geq 0}$	Switching costs of a server
$f : [0, 1] \to \mathbb{R}$	Convex operating cost function of a server. For Chapter 5, f is assumed to be non-negative and monotonically increasing.
$\mathcal{I} := (m, T, \Lambda, \beta, f)$	Input of a problem instance

Problem Statement

$s_{i,t} \in \{0, 1\}$	State of server i at time t; either shut-down (0) or active (1)
$S_i := (s_{i,1}, \dots, s_{i,T})$	Sequence of states for server i
$\lambda_{i,t} \in [0, 1]$	Assigned load for server i at time t
$L_i := (\lambda_{i,1}, \dots, \lambda_{i,T})$	Sequence of assigned loads for server i
$\mathcal{S} := (S_1, \dots, S_m)$	Sequence of all state changes
$\mathcal{L} := (L_1, \dots, L_m)$	Sequence of all assigned loads
$\Sigma := (\mathcal{S}, \mathcal{L})$	Schedule for a problem instance $\mathcal{I}$
$x_t \in [m]_0$	Number of active servers at time t
$\mathcal{X} := (x_1, \dots, x_T)$	Sequence of the number of active servers. Note that due to Chapter 3, $\mathcal{X}$ can be interpreted as a schedule.
$c_{op}(x, \lambda)$	Costs of processing λ with x servers (operating costs)

$c_{sw}(x_{t-1}, x_t)$	Costs of switching from x_{t-1} to x_t servers (switching costs)
$c(x_{t-1}, x_t, \lambda_t)$	Costs of switching from x_{t-1} to x_t servers and processing λ_t with x_t servers
$c(\mathcal{X})$	Costs of a schedule

Conventions

$\forall t \notin [T] : \lambda_t = 0$	There are no loads before and after the scheduling process.
$\forall t \notin [T] : s_{i,t} = x_t = 0$	All servers are shut down before and after the scheduling process.

Miscellaneous

$[n] \coloneqq \{1, \ldots, n\} \subset \mathbb{N}$	Natural numbers from 1 to n
$[n]_0 \coloneqq \{0, \ldots, n\} \subset \mathbb{N}_0$	Integers from 0 to n
Feasible schedule	A schedule is feasible if it processes all loads in due time.
y-approximation	A schedule/operation is called y-approximative if its cost is at most y times as much as the original schedule's/operation's cost.
y-optimal algorithm	A scheduling algorithm is y-optimal if its calculated schedule's cost is at most y times as much as an optimal schedule's cost.