Technische Universität München
Fakultät für Informatik

Algorithms for Energy Conservation in Data Centers

Jens Quedenfeld

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. habil. Alois Chr. Knoll

Prüfer der Dissertation:

1. Prof. Dr. Susanne Albers
2. Dr. Dimitrios Letsios (Lecturer)

Die Dissertation wurde am 26.01.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 13.07.2022 angenommen.

Abstract

Energy consumption in data centers has a high impact on both budget and environment. Unfortunately, much energy is wasted by servers running idle for long periods of time. Therefore, it makes sense to power down these servers so as to dynamically right-size the data center such that the number of active servers matches the arriving job volume. However, powering up also causes costs, so running a server idle for a short period of time is cheaper than powering it down and up immediately thereafter. Hence, algorithms are needed to decide when a server should be powered down.

In this PhD thesis, we study the *discrete* setting of the data center right-sizing problem where the number of active servers must be integral. Thereby, we gain truly feasible solutions. In general, the operating cost of a server is modeled by a time-dependent convex function of its load. We analyze both the offline and the online version of this problem. In the former one, all information is known in advance. In the latter one, the job volumes arrive one-by-one and the algorithm has to decide immediately if servers should be powered down or up.

First, we investigate data centers with identical servers and show how to calculate an optimal offline schedule in polynomial time. Regarding the online problem, we present a 3-competitive deterministic algorithm and show that no deterministic algorithm can achieve a better competitive ratio. Furthermore, we develop a randomized algorithm that is 2-competitive against an oblivious adversary and prove that no smaller competitive ratio is achievable. As a side result, we obtain a lower bound of 2 for the *fractional* setting. All lower bounds still hold in a more restricted model where the operating cost function is fixed in time. In addition, we show that the lower bounds are not affected if the online algorithm has a prediction window with constant length.

In the second part, we analyze heterogeneous data centers with d different server types. We present a $(1+\epsilon)$-approximation algorithm that calculates an offline schedule in polynomial time if d is a constant. We develop different online algorithms depending on the definition of the operating cost. If each server has the same computational power and if its operating cost is time- and load-independent (so it only depends on the server type), then we obtain a competitive ratio of $2d$ with a deterministic algorithm. The algorithm can be randomized such that it achieves a competitive ratio of $\frac{e}{e-1}d \approx 1.582d$ against an oblivious adversary. Afterwards, we consider operating costs that depend on the load of a server. For time-independent operating cost functions, we develop a $(2d+1)$-competitive deterministic algorithm. A modified version of this algorithm is able to handle time-dependent operating cost functions and achieves a competitive ratio of $2d+1+\epsilon$ for any $\epsilon > 0$. Finally, we show a lower bound of $2d$ for deterministic algorithms. The underlying problem instance uses constant operating cost functions, so the lower bound applies to all three variants.

Zusammenfassung

Das Senken des Energieverbrauchs in Rechenzentren ist sowohl aus ökonomischen als auch ökologischen Gründen wichtig. Leider wird viel Energie durch Server verschwendet, die für lange Zeiträume ungenutzt laufen. Daher ist es sinnvoll, die Größe eines Rechenzentrums dynamisch an den Bedarf anzupassen, indem ungenutzte Server heruntergefahren werden. Jedoch verursacht auch das Hochfahren erhöhte Kosten. Daher ist es günstiger, einen Server kurzzeitig im Leerlauf zu lassen, anstatt ihn herunter- und sofort danach wieder hochzufahren. Aus diesem Grund werden Algorithmen benötigt, die entscheiden, wann Server heruntergefahren werden sollen.

In dieser Dissertation erforschen wir diese Fragestellung in der *diskreten* Variante, bei der die Anzahl aktiver Server stets ganzzahlig sein muss. Dadurch erhalten wir Lösungen, die direkt in der Praxis eingesetzt werden könnten. Im Allgemeinen werden dabei die Betriebskosten eines Servers durch eine zeitabhängige konvexe Funktion der Last modelliert. Das Problem kann sowohl in der Offline- als auch in der Online-Version betrachtet werden. Bei der Ersteren ist im Voraus bekannt, wann wie viele Jobs in Zukunft eintreffen werden. Bei der Letzteren dagegen muss der Algorithmus sofort nach Eintreffen neuer Jobs entscheiden, welche Server hoch- oder heruntergefahren werden sollen.

Zunächst untersuchen wir Rechenzentren mit identischen Servern und präsentieren einen optimalen Polynomialzeit-Algorithmus für die Offline-Version. Anschließend entwickeln wir einen 3-kompetitiven deterministischen Online-Algorithmus und zeigen, dass kein deterministischer Algorithmus ein besseres Resultat erzielen kann. Außerdem entwerfen wir einen 2-kompetitiven randomisierten Online-Algorithmus und beweisen dessen Optimalität. Als Zwischenresultat erhalten wir eine untere Schranke von 2 für die *fraktionale* Variante. Alle unteren Schranken gelten auch für ein eingeschränktes Modell, bei dem die Betriebskosten-Funktion zeitlich konstant ist.

Im zweiten Teil der Arbeit analysieren wir heterogene Rechenzentren mit d verschiedenen Server-Typen. Wir präsentieren einen $(1+\epsilon)$-Approximationsalgorithmus mit polynomieller Laufzeit (sofern d konstant ist). Abhängig von der Definition der Betriebskosten entwickeln wir mehrere Online-Algorithmen. Falls alle Server dieselbe Rechenleistung haben und die Betriebskosten zeit- und lastunabhängig sind (also nur von dem Server-Typ abhängen), erreichen wir einen kompetitiven Faktor von $2d$. Durch Randomisierung lässt sich dieses Ergebnis auf $\frac{e}{e-1}d \approx 1.582d$ verbessern. Anschließend betrachten wir lastabhängige Betriebskosten und präsentieren einen $(2d+1)$-kompetitiven deterministischen Online-Algorithmus. Eine modifizierte Version ist in der Lage zeitabhängige Betriebskosten-Funktionen zu verarbeiten und erreicht dabei einen kompetitiven Faktor von $2d+1+\epsilon$ für ein beliebiges $\epsilon > 0$. Zum Abschluss beweisen wir für alle drei Varianten eine untere Schranke von $2d$ für deterministische Algorithmen.

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Dr. Susanne Albers, for suggesting this great topic, for her support and also for financing my doctorate by offering me a PhD position at her chair. I thank her for several long research talks, especially at the beginning of my doctorate.

Special thanks go to Kevin Schewior for two weeks of intense discussions about the right-sizing of heterogeneous data centers. By developing ideas together, I made much progress over the next months. Unfortunately, he left our chair quite soon.

I thank my present and former colleagues, including Ernst Bayer, Gunther Bidlingmaier, Alexander Eckl, Waldo Gálvez, Maximilian Janke, Arindam Khan, Dennis Kraft, Malte Kriegelsteiner, Leon Ladewig, Kelin Luo, Luisa Peter, Harald Räcke, Sebastian Schraink, Sebastian Schubert, Richard Stotz, Wessel van der Heijden, and Ruslan Zabrodin. We had a great and enjoyable time during and after work.

Many thank go to my friends Sven Schrinner, Henning Stute and John Maleki for proofreading parts of this thesis.

The biggest thanks of all go to my parents, Martin and Gabi Quedenfeld, for their limitless love, patience and support.

Contents

1 Introduction

Energy management in data centers is a major issue for both operators and environment. In 2019, worldwide about 350 TWh of electricity were consumed in data centers [RPSS20]. This exceeds the total power consumption in the UK [Baw16] and is about 1.5% of the global electricity production in 2019 [Ene20, VHLL+14]. Other sources even determined that about 3% of the worldwide produced electricity was used in data centers [Baw16].

Since about 18–28% of a data center's budget is invested in power [Ham08, Bri07], energy conservation can reduce the total cost of a data center significantly. There are data centers where the power-related costs even exceed the cost of purchasing the hardware [RSR+07]. It is expected that the worldwide power consumption of data centers will further increase [SSS+16, VHLL+14]. Andrae and Edler [AE15] estimated that the global power consumption of data centers will increase by at least 13% per year in the best-case scenario or up to 21% in the worst-case scenario. They expect that data centers will use 3%–13% of the worldwide electricity in 2030. Since 63% of the global electricity comes from fossil fuels like coal and gas [Rit20], reducing the energy consumption in data centers is not only important for economical but also for ecological reasons. Depending on the location of the data center, increasing its energy efficiency can reduce greenhouse gas emissions drastically.

The server utilization in a data center can be quite low. In fact, a common data center only achieves an average server utilization of 12%–18% [Dea14]. Other publications give even lower values of 10%–50% [BH07, Mit14] or 5%–20% [AFG+09]. Only for a few days a year there are activity bursts which require full processing power [Mit14]. However, the energy consumption of a server does not increase proportional with the load. Idle servers still consume 30%–50% of their peak power [SR09]. Lu et al. [LCA13] even states an energy consumption of more than 60% while being idle. This waste of energy can be reduced by powering down servers that are currently not needed so that the capacity of the data center is dynamically right-sized. However, powering up also causes costs like an increased power consumption for restarting the operating system and all required services [LWAT11a]. Furthermore, each state toggling causes a delay cost. When migrating a large virtual machine, this can be equivalent to running a server for several minutes [CFH+05]. Finally, power down and up operations lead to increased wear and tear of the hardware, so there is the risk that the server does not work properly after restarting. This wear-and-tear cost is in the order of the cost to run a server for an hour [BAC+08]. Therefore, holding an idle server in the active state for a short period of time is cheaper than powering it down and up again shortly after. Algorithms are needed to decide when it is beneficial to power down idle servers.

Modern data centers often consist of heterogeneous servers [GZ16]. For example, in 2020, about 30% of the Top 500 supercomputers use heterogeneous architectures [MSD+20]. Usually, heterogeneity results from servers with GPUs which perform massive parallel calculations much faster than common CPUs [MV15]. However, tasks that contain many branches are unsuitable for GPUs [Enc21a] and should be executed on a CPU. Heterogeneous architectures also include field-programmable gate arrays (FPGA) or other specialized hardware [Enc21b]. If the capacity of a data center is not sufficient any more, it is common practice to extend it with new servers. These new servers mostly use the same instruction set architecture, so from a programmer's point of view they behave equally. However, for deciding which server should be powered down (or up), the higher efficiency and the different switching cost of the new servers must be considered. We will see that the right-sizing problem for heterogeneous data centers is much harder than right-sizing a homogeneous data center where all servers are equal.

Modern hardware has several low-power states [UF21]. Instead of completely shutting down a server, it can be transferred to a sleep state where the main memory is still supplied with power to retain its data while most of the other hardware is powered down. Powering up from such a sleep state is much cheaper and faster than booting the server. In this work, we only investigate two-state problems where each server has an active and one inactive state.

The energy consumption of a server in the active state is not constant, but increases with the load [ALW10]. If a modern CPU runs with low load or is idle, its frequency is lowered to save energy [Mit14]. For high frequencies, the CPU voltage has to be raised, which leads to a superlinear increase in energy consumption [WAT09]. In general, the power consumption of a single server can be modeled by a non-negative increasing convex function of the load. For example, a commonly used function is $f(z) = z^\alpha + \beta$ for the constants $\alpha > 1$ and $\beta > 0$ [BGK+15]. More complicated models can be found in [DWF16] and [IM20].

1.1 Problem formulation

In this work, we study different variations of the *data center right-sizing problem*. In general, we consider a data center with d different server types. The data center contains m_j servers of type $j \in \{1, \dots, d\}$. Each server has an active state where it is able to process jobs and an inactive state where no cost occurs. If a server of type j is powered up, i.e., it is switched from the inactive to the active state, it causes a cost of β_j, called *switching cost*. Powering down does not cost anything. We consider a discrete finite time horizon consisting of the time slots $\{1, \dots, T\}$. For each time slot $t \in \{1, \dots, T\}$, a convex function g_t models the operating cost of the data center depending on the number of active servers. More precisely, $g_t(x_1, \dots, x_d)$ is the incurred cost if there are x_j active servers of each server type j. The function g_t models the arriving job volume, the best distribution of this job volume to the active servers as well as the individual

cost of each server depending on its load. In Chapter 3, we will see how g_t can be defined in detail.

The goal is to determine a *schedule*, specifying the number of active servers for each time slot, that minimizes the total cost. Formally, a schedule X is a sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ with $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,d})$ where each $x_{t,j}$ denotes the number of active servers of type j at time t. At the beginning and the end of the time horizon all servers are inactive, so $\boldsymbol{x}_0 = \boldsymbol{x}_{T+1} = \mathbf{0}$ where $\mathbf{0}$ is the d-dimensional zero vector. The total cost of the schedule X is given by

$$C(X) \coloneqq \sum_{t=1}^{T} g_t(x_{t,1}, \ldots, x_{t,d}) + \sum_{t=1}^{T} \sum_{j=1}^{d} \beta_j (x_{t,j} - x_{t-1,j})^+ \tag{1.1}$$

where $(x)^+ \coloneqq \max\{x, 0\}$. As already mentioned, the switching cost is only paid for powering up. However, this is not a restriction, since at the beginning and the end of the time horizon all servers are inactive, so the cost for powering down can be folded into the cost for powering up.

1.1.1 Offline and online problems

The data center right-sizing problem described above can be studied in the offline and the online version. In the offline version, all operating cost functions g_t are known in advance. The goal is to calculate an optimal schedule as fast as possible. For NP-hard problems an optimal solution cannot be calculated in polynomial time (unless P=NP). In this case, we are interested in approximation algorithms that calculate an approximation of the optimal solution in polynomial time. Formally, an algorithm that determines, for any input, a solution whose cost is at most c times larger than the cost of the optimal solution is called c-approximation. An algorithm that takes a parameter ϵ and calculates a $(1+\epsilon)$-approximation in polynomial time is called *polynomial time approximation scheme* (PTAS). If the runtime is not only polynomial in the problem size, but also in $1/\epsilon$, then the approximation algorithm is a *fully polynomial time approximation scheme* (FPTAS) [Vaz13].

An algorithm whose runtime is polynomial in the largest *numeric value* of the input, but not in the encoding length, has a *pseudo-polynomial* runtime [GJ79]. For example, if the runtime of an algorithm depends linearly on the number m of available servers, it does not run in polynomial time since encoding m only requires $\lceil \log_2 m \rceil$ bits.

In the online version of the data center right-sizing problem, the operating cost functions g_t arrive one-by-one. After receiving g_t, the algorithm has to choose the number of active servers $\boldsymbol{x}_t$ for the current time slot t without the knowledge of the future cost functions $g_{t+1}, g_{t+2}, \ldots$. In the worst case, a server is powered down at time t, then a huge job volume arrives so that the server has to be powered up again to process the incoming jobs. In the online setting, a schedule is compared with the best offline solution. An algorithm is called c-competitive if it is always guaranteed that the cost of the calculated schedule is at most c times larger than the cost of the optimal offline solution.

More formally, let $C^{\mathcal{I}}(X^{\mathcal{A}})$ denote the cost of a deterministic online algorithm $\mathcal{A}$ on the problem instance $\mathcal{I}$ and let $C^{\mathcal{I}}(X^*)$ be the cost of the optimal offline solution. Let $\alpha \geq 0$ be any constant independent of $\mathcal{I}$. Algorithm $\mathcal{A}$ is said to be c-competitive if the inequality

$$C^{\mathcal{I}}(X^{\mathcal{A}}) \leq c \cdot C^{\mathcal{I}}(X^*) + \alpha \tag{1.2}$$

is satisfied for any problem instance $\mathcal{I}$ [BEY98]. The factor c is also called *competitive ratio*. The runtime of an online algorithm is usually not analyzed.

The competitive ratio can be improved by taking random choices. To analyze the competitive ratio of a randomized online algorithm, an adversary is used which generates the input sequence and tries to maximize the ratio between the cost of the algorithm and its own cost. One distinguishes between different adversary models depending on the knowledge of the random choices. An oblivious adversary knows the online algorithm and solves the problem instance offline, but has no knowledge about the random choices of the algorithm. A randomized online algorithm $\mathcal{A}$ is said to be c-competitive against an oblivious adversary if, for any problem instance $\mathcal{I}$, its expected cost satisfies the inequality

$$\mathbb{E}[C^{\mathcal{I}}(X^{\mathcal{A}})] \leq c \cdot C^{\mathcal{I}}(X^*) + \alpha \tag{1.3}$$

for a constant $\alpha \geq 0$ independent of $\mathcal{I}$. There are other adversary models, namely the *adaptive online adversary* and the *adaptive offline adversary* [BDBK+94]. In this work, we only consider the oblivious adversary which is the most common adversary model[1].

It is not only interesting to design an algorithm with a small competitive ratio, but also to show that no algorithm exists that is able to beat a certain competitive ratio. If such a lower bound matches the competitive ratio of an algorithm, we know that this algorithm is optimal.

In the field of online learning, the *regret* is used to measure the performance of an online algorithm. The regret is the difference of the cost caused by the algorithm and the cost of the best *static* offline solution that never changes the number of active servers. In this work, we will only consider the competitive ratio which is the common performance measure for online algorithms [Kar92, BEY98].

1.1.2 Discrete and fractional setting

The data center right-sizing problem can be analyzed in the discrete and in the fractional setting. In the discrete setting, the number of active servers must be an integer, i.e., $x_{t,j} \in \{0, 1, \ldots, m_j\}$. This setting leads to truly feasible solutions.

On the other hand, there is the fractional setting where the number of active servers can be any real number, i.e., $x_{t,j} \in [0, m]$. This setting is reasonable, since the total number of servers in a data center is usually very large [LWAT13]. However, in practice the calculated schedule has to be rounded which may increase the competitive ratio

[1] In November 2021, Google Scholar [LLC21] found 845 publications containing the phrases "oblivious adversary" and "online algorithm", but only 157 or 92, if "oblivious adversary" is replaced by "adaptive offline adversary" or "adaptive online adversary", respectively.

significantly. For example, in the fractional setting, there is a $(d+1)$-competitive algorithm for the right-sizing problem with arbitrary convex operating cost functions [Sel20], while the best achievable competitive ratio in the discrete setting is at least exponential (see Section 3.2.5). The design and especially the analysis of algorithms solving the fractional setting can be quite different from algorithms for the discrete setting. In this work, we mostly study the discrete setting, since only integral schedules are feasible in practice.

1.2 Related work

Energy conservation in processors, mobile devices and especially data centers has received much attention in recent years, see for example [IP05] and [SBM+14] and references therein. One successful approach is *dynamic power management* (DPM), where unused components are powered down or where the performance of components is adapted to the current load [BD97, PBBDM98]. In case of a data center, servers can be shut down or transferred to a sleep state with low power consumption. Furthermore, the speed of the servers' processors can be decreased which is called *speed scaling* [ISG07].

In the following, we will first introduce publications examining dynamic power management on a single machine. Then, we present DPM literature for homogeneous and heterogeneous data centers and related problems. Afterwards, we show how the problem of right-sizing a data center is related to metrical task systems. Finally, we give an overview of other related problems and applications.

1.2.1 Single machine

The data center right-sizing problem for only one server with two states and constant operating cost is equivalent to the famous Ski-Rental problem. Each day, the online player can rent skis at cost r per day or buy it at cost β. At some day, it starts to rain and skiing is not possible any more. The optimal deterministic strategy is to rent the skis for $\beta/r - 1$ days and then to buy them if the weather is still fine. This results in a competitive ratio of 2 which is optimal [KMRS88]. The cost of renting skis corresponds to the constant operating cost of a server. Buying skis is equivalent to powering down a server. The best randomized algorithm achieves a competitive ratio of $e/(e-1) \approx 1.582$ [KKR01].

Irani et al. [ISG03] analyzed the power management of a single server with multiple sleep states in a continuous-time setting. In their model, the state transition costs are additive, i.e., switching form state i to j and then switching from j to k is as expensive as switching from i directly to k. They developed the 2-competitive deterministic *lower envelope algorithm* (LEA). Furthermore, they give a randomized version of LEA that achieves a competitive ratio of $e/(e-1) \approx 1.582$. Both competitive ratios are optimal, since the problem is a generalization of the Ski-Rental problem.

For non-additive transition costs, Augustine et al. [AIS08] give a simple algorithm achieving a competitive ratio of 5.828. Furthermore, they developed an algorithm that

produces a deterministic strategy for a given problem instance that is arbitrarily close to the optimal competitive ratio achievable for this problem instance. A related problem is the capital investment problem analyzed by [ABF+99] and [Dam03]. One has to buy devices with different purchase and operating costs. In general, not all devices are available at the beginning. If a new device is bought, the old one is powered down and causes no cost any more. In [ABF+99], at specific times an order arrives and one machine has to process it immediately by paying the operating cost. In [Dam03], there is always exactly one active device that causes the operating cost. Contrary to the right-sizing problem, it is not possible to power down the device; the only way to reduce the operating cost is to replace the machine by a cheaper one. Further generalizations of the Ski-Rental problem are for example analyzed in [FI05, LPSR12, FKF16].

1.2.2 Homogeneous data centers

Lin et al. [LWAT11a, LWAT13] analyzed the right-sizing problem for homogeneous data centers in the fractional setting, i.e., there is only one server type and the number of active servers can be fractional. In the conference version of their work [LWAT11a], the operating cost of a single server is modeled by a convex function f. Due to convexity, it is optimal to distribute the arriving job volume λ equally to the active servers, so the total operating cost at time t is given by $x \cdot f(\lambda/x)$ where x is the number of active servers. They developed an algorithm called Lazy Capacity Provisioning (LCP) that achieves a competitive ratio of 3. Furthermore, they tested their algorithm with real workloads and showed that it achieves nearly optimal results and leads to significant energy conservation. In the journal version [LWAT13], they showed that their results still hold in a generalized model where the operating costs at each time t are modeled by arbitrary convex functions $g_t(x)$. Further theoretical and empirical analyses of the LCP algorithm can be found in [WLC+15]. They combined stochastic models with the cost guarantees of the LCP algorithm.

Bansal et al. [BGK+15] developed a 2-competitive algorithm for the right-sizing problem on homogeneous data centers and showed a lower bound of 1.86. In [AQ18a], we showed that there is a lower bound of 2 for the fractional setting, so Bansal et al.'s algorithm is optimal (see Section 2.4.2). This result was independently found by Antoniadis and Schewior [AS17].

Andrew et al. [ABL+13] analyzed both the competitive ratio and the regret of online algorithms for the right-sizing problem. They showed that no algorithm can simultaneously achieve a constant competitive ratio and a sublinear regret. Furthermore, they developed a $(1+\gamma)$-competitive algorithm called Randomly Biased Greedy (RBG) that achieves a regret of $\mathcal{O}(\max\{T/\gamma, \gamma\})$ for $\gamma \geq 1$. By setting $\gamma = 1$, RBG achieves the optimal competitive ratio of 2. In the original paper [ABL+13], there is an error in the analysis of the RBG algorithm [ABN+16]. A revised version of the proof can be found in [ABL+15].

Gandhi et al. [GHBA10, GHB11] model a data center with m identical servers as a M/M/m queuing system. The arrivals of the jobs are modeled by a Poisson process and the job service times are exponentially distributed.

1.2.3 Heterogeneous data centers

The online version of the right-sizing problem for heterogeneous data centers in the *fractional* setting is a special case of convex function chasing (CFC) [Sel20], also known as smoothed online convex optimization (SOCO) [GCW17]. For each time slot t, a convex function $g_t : \mathbb{R}^d \to \mathbb{R}$ arrives. The online algorithm has to choose a point $\boldsymbol{x}_t \in \mathbb{R}^d$ and pays the operating cost $g_t(\boldsymbol{x}_t)$ plus the movement cost $\|\boldsymbol{x}_t - \boldsymbol{x}_{t-1}\|$ where $\|\cdot\|$ is any metric. In case of data center right-sizing, $\|\cdot\|$ is a Manhattan metric where each dimension is scaled by the switching cost.

A special case of CFC is convex body chasing (CBC) [Sel20]. Instead of a function g_t, a convex body F_t arrives at each time slot. The online algorithm has to choose a point inside this body, i.e., $x_t \in F_t$. It can be shown that CFC in d dimensions can be reduced to CBC in $d+1$ dimensions. Let $\|\cdot\|_{\text{CFC}}$ be the metric of the original CFC problem instance $\mathcal{I}$. Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{d+1}$ be points in the $(d+1)$-dimensional space and let $\|\boldsymbol{x} - \boldsymbol{y}\|_{\text{CBC}} \coloneqq \|(x_1, \ldots, x_d) - (y_1, \ldots, y_d)\|_{\text{CFC}} + \frac{1}{2}\|x_{d+1} - y_{d+1}\|$ be the metric used for the CBC problem instance $\mathcal{J}$. Each function g_t is replaced by its epigraph (i.e., the point set lying above the graph of g_t), followed by the hyperplane $\mathbb{R}^d \times \{0\}$. For an optimal solution $X^*_{\mathcal{J}}$, the movement cost along the $(d+1)$-th dimension in $\mathcal{J}$ is equal to the operating cost in $\mathcal{I}$. Therefore, the cost of the optimal solutions of both problems are equal. A schedule for the CBC problem instance $\mathcal{J}$ can be converted to a schedule for $\mathcal{I}$ without increasing the cost by only considering the points lying on the epigraphs and ignoring the $(d+1)$-th dimension. Therefore, a $f(d)$-competitive algorithm for $\mathcal{J}$ implies a $f(d+1)$-competitive algorithm for the CFC problem instance $\mathcal{I}$ [BLLS19]. Note that the alternative reduction proof from CFC to CBC in [ABN+16] contains an error [Pru21].

Convex body chasing was introduced by Friedman and Linial in 1993 [FL93]. They conjectured that there is a competitive algorithm for all metric spaces. This conjecture was unsolved for a long time. In 2018, Bansal et al. [BBE+18] found a competitive algorithm for a special case of CBC, namely the *nested* convex body chasing problem where each arriving body is located inside the previous one, i.e., $F_t \subset F_{t-1}$. The exponential competitive ratio of [BBE+18] was drastically improved in [ABC+19] and [BKL+20]. Bubeck et al. [BLLS19] found the first competitive algorithm for the general convex body chasing problem, however, the competitive ratio depends exponentially on d. Recently, Sellke [Sel20] developed a d-competitive algorithm for convex body chasing and a $(d+1)$-competitive algorithm for convex function chasing. A similar result was independently found by Argue et al. [AGGT20]. For CBC in a general metric, there is a lower bound of d, so the algorithms by Sellke and Argue et al. for the CBC problem are optimal. The proof uses the ℓ_∞ metric and is based on chasing the faces of a d-dimensional cube. For the Manhattan metric which is related to the data center

right-sizing problem, the best known lower bound is $\Omega(\log d)$ [ABC+19], so there is still a large gap.

Chen et al. [CGW18] studied a special case of convex function chasing in Euclidean space where the arriving functions are locally α-polyhedral. A function $g : \mathbb{R}^d \to \mathbb{R}$ with minimizer v is called locally α-polyhedral if there exists some $\epsilon > 0$ such that for all $x \in \mathbb{R}^d$ with $\|x - v\| \leq \epsilon$, the inequality $g(x) - g(v) \geq \alpha\|x - v\|$ holds. In other words, the arriving functions grow at least linearly with slope α as one moves away from their minimizer. Chen et al. developed an online algorithm called Online Balanced Descent (OBD) that achieves a competitive ratio of $3 + O(1/\alpha)$.

Goel and Wierman [GW19] showed that OBD achieves a competitive ratio of $3 + O(1/\mu)$, if the arriving functions are μ-strongly convex. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is called μ-strongly convex, if for all $x, y \in \mathbb{R}^d$, the inequality $f(y) \geq f(x) + \nabla f(x)^T \cdot (y - x) + \frac{m}{2}\|y - x\|$ is satisfied [Nes03]. For $d = 1$ and a twice continuously differentiable function f, this is equivalent to $f''(x) \geq m$ for all $x \in \mathbb{R}$. In [GLSW19], Goel et al. further analyzed different variants of the OBD algorithm.

Another algorithm that received much research is Receding Horizon Control (RHC) [LWAT11b]. It has a long history in control theory literature, see for example [KP77, MM88]. Lin et al. [LWAT11b] analyzed the data center right-sizing problem for online algorithms with a prediction window. The online algorithm knows not only the operating cost function for the current time slot, but also for the following w time slots. Lin et al. showed that RHC achieves a competitive ratio of $1 + \frac{\beta}{(w+1)g^{\min}}$ for homogeneous data centers where β is the switching cost and $g^{\min}$ is the minimal operating cost (if the operating cost functions are increasing, this is $\min_{t\in\{1,\dots,T\}} g_t(0)$). Note that the algorithm performs badly if the switching cost is much larger than the minimal operating cost. For heterogeneous data centers, RHC has a poor performance, so they used a modified version of RHC called Averaging Fixed Horizon Control (AFHC) to achieve a competitive ratio of $1 + \max_{j\in\{1,\dots,d\}} \frac{\beta_j}{(w+1)g_j^{\min}}$ where $g_j^{\min}$ is the minimal operating cost of server type j.

Chen et al. [CAW+15] analyzed AFHC under stochastic prediction error models. They developed a general prediction model that makes neither assumptions on the stochastic process modeling the jobs arrivals nor on the design of the predictor. They prove that AFHC achieves a constant competitive ratio and a sublinear regret which is only possible by using predictors [ABL+13]. Further algorithms using predictions are, for example, presented in [CCL+16, LQL18]. Lin et al. [LGW20] developed a randomized algorithm called *Synchronized Fixed Horizon Control* that is able to handle non-convex operating cost functions if two basic conditions between the operating and switching costs are satisfied. It achieves a competitive ratio of $1 + \mathcal{O}(1/w)$ for a prediction window of length w.

Albers [Alb19] analyzed the offline version of the right-sizing problem for heterogeneous data centers with load-*independent* operating costs in the discrete setting. For the two-state problem, she presented a polynomial-time algorithm based on a minimum-cost flow computation. If the servers have multiple sleep states, there is a d-approximation

algorithm that uses a fractional two-commodity minimum-cost flow combined with a rounding technique to gain an integral schedule. In [Sch18], it is shown that this problem is NP-hard if the number of server types and sleep states are unbounded.

For the fractional setting, in [BGK+15] it is mentioned that for homogeneous data centers the offline problem can be solved in polynomial time. However, this does not hold for heterogeneous data centers in general (unless P=NP), as several NP-hard problems can be formulated as a convex optimization problem [DKP02], so finding the minimum of g_t might be NP-hard.

1.2.4 Metrical task systems

The data center right-sizing problem in the *discrete* setting is a special case of *metrical task systems*. A metrical task system is defined by a finite set of n states $S = \{s_1, \dots, s_n\}$ and a distance function $d : S \times S \to \mathbb{R}_{\geq 0}$. For each time slot t, a cost function $g_t : S \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ arrives. Afterwards, the online algorithm has to choose a state $x_t \in S$ causing a cost of $g_t(x_t) + d(x_{t-1}, x_t)$.

For general metrical task systems there is a $(2n-1)$-competitive deterministic online algorithm and no deterministic algorithm can achieve a better competitive ratio [BLS92]. For randomized algorithms, there exists an algorithm with competitive ratio $\mathcal{O}(\log^2 n)$ against an oblivious adversary [BCLL21] and a lower bound of $\Omega(\log n / \log\log n)$ [BLMN03]. There are many results for special cases of metrical task systems. A famous special case is the k-server problem that was first mentioned in [MMS88]. There are k servers in a metric space S that have to serve requests consisting of a single point in S by moving any server to the request. Koutsoupias and Papadimitriou [KP95] showed that the deterministic *Work Function Algorithm* (WFA) has a competitive ratio of $2k-1$. It is an open question if there is any deterministic online algorithm that achieves the optimal competitive ratio of k [MMS90]. There are many results for randomized algorithms with polylogarithmic competitive ratios, see for example [BBMN15, BCL+18, Lee18]. Furthermore, many publications studied specializations of the general k-server problem, for example, the 2-server problem [Sit14] or the CNN-problem [KT04].

As already mentioned, the discrete setting of the data center right-sizing problem is also a special case of a metrical task system. The state set is the set of all possible server configurations, the distance function is equivalent to the switching cost and the cost functions g_t must be convex. Formally, $S = \{\boldsymbol{x} = (x_1, \dots, x_d) \mid \forall j \in \{1, \dots, d\} : x_j \in \{0, 1, \dots, m_j\}\}$ and $d(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \sum_{j=1}^{d} \beta_j (y_j - x_j)^+$. For homogeneous data centers with m servers, this simplifies to $S = \{0, 1, \dots, m\}$ and $d(x, y) = \beta(y - x)^+$. Note that the optimal deterministic algorithm for general metrical task systems achieves a competitive ratio of $2m - 1$, however, this competitive ratio is not desirable since it depends on the number of available servers. We will see in Chapter 2 that a much better competitive ratio is achievable for this special metrical task system.

1.2.5 Other related problems

There are many problems that are related to the data center right-sizing problem. One generalization is Geographical Load Balancing (GLB) with several data centers located at different places [LLW+11]. At each data center, a job volume $\lambda_{t,j}$ arrives at time t. The jobs can be distributed to other data centers by paying delay and energy costs depending on the source and destination data center. Liu et al. [LLWA12] analyzed the algorithms RHC and AFHC for the GLB problem and got the same competitive ratios as [LWAT11b] for right-sizing heterogeneous data centers. They also gave some experimental results and compared RHC and AFHC on real data with the optimal offline solution and a *local* solution without any load balancing. More experimental results for other GLB algorithms can be found in [GP13, LLW+15, ZZS18].

In this thesis, we always assume that the arriving jobs are very small and can be distributed arbitrarily to the active servers. Although this is common practice when analyzing large data centers [Alb19], in general this is not the case. There is a large variety of *scheduling* problems, where arriving jobs with different processing times have to be assigned to a set of machines [GLLK79, Alb03, PST04]. The jobs may have deadlines, i.e., a job must be finished before a specific time slot. Preemption, i.e., suspending a job and resuming it on another machine, may or may not be allowed. Depending on the problem, the machines can be identical, *related* or *unrelated*. If each machine has its individual speed, the machines are called *related*. On *unrelated* machines, each job has its individual processing time on each machine. Scheduling problems can be analyzed offline or online. There are many different objectives, for example, minimizing the makespan (this is the completion time of the last job) or the total energy consumption.

Baptiste [Bap06] combined a basic scheduling problem with dynamic power management where machines can be powered down to save energy. He analyzed offline scheduling for a single machine with a sleep state. The jobs have an arrival time, a deadline and a unit-sized processing time. The switching cost equals the cost of running the machine for one time slot. Baptiste developed a polynomial time algorithm that minimizes the total energy consumption. Later it was shown that the problem is still solvable in polynomial time for arbitrary processing times and switching costs [BCD12]. Demaine et al. [DGH+13] studied offline scheduling for identical machines with a sleep state and unit-sized jobs. They developed a polynomial time algorithm that minimizes the total power consumption and another one minimizing the number of gaps in the schedule. Antoniadis et al. [AGKK20] analyzed the generalization where the jobs have arbitrary processing times. They developed a 2-approximation for a single machine and a 3-approximation for m machines. Both algorithms have pseudo-polynomial runtimes, however, the running time can be made polynomial by only increasing the cost by a factor of $1 + \epsilon$.

Another technique to save energy is *speed scaling* where the frequency of a processor is reduced to decrease its energy consumption. Yao et al. [YDS95] investigated the speed scaling problem on a single machine. The energy consumption is modeled by a convex function of the machine's frequency. Again, the arriving jobs are described by

their arrival time, processing time and deadline. They developed a polynomial time algorithm that minimizes the energy consumption. Irani et al. [ISG07] extended the problem by adding a sleep state to the server. They gave a 2-approximation algorithm for the offline case and an online algorithm with a constant competitive ratio. Albers and Antoniadis [AA14] showed the NP-hardness of this problem and improved the approximation factor to 4/3. Finally, Antoniadis et al. [AHO15] developed a FPTAS for this problem.

Khuller et al. [KLS10] analyzed a similar problem where one has to select (related or unrelated) machines with different activation costs. The algorithm has an activation cost budget and wants to minimize the makespan. A generalization of this problem can be found in [LK11].

1.2.6 Other applications

In this thesis, we always consider the problem described by equation (1.1) in the context of right-sizing a data center. Nevertheless, the underlying mathematical problem has many other applications. For example, in portfolio management [Das14], one wants to maximize the profit while each purchase or sale of assets causes a transaction cost that is equivalent to the switching cost of the data center right-sizing problem. Another example is electric vehicle charging where the prices ($\boldsymbol{x}_t$) are adapted to smooth the energy consumption and to reduce peak loads [KG14]. To avoid rapid price changes, a switching cost is added to the objective function. Further applications are listed in [CGW18] and [LLWA12] including, e.g., video streaming [JDV12], speech animation [KYTM15] and power generation with varying demand [BLW15].

1.3 Contributions

In this work, we analyze different variants of the data center right-sizing problem in the discrete setting. In Chapter 2, we study homogeneous data centers where all servers are identical, i.e., $d = 1$. First, we investigate the offline problem and present a polynomial-time algorithm that relies on a grid-structured graph representation. The shortest path in that graph corresponds to an optimal schedule and is calculated with a binary search approach.

Regarding the online problem, we show that the deterministic LCP algorithm by Lin et al. [LWAT13] is still 3-competitive for the discrete setting. Our analysis is completely different from that by Lin et al. who studied the fractional setting and used methods that are not applicable to the discrete setting. Furthermore, we developed a 2-competitive randomized algorithm. It uses the algorithm of Bansal et al. [BGK+15] for fractional setting to obtain a 2-competitive fractional schedule. This schedule is rounded randomly in a specific way such that the resulting integral schedule is 2-competitive against an oblivious adversary.

In Section 2.4, we present several lower bounds for online algorithms. Initially, we show that no deterministic online algorithm can obtain a competitive ratio smaller than 3,

so the LCP algorithm is optimal in the discrete setting. For the fractional setting, we prove that there is no deterministic online algorithm that is better than 2-competitive. Based on this lower bound, we show that the smallest achievable competitive ratio for randomized algorithms in the discrete setting is 2. Hence, our randomized algorithm is optimal too. We prove that all lower bounds still hold for a more restricted model introduced by Lin et al. [LWAT11a]. In this model, the operating cost of a single server is given by a time-independent convex function $f(z)$ of the load z. All lower bounds still apply if the online algorithm has a prediction window with a constant length w, i.e., at time t, it knows the operating cost functions $g_t, g_{t+1}, \ldots, g_{t+w}$.

In Chapter 3, we analyze heterogeneous data centers with d different server types. The operating cost of a single server of type j running with load z at time t is modeled by a non-negative increasing convex function $f_{t,j}(z)$. First, we study the offline problem and describe an algorithm that calculates an optimal schedule by converting the problem instance to a graph and determining a shortest path. However, the runtime of this algorithm is only pseudo-polynomial. Therefore, we developed a $(1+\epsilon)$-approximation algorithm that uses a polynomial-sized subset of the vertices and has a polynomial runtime if d is a constant. Both the optimal offline and the approximation algorithm still work if the number of available servers depends on time.

Section 3.2 examines the online problem. We begin with the most simple case where all servers have the same computational power and their operating cost is time- and load-independent, i.e., $f_{t,j}(z) = r_j = \text{const}$. Furthermore, we exclude *inefficient* server types, i.e., a server with a higher switching cost has a lower operating cost and vice versa. For this simplified model, we devise a $2d$-competitive deterministic online algorithm and prove that no deterministic algorithm can achieve a smaller competitive ratio. Thus, our algorithm is optimal. Furthermore, we give a randomized version of our algorithm that achieves a competitive ratio of $\frac{e}{e-1}d \approx 1.582d$.

Afterwards, we study load-dependent operating costs and allow inefficient server types. The operating cost functions are still fixed in time (i.e., $f_{t,j} = f_j$), but they are able to model server types with different computational power. For this setting, we devise a $(2d+1)$-competitive deterministic online algorithm. If the operating cost is load-independent, our algorithm achieves the optimal competitive ratio of $2d$. Additionally, we demonstrate how our algorithm can be modified such that it can process time-dependent operating cost functions $f_{t,j}(z)$. We achieve a competitive ratio of $2d+1+\epsilon$ for any $\epsilon > 0$. Note that our lower bound uses the most restricted model ($f_{t,j}(z) = \text{const}$), so all presented deterministic algorithms are almost optimal. Finally, we show that arbitrary convex operating cost functions g_t lead to an at least exponential competitive ratio.

1.4 Outline and publication summary

This PhD thesis is publication based, so it only contains short proof ideas. The corresponding peer-reviewed papers including the full proofs were published as open access and are included in the appendix.

Chapter 2 handles homogeneous data centers. The underlying paper was published in the conference proceedings of the *30th ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA) in 2018 and is presented in Appendix B. Due to space limitations, the paper does not contain all proofs. A full version was submitted to the ACM journal *Transactions on Parallel Computing* on 16th March 2021, but has not yet been accepted. A preprint of this version can be found on arXiv, see [AQ18b].

Chapter 3 examines heterogeneous data centers. The presented results were published in two conference papers that can be found in Appendix C and D. The first one was published in the proceedings of the *12th International Conference on Algorithms and Complexity* (CIAC) in 2021. The second one appeared in the proceedings of the *33rd ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA) in 2021. Again, due to space limitations, some proofs are missing. A full version of the CIAC paper was recently published in a special issue of *Theoretical Computer Science* [AQ21a]. A freely accessible preprint can be found on arXiv [AQ21b]. A full version of the SPAA paper was recently submitted to a special issue of the ACM journal *Transactions on Parallel Computing*, but has not yet been accepted. A preprint of this version was uploaded on arXiv [AQ21d].

Chapter 4 concludes the results and presents several open questions for future work.

1.5 Notation

In this work, we use the following notation. By $\mathbb{N}$ we denote the set $\{1, 2, \dots\}$ of positive integers. For $k \in \mathbb{N}$, let $[k] \coloneqq \{1, 2, \dots k\}$ and $[k]_0 \coloneqq \{0, 1, \dots k\}$ be the sets of positive integers up to k excluding or including 0, respectively. Let $[k : l] \coloneqq \{k, k+1, \dots, l\}$ with $k, l \in \mathbb{Z}$ denote the integers from k to l. For $x \in \mathbb{R}$, let $(x)^+ \coloneqq \max\{0, x\}$, i.e., for a negative x, $(x)^+$ equals 0. Let $[x]_a^b \coloneqq \max\{a, \min\{b, x\}\}$ be the projection of x into the interval $[a, b]$. We write d-dimensional vectors in bold face, for example, $\boldsymbol{x}_t$ or $\boldsymbol{\beta}$. Let $\mathbb{E}[Z]$ be the expected value of the random variable Z.

A tabular overview of the variables used in this work is shown in Appendix A.

2 Homogeneous data centers

In this chapter, we study the right-sizing of *homogeneous* data centers where all servers are identical. Formally, we consider the problem formulation presented in Section 1.1 for only one server type, i.e., $d = 1$. In this case, the index j of several variables that represents the server type is not needed any more. To simplify the notation, let $m \coloneqq m_1$ be the number of available servers and let $\beta \coloneqq \beta_1$ be the switching cost. A schedule is a sequence $X = (x_1, \dots, x_T)$ where x_t denotes the number of active servers during time slot t. At the beginning and the end of the workload, all servers are powered down, i.e., $x_0 = x_{T+1} = 0$. The operating cost is modeled by non-negative convex functions g_t for any $t \in [T]$. More precisely, x_t active servers cause an operating cost of $g_t(x_t)$ during time slot t. The total cost of a schedule X is given by

$$C(X) \coloneqq \sum_{t=1}^{T} g_t(x_t) + \sum_{t=1}^{T} \beta(x_t - x_{t-1})^+.$$

A problem instance is defined by the tuple $\mathcal{I} = (T, m, \beta, \mathcal{G})$ with $\mathcal{G} = (g_1, \dots, g_T)$. To simplify the notation, let $g_{T+1}(x) \coloneqq 0$.

For our lower bounds, we resort to a more restricted model introduced by Lin, Wierman, Andrew and Thereska [LWAT11a]. In this model, at each time slot t a job volume $\lambda_t \in \mathbb{R}_{\geq 0}$ arrives that can be arbitrarily distributed to the active servers. The operating cost of a single server is modeled by a non-negative convex function f that is fixed for the whole time horizon. A single server that runs with a load $z \in [0, 1]$ causes an operating cost of $f(z)$. Since f is a convex function, it is optimal to distribute the arriving job volume equally to the active servers [AQ21e, Lemma 2.2]. Therefore, the operating cost of a single server at time t is equal to $f(\lambda_t / x_t)$ and the total operating cost at time t is given by $g_t(x_t) \coloneqq x_t \cdot f(\lambda_t / x_t)$. Note that in contrast to heterogeneous data centers (see Chapter 3), the function f is not required to be increasing. Formally, a problem instance of the restricted model is defined by the tuple $\mathcal{I} = (T, m, \beta, f, \Lambda)$ with $\Lambda = (\lambda_1, \dots, \lambda_T)$.

In the following sections, we present our results for both the offline and online version of this problem. If nothing else is mentioned, we consider the discrete setting where the number of active servers must be integral, i.e., $x_t \in [m]_0$ for all $t \in [T]$.

First, we examine the offline version and present a polynomial-time algorithm based on a binary search approach (Section 2.1). Afterwards, we investigate the online version. In Sections 2.2 and 2.3, we present a 3-competitive deterministic and 2-competitive randomized online algorithm, respectively. Section 2.4 examines lower bounds for the discrete and fractional setting. First, we give a lower bound of 3 for deterministic

algorithms in the discrete setting. Then, we show a lower bound of 2 for the fractional setting where the number of active servers is allowed to be fractional, i.e., $x_t \in [0, m]$. Based on this lower bound, we are able to prove that no randomized algorithm in the discrete setting is better than 2-competitive against an oblivious adversary. All lower bounds hold for the more restricted model introduced by Lin et al. [LWAT11a]. Finally, we show that prediction windows with a constant length cannot improve the competitive ratio of an algorithm.

Most results presented in this chapter were published in the conference proceedings of the *30th ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA) [AQ18a] that is shown in Appendix B. This conference paper does not include the 2-competitive randomized algorithm of Section 2.3 as well as the lower bound for randomized online algorithms (Section 2.4.3). Furthermore, several proofs are missing due to space limitations. An extended version containing all proofs as well as the results for randomized algorithms can be found on arXiv [AQ18b].

The notation introduced above slightly differs from the notation in [AQ18a] and [AQ18b] where the operating cost functions are denoted by f_t. Furthermore, in this thesis a problem instance is denoted by $\mathcal{I}$ instead of $\mathcal{P}$. We choose the different notation to make it consistent with Chapter 3 and our other publications.

2.1 Optimal offline algorithm

In this section, we present an offline algorithm that calculates an optimal schedule in $\mathcal{O}(T \log m)$ time. The basic idea is to convert the problem instance into a grid-structured graph and then calculate a shortest path. The graph contains $m + 1$ vertices for each time slot, so calculating a shortest path directly would only lead to a pseudo-polynomial runtime, since the encoding length of the problem instance depends logarithmically on m. To achieve a polynomial time, we first use only a constant number of vertices for each time slot. Then, we improve the solution iteratively using binary search such that we finally obtain an optimal schedule.

More precisely, a problem instance $\mathcal{I}$ can be represented as a directed, acyclic, weighted graph $G = (V, E)$. For each time slot $t \in [T]$ and for each possible number of active servers $x \in [m]_0$, there is a vertex $v_{t,x}$. Furthermore, the graph contains two vertices $v_{0,0}$ and $v_{T+1,0}$ for the beginning and the end of the time horizon where all servers are powered down. For any $x \in [m]_0$, the vertex set $\mathcal{R}_x = \{v_{t,x} \in V \mid t \in [T]\}$ is called *row* x, and for any $t \in [T]$, the vertex set $\{v_{t,x} \in V \mid x \in [m]_0\}$ is called *column* t.

For each time slot $t \in [T + 1]$ and for each $x, y \in [m]_0$, there is a directed edge from $v_{t-1,x}$ to $v_{t,y}$ with weight $g_t(y) + \beta(y - x)^+$ (if the vertices exist). The weight corresponds to the operating cost for y active servers at time t plus the switching cost for powering up $y - x$ servers if $y > x$. The structure of G is visualized in Figure 2.1.

Each path from $v_{0,0}$ to $v_{T+1,0}$ represents one specific schedule and vice versa. Due to the grid structure, a path contains exactly one vertex of each column. If the path passes

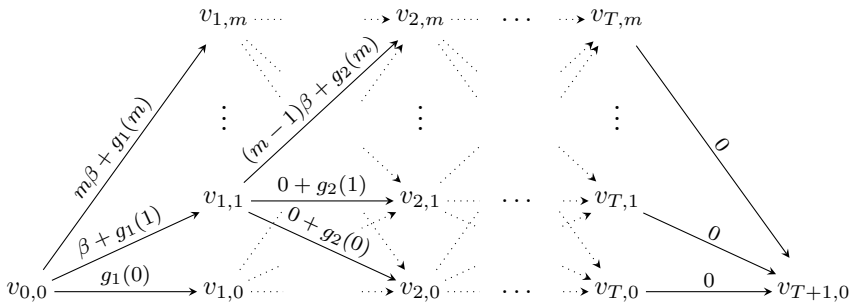

Figure 2.1: Construction of the graph G. Each path from $v_{0,0}$ to $v_{T+1,0}$ corresponds to a schedule and vice versa. For example, if the path passes through the vertex $v_{2,1}$, then the corresponding schedule uses one active server during time slot 2.

through $v_{t,x}$, then the corresponding schedule uses $x_t = x$ active servers at time t. The total weight of a path is equal to the cost of the corresponding schedule.

A shortest path in G can be calculated in $\mathcal{O}(Tm)$ time by using dynamic programming. However, as already mentioned, this runtime is only pseudo-polynomial, since the encoding length of the problem instance is in $\mathcal{O}(T + \log m)$. Therefore, we devise an algorithm that determines a shortest path using binary search. We assume that the total number of available servers m is a power of two. If this is not the case, the problem instance can be extended by increasing the number of available servers to next greater power of two and expanding the domain of the operating cost functions such that it is never beneficial to use more than m active servers (see [AQ18a] for more information).

Our algorithm runs in $\log_2 m - 1$ iterations denoted reversely by $k = K := \log_2 m - 2$ for the first iteration and $k = 0$ for the last one. The number of active servers considered in iteration k is always a multiple of 2^k. The problem instance solved in iteration k is denoted by $\mathcal{I}^k = (T, m, \beta, \mathcal{G}, M^k)$ where $M^k := \{n \in [m]_0 \mid n \mod 2^k = 0\}$ is the set of allowed states, i.e., a feasible schedule X for this problem instance has to satisfy $x_t \in M^k$ for all $t \in [T]$. Note that the original problem instance is given by $\mathcal{I} = \mathcal{I}^0$.

In the first iteration, we only use the rows $\mathcal{R}_0, \mathcal{R}_{m/4}, \mathcal{R}_{m/2}, \mathcal{R}_{3m/4}, \mathcal{R}_m$. The corresponding graph contains 5 vertices per column, so a shortest path can be computed in $\mathcal{O}(T)$ time by dynamic programming. After determining an optimal schedule $\hat{X}^k = (\hat{x}_1^k, \ldots, \hat{x}_T^k)$ for $\mathcal{I}^k$, we use the neighborhood of each state $\hat{x}_t^k$ for the next iteration. More precisely, in column t of the next iteration $k-1$, we consider the vertices $v_{t,x}$ with $x \in \{\hat{x}_t^k + \xi \cdot 2^{k-1} \in [m]_0 \mid \xi \in \{-2, -1, 0, 1, 2\}\}$. The vertices with $\xi \in \{-2, 0, 2\}$ were already used in the previous iteration, so we just add the intermediate vertices with $\xi = -1$ and $\xi = 1$. If $\hat{x}_t^k = 0$ or $\hat{x}_t^k = m$, then there are only three instead of five vertices in column t of iteration $k-1$. Figure 2.2 shows an example of one iteration. Since there are at most five vertices in each column, we achieve a runtime of $\mathcal{O}(T)$ per iteration

and therefore a total runtime of $\mathcal{O}(T \log m)$. The following pseudo code clarifies how our algorithm works. The set V^k contains the vertices considered in iteration k.

Algorithm 1 GraphBasedBinarySearch

Input: $\mathcal{I} = (T, m, \beta, \mathcal{G})$
Output: Optimal offline schedule

Extent the problem instance such that m is a power of two
$K \coloneqq \log_2 m - 2$
for $k \coloneqq K$ **down to** 0 **do**
  if $k = K$ **then**
    $V^k \coloneqq \{v_{0,0}, v_{T+1,0}\} \cup \left\{v_{t,x} \in V \mid t \in [T], x \in M^K\right\}$
  else
    $M_t \coloneqq \left\{\hat{x}_t^k + \xi \cdot 2^{k-1} \in [m]_0 \mid \xi \in \{-2, -1, 0, 1, 2\}\right\}$ for all $t \in [T]$
    $V^k \coloneqq \{v_{0,0}, v_{T+1,0}\} \cup \{v_{t,x} \in V \mid t \in [T], x \in M_t\}$
  $G^k \coloneqq (V^k, E \cap (V^k \times V^k))$
  Calculate a shortest path from $v_{0,0}$ to $v_{T+1,0}$ in G^k
  Set $\hat{X}^k$ to the schedule corresponding to the shortest path
return $\hat{X}^0$

Correctness. So far, we have not proven that the schedule calculated by GraphBasedBinarySearch is actually optimal. Here, we only give a short proof idea, for more information see [AQ18a]. We consider the continuous extension $\bar{\mathcal{I}} = (T, m, \beta, \bar{\mathcal{G}}, [0, m])$ of the original problem instance where the operating cost functions $\bar{g}_t$ are defined as piecewise linear functions such that $\bar{g}_t(x) = g_t(x)$ for all $x \in [m]_0$. Formally, $\bar{\mathcal{G}} \coloneqq (\bar{g}_1, \ldots, \bar{g}_T)$ with

$$\bar{g}_t(x) \coloneqq \begin{cases} g_t(x) & \text{if } x \in [m]_0 \\ (\lceil x \rceil - x) \cdot g_t(\lfloor x \rfloor) + (x - \lfloor x \rfloor) \cdot g_t(\lceil x \rceil) & \text{else.} \end{cases} \tag{2.1}$$

In other words, $\bar{\mathcal{I}}$ is the fractional setting of the original problem instance $\mathcal{I}$. Let $\Omega(\mathcal{J})$ be the set of the optimal schedules for a given problem instance $\mathcal{J}$.

An optimal fractional solution for $\bar{\mathcal{I}}$ can be converted to an integral one without increasing the cost, as the following lemma shows.

Lemma 2.1 [AQ18a, Lemma 2.4]**.** *Let $X^* \in \Omega(\bar{\mathcal{I}})$. Then, the schedule $\lfloor X^* \rfloor \coloneqq (\lfloor x_1^* \rfloor, \ldots, \lfloor x_T^* \rfloor)$ is optimal , i.e., $\lfloor X^* \rfloor \in \Omega(\bar{\mathcal{I}})$.*

The correctness of this statement follows from the piecewise linear definition of $\bar{g}_t$ and its convexity. The lemma implies that both the original problem instance $\mathcal{I}$ and its continuous extension $\bar{\mathcal{I}}$ have a common optimal solution whose cost is equal for both instances. Lemma 2.1 is quite important, as it is not only required to show the correctness of GraphBasedBinarySearch, but also later for analyzing our randomized online algorithm as well as for proving the lower bound of randomized online algorithms.

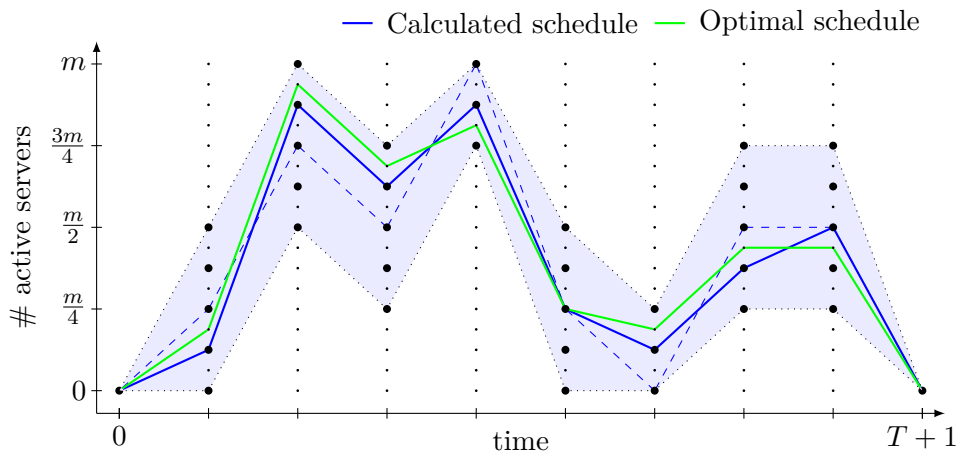

Figure 2.2: Visualization of one iteration of GraphBasedBinarySearch. The dashed blue line shows the schedule $\hat{X}^K$ of the first iteration ($k = K$) where only the states $0, m/4, m/2, 3m/4$ and m are available. The light blue area and the thick dots indicate the allowed states for the next iteration $k = K - 1$. The schedule $\hat{X}^{K-1}$ calculated in this iteration is visualized by the solid blue line. The optimal schedule is drawn in green. Note that it always stays inside the blue area. The vertical distance between the solid blue and green line is at most $m/16$ (i.e., the distance between two neighboring dots) as proven by Lemma 2.3.

For each iteration k and each optimal schedule $\hat{X}^k \in \Omega(\mathcal{I}^k)$, there exists an optimal schedule X^* for the continuous extension $\bar{\mathcal{I}}$ such that the distance between $\hat{X}^k$ and X^* is smaller than 2^k for all time slots. Formally, this is expressed by the following lemma.

Lemma 2.2 [AQ18a, Lemma 2.3]**.**

$$\forall k \in [K]_0 : \forall \hat{X}^k \in \Omega(\mathcal{I}^k) : \exists X^* \in \Omega(\bar{\mathcal{I}}) : \forall t \in [T] : |\hat{x}_t^k - x_t^*| < 2^k.$$

Proof idea. This lemma is proven by contradiction. We assume that there exists an optimal schedule $\hat{X}^k \in \Omega(\mathcal{I}^k)$ such that for all optimal schedules $X^* \in \Omega(\bar{\mathcal{I}})$, the condition $|\hat{x}_t^k - x_t^*| < 2^k$ is violated for at least one time slot t. Given a fractional schedule X, let $J(X)$ be the set of the inclusion maximal time intervals with $|x_t - x_t^*| \geq 2^k$ or $x_t \notin M^k$, i.e., x_t violates the condition or is not a multiple of 2^k. We define a transformation ϕ that reduces the number of time slots in $J(X)$ by at least one without increasing the cost of the schedule. For each time interval in $J(X)$, we move towards the optimal schedule, but do not exceed it, so the sign of $x_t - x_t^*$ does not change after transforming x_t.

Beginning from $\hat{X}^k$, we apply the transformation several times, until we eventually obtain a schedule Z whose states are multiples of 2^k and that fulfills the condition $|z_t - x_t^*| < 2^k$ for all time slots $t \in [T]$. Since the operating and switching costs are

convex functions and since we are moving towards the optimum, the cost of Z is smaller than or equal to the cost of the original schedule $\hat{X}^k$. Since $\hat{X}^k$ is optimal for $\mathcal{I}^k$, the cost of the schedules $\hat{X}^k$ and Z must be equal. Furthermore, convexity implies that the cost of Z equals the cost of X^* regarding the continuous problem instance $\bar{\mathcal{I}}$. Therefore, $\hat{X}^k$ is an optimal schedule for $\bar{\mathcal{I}}$ satisfying the condition $|\hat{x}_t^k - x_t^*| < 2^k$ for all time slots $t \in [T]$. This contradicts our assumption that no such schedule exists, so the statement of the lemma must be correct. □

In simple terms, Lemma 2.2 proved that for each optimal schedule $\hat{X}^k$, there exists an optimal fractional schedule $X^* \in \Omega(\bar{\mathcal{I}})$ within a distance of 2^k for each time slot. The following lemma expands this statement. Given an optimal schedule for $\mathcal{I}^k$, there is an optimal schedule for the subsequent iteration $k-1$ such that the distance between both schedules is at most 2^k for each time slot.

Lemma 2.3 [AQ18a, Lemma 2.5]**.**

$$\forall k \in [K]_0 : \forall \hat{X}^k \in \Omega(\mathcal{I}^k) : \exists \hat{X}^{k-1} \in \Omega(\mathcal{I}^{k-1}) : \forall t \in [T] : |\hat{x}_t^k - \hat{x}_t^{k-1}| \leq 2^k$$

Proof idea. First, we apply Lemma 2.2 with an appropriately scaled version of the original problem instance $\mathcal{I}$. Afterwards, we use the fact that a fractional schedule can be rounded down to an integral one without increasing its cost, as shown by Lemma 2.1. The resulting integral schedule still fulfills the distance condition. By undoing the scaling, we finally get an optimal schedule $\hat{X}^{k-1}$ for the problem instance $\mathcal{I}^{k-1}$ that satisfies $|\hat{x}_t^k - \hat{x}_t^{k-1}| \leq 2^k$ for all time slots $t \in [T]$. □

Now, we are able to prove the correctness of GraphBasedBinarySearch.

Theorem 2.1 [AQ18a, Theorem 2.6]**.** *The algorithm GraphBasedBinarySearch calculates an optimal schedule for the homogeneous data center right-sizing problem in* $\mathcal{O}(T \log m)$ *time.*

Proof. The runtime was already analyzed above, so it is sufficient to prove the optimality of the returned schedule. Let $\hat{X}^k$ be the schedule calculated in iteration k. We will show by induction that $\hat{X}^k$ is optimal for the problem instance $\mathcal{I}^k$.

In the first iteration, the algorithm calculates an optimal schedule for $\mathcal{I}^K$, since all states of M^K are considered. Assume that the schedule $\hat{X}^k$ calculated in iteration k is optimal for $\mathcal{I}^k$. In the next iteration, the algorithm only considers the states $x_t \in M^{k-1}$ with $|\hat{x}_t^k - x_t| \leq 2^k$. By Lemma 2.3, there exists an optimal schedule X for $\mathcal{I}^{k-1}$ with exactly this property. Therefore, the schedule $\hat{X}^{k-1}$ calculated in iteration $k-1$ is optimal for $\mathcal{I}^{k-1}$. Finally, by induction, the algorithm returns a schedule that is optimal for $\mathcal{I}^0$ which is the original problem instance. □

2.2 Deterministic online algorithm

In the online version of the data center right-sizing problem, the operating cost functions g_t arrive one-by-one. The online algorithm has to determine the number x_t of

active servers at time t without the knowledge of the future operating cost functions $g_{t+1}, g_{t+2}, \dots$. In this section, we adapt the *Lazy Capacity Provisioning* (LCP) algorithm by Lin et al. [LWAT13] to the discrete setting and show that it still achieves a competitive ratio of 3 as in the fractional setting. Although the algorithm is quite similar, our analysis is completely different from that by [LWAT13]. Later, in Section 2.4.1, we will see that no deterministic algorithm can achieve a better competitive ratio in the discrete setting, so the LCP algorithm is optimal.

Roughly, the LCP algorithm works as follows. We define a lower and an upper bound denoted by x_t^L and x_t^U, respectively. The LCP algorithm stays lazily between these bounds, i.e., $x_t^L \le x_t^{\mathrm{LCP}} \le x_t^U$ holds for all time slots $t \in [T]$. It only changes its state if it is necessary to satisfy this condition. The lower bound x_t^L is the last state of the optimal offline solution for the shortened problem instance that ends at time slot t. For the upper bound, we pay the switching cost for powering down, while power-up operations do not cost anything. Similarly to the lower bound, we consider the problem instance up to time t. The upper bound x_t^U is the last state of an optimal offline solution for this problem instance.

More precisely, let $X_\tau^L = (x_{\tau,1}^L, \dots, x_{\tau,\tau}^L)$ be the schedule that minimizes

$$C_\tau^L(X) \coloneqq \sum_{t=1}^{\tau} g_t(x_t) + \beta \sum_{t=1}^{\tau} (x_t - x_{t-1})^+ \tag{2.2}$$

with $X = (x_1, \dots, x_\tau)$. Similarly, let $X_\tau^U = (x_{\tau,1}^U, \dots, x_{\tau,\tau}^U)$ be the schedule that minimizes

$$C_\tau^U(X) \coloneqq \sum_{t=1}^{\tau} g_t(x_t) + \beta \sum_{t=1}^{\tau} (x_{t-1} - x_t)^+. \tag{2.3}$$

The only difference between equations (2.2) and (2.3) is the definition of the switching cost. The lower and upper bound are the last states of X_τ^L and X_τ^U, respectively, so $x_\tau^L \coloneqq x_{\tau,\tau}^L$ and $x_\tau^U \coloneqq x_{\tau,\tau}^U$. If there are several optimal schedules X_τ^L or X_τ^U, then x_τ^L is the smallest and x_τ^U is largest possible value.

Let $[x]_a^b \coloneqq \max\{a, \min\{b, x\}\}$ be the projection of x into the interval $[a, b]$. The LCP algorithm is defined by

$$x_\tau^{\mathrm{LCP}} \coloneqq \begin{cases} 0 & \text{if } \tau = 0, \\ [x_{\tau-1}^{\mathrm{LCP}}]_{x_\tau^L}^{x_\tau^U} & \text{if } \tau \ge 1. \end{cases}$$

Figure 2.3 visualizes how the LCP algorithm works. The lower and upper bound can be calculated with the offline algorithm GraphBasedBinarySearch presented in the previous section. This requires $\mathcal{O}(t \log m)$ time per arriving function. Alternatively, one can use a dynamic program that is updated when a new function arrives. In this case, we only need to store the minimal cost of the shortest path to the vertices of the last column, but not the operating cost functions. This leads to a runtime of $\mathcal{O}(m)$ per time slot.

Theorem 2.2 [AQ18a, Theorem 3.13]**.** *The LCP algorithm is 3-competitive.*

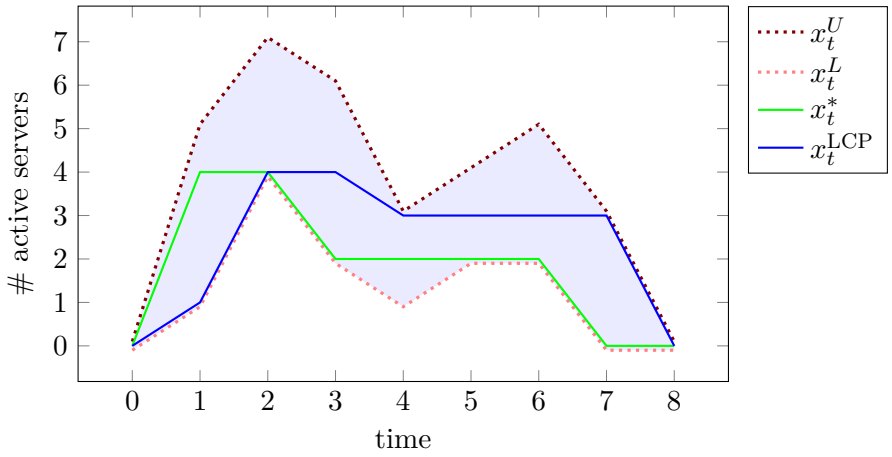

Figure 2.3: Visualization of the LCP algorithm. The pink and brown plot show the lower and upper bound. The LCP schedule (printed in blue) stays lazily between them. Similarly, the optimal schedule (shown in green) stays lazily between them *backwards* in time.

The proof consists of several lemmas. Here, we only present the most important ones. First, we observe that an optimal offline schedule always stays between the lower and upper bound.

Lemma 2.4 [AQ18a, Lemma 3.1]**.** *Given an optimal schedule X^*, the inequality*

$$x_\tau^L \leq x_\tau^* \leq x_\tau^U$$

holds for all $\tau \in [T]$.

The lemma can be proven by using the definitions of x_τ^L and x_τ^U. The next lemma shows that an optimal schedule can be constructed by staying lazily between the lower and upper bounds *backwards* in time.

Lemma 2.5 [AQ18a, Lemma 3.6]**.** *The schedule $X^* = (x_1^*, \dots, x_T^*)$ defined by*

$$x_t^* \coloneqq \begin{cases} 0 & \text{if } t = T+1 \\ [x_{t+1}^*]_{x_t^L}^{x_t^U} & \text{if } t \in [T] \end{cases}$$

is optimal.

This lemma can be proven by induction in reserve time. The proof uses Lemma 2.4 as well as the convexity of the operating and switching costs.

In the following, let X^* be the optimal schedule given by Lemma 2.5. When comparing the schedules X^{LCP} and X^*, we observe that there are time intervals where LCP has more active servers than the optimal schedule (i.e., $x_t^{\mathrm{LCP}} > x_t^*$) and others where it has

less active servers (i.e., $x_t^{\text{LCP}} < x_t^*$). Between these intervals, there is at least one time slot where both schedules use the same number of servers, i.e., $x_t^{\text{LCP}} = x_t^*$ [AQ18a, Lemma 3.7]. During the time intervals with $x_t^{\text{LCP}} > x_t^*$, the number of active servers of both X^{LCP} and X^* are non-increasing [AQ18a, Lemma 3.8], i.e., $x_t \geq x_{t+1}$ holds for both schedules. We call these intervals *decreasing*. Analogously, during so called *increasing* intervals with $x_t^{\text{LCP}} < x_t^*$, both x_t^{LCP} and x_t^* are non-decreasing, i.e., $x_t \leq x_{t+1}$. This implies that the switching cost of the LCP algorithm must be equal to the switching cost of the optimal offline solution.

Lemma 2.6 [AQ18a, Lemma 3.9]**.** *Let $S(X) \coloneqq \sum_{t=1}^{T} \beta(x_{t+1} - x_t)^+$ be the switching cost of the schedule X. It holds that $S(X^{LCP}) = S(X^*)$.*

Note that in the original paper, only $S(X^{\text{LCP}}) \leq S(X^*)$ was shown. However, the case $x_T^{\text{LCP}} < x_T^*$ during an increasing interval can never occur, since $x_{T+1}^* = 0$ and thus $x_T^* = x_T^L \leq x_T^{\text{LCP}}$.

We still need to analyze the operating cost given by $R(X) \coloneqq \sum_{t=1}^{T} g_t(x_t)$. The following lemma estimates the operating of the LCP algorithm.

Lemma 2.7 [AQ18a, Lemmas 3.11 and 3.12]**.** *It holds that*

$$R(X^{LCP}) \leq R(X^*) + 2 \cdot S(X^*).$$

Proof idea. Let

$$\hat{C}_\tau^Y(x) \coloneqq \min_{x_1,\dots,x_{\tau-1}} C_\tau^Y((x_1,\dots,x_{\tau-1},x))$$

with $Y \in \{L, U\}$ be the minimal cost up to time τ achievable for a schedule that ends in the state x at time τ. The functions $\hat{C}_\tau^Y(x)$ are convex [AQ18a, Lemma 3.3]. Based on this property, it can be shown that during increasing intervals,

$$\hat{C}_\tau^L(x_\tau^{\text{LCP}}) + g_{\tau+1}(x_{\tau+1}^{\text{LCP}}) \leq \hat{C}_{\tau+1}^L(x_{\tau+1}^{\text{LCP}}) \tag{2.4}$$

holds. Analogously, for decreasing intervals, we have

$$\hat{C}_\tau^U(x_\tau^{\text{LCP}}) + g_{\tau+1}(x_{\tau+1}^{\text{LCP}}) \leq \hat{C}_{\tau+1}^U(x_{\tau+1}^{\text{LCP}}). \tag{2.5}$$

We add the inequalities given by (2.4) or (2.5) for all time slots t of an interval I. By using the fact that both X^{LCP} and X^* are in the same state at the beginning and the end of an interval and by transforming the result appropriately, we get

$$\sum_{t\in I} g_{t+1}(x_{t+1}^{\text{LCP}}) \leq \sum_{t\in I} g_{t+1}(x_{t+1}^*) + \sum_{t\in I} \beta|x_{t+1}^* - x_t^*|.$$

We add the operating cost of all time intervals and finally get

$$R(X^{\text{LCP}}) \leq R(X^*) + \sum_{t=1}^{T} \beta|x_{t+1}^* - x_t^*|.$$

The term $\beta|x_{t+1}^* - x_t^*|$ describes the switching cost when both powering up and powering down cause a cost of β. Since all servers are powered down at the end of the workload, $\sum_{t=1}^{T} \beta|x_{t+1}^* - x_t^*| = 2 \cdot S(X^*)$. □

Now, we are able to prove Theorem 2.2.

Proof of Theorem 2.2. By using Lemmas 2.6 and 2.7, we get

$$C(X^{\mathrm{LCP}}) = R(X^{\mathrm{LCP}}) + S(X^{\mathrm{LCP}}) \leq R(X^*) + 3 \cdot S(X^*) \leq 3 \cdot C(X^*). \qquad \square$$

Interestingly, the LCP algorithm can achieve a much better performance if the operating cost of the optimal schedule is large in comparison to its switching cost. For example, for a problem instance where both $R(X^*)$ and $S(X^*)$ are equal, LCP achieves a competitive ratio of 2. If the ratio $S(X^*)/R(X^*)$ approaches zero (i.e., the operating cost is much larger than the switching cost), then the LCP algorithm calculates an almost optimal schedule.

2.3 Randomized online algorithm

In the previous section, we saw that the LCP algorithm is 3-competitive in the discrete setting. The competitive ratio can be improved by using randomization. Bansal et al. [BGK+15] developed a 2-competitive deterministic online algorithm for the right-sizing problem of homogeneous data centers in the *fractional* setting. We use this algorithm to obtain a 2-competitive *fractional* schedule for a given integral problem instance. By rounding the number of active servers properly, our algorithm RandomizedRounding creates an integral schedule while still achieving a competitive ratio of 2 against an oblivious adversary.

We will see that our competitiveness proof only uses the 2-competitiveness of the underlying algorithm. Therefore, any 2-competitive algorithm for the fractional setting can be used, for example, the algorithm *Randomly Biased Greedy* (RBG) by Andrew et al. [ABL+13] with $\gamma = 1$.

Formally, our algorithm works as follows. Let $\bar{\mathcal{I}}$ be the continuous extension of the original problem instance as defined in Section 2.1 and let $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_T)$ be a 2-competitive online schedule for $\bar{\mathcal{I}}$. In the following, we describe how RandomizedRounding converts this solution into an integral schedule $X^{\mathcal{R}} = (x_1^{\mathcal{R}}, \ldots, x_T^{\mathcal{R}})$.

First, we define a modified version of the ceiling operator. Let $\lceil x \rceil^* \coloneqq \min\{n \in \mathbb{Z} \mid n > x\}$ be the smallest integer that is *greater than* x, so for an integer $k \in \mathbb{Z}$ we have $\lceil k \rceil^* = k + 1$ and for a non-integral number $x \in \mathbb{R} \setminus \mathbb{Z}$, we have $\lceil x \rceil^* = \lceil x \rceil$. This definition ensures that the identity $\lceil x \rceil^* = \lfloor x \rfloor + 1$ holds for all numbers $x \in \mathbb{R}$. Let $\mathrm{frac}(x) \coloneqq x - \lfloor x \rfloor$ be the fractional part of x.

The pseudo code below describes how the states $x_t^{\mathcal{R}}$ are determined. After calculating $\bar{x}_t$ in line 2, the algorithm checks whether the number of active servers in the fractional schedule $\bar{X}$ increases (line 4) or decreases (line 10). In the first case, we have $\bar{x}_{t-1} \leq \bar{x}_t$. If $x_{t-1}^{\mathcal{R}}$ is already in the upper state $\lceil \bar{x}_t \rceil^*$, we do not change the number of active servers, so $x_t^{\mathcal{R}} \coloneqq \lceil \bar{x}_t \rceil^*$. Otherwise, $\bar{x}_t$ is rounded up with probability $p_t^{\uparrow} \coloneqq \frac{\bar{x}_t - x'_{t-1}}{1 - \mathrm{frac}(x'_{t-1})}$ where $x'_{t-1} \coloneqq [\bar{x}_{t-1}]_{\lfloor \bar{x}_t \rfloor}^{\lceil \bar{x}_t \rceil^*}$ is the projection of the last fractional state $\bar{x}_{t-1}$ into the unit size

Algorithm 2 RandomizedRounding

Input: $\mathcal{I} = (T, m, \beta, \mathcal{G})$

1: **for** $t \coloneqq 1$ **to** T **do**
2: Determine $\bar{x}_t$ with a 2-competitive algorithm for the fractional setting
3: $x'_{t-1} \coloneqq [\bar{x}_{t-1}]_{\lfloor \bar{x}_t \rfloor}^{\lceil \bar{x}_t \rceil^*}$
4: **if** $\bar{x}_{t-1} \leq \bar{x}_t$ **then**
5: **if** $x^{\mathcal{R}}_{t-1} = \lceil \bar{x}_t \rceil^*$ **then**
6: $x^{\mathcal{R}}_t \coloneqq \lceil \bar{x}_t \rceil^*$
7: **else**
8: $p^{\uparrow}_t \coloneqq \frac{\bar{x}_t - x'_{t-1}}{1 - \mathrm{frac}(x'_{t-1})}$
9: $x^{\mathcal{R}}_t \coloneqq \begin{cases} \lceil \bar{x}_t \rceil^* & \text{with probability } p^{\uparrow}_t \\ \lfloor \bar{x}_t \rfloor & \text{with probability } 1 - p^{\uparrow}_t \end{cases}$
10: **else**
11: **if** $x^{\mathcal{R}}_{t-1} = \lfloor \bar{x}_t \rfloor$ **then**
12: $x^{\mathcal{R}}_t \coloneqq \lfloor \bar{x}_t \rfloor$
13: **else**
14: $p^{\downarrow} \coloneqq \frac{x'_{t-1} - \bar{x}_t}{\mathrm{frac}(x'_{t-1})}$
15: $x^{\mathcal{R}}_t \coloneqq \begin{cases} \lfloor \bar{x}_t \rfloor & \text{with probability } p^{\downarrow}_t \\ \lceil \bar{x}_t \rceil^* & \text{with probability } 1 - p^{\downarrow}_t \end{cases}$

interval $[\lfloor \bar{x}_t \rfloor, \lceil \bar{x}_t \rceil^*]$ defined by the current state $\bar{x}_t$. The second case, i.e., $\bar{x}_{t-1} > \bar{x}_t$, is symmetrical (see lines 11–15).

Theorem 2.3 [AQ18b, Theorem 3]**.** *The* RandomizedRounding *algorithm is 2-competitive against an oblivious adversary.*

Before we prove this theorem, we show an important relationship between the rounded schedule $X^{\mathcal{R}}$ and the fractional schedule $\bar{X}$.

Lemma 2.8 [AQ18b, Lemma 18]**.** *The probability that the* RandomizedRounding *algorithm uses the upper state $\lceil \bar{x}_t \rceil^*$ at time t is equal to the fractional part of $\bar{x}_t$. Formally,* $\Pr[x^{\mathcal{R}}_t = \lceil \bar{x}_t \rceil^*] = \mathit{frac}(\bar{x}_t)$ *holds for all $t \in [T]$.*

Proof idea. The lemma can be proven by induction over t. In the induction step, we differentiate whether or not $\bar{x}_{t-1} \leq \bar{x}_t$ holds (line 4). Here, we only consider $\bar{x}_{t-1} \leq \bar{x}_t$, since both cases are symmetrical. By the law of total probability, $\Pr[x^{\mathcal{R}}_t = \lceil \bar{x}_t \rceil^*]$ can be written as

$$\begin{aligned} \Pr[x^{\mathcal{R}}_t = \lceil \bar{x}_t \rceil^*] = \Pr[x^{\mathcal{R}}_t = \lceil \bar{x}_t \rceil^* \mid x^{\mathcal{R}}_{t-1} = \lceil \bar{x}_t \rceil^*] \cdot \Pr[x^{\mathcal{R}}_{t-1} = \lceil \bar{x}_t \rceil^*] \\ + \Pr[x^{\mathcal{R}}_t = \lceil \bar{x}_t \rceil^* \mid x^{\mathcal{R}}_{t-1} \leq \lfloor \bar{x}_t \rfloor] \cdot \Pr[x^{\mathcal{R}}_{t-1} \leq \lfloor \bar{x}_t \rfloor]. \end{aligned}$$

The probabilities $\Pr[x^{\mathcal{R}}_{t-1} = \lceil \bar{x}_t \rceil^*]$ and $\Pr[x^{\mathcal{R}}_{t-1} \leq \lfloor \bar{x}_t \rfloor]$ are given by our induction hypothesis. The conditional probabilities are defined by our algorithm (line 6 and 8–9,

respectively). Inserting the definition of $p_t^{\uparrow}$ and simplifying the resulting expression gives us the equation $\Pr[x_t^{\mathcal{R}} = \lceil \bar{x}_t \rceil^*] = \text{frac}(\bar{x}_t)$. □

Now, we are able to prove Theorem 2.3.

Proof idea of Theorem 2.3. Let $C^{\mathcal{J}}(X)$ be the cost of the schedule X regarding the problem instance $\mathcal{J} \in \{\mathcal{I}, \bar{\mathcal{I}}\}$. We have to show that the expected cost of RandomizedRounding is smaller than two times the cost of the optimal offline schedule, i.e.,

$$\mathbb{E}[C^{\mathcal{I}}(X^{\mathcal{R}})] \leq 2 \cdot C^{\mathcal{I}}(X^*).$$

First, we will show that the expected cost of RandomizedRounding equals the cost of the fractional online schedule $\bar{X}$. The definition of the continuous operating cost functions $\bar{g}_t$ (see equation (2.1)) in combination with Lemma 2.8 implies that the expected operating cost of $X^{\mathcal{R}}$ equals the operating cost of $\bar{X}$ regarding the fractional problem instance $\bar{\mathcal{I}}$ [AQ18b, Lemma 19]. Formally, $\mathbb{E}[R^{\mathcal{I}}(X^{\mathcal{R}})] = R^{\bar{\mathcal{I}}}(\bar{X})$ holds where $R^{\mathcal{J}}(X)$ denotes the operating cost of X regarding the problem instance $\mathcal{J}$.

For the analysis of the switching cost, we distinguish between the cases (1) $\bar{x}_{t-1} < \lfloor \bar{x}_t \rfloor$ and (2) $\bar{x}_{t-1} \in [\lfloor \bar{x}_t \rfloor, \bar{x}_t]$. Note that $\bar{x}_{t-1} > \bar{x}_t$ implies that no servers are powered up in $\bar{X}$ and $X^{\mathcal{R}}$, so we do not need to consider this case as there is no switching cost. By applying Lemma 2.8, it can be shown that in both cases $\mathbb{E}[\beta(x_t^{\mathcal{R}} - x_{t-1}^{\mathcal{R}})^+] = \beta(\bar{x}_t - \bar{x}_{t-1})^+$ holds [AQ18b, Lemma 20]. By summing over all time slots, we get $\mathbb{E}[S^{\mathcal{I}}(X^{\mathcal{R}})] = S^{\bar{\mathcal{I}}}(\bar{X})$ where $S^{\mathcal{J}}(X)$ denotes the switching cost of X regarding the problem instance $\mathcal{J}$. Therefore, $\mathbb{E}[C^{\mathcal{I}}(X^{\mathcal{R}})] = C^{\bar{\mathcal{I}}}(\bar{X})$ holds.

Let $\bar{X}^*$ be an optimal offline solution for the continuous problem instance $\bar{\mathcal{I}}$. The fractional schedule $\bar{X}$ is 2-competitive regarding $\bar{\mathcal{I}}$, so we have $C^{\bar{\mathcal{I}}}(\bar{X}) \leq 2 \cdot C^{\bar{\mathcal{I}}}(\bar{X}^*)$. By Lemma 2.1, an optimal fractional schedule can be rounded to an integral schedule without increasing its cost. Therefore, we have $C^{\bar{\mathcal{I}}}(\bar{X}^*) = C^{\mathcal{I}}(X^*)$. All in all, we get

$$\mathbb{E}[C^{\mathcal{I}}(X^{\mathcal{R}})] = C^{\bar{\mathcal{I}}}(\bar{X}) \leq 2 \cdot C^{\bar{\mathcal{I}}}(\bar{X}^*) = 2 \cdot C^{\mathcal{I}}(X^*).$$ □

2.4 Lower bounds for online algorithms

In this section, we present several lower bounds for deterministic and randomized online algorithms. First, we will prove that no deterministic online algorithm can achieve a competitive ratio smaller than 3 in the discrete setting (Section 2.4.1). Afterwards, we switch to the fractional setting and show that the smallest achievable competitive ratio is 2 (see Section 2.4.2). Based on this result, we show in Section 2.4.3 that no randomized online algorithm for the discrete setting is better than 2-competitive.

Each lower bound is first shown for the *general* model where the operating costs are given by arbitrary non-negative convex functions g_t. Then we switch to the *restricted* model and show how f and λ_t have to be chosen such that the particular lower bound still holds. Recall that the operating cost of the restricted model at time t is given by $g_t(x_t) \coloneqq x_t \cdot f(\lambda_t / x_t)$. The inequality $x_t \geq \lambda_t$ must be satisfied for all time slots $t \in [T]$.

Finally, in Section 2.4.4, we prove that all lower bounds still hold if the online algorithm has a prediction window whose length does not depend on the problem instance.

To simplify the analysis, we pay the switching cost for both powering up and powering down. Remember that this is not a restriction, since at the beginning and at the end of the time horizon all servers are powered down. We set $\beta = 2$, so changing the state of a server causes a cost of $\beta/2 = 1$. Hence, the total cost of a schedule X is given by

$$C(X) \coloneqq \sum_{t=1}^{T} g_t(x_t) + \sum_{t=1}^{T+1} |x_t - x_{t-1}|$$

with $x_0 \coloneqq x_{T+1} \coloneqq 0$.

2.4.1 Discrete setting, deterministic algorithms

Theorem 2.4 [AQ18a, Theorem 4.1]**.** *There is no* deterministic *online algorithm that achieves a competitive ratio of $c < 3$ for the* general *model of the homogeneous data center right-sizing problem in the* discrete *setting.*

Proof idea. We consider a data center with only one server, so $m = 1$. Let $\mathcal{A}$ be an arbitrary deterministic online algorithm generating the schedule $X^{\mathcal{A}}$. The adversary uses the functions $\varphi_0(x) = \epsilon|x|$ and $\varphi_1(x) = \epsilon|1 - x|$ with $\epsilon \to 0$. If $\mathcal{A}$ has an active server, the adversary sends φ_0 as operating cost function. Otherwise, φ_1 is used, so $\mathcal{A}$ always has to pay the operating cost ϵ except for the time slots where it changes its state. The total cost of $\mathcal{A}$ depends on the length T of the problem instance and is given by $C(X^{\mathcal{A}}) = (T - S)\epsilon + S$ where S is the switching cost of $\mathcal{A}$.

We use two strategies to find an upper bound for the cost of the optimal schedule X^*. In the first strategy, we remain in one state for the whole time horizon. If the adversary sends φ_0 more often than φ_1, this is state $x = 0$, otherwise it is state $x = 1$. The operating cost is at most $T\epsilon/2$ and the switching is at most 2 (for powering up and down one server at the beginning and the end of the workload if necessary). The second strategy avoids any operating cost by always choosing the state without any cost. However, it causes a switching cost of at most $S + 2$. Therefore, the total cost of an optimal schedule is at most

$$C(X^*) \leq \min\{T\epsilon/2 + 2, S + 2\}.$$

By distinguishing between the cases $S \geq T\epsilon/2$ and $S < T\epsilon/2$, it can be shown that the competitive ratio is

$$\frac{C(X^{\mathcal{A}})}{C(X^*)} \geq 3 - \epsilon - \frac{2(1-\epsilon) + 4}{T\epsilon/2 + 2}.$$

We set the length of our problem instance to $T \geq 1/\epsilon^2$. If ϵ goes to zero, the competitive ratio converges to 3. Therefore, no algorithm can achieve a competitive ratio smaller than 3. □

Theorem 2.4 still holds in the restricted model.

Theorem 2.5 [AQ18a, Theorem 4.2]**.** *There is no* deterministic *online algorithm that achieves a competitive ratio of* $c < 3$ *for the* restricted *model of the homogeneous data center right-sizing problem in the* discrete *setting.*

Proof idea. We adapt the proof of Theorem 2.4, but use $m = 2$ instead of one server. The basic idea is to force the online algorithm to switch between the states 1 and 2 instead of 0 and 1. To achieve this, the adversary uses the job volumes $v_0 \coloneqq 0.5$ and $v_1 \coloneqq 1$ instead of φ_0 and φ_1. In combination with the fixed operating cost function $f(z) \coloneqq \epsilon|1 - 2z|$, this results in the same operating cost as in the proof of Theorem 2.4. Formally, $x_t f(v_k/x_t) = \varphi_k(x_t - 1)$ for all $x_t \in \{1, 2\}$ and $k \in \{0, 1\}$. Note that it is not allowed to use $x_t = 0$ for $t \in [T]$, because in the restricted model $x_t \geq \lambda_t \geq 0.5$ must always be fulfilled. The additional switching cost of 1 at the beginning and the end of the workload does not influence the competitive ratio if we set $T \geq 1/\epsilon^2$ and $\epsilon \to 0$. □

2.4.2 Fractional setting

Before we analyze the lower bound of randomized online algorithms in the discrete setting, we switch to the fractional setting where the number of active servers is not required to be integral. In fact, we need the results of the fractional setting to derive a lower bound for randomized online algorithms in the discrete setting. Bansal et al. [BGK+15] did not only develop a 2-competitive online algorithm for the fractional setting, but also show a lower bound of 1.86. Although they conjectured that a better online algorithm exists, we found a lower bound of 2, so Bansal et al.'s algorithm is actually optimal. This result was independently found by [AS17]. In addition to [AS17], we will extend our lower bound to the restricted model.

Theorem 2.6 [AQ18b, Theorem 6]**.** *There is no online algorithm that achieves a competitive ratio of* $c < 2$ *for the* general *model of the homogeneous data center right-sizing problem in the* fractional *setting.*

Bansal et al. [BGK+15] proved that in the fractional setting, a randomized online algorithm can be derandomized without increasing the competitive ratio. Therefore, it is sufficient to determine a lower bound for deterministic algorithms.

Proof idea. We use a data center with $m = 1$ server. As in the proof of Theorem 2.4, the adversary uses the functions $\varphi_0(x) \coloneqq \epsilon|x|$ and $\varphi_1(x) \coloneqq \epsilon|1 - x|$. The theorem is proven in two steps. First, we define an algorithm $\mathcal{B}$ whose competitive ratio is greater than $2 - \delta$ for any $\delta > 0$. Then, we show that any online algorithm $\mathcal{A}$ that differs from $\mathcal{B}$ has a greater competitive ratio than $\mathcal{B}$.

Algorithm $\mathcal{B}$ moves in small steps of size $\epsilon/2$ towards 0 or 1 if the adversary sends φ_0 or φ_1, respectively. It can be shown that this strategy results in a competitive ratio of $2 - \epsilon$. In fact, for any input sequence that only contains the functions φ_0 and φ_1, algorithm $\mathcal{B}$ behaves exactly like the 2-competitive algorithm of Bansal et al. [BGK+15].

Let $\mathcal{A}$ be an arbitrary deterministic online algorithm and let a_t and b_t denote the number of active servers of algorithm $\mathcal{A}$ and $\mathcal{B}$, respectively. If the state of algorithm $\mathcal{A}$ is smaller than that of $\mathcal{B}$, i.e., $a_{t-1} < b_{t-1}$, then the adversary sends $g_t = \varphi_1$. If $a_{t-1} > b_{t-1}$, the adversary chooses φ_0.

Assume that the adversary sends $g_t = \varphi_1$, but $\mathcal{A}$ stays below $\mathcal{B}$, i.e., $a_t < b_t$. Then, $\mathcal{A}$ has to pay a greater operating cost, but reduces the switching cost in comparison to $\mathcal{B}$. However, the adversary continues sending φ_1, so at some point, $\mathcal{A}$ has to reach (or exceed) the state of $\mathcal{B}$ to avoid infinite operating costs. However, this nullifies the switching cost savings and the greater operating cost of $\mathcal{A}$ results in a higher total cost.

On the other hand, if the adversary sends φ_1 and $\mathcal{A}$ tries to minimize the operating cost by using a state greater than the state of $\mathcal{B}$, i.e., $a_t > b_t$, then the adversary will use φ_0 for the next time slot causing an additional switching cost for $\mathcal{A}$ in comparison to $\mathcal{B}$. This switching cost is greater than the saved operating cost, so again, the total cost of $\mathcal{A}$ is greater than the total cost of $\mathcal{B}$.

For $g_t = \varphi_0$, the argumentation is analogous. All in all, we get

$$C(\mathcal{A}) \geq C(\mathcal{B}) \geq (2 - \epsilon) \cdot C(X^*)$$

for any $\epsilon > 0$, so there is no deterministic online algorithm that achieves a competitive ratio of $c < 2$. □

Similar to the previous section, Theorem 2.6 still holds in the restricted model.

Theorem 2.7 [AQ18b, Theorem 7]**.** *There is no online algorithm that achieves a competitive ratio of $c < 2$ for the* restricted *model of the homogeneous data center right-sizing problem in the* fractional *setting.*

Proof idea. We choose $f(z) \coloneqq \epsilon|1 - kz|$ with $k \to \infty$ and $\epsilon \to 0$. Instead of φ_0 and φ_1, the adversary uses the job volumes $v_0 = 0$ and $v_1 = 1/k$ leading to the same operating cost, i.e., $x_t f(v_k/x_t) = \varphi_k(x_t)$ for $k \in \{0, 1\}$. Therefore, the lower bound remains the same. □

2.4.3 Discrete setting, randomized algorithms

Based on the lower bound for the fractional setting, we are now able to prove lower bounds for randomized online algorithms in the *discrete* setting.

Theorem 2.8 [AQ18b, Theorem 8]**.** *There is no* randomized *online algorithm that achieves a competitive ratio of $c < 2$ against an oblivious adversary for the* general *model of the homogeneous data center right-sizing problem in the* discrete *setting.*

Proof idea. Let $\mathcal{A}$ be an arbitrary randomized online algorithm and let $X^{\mathcal{A}}$ be the generated schedule. Let $\bar{x}_t^{\mathcal{A}}$ be the probability that $\mathcal{A}$ is in state 1 at time slot t. It can be shown that the expected cost of $X^{\mathcal{A}}$ is at least as large as the cost of the fractional schedule $\bar{X}^{\mathcal{A}} \coloneqq (\bar{x}_1^{\mathcal{A}}, \ldots, \bar{x}_T^{\mathcal{A}})$ regarding the continuous extension of the original problem

instance [AQ18b, Lemma 24]. For this, the expected operating and switching costs of $X^{\mathcal{A}}$ are analyzed separately. Formally, we get

$$\mathbb{E}[C^{\mathcal{I}}(X^{\mathcal{A}})] \geq C^{\bar{\mathcal{I}}}(\bar{X}^{\mathcal{A}}).$$

By Theorem 2.6, we know that a deterministic online algorithm that generates the fractional schedule $\bar{X}^{\mathcal{A}}$ cannot achieve a competitive ratio smaller than 2, i.e., $C^{\bar{\mathcal{I}}}(\bar{X}^{\mathcal{A}}) \geq 2 \cdot C^{\bar{\mathcal{I}}}(\bar{X}^*)$. Furthermore, Lemma 2.1 states that the optimal schedules of $\mathcal{I}$ and $\bar{\mathcal{I}}$ have the same cost, so $C^{\bar{\mathcal{I}}}(\bar{X}^*) = C^{\mathcal{I}}(X^*)$. Altogether, we get

$$\mathbb{E}[C^{\mathcal{I}}(X^{\mathcal{A}})] \geq C^{\bar{\mathcal{I}}}(\bar{X}^{\mathcal{A}}) \geq 2 \cdot C^{\bar{\mathcal{I}}}(\bar{X}^*) \geq 2 \cdot C^{\mathcal{I}}(X^*),$$

i.e., the competitive ratio of $\mathcal{A}$ is at least 2. □

Theorem 2.9 [AQ18b, Theorem 9]**.** *There is no* randomized *online algorithm that achieves a competitive ratio of $c < 2$ against an oblivious adversary for the* restricted *model of the homogeneous data center right-sizing problem in the* discrete *setting.*

Proof idea. Similar to the proof of Theorem 2.5, we set $m = 2$, $f(z) \coloneqq \epsilon|1-2z|$, $v_0 \coloneqq 0.5$ and $v_1 \coloneqq 1$ such that $x_t f(v_k/x_t) = \varphi_k(x_t - 1)$ for $x_t \in \{1, 2\}$ and $k \in \{0, 1\}$. □

2.4.4 Online algorithms with prediction window

So far, we always assume that the online algorithm only knows the current operating cost function g_t. An online algorithm with a prediction window of length w can access the functions $g_t, \ldots, g_{t+w}$ to choose the next state x_t. We assume that the length of the prediction window is constant, i.e., it is independent of the problem instance. The following theorem shows that a prediction window with a constant length does not improve the competitive ratio of the respective algorithm.

Theorem 2.10 [AQ18a, Theorem 4.5], [AQ18b, Theorem 10]**.** *Let $w \in \mathbb{N}$. There is no* deterministic *online algorithm with a prediction window of length w that achieves a competitive ratio of $c < 3$ in the discrete setting or $c < 2$ in the fractional setting. There is no* randomized *online algorithm with a prediction window of length w achieving a competitive ratio of $c < 2$ against an oblivious adversary.*

Proof idea. The basic idea is to replace each function g_t with a sequence of $n \cdot w$ equal functions $g'_{t,i}(x) \coloneqq \frac{1}{nw} g_t(x)$ with $n \in \mathbb{N}$ and $i \in [nw]$. By choosing n sufficiently large, the advantage of the prediction window gets arbitrarily small, so the lower bounds proven in the previous sections still hold. □

3 Heterogeneous data centers

Modern data centers often consist of different server types. This can be different architectures, like servers that use the GPU instead of the CPU for computing. Heterogeneity also appears if a data center is extended by new servers while the old ones are kept running. In this chapter, we study the right-sizing problem of a heterogeneous data center with d different server types in the offline and online version. We consider the problem formulation presented in Section 1.1, but restrict the operating cost functions to gain results with more practical relevance. In fact, arbitrary convex operating cost functions without further restrictions lead to an exponential lower bound as we will see in Section 3.2.5. Therefore, we resort to the model introduced by Lin et al. [LWAT11a] and generalize it for heterogeneous data centers. In this model, the operating cost of a single server is modeled by a non-negative increasing convex function of its load. Furthermore, for each time slot, a job volume λ_t arrives that can be distributed arbitrarily to the active servers. We generalize the model by defining an individual operating cost function for each server type. Additionally, we have to consider how the arriving job volume is distributed to the server types.

The operating cost of a single server of type j that runs with load $z \in [0, z_j^{\max}]$ at time t is modeled by a non-negative increasing convex function $f_{t,j}(z)$. The variables $z_j^{\max}$ model the computational power of a single server of type j. More precisely, $z_j^{\max}$ is the maximal job volume that a server of type j can process per time slot. Hence, we define $f_{t,j}(z) \coloneqq \infty$ for $z > z_j^{\max}$. Let $q_{t,j}$ be the job fraction assigned to all servers of type j at time t. Since $f_{t,j}$ is convex, it is optimal to distribute the job fraction equally to the active servers [AQ21e, Lemma 2.2]. Formally,

$$g_{t,j}(x_{t,j}, q_{t,j}) \coloneqq \begin{cases} x_{t,j} f_{t,j}\left(\frac{\lambda_t q_{t,j}}{x_{t,j}}\right) & \text{if } x_{t,j} > 0 \\ \infty & \text{if } x_{t,j} = 0 \text{ and } \lambda_t q_{t,j} > 0 \\ 0 & \text{if } x_{t,j} = 0 \text{ and } \lambda_t q_{t,j} = 0 \end{cases} \tag{3.1}$$

is the operating cost of server type j during time slot t. Remember that $x_{t,j}$ is the number of active servers of type j at time t. The quotient $\frac{\lambda_t q_{t,j}}{x_{t,j}}$ is the job volume assigned to a single server of type j. We apply the operating cost function to get the operating cost of a single server. Multiplying with the number of active servers gives us the total operating cost of server type j at time t. The second case of equation (3.1) (i.e., $x_{t,j} = 0$ and $\lambda_t q_{t,j} > 0$) describes the situation where a job volume is assigned to server type j, however, all servers of type j are inactive, so the job volume cannot be processed and we set the operating cost to infinity. In the last case (i.e., $x_{t,j} = 0$ and $\lambda_t q_{t,j} = 0$), there are no jobs and no active servers, so there is no operating cost.

Let $\mathcal{Q} \coloneqq \{(q_1, \dots, q_d) \in [0,1]^d \mid \sum_{j=1}^{d} q_j = 1\}$ be the set of all possible job assignments. The total operating cost during time slot t is given by

$$g_t(x_1, \dots, x_d) \coloneqq \min_{(q_1,\dots,q_d)\in\mathcal{Q}} \sum_{j=1}^{d} g_{t,j}(x_j, q_j). \tag{3.2}$$

Remember that we already defined the total cost of a schedule $X = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$ in equation (1.1) (see Section 1.1). A problem instance is defined by the tuple $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$ with $\boldsymbol{m} = (m_1, \dots, m_d)$, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_d)$, $F = (f_{1,1}, \dots, f_{T,d})$ and $\Lambda = (\lambda_1, \dots, \lambda_T)$. Note that $z_j^{\max}$ is implicitly defined by the operating cost functions.

Finding the optimal job assignment, i.e., calculating the value of g_t for a given server configuration $\boldsymbol{x}$ is a convex optimization problem. In general, such problems are NP-hard [DKP02], however, in our case it is solvable in polynomial time, since the objective function $\sum_{j=1}^{d} g_{t,j}(x_j, q_j)$ is additively separable [Chu16]. A function $H : \mathbb{R}^d \to \mathbb{R}$ is called additively separable if there exists $h_1, \dots, h_d : \mathbb{R} \to \mathbb{R}$ such that $H(x_1, \dots, x_d) = \sum_{j=1}^{d} h_j(x_j)$ which is true in our case. In this thesis, we do not deepen the problem of calculating g_t, we just assume that there is an oracle that calculates g_t for a given server configuration $\boldsymbol{x}$ in $\mathcal{O}(1)$ time.

In this chapter, we first consider the offline problem and present an optimal offline algorithm that generalizes the graph-based approach of Section 2.1 to an arbitrary number of server types. However, even for a constant d, the runtime of this algorithm is only pseudo-polynomial. By using a polynomial-sized subset of the vertices, we obtain a $(1+\epsilon)$-approximation algorithm.

Section 3.2 investigates the online problem. First, we examine the special case where all servers have the same computational power and constant operating costs, i.e., the operating cost does neither depend on time nor on the load of the server. We present a $2d$-competitive deterministic and a $1.582d$-competitive randomized online algorithm for this setting. Then we consider load-dependent, but time-independent operating costs and develop a $(2d+1)$-competitive deterministic algorithm. For time- and load-dependent operating costs, we present a deterministic online algorithm that achieves a competitive ratio of $2d + 1 + \epsilon$ for any $\epsilon > 0$. Finally, we prove a lower bound of $2d$ for deterministic algorithms that only uses constant operating costs, so it is a lower bound for all three algorithms. Furthermore, we show that handling arbitrary convex operating cost functions g_t (that do not necessarily satisfy equation (3.2)) leads to an at least exponential competitive ratio for deterministic algorithms.

The lower bound of $2d$ as well as the $2d$- and $1.582d$-competitive algorithms for constant operating costs were published in the proceedings of the *12th International Conference on Algorithms and Complexity* (CIAC) [AQ21c] that is shown in Appendix C. Due to space limitations, several proofs are missing. A full version of this paper has recently been published in a special issue of the journal *Theoretical Computer Science* [AQ21a]. A freely accessible preprint of this version can be found on arXiv [AQ21b].

The offline algorithms as well as the online algorithms for load-dependent operating costs were published in the conference proceedings of the *33rd ACM Symposium on Par-*

allelism in Algorithms and Architectures (SPAA) [AQ21e] that is shown in Appendix D. Again, a full version containing all proofs can be found on arXiv [AQ21d].

The notation introduced above slightly differs from the published papers in the appendix. In [AQ21e], both the job fraction $q_{t,j}$ and the maximal load $z_j^{\max}$ are denoted with z (i.e., $z_{t,j}$ and $z_j^{\max}$). To avoid this overload of variables, we denote the job fraction with $q_{t,j}$. The constant operating cost of [AQ21c] will be denoted by r_j instead of l_j, similar to [Alb19].

3.1 Offline problem

The grid-structured graph for homogeneous data centers introduced in Section 2.1 can be generalized for heterogeneous data centers. We use one dimension for each server type, so a problem instance can be represented by a $(d+1)$-dimensional grid (remember that one dimension is used for time). The optimal schedule can be determined by finding a shortest path in the graph. However, it is not possible to generalize the binary search approach for multiple server types as the example in Figure 3.1 shows. Therefore, we focused on developing an approximation algorithm.

As a preparation, we first devise an optimal offline algorithm that calculates a shortest path by using dynamic programming (Section 3.1.1). However, its runtime is not polynomial since the graph contains $\Theta(T \cdot \prod_{j=1}^{d} m_j)$ vertices. By using only a polynomial-sized subset of the vertices, we gain a $(1+\epsilon)$-approximation algorithm that runs in polynomial time if d is a constant (Section 3.1.2). Before we start explaining our algorithms, we give an overview on previous results in the literature regarding specialized or related problems.

It is an open question whether or not the problem is NP-hard. If the operating cost is load- and time-independent and if each server type has the same computational power, i.e., $f_{t,j}(z) = r_j = \text{const}$ for $z \in [0,1]$ and $f_{t,j}(z) = \infty$ for $z > 1$, then there is a polynomial-time algorithm based on a minimum-cost flow computation [Alb19, Theorem 3.1]. Actually, the algorithm presented in [Alb19] was designed for m unique servers. Note that the total number of servers $m = \sum_{j=1}^{d} m_j$ is not polynomial in the encoding length of the problem instance that is given by $\mathcal{O}(T + \sum_{j=1}^{d} \log m_j)$. Hence, using the algorithm directly leads to an exponential runtime. The graph consists of one component for each server representing its state (active or inactive). The construction can be extended by increasing the capacity of the edges in each component from 1 to the number m_j of available servers of the particular server type so that each component represents one server type. The proofs presented in [Alb19] are not affected by this change, so we gain a polynomial runtime.

It is even possible to determine a minimum-cost flow where the edge weights are modeled by a convex function of the flow passing through the edge [Vég16]. One may think that this approach can be used to generalize the algorithm by [Alb19] for load-dependent operating costs. However, it is not sufficient to replace each edge weight by a convex function, because the operating cost of a server type depends on the job

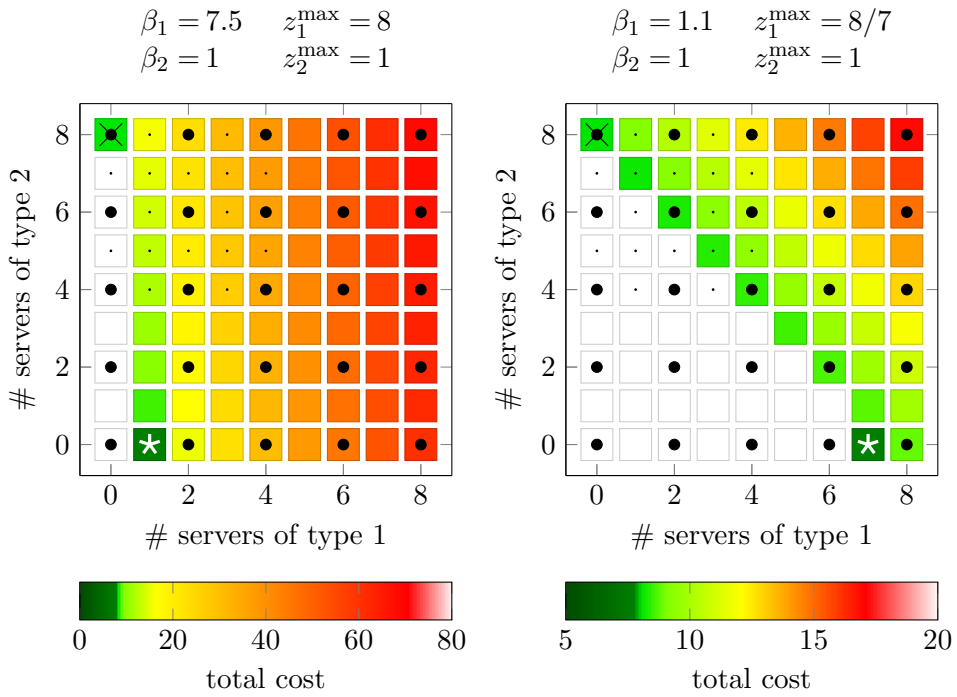

Figure 3.1: Demonstration why GraphBasedBinarySearch (GBBS) cannot be generalized for two or more server types. Both plots show one problem instance each with $d = 2$, $(m_1, m_2) = (8, 8)$, $T = 1$, $\lambda_1 = 8$ and $f_{t,j}(z) = 0$ for $z \in [0, z_j^{\max}]$ (and $f_{t,j}(z) = \infty$ for $z > z_j^{\max}$). In the left plot, we have $(\beta_1, \beta_2) = (7.5, 1)$ and $(z_1^{\max}, z_2^{\max}) = (8, 1)$. The colors represent the cost of the corresponding schedule as shown in the color bar below the plot. A white color indicates an infeasible server configuration. The cheapest schedule is $\boldsymbol{x}_1^* = (x_{1,1}^*, x_{1,2}^*) = (1, 0)$ (highlighted with a white star) causing a total cost of $C(X^*) = 7.5$. However, in the first iteration, the algorithm only checks a 5×5 grid indicated by the large black circles, so it will find $\boldsymbol{x}_1 = (0, 8)$ with a cost of 8 (marked with a large cross). A natural generalization of GBBS would be to assume that the optimal schedule is inside $[0 : 4] \times [4 : 8]$ (marked with small dots). However, this is not the case, so GBBS does not find the optimal schedule. The right plot presents another problem instance. Here, we have $\boldsymbol{\beta} = (1.1, 1)$ and $\boldsymbol{z}^{\max} = (8/7, 1)$, so $\boldsymbol{x}_1^* = (7, 0)$ is optimal, although GBBS again finds $\boldsymbol{x}_1 = (0, 8)$ in its first iteration.

fraction $q_{t,j}$ assigned to this server type. This job fraction depends on the number of active servers of all types. Therefore, the problem cannot be solved by a minimum-cost flow computation any more.

3.1.1 Optimal offline algorithm

As already mentioned, we use a $(d+1)$-dimensional grid-structured graph to represent a given problem instance $\mathcal{I}$. There is one dimension for each server type and one dimension for time. To avoid a quadratic number of edges between two time slots as in the construction presented in Section 2.1, we use two vertices for each time slot, one for the beginning and one for the end of a time slot. Between these vertex pairs, we have an edge weighted with the corresponding operating cost. The vertices at the beginning of a time slot are interconnected with edges that allows us to power up servers. Similarly, at the end of a time slot there are edges for powering down.

Formally, a problem instance $\mathcal{I}$ can be represented by a directed acyclic graph $G(\mathcal{I})$. For each time slot $t \in [T]$ and each server configuration $\boldsymbol{x} \in \mathcal{X} \coloneqq \bigtimes_{j=1}^{d} [m_j]_0$, there are the vertices $v^{\uparrow}_{t,\boldsymbol{x}}$ and $v^{\downarrow}_{t,\boldsymbol{x}}$. Let $\boldsymbol{e}_j = (\delta_{j=1}, \dots, \delta_{j=d})$ (with $\delta_{j=i} = 1$ if $i = j$ and $\delta_{j=i} = 0$ otherwise) be the standard unit vectors. There is an edge from $v^{\uparrow}_{t,\boldsymbol{x}}$ to $v^{\uparrow}_{t,\boldsymbol{x}+\boldsymbol{e}_j}$ with weight β_j for all $t \in [T]$, $j \in [d]$ and $\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{e}_j \in \mathcal{X}$ representing the power-up operation of a single server. Similarly, we have an edge from $v^{\downarrow}_{t,\boldsymbol{x}}$ to $v^{\downarrow}_{t,\boldsymbol{x}-\boldsymbol{e}_j}$ with weight 0 for all $t \in [T]$, $j \in [d]$ and $\boldsymbol{x}, \boldsymbol{x}-\boldsymbol{e}_j \in \mathcal{X}$ representing the power-down operations. The operating cost for the server configuration $\boldsymbol{x} \in \mathcal{X}$ at time $t \in [T]$ is modeled by an edge from $v^{\uparrow}_{t,\boldsymbol{x}}$ to $v^{\downarrow}_{t,\boldsymbol{x}}$ with weight $g_t(\boldsymbol{x})$. Finally, there are edges from $v^{\downarrow}_{t,\boldsymbol{x}}$ to $v^{\uparrow}_{t+1,\boldsymbol{x}}$ with weight 0 for all $t \in [T-1]$ and $\boldsymbol{x} \in \mathcal{X}$ to switch to the next time slot. The construction is visualized in Figure 3.2.

Each schedule can be represented by a path from $v^{\uparrow}_{1,\mathbf{0}}$ to $v^{\downarrow}_{T,\mathbf{0}}$ and vice versa. Given the server configurations $\boldsymbol{x}_t$ of a schedule X, the corresponding path passes through the edges from $v^{\uparrow}_{t,\boldsymbol{x}_t}$ to $v^{\downarrow}_{t,\boldsymbol{x}_t}$. By choosing an arbitrary shortest sub path from $v^{\downarrow}_{t,\boldsymbol{x}_t}$ to $v^{\uparrow}_{t+1,\boldsymbol{x}_{t+1}}$ for each $t \in [T-1]$, we obtain a path whose total weight is equal to the total cost of the schedule X.

On the other hand, each path from $v^{\uparrow}_{1,\mathbf{0}}$ to $v^{\downarrow}_{T,\mathbf{0}}$ represents a schedule. If the path passes through the edge from $v^{\uparrow}_{t,\boldsymbol{x}}$ to $v^{\downarrow}_{t,\boldsymbol{x}}$, then the corresponding schedule uses the server configuration $\boldsymbol{x}$ at time t. Note that the total weight of a path can be greater than the total cost of the corresponding schedule, if any sub path from $v^{\downarrow}_{t,\boldsymbol{x}_t}$ to $v^{\uparrow}_{t+1,\boldsymbol{x}_{t+1}}$ does not contain the minimal number of edges. For example, when replacing the edge from $v^{\downarrow}_{1,(1,0)}$ to $v^{\uparrow}_{2,(1,0)}$ by the sub path $(v^{\downarrow}_{1,(1,0)}, v^{\downarrow}_{1,(0,0)}, v^{\uparrow}_{2,(0,0)}, v^{\uparrow}_{2,(1,0)})$, the resulting path still represents the same schedule, however its total weight increases by β_1. Nevertheless, an optimal schedule can be determined by calculating a shortest path.

The algorithm OptimalGraphSearch calculates a shortest path in $G(\mathcal{I})$ by dynamic programming. Its pseudo code is shown below. The variable $d(v)$ stores the distances from the start vertex $v^{\uparrow}_{1,\mathbf{0}}$ to v. Lines 3 and 4 calculate the cost for powering up. For each vertex v of the current time slot, we access the neighbors and update the distance $d(v)$.

The condition $x_j \geq 1$ ensures that we only access valid server configurations, i.e., $\boldsymbol{x} - \boldsymbol{e}_j \in \mathcal{X}$. Line 5 applies the operating cost. In lines 6 and 7, servers can be powered down and line 9 switches to the next time slot (if $t < T$). Afterwards, we know the shortest distance from $v^{\uparrow}_{1,\mathbf{0}}$ to each vertex, so we can reconstruct the shortest path by traversing the graph backwards from $v^{\downarrow}_{T,\mathbf{0}}$ to $v^{\uparrow}_{1,\mathbf{0}}$ using the distance values d. Finally, the algorithm returns the corresponding schedule of the shortest path.

Algorithm 3 OptimalGraphSearch

Input: $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$
Output: Optimal schedule
1: $d(v^{\uparrow}_{1,\mathbf{0}}) \coloneqq 0$ and $d(v^{\uparrow}_{1,\boldsymbol{x}}) \coloneqq \infty$ for all $\boldsymbol{x} \in \mathcal{X} \setminus \{\mathbf{0}\}$
2: **for** $t \coloneqq 1$ **to** T **do**
3: **for** $\boldsymbol{x} \coloneqq \mathbf{0}$ **to** $(m_1, \dots, m_d)$ **do**
4: $d(v^{\uparrow}_{t,\boldsymbol{x}}) \coloneqq \min\left\{d(v^{\uparrow}_{t,\boldsymbol{x}}), \min\{d(v^{\uparrow}_{t,\boldsymbol{x}-\boldsymbol{e}_j}) + \beta_j \mid j \in [d], x_j \geq 1\}\right\}$
5: $d(v^{\downarrow}_{t,\boldsymbol{x}}) \coloneqq d(v^{\uparrow}_{t,\boldsymbol{x}}) + g_t(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{X}$
6: **for** $\boldsymbol{x} \coloneqq (m_1, \dots, m_d)$ **down to** $\mathbf{0}$ **do**
7: $d(v^{\downarrow}_{t,\boldsymbol{x}}) \coloneqq \min\left\{d(v^{\downarrow}_{t,\boldsymbol{x}}), \min\{d(v^{\downarrow}_{t,\boldsymbol{x}+\boldsymbol{e}_j}) \mid j \in [d], x_j \leq m_j - 1\}\right\}$
8: **if** $t < T$ **then**
9: $d(v^{\uparrow}_{t+1,\boldsymbol{x}}) \coloneqq d(v^{\downarrow}_{t,\boldsymbol{x}})$ for all $\boldsymbol{x} \in \mathcal{X}$
10: Construct the shortest path by traversing backwards from $v^{\downarrow}_{T,\mathbf{0}}$ to $v^{\uparrow}_{1,\mathbf{0}}$
11: **return** schedule X that corresponds to the shortest path

The inner for-loops in lines 3 and 6 must be executed in lexicographical order such that we only take the minimum of values that were already updated. More precisely, in line 3, $\boldsymbol{x}$ must be updated before $\boldsymbol{y}$ if $x_j \leq y_j$ holds for all $j \in [d]$. In line 7, we access the vertices in the other way round.

The runtime of OptimalGraphSearch is given by $\mathcal{O}(T \cdot d \cdot \prod_{j=1}^{d} m_j)$. The factor d appears, since the minimum calculations in lines 4 and 7 access $d+1$ values. However, if we assume that d is constant, the runtime simplifies to $\mathcal{O}(T \cdot \prod_{j=1}^{d} m_j)$. Note that this runtime is still only pseudo-polynomial, since the encoding length of the problem instance is $\mathcal{O}(T + \sum_{j=1}^{d} \log m_j)$.

We summarize our findings in the following theorem.

Theorem 3.1 [AQ21e, Section 4.1]**.** *The algorithm* OptimalGraphSearch *calculates an optimal offline solution for the heterogeneous data center right-sizing problem in* $\mathcal{O}(T \cdot d \cdot \prod_{j=1}^{d} m_j)$ *time.*

3.1.2 $(1+\epsilon)$-approximation

The optimal offline algorithm does not have a polynomial runtime. Therefore, we developed a $(1+\epsilon)$-approximation algorithm that runs in polynomial time if the number d

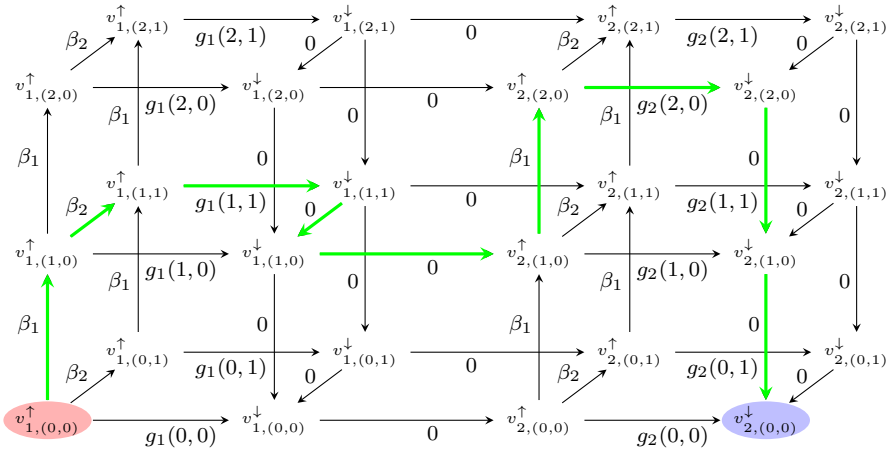

Figure 3.2: Graph representation of a problem instance with $d = 2$ server types and $T = 2$ time slots. There are $m_1 = 2$ servers of type 1 and $m_2 = 1$ server of type 2. The algorithm OptimalGraphSearch calculates a shortest path from $v^{\uparrow}_{1,(0,0)}$ (highlighted in red) to $v^{\downarrow}_{2,(0,0)}$ (highlighted in blue). The shortest path is drawn in green and corresponds to the optimal schedule $\boldsymbol{x}_1 = (1, 1)$ and $\boldsymbol{x}_2 = (2, 0)$.

of server types is constant. The basic idea is to use only a polynomial-sized subset of the vertices in the graph presented above. The number of active servers can take on only specific values. For example, we gain a 3-approximation if we only consider the values $x_{t,j} \in \{0, 1, 2, 4, 8, \ldots, m_j\}$ for each server type j, i.e., each power of two up to m_j as well as 0 and m_j.

In the following, we first construct a reduced graph that depends on a parameter $\gamma > 1$. The resulting approximation factor is analyzed afterwards. Formally, let M_j^{γ} denote the set of values that the number of active servers of type j can take on. The ratio between two consecutive values in M_j^{γ} is at most γ, if the difference between these values is greater than 1. If γ is an integral number, then we can just take the powers of γ to achieve this property. If γ has a fractional value, the construction is more complicated, since the elements of M_j^{γ} must be integral. In our original paper [AQ21e], we use both the rounded down and rounded up values of γ^k to ensure that the ratio between two consecutive values is at most γ. Formally, $M_j^{\gamma} := \{0, 1, \lfloor\gamma^1\rfloor, \lceil\gamma^1\rceil, \lfloor\gamma^2\rfloor, \lceil\gamma^2\rceil, \ldots, m_j\}$.

An alternative way to calculate M_j^{γ} is shown in Algorithm 4. Beginning with $x = 1$, we iteratively add values to M_j^{γ} by multiplying with γ and rounding down. For small values of x, this may result in the same value, so we add 1 if this is the case.

Note that both methods for calculating M_j^{γ} result in the same asymptotic size, namely $|M_j^{\gamma}| \in \mathcal{O}(\log_{\gamma} m_j)$. However, the iterative method has more practical relevance, since the original method needs up to twice as much states for each server type to achieve the same approximation factor.

Algorithm 4 Alternative calculation of M_j^γ

Input: m_j, γ
Output: M_j^γ
1: $M \coloneqq \{0, m_j\}$
2: $x \coloneqq 1$
3: **while** $x < m_j$ **do**
4: $\quad M \coloneqq M \cup \{x\}$
5: $\quad x \coloneqq \max\{\lfloor \gamma x \rfloor, x+1\}$
6: **return** M

The reduced graph G^γ contains the vertices $v_{t,\boldsymbol{x}}^s$ for $s \in \{\uparrow, \downarrow\}$, $t \in [T]$ and $\boldsymbol{x} \in \mathcal{X}^\gamma$ where $\mathcal{X}^\gamma \coloneqq \bigtimes_{j=1}^d M_j^\gamma$ is the set of server configurations considered for the approximation. Analogously to the original graph of the previous section, we have edges from $v_{t,\boldsymbol{x}}^\uparrow$ to $v_{t,\boldsymbol{x}}^\downarrow$ with weight $g_t(\boldsymbol{x})$ for all $t \in [T]$ and $\boldsymbol{x} \in \mathcal{X}^\gamma$ as well as edges from $v_{t,\boldsymbol{x}}^\downarrow$ to $v_{t+1,\boldsymbol{x}}^\uparrow$ with weight 0 for all $t \in [T-1]$ and $\boldsymbol{x} \in \mathcal{X}^\gamma$. The edges for changing the number of active servers must be changed, since several vertices are missing. Let $N_j(x_j) \coloneqq \min\{x \in M_j^\gamma \mid x > x_j\}$ denote the next greater value of x_j in M_j^γ. For each $j \in [d]$ and for each $\boldsymbol{x} \in \mathcal{X}^\gamma$ with $x_j \neq m_j$, let $\boldsymbol{x}' = (x_1, \dots, x_{j-1}, N_j(x_j), x_{j+1}, \dots, x_d)$. There is an edge from $v_{t,\boldsymbol{x}}^\uparrow$ to $v_{t,\boldsymbol{x}'}^\uparrow$ with weight $\beta_j(N_j(x_j) - x_j)$ and an edge from $v_{t,\boldsymbol{x}}^\downarrow$ to $v_{t,\boldsymbol{x}'}^\downarrow$ with weight 0.

The OptimalGraphSearch algorithm can be adapted to work on the reduced graph G^γ. We call the new algorithm ApproximateGraphSearch (AGS). The following lemma shows that the schedule calculated by AGS is a $(2\gamma - 1)$-approximation.

Lemma 3.1 [AQ21e, Theorem 4.1]**.** *Let X^γ be a schedule that corresponds to a shortest path in G^γ and let X^* be an optimal schedule for the original problem instance. Then, the inequality*

$$C(X^\gamma) \leq (2\gamma - 1) \cdot C(X^*) \tag{3.3}$$

is always satisfied, i.e., X^γ is a $(2\gamma - 1)$-approximation.

To achieve an approximation factor of $1 + \epsilon$, we set $\gamma = 1 + \epsilon/2$. If the number of server types d is constant, then the resulting runtime is polynomial in both the input length and ϵ as the following theorem shows. Therefore, AGS is a fully polynomial time approximation scheme.

Theorem 3.2 [AQ21e, Theorem 4.2]**.** *The algorithm ApproximateGraphSearch calculates a $(1+\epsilon)$-approximation in $\mathcal{O}(T \cdot d \cdot \epsilon^{-d} \cdot \prod_{j=1}^d \log m_j)$ time.*

Analysis. In the following, we present the proof ideas of Lemma 3.1 and Theorem 3.2.

Proof idea of Lemma 3.1. Let $\alpha \coloneqq 2\gamma - 1$ be the approximation factor. Instead of directly analyzing the shortest path X^γ, we construct an alternative schedule X' that

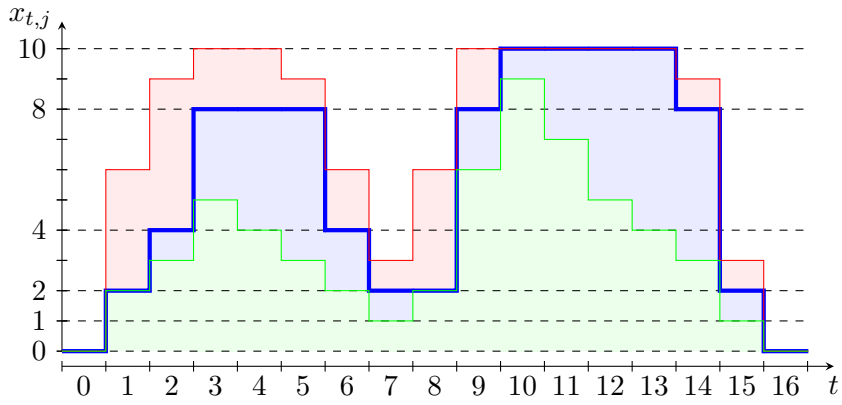

Figure 3.3: Construction of X' (shown in blue) for one specific server type j. In this example, we have $\gamma = 2$ and $m_j = 10$, so the allowed states for X' are $M_j^\gamma = \{0, 1, 2, 4, 8, 10\}$ (indicated by the dashed horizontal lines). The optimal schedule X^* is printed in green. The red line shows the value of $\min\{m_j, \alpha \cdot x^*_{t,j}\}$ with $\alpha = 2\gamma - 1 = 3$. Note that the schedule X' always stays between the red and green line and only changes the number of active servers to satisfy the invariant $x^*_{t,j} \leq x'_{t,j} \leq \alpha \cdot x^*_{t,j}$.

also only uses the server configurations in $\mathcal{X}^\gamma$. The schedule X' might not correspond to a shortest path in G^γ, however, we will show that it satisfies the inequality $C(X') \leq \alpha \cdot C(X^*)$. The optimality of X^γ regarding G^γ implies $C(X^\gamma) \leq C(X')$ and therefore inequality (3.3) is satisfied.

X' lazily stays between the optimal solution $x^*_{t,j}$ and $\alpha \cdot x^*_{t,j}$ while only using the states in M_j^γ. Formally, $x'_{t,j}$ is defined by

$$x'_{t,j} \coloneqq \begin{cases} \min\{x \in M_j^\gamma \mid x \geq x^*_{t,j}\} & \text{if } x'_{t-1,j} \leq x^*_{t,j} \\ x'_{t-1,j} & \text{if } x^*_{t,j} < x'_{t-1,j} \leq \alpha \cdot x^*_{t,j} \\ \max\{x \in M_j^\gamma \mid x \leq \alpha \cdot x^*_{t,j}\} & \text{if } \alpha \cdot x^*_{t,j} < x'_{t-1,j} \end{cases}$$

An example of the construction of X' can be found in Figure 3.3.

In the schedule X', there are always more active servers of each type than in X^*. Therefore, the operating cost of a single server is reduced (or remains equal), as $f_{t,j}$ are increasing functions. The number of active servers of each type is at most α times greater than the number in X^*. Thus, the operating cost of X' is a α-approximation, i.e., $R(X') \leq \alpha \cdot R(X^*)$ where $R(X)$ is the operating cost of X [AQ21d, Lemma 19].

The analysis of the switching cost uses the fact that the ratio between two consecutive states in M_j^γ is at most γ. Together with the inequality $x^*_{t,j} \leq x'_{t,j} \leq \alpha \cdot x^*_{t,j}$, it can be shown that the switching cost of X' is a α-approximation, i.e., $S(X') \leq \alpha \cdot S(X^*)$ where $S(X)$ is the switching cost of X [AQ21d, Lemma 20].

By adding both inequalities, we get $C(X') = R(X') + S(X') \leq \alpha R(X^*) + \alpha S(X^*) = \alpha C(X^*)$. □

Proof idea of Theorem 3.2. The approximation factor directly follows form Lemma 3.1 with $\gamma = 1+\epsilon/2$. We still have to analyze the runtime. AGS uses $|M_j^\gamma| = \mathcal{O}(\log_{1+\epsilon/2} m_j)$ different states for server type j. By Theorem 3.1, this leads to a runtime of $\mathcal{O}(T \cdot d \cdot \prod_{j=1}^{d} \log_{1+\epsilon/2} m_j)$. By using the estimation $\ln(x) \geq 1 - 1/x$, we can transform the term $\log_{1+\epsilon/2} m_j$ as follows:

$$\log_{1+\epsilon/2} m_j = \frac{\ln m_j}{\ln(1+\epsilon/2)} \leq \frac{\ln m_j}{1 - \frac{1}{1+\epsilon/2}} = \left(1 + \frac{1}{\epsilon/2}\right) \ln m_j \in \mathcal{O}(\epsilon^{-1} \log m_j).$$

Therefore, we get a total runtime of $\mathcal{O}(T \cdot d \cdot \epsilon^{-d} \cdot \prod_{j=1}^{d} \log m_j)$. □

Time-dependent data center size. The size of a real data center usually changes over time. If hardware fails or if old servers are shut down permanently, then the number m_j of available servers decreases. If a data center is extended by new servers, then m_j increases. Let $m_{t,j}$ denote the number of servers of type j that are available during time slot t. This can be modeled by removing the corresponding vertices from the graph G^γ. The AGS algorithm can simply be adapted for this setting leading to the following result.

Theorem 3.3 [AQ21e, Theorem 4.3]**.** *Let $\mathcal{I}$ be an instance of the heterogeneous data center right-sizing problem where the number of available servers depends on time. A $(1+\epsilon)$-approximation can be calculated in*

$$\mathcal{O}\left(d \cdot \epsilon^{-d} \cdot \sum_{t=1}^{T} \prod_{j=1}^{d} \log m_{t,j}\right) \subseteq \mathcal{O}\left(T \cdot d \cdot \epsilon^{-d} \cdot \prod_{j=1}^{d} \log \max_{t \in [T]} m_{t,j}\right)$$

time.

3.2 Online problem

In this section, we investigate the online problem and present different results depending on the complexity of the operating cost functions. We begin with the most simple case where the operating costs are time- and load-independent, so they only depend on the server type. Furthermore, we assume that all servers have the same computational power and we exclude inefficient server types, so a server with a higher switching cost always has a lower operating cost. For this simplified setting, we present a $2d$-competitive deterministic and a $1.582d$-competitive randomized online algorithm.

Afterwards, we handle operating costs that depend on the load of the server, but are still fixed in time. We drop the other restrictions, i.e., we allow different computational power as well as inefficient server types. For this case, we develop a deterministic algorithm that achieves a competitive ratio of $2d+1$ (see Section 3.2.2). The algorithm can be modified such that it is able to process time-dependent operating cost functions while still achieving a competitive ratio of $2d+1+\epsilon$ for any $\epsilon > 0$ (see Section 3.2.3).

Finally, in Sections 3.2.4 and 3.2.5, we investigate lower bounds. First, we prove that no deterministic online algorithm can achieve a competitive ratio that is smaller than $2d$. The problem instance used for this proof only contains constant operating cost functions, so the lower bound holds for all problem variants. Especially, it shows that the deterministic algorithm presented in Section 3.2.1 is optimal and the other algorithms are nearly optimal. In Section 3.2.5, we investigate the data center right-sizing problem with arbitrary convex operating cost functions g_t (that do not necessarily satisfy equation (3.2)) and prove a lower bound of $2^d + \frac{2^d-1}{d}$. This exponential lower bound was the reason, why we only developed online algorithms for the case where each server type has its own operating cost function.

3.2.1 Time- and load-independent operating cost

The most simple case are time- and load-independent operating costs and servers with equal computational power. Formally, we have $f_{t,j}(z) \coloneqq r_j = \text{const}$ with $z \in [0,1]$, i.e., each server of type j can process a job volume of at most $z_j^{\max} = 1$ per time slot and causes a constant operating cost of r_j regardless of its load. Furthermore, for our algorithm we have to exclude *inefficient* server types. A server is called *inefficient*, if there is another server with both lower switching and operating costs. The exclusion of inefficient servers implies that a server with a higher switching cost always has a lower operating cost. W.l.o.g., let $\beta_1 \leq \cdots \leq \beta_d$ and $r_1 \geq \cdots \geq r_d$, so the server types are sorted ascending by their switching cost and descending by their operating cost. A reasonable online algorithm would usually first power up a server of type 1, since it has the lowest switching cost. If it is needed for a long period of time, then at some point, it is replaced with a higher server type that has a lower operating cost.

Note that the exclusion of inefficient server types is only a small restriction, since an inefficient server would only be powered up if all more efficient servers are already running. However, if an inefficient server is active, but not needed any more, then the decision whether the inefficient or an efficient server should be powered down is non-trivial. On the one hand, the inefficient server has a higher operating cost, so it should be avoided to run it idle. On the other hand, it may be necessary to restart the server that was powered down, since a large job volume arrives after a short time. In this case, powering the efficient server down and up is cheaper due to its lower switching cost.

The exclusion of inefficient servers is similar to the convex case in capital investment [Dam03] where the online player can purchase devices with different prices and operating costs. In the convex case, a higher purchase price implies a lower operating cost. Another example is the exclusion of inefficient sleep states in [AIS08] when operating a single server.

The simplification with constant operating costs seems to be related to the Parking Permit Problem introduced by Meyerson [Mey05]. There are d permits which cost β_j and have a duration of D_j with $j \in [d]$. Certain days are driving days where at least one permit is needed. The switching cost in the data center right-sizing problem

corresponds to the purchase price of the permits. Driving days can be modeled by setting the arriving job volume to $\lambda_t = 1$. However, the Parking Permit Problem has no operating cost. Furthermore, there is no analogy for the duration D_j of the permits. If a given problem instance of the data center right-sizing problem is transferred to the Parking Permit Problem by replacing each server type with an infinite number of permits with duration t and cost $\beta_j + t \cdot r_j$, then we still have a different problem, as the online algorithm has to decide how long a server will run already when it is powered up.

In the introduction of this thesis, we mentioned the Online Balanced Descent (OBD) algorithm that achieves a competitive ratio of $3 + \mathcal{O}(1/\mu)$ if the arriving operating cost functions are μ-strongly convex [GW19]. In our case, g_t is a (piecewise) linear function, so $\mu = 0$. Hence, the result of Goel and Wierman would lead to an infinite competitive ratio (besides the problem that the switching cost is not Euclidean). A similar result was achieved by [CGW18] who showed that OBD is $(3 + \mathcal{O}(1/\alpha))$-competitive if the arriving functions are locally α-polyhedral. After replacing $x_{t,j}$ with $x'_{t,j}/\beta_j$ such that the switching cost is equal in each direction, the minimal slope of g_t is r_j/β_j, so it can be arbitrarily small, especially if there is a server type with a tiny operating and huge switching cost. Then, α is almost zero and the resulting competitive ratio of OBD is quite large.

Contribution. In this section, we present a $2d$-competitive deterministic and a $1.582d$-competitive randomized online algorithm for the problem described above. The basic idea of both algorithms is to calculate an optimal schedule for the problem instance that ends at the current time slot. The algorithms ensure that the types of their active servers are at least as large as the server types in the optimal schedule. If this is not the case, such servers are powered down and replaced by the server types used in the optimal schedule. If a server is idle for a specific time depending on the server types used in the previous optimal schedules, it is powered down. The randomized algorithm improves the competitive ratio by randomizing the running time of a server similar to the Ski-Rental problem.

Preliminaries. Before we can explain our algorithms in detail, we have to introduce an alternative way to describe a schedule. A schedule can be separated into $m \coloneqq \sum_{j=1}^{d} m_j$ lanes such that there is at most one server in each lane. The arriving job volume is distributed to the λ_t lowest lanes, i.e., one job each to the lanes 1 to λ_t and no jobs for the lanes $\lambda_t + 1$ to m. The server types of a given schedule are always sorted in descending order, so the server with the greatest switching cost is in the lowest lane.

Given a schedule X, let $y_{t,k}$ be the server type used in lane $k \in [m]$ at time t. If there is no active server, we set $y_{t,k}$ to 0. Formally, $y_{t,k}$ is defined by

$$y_{t,k} \coloneqq \begin{cases} \max\{j \in [d] \mid \sum_{j'=j}^{d} x_{t,j'} \geq k\} & \text{if } k \in \left[\sum_{j=1}^{d} x_{t,j}\right] \\ 0 & \text{else.} \end{cases}$$

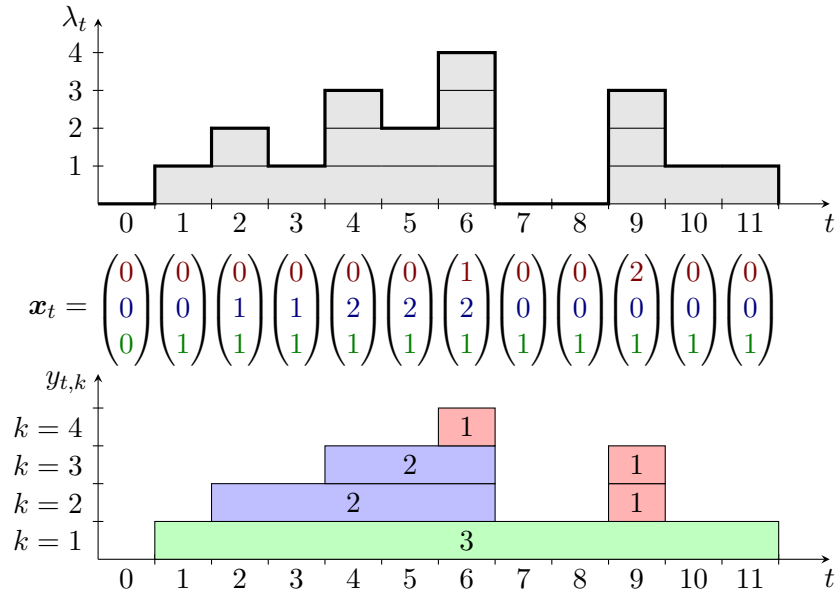

Figure 3.4: Example of the alternative schedule representation. The upper plot shows a job sequence. A feasible schedule is printed in the middle with the common notation $x_{t,j}$. The alternative representation of X is shown in the lower plot. Outside of the colored rectangles, the value of $y_{t,k}$ is 0. Note that the colored rectangles of a feasible schedule always cover the job sequence plot.

The tuple of all $y_{t,k}$ for $t \in [T]$ and $k \in [m]$ is an alternative representation of a schedule. Given $y_{t,k}$, the number $x_{t,j}$ of active servers of type j at time t can be determined by

$$x_{t,j} = |\{k \in [m] \mid y_{t,k} = j\}|.$$

Figure 3.4 shows an example schedule in both notations.

A problem instance is described by the tuple $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{r}, \Lambda)$ with $\boldsymbol{r} = (r_1, \ldots, r_d)$. In contrast to problem instance definition of the general problem at the beginning of this chapter, we replaced the operating cost function vector F with the vector $\boldsymbol{r}$ containing the constant operating costs. Let $\mathcal{I}^t = (t, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{r}, \Lambda^t)$ with $\Lambda^t := (\lambda_1, \ldots, \lambda_t)$ be the shortened problem instance that ends at time slot t. Let $\hat{X}^t$ be an optimal schedule for $\mathcal{I}^t$. For the alternative representation of $\hat{X}^t$, we use the symbols $\hat{y}^t_{t',k}$. There are potentially several optimal schedules for a given problem instance. Our algorithm requires that $\hat{X}^t$ does not use smaller server types in any lane than the previous optimal schedule $\hat{X}^{t-1}$, i.e., $\hat{y}^t_{t',k} \geq \hat{y}^{t-1}_{t',k}$ must hold for all $t' \in [t]$ and $k \in [m]$ with $\hat{y}^t_{t',k} > 0$. We will later describe how such a schedule can be constructed.

Deterministic algorithm. Now, we are able to describe how our deterministic online algorithm $\mathcal{A}$ works. Let $X^{\mathcal{A}}$ be the schedule calculated by our algorithm and let $y^{\mathcal{A}}_{t,k}$ denote its lane representation. When a new job volume λ_t arrives, first of all an optimal

schedule $\hat{X}^t$ with the desired property ($\hat{y}^t_{t',k} > 0 \to \hat{y}^t_{t',k} \geq \hat{y}^{t-1}_{t',k}$) is calculated. Then, beginning from the lowest lane $k = 1$, the algorithm ensures that the server type in lane k is at least as large as the corresponding server type in the optimal schedule. If this is not the case (i.e., $y^{\mathcal{A}}_{t-1,k} < \hat{y}^t_{t,k}$), the old server of type $y^{\mathcal{A}}_{t-1,k}$ is powered down and a server of type $j = \hat{y}^t_{t,k}$ is powered up. The server will run for $\bar{t}_j \coloneqq \lfloor \beta_j / r_j \rfloor$ time slots (providing that it is not replaced by another server). If the optimal schedule uses a smaller or an equal server type, i.e., $y^{\mathcal{A}}_{t-1,k} \geq \hat{y}^t_{t,k}$, then the particular server will stay active for at least $\bar{t}_j$ time slots with $j = \hat{y}^t_{t,k}$. Note that this operation never decrease the running time of a server.

The pseudocode below summarizes how the algorithm works. The variables e_k store the time slot when the server running in lane k will be powered down. Figure 3.5 shows an example how the schedule changes from $t-1$ to t.

Algorithm 5 Algorithm $\mathcal{A}$ [AQ21c]

Input: $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{r}, \Lambda)$
Output: $X^{\mathcal{A}} = (\boldsymbol{x}^{\mathcal{A}}_1, \ldots, \boldsymbol{x}^{\mathcal{A}}_T)$

```
e_k ≔ 0 for all k ∈ [m]
for t ≔ 1 to T do
    Calculate X̂^t such that ŷ^t_{t',k} > 0 → ŷ^t_{t',k} ≥ ŷ^{t-1}_{t',k} for all t' ∈ [t] and k ∈ [m]
    for k ≔ 1 to m do
        if y^A_{t-1,k} < ŷ^t_{t,k} or t ≥ e_k then
            y^A_{t,k} ≔ ŷ^t_{t,k}
            e_k ≔ t + t̄_{y^A_{t,k}}
        else
            y^A_{t,k} ≔ y^A_{t-1,k}
            e_k ≔ max{e_k, t + t̄_{ŷ^t_{t,k}}} where t̄_0 ≔ 0
```

So far, we did not explain how the optimal schedule with the desired property can be determined. As mentioned earlier, an optimal schedule $\hat{X}^t$ can be calculated in polynomial time by a modified version of the minimum-cost flow computation presented in [Alb19]. To ensure that $\hat{y}^t_{t',k} > 0$ implies $\hat{y}^t_{t',k} \geq \hat{y}^{t-1}_{t',k}$, we determine the so called *maximum* schedule. Given the optimal schedules $\hat{X}^{t-1}$ and $\hat{X}^t$, let $y^{\max}_{t',k} \coloneqq \max\{\hat{y}^{t-1}_{t',k}, \hat{y}^t_{t',k}\}$ be the maximal server type of both schedules. To avoid immediate server changes (that cannot happen in an optimal schedule [AQ21a, Lemma 3]), we replace the lower server type with the greater one if the server type changes immediately. Formally, if $y^{\max}_{t'-1,k} \neq y^{\max}_{t',k}$ and $y^{\max}_{t'-1,k} > 0$ and $y^{\max}_{t',k} > 0$, then we set both $y^{\max}_{t'-1,k}$ and $y^{\max}_{t',k}$ to the greater value $\max\{y^{\max}_{t'-1,k}, y^{\max}_{t',k}\}$. This procedure is repeated until $y^{\max}_{t'-1,k} > 0 \wedge y^{\max}_{t',k} > 0$ implies $y^{\max}_{t'-1,k} = y^{\max}_{t',k}$ for all $t' \in [t]$. The feasibility and optimality of the resulting schedule is shown in [AQ21c, Lemmas 1–2].

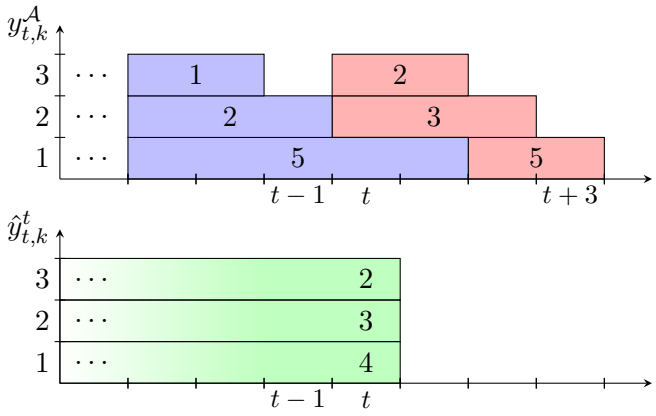

Figure 3.5: Example of an update in algorithm $\mathcal{A}$ from time slot $t-1$ to t. The schedule of $\mathcal{A}$ (upper plot) at $t-1$ is shown in blue, the changes after reacting to λ_t are printed in red. The lower plot displays the optimal schedule $\hat{X}^t$ in green. In this example, the running times are defined as $\bar{t}_j \coloneqq j$. In the lowest lane $k=1$, we have $y^{\mathcal{A}}_{t,1} = 5 \geq 4 = \hat{y}^t_{t,1}$, so server type $y^{\mathcal{A}}_{t,1}$ will run for at least $\bar{t}_4 = 4$ further time slots (including the current time slot t), i.e., it will be powered down after time slot $t+3$. In lane $k=2$, server type $y^{\mathcal{A}}_{t-1,2} = 2$ is powered down and replaced by $\hat{y}^t_{t,2} = 3$, since $y^{\mathcal{A}}_{t-1,2} < \hat{y}^t_{t,2}$. In lane $k=3$, there is no active server at time slot $t-1$, so server type $\hat{y}^t_{t,3} = 2$ is powered up.

Theorem 3.4 [AQ21c, Theorem 1]**.** *Given the heterogeneous data center right-sizing problem with time- and load-independent operating costs, no inefficient servers and equal computational power for all server types, algorithm $\mathcal{A}$ achieves a competitive ratio of $2d$.*

Before we can prove the competitive ratio, we have to show that the schedule is actually feasible. This is not trivial, since each lane is handled separately and it is not clear, why algorithm $\mathcal{A}$ will never use more servers of a given type than available.

Lemma 3.2 [AQ21c, Lemma 4]**.** *Algorithm $\mathcal{A}$ calculates a feasible schedule.*

Proof idea. We have to show that (1) there are enough active servers to handle the arriving job volume (i.e., $\sum_{j=1}^{d} x^{\mathcal{A}}_{t,k} \geq \lambda_t$) and that (2) there are not more active servers than available (i.e., $x^{\mathcal{A}}_{t,j} \in [m_j]_0$). The first property directly follows from the invariant $y^{\mathcal{A}}_{t,k} \geq \hat{y}^t_{t,k}$ that is always satisfied. The proof of the second property is more complicated. The order of the server types and the exclusion of inefficient servers imply that the running times $\bar{t}_j$ are sorted in ascending order, i.e., $\bar{t}_1 \leq \cdots \leq \bar{t}_d$. This fact is used to prove that the server types in the lanes of $X^{\mathcal{A}}$ are sorted in descending order, i.e., $y^{\mathcal{A}}_{t,1} \geq \cdots \geq y^{\mathcal{A}}_{t,m}$ [AQ21c, Lemma 3]. To deduce a contradiction, we assume that the condition $x^{\mathcal{A}}_{t,j} \leq m_j$ is violated at some point. This can happen in line 6 or 9. In both cases, we obtain a contradiction, so $X^{\mathcal{A}}$ must be a feasible schedule. □

To analyze the competitive ratio of algorithm $\mathcal{A}$, we divide its schedule into blocks $A_{t,k}$ with $t \in [T]$ and $k \in [m]$. There are two types of blocks: *new* blocks and *extended* blocks. If a new server is powered up at time slot t in lane k, then there is the *new* block. If the running time of an active server is extended, then we call $A_{t,k}$ an *extended* block. If neither the running time is extended nor a new server is powered up at time t in lane k, then the block $A_{t,k}$ does not exist.

Let $C(A_{t,k})$ denote the cost caused by $A_{t,k}$. The cost of a *new* block is upper bounded by $\beta_j + \bar{t}_j \cdot r_j$ where $j = \hat{y}^t_{t,k}$ is the server type that was powered up. This term is at most $2\beta_j$ due to the definition of $\bar{t}_j$ [AQ21c, Lemma 5]. For an extended block, the cost depends on how many time slots the running time was increased. If $A_{t,k}$ does not exist, we set $C(A_{t,k}) = 0$. For a more precise definition of $A_{t,k}$ and its cost $C(A_{t,k})$, we refer to [AQ21c].

For our analysis, we have to introduce further notations. Let

$$C_{t,k}(X) \coloneqq \begin{cases} f_{y_{t,k}}(0) + \beta_{y_{t,k}} & \text{if } y_{t-1,k} \neq y_{t,k} > 0 \\ f_{y_{t,k}}(0) & \text{if } y_{t-1,k} = y_{t,k} > 0 \\ 0 & \text{otherwise.} \end{cases}$$

be the cost of the schedule X at time t in lane k. Note that the total cost of an arbitrary schedule X can be written as $C(X) = \sum_{t=1}^{T} \sum_{k=1}^{m} C_{t,k}(X)$.

Let $\tilde{y}^u_{t,k} \coloneqq \max_{t' \in [t:u]} \hat{y}^{t'}_{t',k}$ denote the largest server type used in lane k by the schedule $\hat{X}^{t'}$ at its last time slot t' for $t' \in [t:u]$. For example, a value of $\tilde{y}^{10}_{5,k} = 4$ means that during the time interval from $t = 5$ up to the current time slot $u = 10$, our algorithm has never accessed a server type in lane k that was greater than 4.

Given the optimal schedules $\hat{X}^u$ and $\hat{X}^v$ with $u < v$, it is clear that

$$\sum_{k=1}^{m} \sum_{t=1}^{u} C_{t,k}(\hat{X}^u) \leq \sum_{k=1}^{m} \sum_{t=1}^{u} C_{t,k}(\hat{X}^v)$$

since $\hat{X}^u$ is optimal for the problem instance that ends at time u. The following quite technical lemma shows that this property still holds if the cost $C_{t,k}(\cdot)$ is scaled by the factor $\tilde{y}^u_{t,k}$.

Lemma 3.3 [AQ21c, Lemma 7]**.** *Let $u, v \in [T]$ with $u < v$. It holds that*

$$\sum_{k=1}^{m} \sum_{t=1}^{u} \tilde{y}^u_{t,k} C_{t,k}(\hat{X}^u) \leq \sum_{k=1}^{m} \sum_{t=1}^{u} \tilde{y}^u_{t,k} C_{t,k}(\hat{X}^v).$$

The proof uses monotony properties of $\tilde{y}^u_{t,k}$ in combination with the optimality of $\hat{X}^u$. The next lemma demonstrates how the cost of a block can be folded into the term $2\sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^v)$.

Lemma 3.4 [AQ21c, Lemma 8]**.** *For all lanes $k \in [m]$ and time slots $v \in [T]$, it holds that*

$$2 \sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^v) + C(A_{v,k}) \leq 2 \sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v). \tag{3.4}$$

We will only show the proof idea for new blocks. For *extended* blocks the proof is quite similar and can be found in [AQ21b]. If $A_{v,k}$ does not exist, then $C(A_{v,k}) = 0$ and equation (3.4) is obviously fulfilled by the definition of $\tilde{y}^v_{t,k}$.

Proof idea. If $A_{v,k}$ is a new block, then we know that its corresponding server type $j \coloneqq \hat{y}^v_{v,k}$ was not used in the last time slot of the last $\bar{t}_j$ optimal schedules. Therefore, $\tilde{y}^{v-1}_{t,k} \leq \tilde{y}^v_{t,k} - 1$ must hold for $t \in [v - \bar{t}_j : v - 1]$. The cost of $\hat{X}^v$ in lane k during the last $\bar{t}_j$ time slots is at least β_j, since $y^{\mathcal{A}}_{v,k} = j$ implies that a server of type j was either powered up during the last $\bar{t}_j$ time slots or runs for more than $\bar{t}_j$ time slots. In the former case, there is the switching cost β_j, and in the latter case, we have an accumulated operating cost of $r_j \cdot (\bar{t}_j + 1) \geq \beta_j$. Furthermore, we already noticed that the cost caused by $A_{v,k}$ is at most $2\beta_j$. By putting these facts together, we finally obtain inequality (3.4). □

Now, we are able to prove the competitive ratio of algorithm $\mathcal{A}$.

Proof idea for Theorem 3.4. We will prove the inequality

$$C_v(X^{\mathcal{A}}) \leq 2 \sum_{k=1}^{m} \sum_{t=1}^{v} \tilde{y}^v_{t,k} C_{t,k}(\hat{X}^v) \tag{3.5}$$

by induction. For $v = 0$, the inequality is obviously fulfilled. Assume that inequality (3.5) holds for $v - 1$. The cost of $X^{\mathcal{A}}$ up to time v can be split into the cost up to time $v - 1$ plus the cost for the blocks created at time slot v. For the first term, we can apply the induction hypothesis and transform the resulting sum with Lemma 3.3. Afterwards, we apply Lemma 3.4 for each lane k to eliminate the cost of the blocks. Formally, we get

$$\begin{aligned}
C_v(X^{\mathcal{A}}) &= C_{v-1}(X^{\mathcal{A}}) + \sum_{k=1}^{m} C(A_{v,k}) \\
&\overset{\text{I.H.}}{\leq} 2 \sum_{k=1}^{m} \sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^{v-1}) + \sum_{k=1}^{m} C(A_{v,k}) \\
&\overset{L3.3}{\leq} 2 \sum_{k=1}^{m} \sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^{v}) + \sum_{k=1}^{m} C(A_{v,k}) \\
&\overset{L3.4}{\leq} 2 \sum_{k=1}^{m} \sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^{v}).
\end{aligned}$$

Remember that $\tilde{y}^v_{t,k}$ represents a server type, so $\tilde{y}^v_{t,k} \leq d$. Thus, at the end of the time horizon ($v = T$), we get $C(X^{\mathcal{A}}) \leq 2d \cdot C(\hat{X}^T)$. □

Randomization. The competitive ratio of algorithm $\mathcal{A}$ can be improved by randomization. Similar to the Ski Rental problem where the time point for buying skis can be randomized to achieve a better competitive ratio, we will randomize the running time of a server. More precisely, at the beginning we choose $\sigma \in [0,1]$ according to the probability density function $f_\sigma(x) = e^x/(e-1)$ for $x \in [0,1]$. Then, we redefine the running times as $\bar{t}_j \coloneqq \lfloor \sigma \cdot \beta_j / r_j \rfloor$. Afterwards, we execute algorithm $\mathcal{A}$ with these new running times. We call this new procedure algorithm $\mathcal{B}$. Note that σ is determined at the beginning of the algorithm and is not changed afterwards. The pseudo code below summarizes how algorithm $\mathcal{B}$ works.

Algorithm 6 Algorithm $\mathcal{B}$

Input: $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{r}, \Lambda)$
Output: $X^{\mathcal{B}} = (\boldsymbol{x}_1^{\mathcal{B}}, \dots, \boldsymbol{x}_T^{\mathcal{B}})$
1: Choose $\sigma \in [0,1]$ according to density $f_\sigma(x) = e^x/(e-1)$ for $x \in [0,1]$
2: Set $\bar{t}_j \coloneqq \lfloor \sigma \cdot \beta_j / l_j \rfloor$ for all $j \in [d]$
3: Execute algorithm $\mathcal{A}$ and set $\boldsymbol{x}_t^{\mathcal{B}} \coloneqq \boldsymbol{x}_t^{\mathcal{A}}$ after each iteration

Theorem 3.5 [AQ21b, Theorem 2]**.** *Given the heterogeneous data center right-sizing problem with time- and load-independent operating costs, no inefficient servers and equal computational power for all server types, algorithm $\mathcal{B}$ achieves a competitive ratio of $\frac{e}{e-1} \cdot d \approx 1.582d$ against an oblivious adversary.*

Proof idea. Most parts of the proof of the determiinistic algorithm can be reused without any changes, since they do not depend on the exact values of $\bar{t}_j$. Instead of the cost of a block, we now consider its expected cost $\mathbb{E}[C(A_{t,k})]$. Afterwards, we have to adapt Lemma 3.4 by proofing the inequality

$$\frac{e}{e-1} \cdot \sum_{t=1}^{v-1} \tilde{y}_{t,k}^{v-1} C_{t,k}(\hat{X}^v) + \mathbb{E}[C(A_{v,k})] \leq \frac{e}{e-1} \cdot \sum_{t=1}^{v} \tilde{y}_{t,k}^{v} C_{t,k}(\hat{X}^v).$$

The final proof of Theorem 3.5 is analogous to the proof of Theorem 3.4. More details can be found in [AQ21b]. □

3.2.2 Time-independent operating cost

Usually, different server types can process different amounts of jobs per time slot. Furthermore, the energy consumption depends on the load. As we learn in the introduction, an idle server usually consumes only half of its peak power [SR09]. Hence, the model analyzed in the previous section greatly simplifies the reality. Therefore, in this section, we handle load-dependent operating cost functions whose domain is no longer restricted to $[0,1]$. Formally, we have $f_{t,j}(z) \coloneqq f_j(z)$ with $z \in [0, z_j^{\max}]$. This makes the problem much harder. Interestingly, we still achieve a competitive ratio of $2d+1$ which is only slightly greater than the competitive ratio of the deterministic algorithm presented in

the previous section. In fact, the new algorithm $\mathcal{C}$ is $2d$-competitive for load-independent operating costs, even if there are inefficient server types. Nevertheless, the results of the previous section are still valuable, as the randomized algorithm $\mathcal{B}$ obtains a competitive ratio of $1.582d$. It is an open question whether the competitive ratio of algorithm $\mathcal{C}$ can also be improved by randomization.

Note that algorithm $\mathcal{C}$ introduced in this section was called algorithm $\mathcal{A}$ in the original paper [AQ21e]. This renaming was necessary because both the algorithm of the previous section and the one of this section were called algorithm $\mathcal{A}$ in [AQ21c] and [AQ21e], respectively.

Let $\hat{X}^t$ be an arbitrary optimal schedule for the problem instance that ends at time slot t. Such a schedule can be calculated with the optimal offline algorithm presented in Section 3.1.1. In practice, it might make sense to use the approximation algorithm of Section 3.1.2 to avoid an exponential runtime. In contrast to algorithms $\mathcal{A}$ and $\mathcal{B}$, we do not need the alternative schedule representation, but directly use the number of active servers $\hat{x}^t_{t',j}$.

Algorithm $\mathcal{C}$ calculates its schedule $X^{\mathcal{C}}$ as follows. At any time, it ensures that the number of active servers of each type is always at least as large as the number of active servers of the same type in the optimal schedule $\hat{X}^t$ during its last time slot. If this is not the case, servers are powered up such that the inequality $x^{\mathcal{C}}_{t,j} \geq \hat{x}^t_{t,j}$ is satisfied for all server types $j \in [d]$. The running time of a server is given by $\bar{t}_j \coloneqq \lceil \beta_j / f_j(0) \rceil$, so a server is powered down after its accumulated *idle* operating cost exceeds its switching cost for the first time. Note that the workload of a server does not influence its running time.

A formal definition of algorithm $\mathcal{C}$ is shown in the pseudocode below. The variables $w_{t,j}$ store the number of servers of type j that were powered up at time slot t. Thus, $w_{t-\bar{t}_j,j}$ is the number of servers of type j that has to be powered down at the beginning of time slot t. Figure 3.6 shows an example how algorithm $\mathcal{C}$ operates.

Algorithm 7 Algorithm $\mathcal{C}$ [AQ21e]

Input: $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{f}, \Lambda)$ with $\boldsymbol{f} = (f_1, \ldots, f_d)$
Output: $X^{\mathcal{C}} = (\boldsymbol{x}^{\mathcal{C}}_1, \ldots, \boldsymbol{x}^{\mathcal{C}}_T)$

```
x^C_{0,j} := 0 for all j ∈ [d]
w_{t,j} := 0 for all t ∈ ℤ and j ∈ [d]
for t := 1 to T do
    Calculate X̂^t
    for j := 1 to d do
        x^C_{t,j} := x^C_{t-1,j} − w_{t−t̄_j,j}
        if x^C_{t,j} < x̂^t_{t,j} then
            w_{t,j} := x̂^t_{t,j} − x^C_{t,j}
            x^C_{t,j} := x̂^t_{t,j}
```

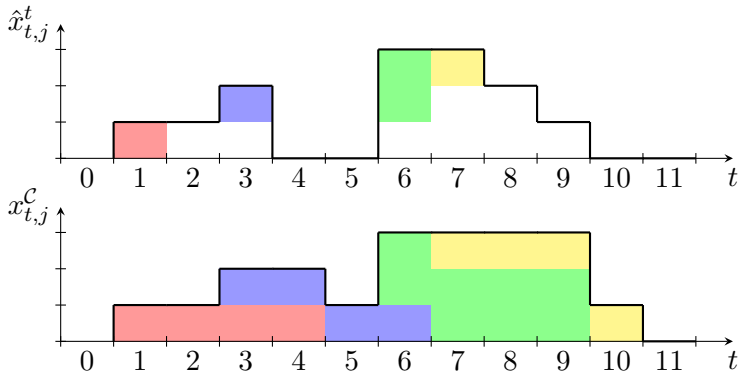

Figure 3.6: Visualization of algorithm $\mathcal{C}$ for one server type j with $\bar{t}_j = 4$. The upper plot shows the values of $\hat{x}^t_{t,j}$ that are needed to determine the schedule $X^{\mathcal{C}}$ displayed in the lower plot. Note that the upper plot is not an optimal schedule, but the last state of each optimal schedule $\hat{X}^1, \hat{X}^2, \ldots, \hat{X}^T$. Each colored square in the upper plot indicates that $\mathcal{C}$ has to power up a server to satisfy the invariant $\hat{x}^{\mathcal{C}}_{t,j} \geq \hat{x}^t_{t,j}$. The running time of the activated server is visualized with the same color in the lower plot.

The feasibility of the calculated schedule $X^{\mathcal{C}}$ directly follows from the invariant $x^{\mathcal{C}}_{t,j} \geq \hat{x}^t_{t,j}$ that is always fulfilled [AQ21e, Lemma 2.1].

Theorem 3.6 [AQ21e, Theorem 2.7]**.** *Algorithm $\mathcal{C}$ solves the heterogeneous data center right-sizing problem with time-independent operating cost functions and achieves a competitive ratio of $2d + 1$.*

Analysis. For the analysis of the competitive ratio, we divide the operating cost into an *idle* and a *load-dependent* part and analyze them separately. The idle operating cost of an active server of type j running for a single time slot is given by $f_j(0)$. Given a schedule X, the load-dependent operating cost of all active servers of type j during time slot t is defined as

$$L_{t,j}(X) \coloneqq x_{t,j} \left(f_j \left(\frac{\lambda_t q_{t,j}}{x_{t,j}} \right) - f_j(0) \right) \tag{3.6}$$

where $q_{t,j}$ is the job ratio assigned to server type j at time slot t, implicitly given by equation (3.2). Formally, these values are defined by

$$(q_{t,1}, \ldots, q_{t,d}) \coloneqq \operatorname*{argmin}_{(q_1,\ldots,q_d)\in\mathcal{Q}} \sum_{j=1}^{d} g_{t,j}(x_{t,j}, q_j).$$

Note that $L_{t,j}(X) \geq 0$, since f_j is an increasing function. The next lemma gives us an upper bound for the total load-dependent operating cost caused by algorithm $\mathcal{C}$.

Lemma 3.5 [AQ21e, Lemma 2.4]**.** *The total load-dependent operating cost of algorithm $\mathcal{C}$ is bounded by the cost of the optimal schedule. Formally,*

$$\sum_{t=1}^{T}\sum_{j=1}^{d} L_{t,j}(X^{\mathcal{C}}) \leq C(\hat{X}^T). \tag{3.7}$$

Proof idea. First of all, it is shown that the load-dependent operating cost of server type j at time slot t in the schedule $X^{\mathcal{C}}$ never exceeds the load-dependent operating cost in the optimal schedule $\hat{X}^t$, i.e., $L_{t,j}(X^{\mathcal{C}}) \leq L_{t,j}(\hat{X}^t)$ [AQ21e, Lemma 2.3]. This fact follows from the invariant $x^{\mathcal{C}}_{t,j} \geq \hat{x}^t_{t,j}$ and from the convexity of f_j. Afterwards, inequality (3.7) can be proven by induction over t. □

To analyze the switching and idle operating cost, we divide the schedule $X^{\mathcal{C}}$ into blocks. A block begins when a server is powered up and ends when it is powered down. Let n_j be the total number of power-up operations[1] of server type j. The blocks are denoted by $A_{j,i}$ with $j \in [d]$ and $i \in [n_j]$. Note that the length of a block only depends on the server type j and is given by $\bar{t}_j$. Thus, its switching and idle operating costs are $C(A_{j,i}) = \beta_j + \bar{t}_j f_j(0)$. In the following lemma, we estimate the switching and idle operating costs of one server type.

Lemma 3.6 [AQ21e, Lemma 2.6]**.** *The cost of all blocks of server type $j \in [d]$ is at most two times as large as the cost of the optimal schedule. Formally,*

$$\sum_{i=1}^{n_j} C(A_{j,i}) \leq 2 \cdot C(\hat{X}^T) \tag{3.8}$$

holds for all $j \in [d]$.

Proof idea. First, we notice that the cost of a block is bounded by

$$C(A_{j,i}) \leq 2 \cdot \min\{\beta_j + f_j(0), \bar{t}_j \cdot f_j(0)\}$$

due to the definition of $\bar{t}_j$ [AQ21e, Lemma 2.5]. To simplify the notation, we abbreviate the minimum term as $\min\{\ldots\}$.

For each server type j, we define special time slots so that each block $A_{j,i}$, $i \in [n_j]$, is covered by exactly one special time slot as shown in Figure 3.7. Furthermore, we ensure that at least one server is powered up during each special time slot. The distance between two special time slots is at least the running time $\bar{t}_j$ of a server. If there are n blocks covered by a special time slot τ, then we know that their cost is bounded by $n \cdot 2 \cdot \min\{\ldots\}$. On the other hand, the optimal schedule $\hat{X}^\tau$ also uses exactly n servers of type j at time τ, because otherwise algorithm $\mathcal{C}$ would not have powered up any server at time τ or even more servers to ensure the invariant $x^{\mathcal{C}}_{\tau,j} \geq \hat{x}^\tau_{\tau,j}$. During

[1] If a server is powered down in line 6 and powered up in the same time slot in line 9, the algorithm does not have to pay a switching post. However, here we count this as a power-up operation to simplify the analysis and assume that the switching cost has to be paid.

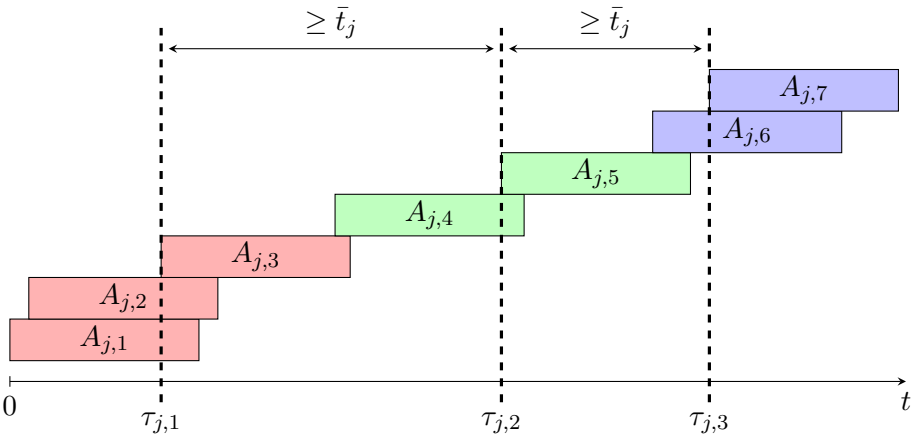

Figure 3.7: Visualization of the special time slots for one specific server type j. The special time slots are marked with dashed lines. The rectangles represent the blocks $A_{j,i}$. Each block is covered by exactly one special time slot, as indicated by the colors. The distance between two consecutive special time slots is always greater than or equal to $\bar{t}_j$.

the time interval between the previous and the current special time slot, the servers of type j in the optimal schedule $\hat{X}^\tau$ cause a switching cost of β_j or an idle operating cost of $\bar{t}_j f_j(0)$. Formally, this is $n \cdot \min\{\ldots\}$, i.e., half of the cost caused by the covered blocks. By induction over the special time slots, we can prove that the cost of the blocks of a specific server type created during or before the special time slot τ is at most $2 \cdot C(\hat{X}^\tau)$. For the last special time slot, we finally obtain inequality (3.8). □

Now, Theorem 3.6 directly follows from Lemmas 3.5 and 3.6.

Proof of Theorem 3.6. The total cost of algorithm $\mathcal{C}$ is at most

$$\begin{aligned} C(X^{\mathcal{C}}) &= \sum_{j=1}^{d} \sum_{i=1}^{n_j} C(A_{j,i}) + \sum_{t=1}^{T} \sum_{j=1}^{d} L_{t,j}(X^{\mathcal{C}}) \\ &\overset{\text{L3.5}}{\underset{\text{L3.6}}{\leq}} \sum_{j=1}^{d} 2 \cdot C(\hat{X}^T) + C(\hat{X}^T) \\ &= (2d+1) \cdot C(\hat{X}^T). \end{aligned}$$

Since $\hat{X}^T$ is an optimal schedule for the original problem instance $\mathcal{I}$, algorithm $\mathcal{C}$ is $(2d+1)$-competitive. □

If the operating cost of all server types is constant (i.e., $f_j(z) = r_j = \text{const}$ for $z \in [0, z_j^{\max}]$), then the load-dependent operating cost $L_{t,j}(X)$ is always zero, so the total cost of algorithm $\mathcal{C}$ is bounded by $2d \cdot C(\hat{X}^T)$. We gain the same result as of

Theorem 3.4, but without the exclusion of inefficient server types. Furthermore, the server are not required to have the same computational power. We summarize this finding in the following corollary.

Corollary 3.1 [AQ21e, Corollary 2.8]**.** *Algorithm $\mathcal{C}$ achieves a competitive ratio of $2d$, if the operating cost does not depend on the load, i.e., $f_j(z) \coloneqq r_j = const$ for $z \in [0, z_j^{max}]$.*

3.2.3 Time-dependent operating cost

In practice, the operating cost may change over time, for example, due to varying energy prices. We can extend our algorithm of the previous section such that it can handle time-dependent operating cost functions $f_{t,j}$. The extension consists of two steps. First, we only adjust the running time of a server which leads to a competitive ratio of $2d + 1 + c(\mathcal{I})$ as an intermediate result. The constant $c(\mathcal{I})$ depends on the problem instance $\mathcal{I}$ and is defined as $c(\mathcal{I}) \coloneqq \sum_{j=1}^{d} \max_{t\in[T]} \frac{f_{t,j}(0)}{\beta_j}$. Unfortunately, this constant can be arbitrarily large. Therefore, in the second step, we modify the problem instance by dividing each time slot into several sub time slots. The resulting schedule is adapted without increasing the cost such that it fits to the original problem instance. By choosing the number of sub time slots properly, we can make the constant arbitrarily small and achieve a competitive ratio of $2d + 1 + \epsilon$ for any $\epsilon > 0$.

When handling time-dependent operating cost functions, the idle operating cost $f_{t,j}(0)$ changes over time, so we have to adapt the running time of a server. Nevertheless, the basic idea stays the same: a server is powered down when its accumulated idle operating cost exceeds its switching cost. However, the operating cost of the first time slot at which the server was powered up is ignored. More precisely, in algorithm $\mathcal{D}$, a server of type j that was powered up at time slot t runs for

$$\bar{t}_{t,j} \coloneqq \max \left\{ \bar{t} \in [T-t] \mid \sum_{u=t+1}^{t+\bar{t}} f_{u,j}(0) \leq \beta_j \right\}$$

further time slots, so it is kept in the active state for a total of $\bar{t}_{t,j} + 1$ time slots. Note that this definition differs from the previous section where a server stays active for $\bar{t}_j$ time slots.

The pseudo code below clarifies how algorithm $\mathcal{D}$ operates[2]. Only lines 6 and 7 change in comparison to algorithm $\mathcal{C}$. Again, $w_{t,j}$ are the number of servers of type j that were powered up at time t. The set W_t contains the time slots u with $u + \bar{t}_{u,j} + 1 = t$. Hence, the number of servers of type j that have to be powered down at time slot t is given by $\sum_{u\in W_t} w_{t,j}$. Figure 3.8 shows an example of algorithm $\mathcal{D}$ including the values of $\bar{t}_{t,j}$ and W_t.

Theorem 3.7 [AQ21e, Theorem 3.4]**.** *Algorithm $\mathcal{D}$ solves the heterogeneous data center right-sizing problem with time-dependent operating cost functions and achieves a competitive ratio of $2d + 1 + c(\mathcal{I})$ with $c(\mathcal{I}) = \sum_{j=1}^{d} \max_{t\in[T]} \frac{f_{t,j}(0)}{\beta_j}$.*

[2] In the original paper [AQ21e], this algorithm was called algorithm $\mathcal{B}$.

Algorithm 8 Algorithm $\mathcal{D}$ [AQ21e]

Input: $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$
Output: $X^{\mathcal{D}} = (\boldsymbol{x}_1^{\mathcal{D}}, \ldots, \boldsymbol{x}_T^{\mathcal{D}})$

$x_{0,j}^{\mathcal{D}} \coloneqq 0$ for all $j \in [d]$
$w_{t,j} \coloneqq 0$ for all $t \in \mathbb{Z}$ and $j \in [d]$
for $t \coloneqq 1$ **to** T **do**
    Calculate $\hat{X}^t$
    for $j \coloneqq 1$ **to** d **do**
        $W_t \coloneqq \{u \in [t-1] | \sum_{v=u+1}^{t-1} f_{v,j}(0) \leq \beta_j < \sum_{v=u+1}^{t} f_{v,j}(0)\}$
        $x_{t,j}^{\mathcal{D}} \coloneqq x_{t-1,j}^{\mathcal{D}} - \sum_{u \in W_t} w_{u,j}$
        if $x_{t,j}^{\mathcal{D}} < \hat{x}_{t,j}^t$ **then**
            $w_{t,j} \coloneqq \hat{x}_{t,j}^t - x_{t,j}^{\mathcal{D}}$
            $x_{t,j}^{\mathcal{D}} \coloneqq x_{t,j}^t$

Analysis of the competitive ratio. The proof of Theorem 3.7 is quite similar to that of Theorem 3.6. In the definition of the load-dependent operating cost (equation (3.6)), we can just replace f_j with $f_{t,j}$. Lemma 3.5 still holds since it is based on the invariant $x_{t,j}^{\mathcal{D}} \geq \hat{x}_{t,j}^t$ that is still satisfied. Therefore, we only have to adapt Lemma 3.6 that estimates the switching and idle operating costs. The cost of a block is now bounded by $C(A_{j,i}) \leq 2\beta_j + f_{s,j}(0)$ where s is the first time slot of the block $A_{j,i}$. This term can be estimated by $C(A_{j,i}) \leq \beta_j(2 + c_j)$ with $c_j \coloneqq \max_{t \in [T]} f_{t,j}(0)/\beta_j$. We define the special time slots like in the proof of Lemma 3.5, conduct the induction and finally obtain the inequality

$$\sum_{i=1}^{n_j} C(A_{j,i}) \leq (2 + c_j) \cdot C(\hat{X}^T).$$

In contrast to the original lemma, we have the factor $2 + c_j$ instead of 2. This results in a competitive ratio of $2d + 1 + \sum_{j=1}^{d} c_j = 2d + 1 + c(\mathcal{I})$.

Improving the competitive ratio. It is possible to modify the problem instance such that the constant $c(\mathcal{I})$ gets arbitrarily small. For this, we split each time slot into $\tilde{n}_t$ sub time slots to allow intermediate state changes. The arriving operating cost functions $f_{t,j}$ are distributed evenly to these sub time slots. Then we execute $\tilde{n}_t$ time slots of algorithm $\mathcal{D}$ on the modified problem instance. Given the server configurations calculated by $\mathcal{D}$, we choose the one that minimizes the operating cost. To achieve a competitive ratio of $2d + 1 + \epsilon$, we set the number of sub time slots to

$$\tilde{n}_t \coloneqq \frac{d}{\epsilon} \cdot \max_{j \in [d]} \frac{f_{t,j}(0)}{\beta_j}.$$

Formally, the modified problem instance $\tilde{\mathcal{I}} \coloneqq (\tilde{T}, d, \boldsymbol{m}, \boldsymbol{\beta}, \tilde{F}, \tilde{\Lambda})$ is defined as follows. Each time slot t of the original problem instance $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$ is represented

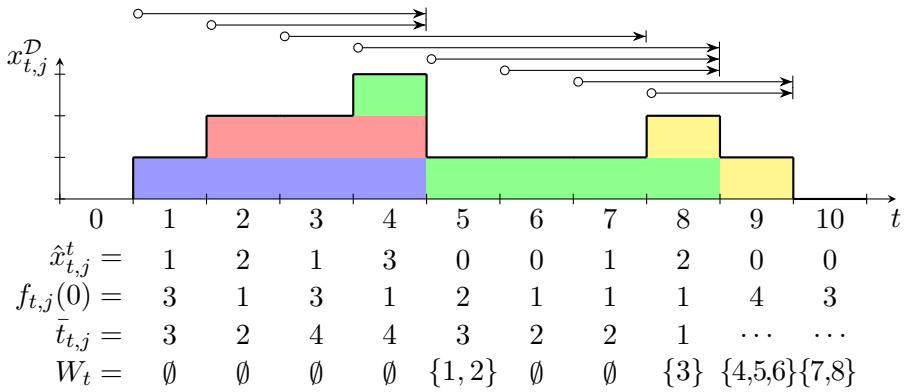

Figure 3.8: Visualization of algorithm $\mathcal{D}$ for one specific server type j with $\beta_j = 5$. The plot shows the number of active servers $x^{\mathcal{D}}_{t,j}$. The colors indicate the running time of each server. The values of $\hat{x}^t_{t,j}$ (that are needed to determine when a server has to be powered up) and the idle operating costs $f_{t,j}(0)$ as well as the resulting values of $\bar{t}_{t,j}$ and W_t are shown below the plot. The running time $\bar{t}_{t,j}$ of a server that is powered up at time slot t is indicated by the arrows, e.g., a server that is powered up at time slot $t = 2$ runs for $\bar{t}_{2,j} = 2$ *additional* time slots, so it is powered down at the end of time slot $t + \bar{t}_{t,j} = 4$. The values $\bar{t}_{t,j}$ are the maximal number of time slots *after* t such that the accumulated idle operating cost does not exceed the switching cost β_j. For example, $\bar{t}_{2,j} = 2$, because $f_{3,j}(0) + f_{4,j}(0) = 3+1 = 4 \leq \beta_j = 5$, but $f_{3,j}(0) + f_{4,j}(0) + f_{5,j}(0) = 6 > \beta_j$. At time slot t, the servers that were powered up at time $u \in W_t$ are shut down, e.g., $W_5 = \{1, 2\}$, so both the blue and red server are powered down at time slot 5. For $t \geq 9$, the values of $\bar{t}_{t,j}$ are not known yet, because they depend on $f_{11,j}(0)$.

by $\tilde{n}_t$ time slots in $\tilde{\mathcal{I}}$ given by $U(t) \coloneqq [u + 1 : u + \tilde{n}_t]$ with $u = \sum_{t'=1}^{t-1} \tilde{n}_{t'}$. We denote time slots in $\mathcal{I}$ by the symbols t or t', while u and u' indicate time slots in $\tilde{\mathcal{I}}$. The total number of time slots in $\tilde{\mathcal{I}}$ is equal to $\tilde{T} \coloneqq \sum_{t=1}^{T} \tilde{n}_t$. Given a time slot $u \in [\tilde{T}]$, let $t = U^{-1}(u)$ be the corresponding time slot in the original problem instance $\mathcal{I}$, i.e., $u \in U(t)$. The operating cost functions of $\tilde{\mathcal{I}}$ are defined as $\tilde{f}_{u,j}(z) \coloneqq \frac{1}{\tilde{n}_t} f_{t,j}(z)$ with $t = U^{-1}(u)$. The job volumes remain unchanged, so $\tilde{\lambda}_u \coloneqq \lambda_{U^{-1}(u)}$.

The pseudocode below summarizes how the new algorithm $\mathcal{E}$ works[3]. In line 6, we choose the server configuration $\boldsymbol{x}^{\mathcal{D}}_u$ with $u \in U(t)$ that minimizes the operating cost $\tilde{g}_u$.

Theorem 3.8 [AQ21e, Theorem 3.6]**.** *Algorithm $\mathcal{E}$ solves the heterogeneous data center right-sizing problem with time-dependent operating cost functions and achieves a competitive ratio of $2d + 1 + \epsilon$ for any $\epsilon > 0$.*

[3] In the original paper [AQ21e], this algorithm was called algorithm $\mathcal{C}$.

Algorithm 9 Algorithm $\mathcal{E}$ [AQ21e]

Input: $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$
Output: $X^{\mathcal{E}} = (\boldsymbol{x}_1^{\mathcal{E}}, \dots, \boldsymbol{x}_T^{\mathcal{E}})$
1: Initialize algorithm $\mathcal{D}$
2: **for** $t \coloneqq 1$ **to** T **do**
3: $\quad \tilde{n}_t \coloneqq d/\epsilon \cdot \max_{j\in[d]} f_{t,j}(0)/\beta_j$
4: $\quad$ Create the next $\tilde{n}_t$ time slots of the modified problem instance $\tilde{\mathcal{I}}$
5: $\quad$ Execute $\tilde{n}_t$ time slots of algorithm $\mathcal{D}$
6: $\quad \boldsymbol{x}_t^{\mathcal{E}} \coloneqq \boldsymbol{x}_{\mu(t)}^{\mathcal{D}}$ with $\mu(t) \coloneqq \operatorname{argmin}_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_u^{\mathcal{D}})$

Proof idea. Let $C^{\mathcal{J}}(X)$ be the cost of the schedule X regarding the problem instance $\mathcal{J} \in \{\mathcal{I}, \tilde{\mathcal{I}}\}$. Furthermore, let $X^*_{\mathcal{J}}$ be an optimal schedule for $\mathcal{J}$. It can be shown that

$$C^{\mathcal{I}}(X^{\mathcal{E}}) \le C^{\tilde{\mathcal{I}}}(X^{\mathcal{D}}),$$

so the total cost of $\mathcal{E}$ cannot be greater than the total cost of $\mathcal{D}$ [AQ21e, Lemma 3.5]. By Theorem 3.7, algorithm $\mathcal{D}$ achieves a competitive ratio of $2d + 1 + c(\tilde{\mathcal{I}})$, i.e.,

$$C^{\tilde{\mathcal{I}}}(X^{\mathcal{D}}) \le (2d + 1 + c(\tilde{\mathcal{I}})) \cdot C^{\tilde{\mathcal{I}}}(X^*_{\tilde{\mathcal{I}}}).$$

By the definition of $\tilde{n}_t = \frac{d}{\epsilon} \cdot \max_{j\in[d]} \frac{f_{t,j(0)}}{\beta_j}$, we get

$$c(\tilde{\mathcal{I}}) = \sum_{j=1}^{d} \max_{u\in\tilde{T}} \frac{\tilde{f}_{u,j}(0)}{\beta_j} = \sum_{j=1}^{d} \max_{t\in T} \frac{f_{t,j}(0)}{\tilde{n}_t \beta_j} \le \sum_{j=1}^{d} \max_{t\in T} \frac{\epsilon}{d} = \epsilon.$$

Finally, we notice that an optimal schedule for $\mathcal{I}$ can be easily converted to a schedule for $\tilde{\mathcal{I}}$, so

$$C^{\tilde{\mathcal{I}}}(X^*_{\tilde{\mathcal{I}}}) \le C^{\mathcal{I}}(X^*_{\mathcal{I}}).$$

All in all, we get $C^{\mathcal{I}}(X^{\mathcal{E}}) \le (2d + 1 + \epsilon) \cdot C^{\mathcal{I}}(X^*_{\mathcal{I}})$. □

3.2.4 Lower bound for constant operating costs

In the previous sections, we have developed several online algorithms solving different variations of the heterogeneous data center right-sizing problem. In this section, we will derive a lower bound for deterministic online algorithms showing that no algorithm can beat a certain competitive ratio.

Theorem 3.9 [AQ21e, Theorem 3]**.** *There is no deterministic online algorithm that achieves a competitive ratio smaller than* $2d$ *for the heterogeneous data center right-sizing problem with constant operating costs, no inefficient servers and equal computational power for all server types.*

This lower bound does not only holds for algorithm $\mathcal{A}$ presented in Section 3.2.1, but also for the algorithms $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$ presented in Sections 3.2.2 and 3.2.3, as they solve more general problems. Algorithm $\mathcal{A}$ achieves the optimal competitive ratio of $2d$. Furthermore, algorithms $\mathcal{C}$ and $\mathcal{E}$ with a competitive ratio of $2d+1$ and $2d+1+\epsilon$, respectively, are almost optimal.

To prove Theorem 3.9, we consider the following problem instance with d server types. There is $m_j = 1$ server of each type. The switching and operating costs are given by $\beta_j = N^{2j}$ and $r_j \coloneqq 1/N^{2j}$ where N is a large number that we will define later. Let $\mathcal{A}$ be an arbitrary deterministic online algorithm generating the schedule $X^{\mathcal{A}}$. Whenever $\mathcal{A}$ has an active server, the adversary sends no jobs ($\lambda_t = 0$). If all servers of $\mathcal{A}$ are inactive, then the adversary sends a job ($\lambda_t = 1$). The following lemma presents an important intermediate result.

Lemma 3.7 [AQ21c, Lemma 9]**.** *Let $k \in [d]$. If $X^{\mathcal{A}}$ only uses servers of type lower than or equal to k and if the cost of $\mathcal{A}$ is at least $C(X^{\mathcal{A}}) \geq N\beta_k$, then the cost of $\mathcal{A}$ is at least*

$$C(X^{\mathcal{A}}) \geq (2k - \epsilon_k) \cdot C(X^*)$$

with $\epsilon_k = 9k^2/N$ and $N \geq 6k$.

Proof idea. The statement is proven by induction over k. The base case $k = 1$ is similar to the well known Ski-Rental problem. For $k \geq 2$, we divide the schedule $X^{\mathcal{A}}$ into phases $L_0, K_1, L_1, \ldots, K_n, L_n$. W.l.o.g., we assume that $\mathcal{A}$ has never two or more active servers. In the phases K_i (also called K-phases), the server type k is used exactly once, so a server of type k is powered up at the beginning of the phase and powered down at the end of it. During the phases L_i (also called L-phases), algorithm $\mathcal{A}$ only uses server types smaller than k. It is possible that an L-phase is empty if a server of type k is powered down and up immediately thereafter. The L-phases are divided into *short* and *long* L-phases. If the cost during L_i is smaller than β_k/N, then L_i is a *short* L-phase, otherwise it is *long*.

We use two strategies to derive an upper bound for the cost of an optimal schedule (see Figure 3.9). In the first strategy, a server of type k is powered up at $t = 1$ and runs for the whole time. Only for K-phases where the total operating cost exceeds the switching cost, the server is switched to the inactive state from the second time slot of the phase up to its end, since this is cheaper than keeping the server active.

The behavior in second strategy depends on the phases. For the K-phases, server type 1 runs for exactly one time slot. This is sufficient because only for the first time slot of a K-phase $\lambda_t = 1$ holds. In the short L-phases, we behave like algorithm $\mathcal{A}$. Since these phases are "short", the portion of the total cost is small. The long L-phases fulfill all conditions to apply the induction hypothesis (i.e., Lemma 3.7 for $k-1$), as algorithm $\mathcal{A}$ only uses servers whose types are smaller than or equal to $k-1$ and its cost is at least $\beta_k/N = N\beta_{k-1}$ during this phase. The induction hypothesis implicitly gives a strategy whose cost is at most $1/\alpha$ times the cost of $\mathcal{A}$ during this phase where $\alpha = 2k - 2 - \epsilon_{k-1}$.

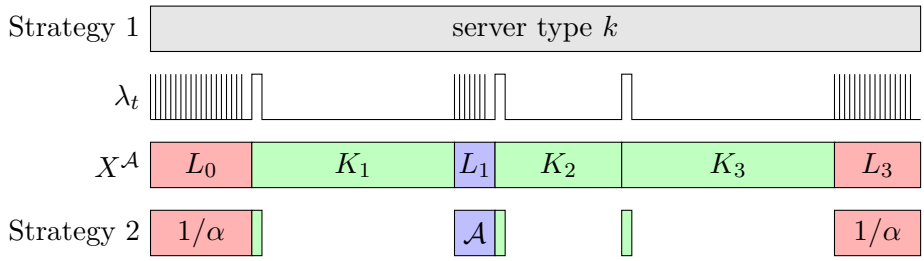

Figure 3.9: Visualization of the two strategies used to determine an upper bound for the cost of an optimal schedule. The arriving job volumes λ_t and the schedule of algorithm $\mathcal{A}$ are shown in the middle. Long L-phases are marked in red, short L-phases are highlighted in blue. Note that L_2 is a short L-phase with zero length. Strategy 1 starts a server of type k and keeps it active for the whole time. During the short L-phases, strategy 2 exactly behaves like algorithm $\mathcal{A}$. For the long L-phases, there exists a solution that only causes $1/\alpha$ of the cost of algorithm $\mathcal{A}$. The small green rectangles in strategy 2 indicate that server type 1 runs for exactly one time slot.

We want to find a lower bound for the competitive ratio $\frac{C(X^{\mathcal{A}})}{C(X^*)}$. The cost of $\mathcal{A}$ is split into the cost during the K- and L-phases denoted by $C_K(X^{\mathcal{A}})$ and $C_L(X^{\mathcal{A}})$, respectively. For the K-phases, $C(X^*)$ is estimated by the first strategy. The cost of the first strategy during the L-phases is negligible, since operating server type k only causes a cost of at most $1/N^2$ times the cost of $\mathcal{A}$ during the L-phases. In contrast to the first strategy, the online algorithm has to pay the switching cost β_k for each K-phase. Similar to the Ski-Rental problem, this results in a competitive ratio of 2 regarding the K-phases. Formally, we get

$$\frac{C_K(X^{\mathcal{A}})}{C(X^*)} \geq 2 - \mathcal{O}\left(\frac{k}{N}\right).$$

The term $\mathcal{O}(k/N)$ results from the small cost of X^* during the L-phases.

To estimate $C_L(X^{\mathcal{A}})/C(X^*)$, we use the second strategy. Here, the cost of X^* during the K-phases and short L-phases are negligible. For the long L-phases, we can use the induction hypothesis resulting in a competitive ratio of α. Formally, we get

$$\frac{C_L(X^{\mathcal{A}})}{C(X^*)} \geq \alpha - \mathcal{O}\left(\frac{k}{N}\right).$$

Adding both inequalities and using the definitions of α and ϵ_k finally gives us

$$\frac{C(X^{\mathcal{A}})}{C(X^*)} \geq 2k - \epsilon_k.$$

A complete and precise calculation of the quotient $\frac{C(X^{\mathcal{A}})}{C(X^*)}$ (including the estimation of the small terms) can be found in [AQ21a]. □

Now, we are able to proof Theorem 3.9.

Proof idea of Theorem 3.9. Assume that $\mathcal{A}$ is a $(2d-\epsilon)$-competitive deterministic online algorithm. Let $N := \max\{6d, \lceil 9k^2/\epsilon + 1\rceil\}$. We construct a workload as described above until the cost of $\mathcal{A}$ exceeds $N\beta_d$. This is always the case at some point, since $\mathcal{A}$ has to pay either a switching or an operating cost for each time slot and both of them are greater than zero. By Lemma 3.7 with $k = d$, we get

$$C(X^{\mathcal{A}}) \geq (2d - \epsilon_d) \cdot C(X^*) > (2d - \epsilon) \cdot C(X^*).$$

For the last step, we use the definitions of ϵ_d and N. The final result is a contradiction to the assumption that $\mathcal{A}$ is $(2d - \epsilon)$-competitive. □

In the proof of Theorem 3.9, we only use one server of each type. Therefore, we get a lower bound of $2m$ for a data center with m unique servers. We summarize this result in the following corollary.

Corollary 3.2 [AQ21a, Corollary 19]**.** *Given a data center with m unique non-inefficient servers with equal computational power and constant operating costs, there is no deterministic online algorithm that achieves a competitive ratio smaller than $2m$.*

3.2.5 Lower bound for arbitrary convex operating cost functions

At the beginning of this chapter, we claimed that no deterministic online algorithm handling arbitrary convex operating cost functions g_t can achieve a sub-exponential competitive ratio. In this setting, g_t is not restricted to the form given by equation (3.2), but it can be an arbitrary convex function. In the following, we want to prove this claim.

Theorem 3.10. *There is no deterministic online algorithm for the heterogeneous data center right-sizing problem with arbitrary convex operating cost functions g_t that achieves a competitive ratio smaller than $2^d + \frac{2^d-1}{d}$.*

The proof of this theorem was not published yet. Therefore, the complete proof is shown below. A weaker, but still exponential lower bound of $\frac{2^d-1}{d}$ was mentioned in [AQ21c] and [AQ21e]. For a homogeneous data center with only one server type (i.e., $d = 1$), Theorem 3.10 gives us a lower bound of 3 which is tight by Theorem 2.2 (showing the 3-competitiveness of the LCP algorithm).

Proof of Theorem 3.10. We consider a data center with $m_j = 1$ server of each type. To simplify the analysis, the switching cost is paid for both power-up and power-down operations. Switching the state of any server always causes a cost of $\beta_j = 1$. We use 2^d different operating cost functions denoted by $h_{\boldsymbol{y}}$ with $\boldsymbol{y} \in \{0,1\}^d$. The functions are defined as

$$h_{\boldsymbol{y}}(\boldsymbol{x}) = \begin{cases} \epsilon & \text{if } \boldsymbol{x} = \boldsymbol{y} \\ 0 & \text{else} \end{cases}$$

with $\epsilon > 0$.

Let $\mathcal{A}$ be an arbitrary deterministic online algorithm with the schedule $X^{\mathcal{A}}$. At time slot t, the adversary will send the function $h_{\boldsymbol{y}}$ with $\boldsymbol{y} = \boldsymbol{x}_t^{\mathcal{A}}$, so if the online algorithm will keep its state, it has to pay the operating cost ϵ. Let S be the number of time slots where algorithm $\mathcal{A}$ changes the state of at least one server. Hence, the total operating cost of algorithm $\mathcal{A}$ is given by $(T - S)\epsilon$. The switching cost of $\mathcal{A}$ is at least S, as switching the state of any server causes a cost of $\beta_j = 1$. Therefore, the total cost of algorithm $\mathcal{A}$ is at least

$$C(X^{\mathcal{A}}) \geq T\epsilon + S(1 - \epsilon). \tag{3.9}$$

If algorithm $\mathcal{A}$ switches its state very often, a good offline strategy is to stay at one state for the whole workload. There exists one state whose cost is at most $T\epsilon/2^d$. Reaching this state and going back to $\mathbf{0}$ at the end of the time horizon, produces a switching cost of at most $2d$. If $\mathcal{A}$ performs only a few state changes, a good offline strategy is to switch to the state that $\mathcal{A}$ does not visit for the longest time. So after $2^d - 1$ state changes of $\mathcal{A}$, the offline algorithm will switch its state causing a cost of at most d. Again, we have to add $2d$ for the beginning and the end of the workload. Therefore, the cost of the optimal offline solution is at most

$$C(X^*) \leq \min \begin{cases} T\epsilon/2^d + 2d \\ \frac{d}{2^d - 1} \cdot S + 2d \end{cases} \tag{3.10}$$

We want to find a lower bound for the competitive ratio given by $\frac{C(X^{\mathcal{A}})}{C(X^*)}$. Let $\kappa \coloneqq \frac{d}{2^d - 1}$ be the coefficient of S in equation (3.10) and let $\nu \coloneqq 2^d$ be the number of states. We distinguish whether or not the inequality

$$T\epsilon \geq S\kappa\nu + 2d\nu + 2d/\kappa \cdot (1 - \epsilon) \tag{3.11}$$

holds.

If it holds (**case 1**), we estimate the cost of the optimal solution with $C(X^*) \leq S\kappa + 2d$ and get

$$\begin{aligned} \frac{C(X^{\mathcal{A}})}{C(X^*)} &\overset{(3.9)}{\underset{(3.10)}{\geq}} \frac{T\epsilon + S(1 - \epsilon)}{S\kappa + 2d} \\ &\overset{(3.11)}{\geq} \frac{S\kappa\nu + 2d\nu + 2d/\kappa \cdot (1 - \epsilon) + S(1 - \epsilon)}{S\kappa + 2d} \\ &= \nu + \frac{1 - \epsilon}{\kappa}. \end{aligned}$$

In the second step, we use inequality (3.11) to estimate $T\epsilon$. In the last step, we just simplify the term. If ϵ goes to 0, we get

$$\lim_{\epsilon \to 0} \frac{C(X^{\mathcal{A}})}{C(X^*)} \geq \nu + \frac{1}{\kappa} = 2^d + \frac{2^d - 1}{d}.$$

In case 2, we have $T\epsilon < S\kappa\nu + 2d\nu + 2d/\kappa \cdot (1-\epsilon)$ which is equivalent to

$$S > T\epsilon/(\kappa\nu) - 2d/\kappa - 2d/(\kappa^2\nu) \cdot (1-\epsilon). \tag{3.12}$$

We estimate the cost of the optimal solution with $C(X^*) \leq T\epsilon/\nu + 2d$ and get

$$\begin{aligned}
\frac{C(X^{\mathcal{A}})}{C(X^*)} &\overset{\substack{(3.9)\\(3.10)}}{\geq} \frac{T\epsilon + S(1-\epsilon)}{T\epsilon/\nu + 2d} \\
&\overset{(3.12)}{\geq} \frac{T\epsilon + \Big(T\epsilon/(\kappa\nu) - 2d/\kappa - 2d/(\kappa^2\nu) \cdot (1-\epsilon)\Big) \cdot (1-\epsilon)}{T\epsilon/\nu + 2d} \\
&= \nu + \frac{1-\epsilon}{\kappa} - \frac{2d\nu + 4d/\kappa + 2d/(\kappa^2\nu) \cdot (1-\epsilon)}{T\epsilon/\nu + 2d} \cdot (1-\epsilon)
\end{aligned} \tag{3.13}$$

By setting $T \coloneqq \frac{1}{\epsilon^2}$, we get $\lim_{\epsilon\to 0} T\epsilon = \infty$. Since the ν and κ only depend on d, the whole subtrahend in (3.13) converges to 0. Therefore, we get

$$\lim_{\epsilon\to 0} \frac{C(X^{\mathcal{A}})}{C(X^*)} \geq \nu + \frac{1}{\kappa} = 2^d + \frac{2^d - 1}{d}$$

which is the same value as in case 1. □

4 Conclusions

Energy conversation in data centers is an important issue for both economical and ecological reasons. In this thesis, we examined the data center right-sizing problem where idle servers can be powered down to save energy. In contrast to related work such as [LWAT13, BGK+15, Sel20], we studied the discrete setting where the number of active servers must be integral. Thereby, we gain truly feasible solutions. We analyzed both the offline and the online version of the data center right-sizing problem. In the following, we conclude our results and present some open questions for future work.

4.1 Results

For homogeneous data centers (Chapter 2), we presented an optimal offline algorithm that runs in $\mathcal{O}(T \log m)$ time. For the online setting, we developed a 3-competitive deterministic and a 2-competitive randomized algorithm. We showed that no algorithm can achieve better results, so our online algorithms are optimal. Furthermore, we proved that no algorithm for the fractional setting exists whose competitive ratio is smaller than 2. All lower bounds still hold for the restricted model introduced by Lin et al. [LWAT11a]. Finally, we showed that the lower bounds do not change if the online algorithm has a prediction window with constant length.

In Chapter 3, we analyzed heterogeneous data centers with d different server types. The operating cost of server type j at time t is modeled by a non-negative increasing convex function $f_{t,j}$ of the load. First, we analyzed the offline version. We presented an optimal offline algorithm that runs in $\mathcal{O}(T \cdot \prod_{j=1}^{d} m_j)$ time. However, this runtime is only pseudo-polynomial since the encoding length of the problem depends logarithmically on m_j. We developed a $(1+\epsilon)$-approximation algorithm that runs in $\mathcal{O}(T \cdot \epsilon^{-d} \prod_{j=1}^{d} \log m_j)$ time which is polynomial if d is a constant. Our algorithm still works if the number of available servers depends on time.

Afterwards, we analyzed the online version. First, we studied the simplified problem where the operating cost is time- and load-independent, i.e., $f_{t,j}(z) = r_j = \text{const}$. Furthermore, we had to exclude inefficient server types and assumed that all servers have the same computational power ($z_j^{\max} = 1$). We developed a $2d$-competitive deterministic online algorithm for this setting and showed a lower bound of $2d$. Hence, our algorithm is optimal. We were able to improve the competitive ratio by randomization and obtained a $1.582d$-competitive randomized algorithm. For load-dependent, but time-independent operating cost functions $f_{t,j}(z) = f_j(z)$, we developed a $(2d+1)$-competitive deterministic online algorithm. Each server type had its own computational power and the exclusion of inefficient server types was no longer necessary. We extended our algorithm such

that it can handle time-dependent operating cost functions and achieved a competitive ratio of $2d + 1 + \epsilon$ for any $\epsilon > 0$. Note that the lower bound of $2d$ still holds for these generalizations, so our online algorithms are almost optimal. Finally, we proved that arbitrary convex operating cost functions g_t lead to an at least exponential competitive ratio.

4.2 Open questions

Although we obtained several new results, there are still many open questions, especially regarding the discrete setting of the data center right-sizing problem. The most important one is the missing NP-hardness proof for heterogeneous data centers with a constant number of server types. For homogeneous data centers we presented a polynomial time algorithm, however, extending this algorithm to $d \geq 2$ dimensions does not lead to optimal schedules any more. The polynomial-time algorithm by [Alb19] only works for load-independent operating costs and cannot be extended to convex operating cost functions. On the other hand, the NP-hardness proof in [Sch18] requires multiple sleep states and cannot be adapted to our problem.

Our results for deterministic online algorithms are almost optimal. We gained an upper bound of $2d + 1 + \epsilon$ and a lower bound of $2d$, so there is only little room for improvements. However, for randomized algorithms there is still a large gap between the best known upper and lower bound. For time- and load-*independent* operating cost functions, we developed a $1.582d$-competitive online algorithm, while the best known lower bound is $\frac{e}{e-1} \approx 1.582$ given by the Ski-Rental problem [KKR01]. The randomization of our online algorithm only affects the running time of a server, but not the selection of the server types itself. Therefore, it might be possible to achieve a sublinear competitive ratio by choosing the server types randomly.

Another open question is whether, and if so, how our algorithms for load-dependent operating costs can be randomized to improve the competitive ratio. We tried to randomize the algorithm for time-independent operating cost functions f_j presented in Section 3.2.2 by randomizing the running time of a server, however, we were not able to prove a better competitive ratio.

We have shown that the competitive ratio of a deterministic algorithm for general operating cost functions g_t is at least exponential. It is an open question if this still holds for randomized algorithms. Since there is a $(d+1)$-competitive algorithm in the fractional setting by [Sel20], it might be possible to achieve a linear competitive ratio by randomly rounding the fractional solution properly.

Convex function chasing on an arbitrary metric has a lower bound of d, so the $(d+1)$-competitive online algorithm by [Sel20] is almost optimal for this general case. The proof of the lower bound uses the maximum metric. However, for the Manhattan metric which is related to the data center right-sizing problem, the best known lower bound is $\Omega(\log d)$ [ABC+19]. It would be interesting if one can improve the lower bound or the competitive ratio for this metric.

Another generalization are time-dependent switching costs $\beta_{t,j}$. One could investigate whether the binary search approach of our polynomial time offline algorithm for homogeneous data centers still leads to an optimal schedule. Furthermore, it is unclear if the approximation algorithm for heterogeneous data centers still produces a $(1+\epsilon)$-approximation. For the online version further restrictions are needed, since in general there is no competitive algorithm for time-dependent switching costs.

In this work, we only consider the two-state problem, where each server has an active and exactly one inactive state. Multiple sleep states were, for example, analyzed by [ISG03, AIS08] for a single machine in the online setting and by [Alb19] for a heterogeneous data center in the offline setting. It would be interesting to combine the results from [ISG03, AIS08] with our algorithms for homogeneous and heterogeneous data centers. Regarding the offline problem, it is unclear if the right-sizing problem of a *homogeneous* data center with *constant* operating costs and *multiple* sleep states can be solved in polynomial time. For a *heterogeneous* data center with two states, this is the case [Alb19], but for an unbounded number of sleep states and server types the problem is NP-hard [Sch18].

Heterogeneity often results from different architectures like GPUs and CPUs. The model introduced in Chapter 3 can handle multiple server types with different operating cost functions and computational speeds. However, we assumed that the incoming job volume is homogeneous. In reality, this is not the case. There might be branchless jobs with massive parallel calculations designed for the GPUs that would run quite slow on a pure CPU server. Thus, the processing time does not only depend on the server type, but also on the job types assigned to a server. A single one-dimensional operating cost function per server type is not sufficient to model this situation. Given e different job types, one possibility is to use e-dimensional operating cost functions $f_{t,j}(z_1, \dots, z_e)$ where z_k denotes the job volume of type k processed by a single server of type j at time t. Regarding the online problem, it would be interesting to see how the upper and lower bounds change when handling, for example, $e = 2$ job types.

A prediction window with constant length does not improve the competitive ratio of any online algorithm as shown in Section 2.4.4, since the time slots can be made arbitrarily short such that the advantage of the prediction window is nullified. Thus, one may consider a prediction window whose length depends on the operating and switching costs. For example, given a fixed fraction $\eta > 0$, at time t the online algorithm knows the operating cost functions for all $t' \geq t$ satisfying the inequality $\sum_{u=t}^{t'} f_{u,j}(0) \leq \eta \cdot \beta_j$. In other words, the algorithm knows the next $t' - t$ operating cost functions, if their accumulated idle operating cost does not exceed the fraction η of the switching cost.

In practice, more information about the job distribution is known. For example, a data center usually receives more jobs during the day than at night, and at the weekends the workload is usually lower than on working days. There are publications investigating the data center right-sizing problem under stochastic models like [CAW+15, LQL18], however, they only analyze the fractional setting. Regarding the discrete setting, nothing is known yet.

Another large open research area is the combination of scheduling problems with the data center right-sizing problem. In this case, we do not have a job volume λ_t that can be distributed arbitrarily, but we have single jobs with different processing times and deadlines. There are some publications handling the offline problem, e.g., [BCD12, DGH+13, AGKK20], however, not much is known yet regarding the online problem.

In this PhD thesis, we analyzed the data-center right-sizing problem from a theoretical point of view proving competitive ratios, lower bounds and approximation factors. From a practical point of view, it would be interesting to see how the algorithms perform on real data centers. Some experimental results of the algorithms developed in this work can be found in the bachelor thesis of Jonas Hübotter [Hüb21]. However, there is still room for further investigation.

Bibliography

[AA14] Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. *ACM Transactions on Algorithms (TALG)*, 10(2):9, 2014.

[ABC+19] CJ Argue, Sébastien Bubeck, Michael B Cohen, Anupam Gupta, and Yin Tat Lee. A nearly-linear bound for chasing nested convex bodies. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 117–122. SIAM, 2019.

[ABF+99] Yossi Azar, Yair Bartal, Esteban Feuerstein, Amos Fiat, Stefano Leonardi, and Adi Rosén. On capital investment. *Algorithmica*, 25(1):22–36, 1999.

[ABL+13] Lachlan Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT'13)*, pages 741–763, 2013.

[ABL+15] Lachlan L. H. Andrew, Siddharth Barman, Katrina Ligett, Minghong Lin, Adam Meyerson, Alan Roytman, and Adam Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret, 2015. arXiv:1508.03769.

[ABN+16] Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, Kevin Schewior, and Michele Scquizzato. Chasing convex bodies and functions. In *Proceedings of the 12th Latin American Symposium on Theoretical Informatics (LATIN'16)*, pages 68–81. Springer, 2016.

[AE15] Anders SG Andrae and Tomas Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report No. UCB/EECS-2009-282, EECS Department, University of California, Berkeley, 2009.

[AGGT20] CJ Argue, Anupam Gupta, Guru Guruganesh, and Ziye Tang. Chasing convex bodies with linear competitive ratio. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1519–1524. SIAM, 2020.

[AGKK20] Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. Parallel machine scheduling to minimize energy consumption. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2758–2769. SIAM, 2020.

[AHO15] Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1102–1113. SIAM, 2015.

[AIS08] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. *SIAM Journal on Computing*, 37(5):1499–1516, 2008.

[Alb03] Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1):3–26, 2003.

[Alb19] Susanne Albers. On energy conservation in data centers. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–26, 2019.

[ALW10] Lachlan LH Andrew, Minghong Lin, and Adam Wierman. Optimality, fairness, and robustness in speed scaling designs. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 37–48, 2010.

[AQ18a] Susanne Albers and Jens Quedenfeld. Optimal algorithms for right-sizing data centers. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA'18)*, pages 363–372. ACM, 2018.

[AQ18b] Susanne Albers and Jens Quedenfeld. Optimal algorithms for right-sizing data centers—extended version, 2018. arXiv:1807.05112.

[AQ21a] Susanne Albers and Jens Quedenfeld. Algorithms for energy conservation in heterogeneous data centers. *Theoretical Computer Science*, 896:111–131, 2021.

[AQ21b] Susanne Albers and Jens Quedenfeld. Algorithms for energy conservation in heterogeneous data centers, 2021. arXiv:2107.14672.

[AQ21c] Susanne Albers and Jens Quedenfeld. Algorithms for energy conservation in heterogeneous data centers. In *Algorithms and Complexity - 12th International Conference, CIAC 2021*. Springer, 2021.

[AQ21d] Susanne Albers and Jens Quedenfeld. Algorithms for right-sizing heterogeneous data centers, 2021. arXiv:2107.14692.

[AQ21e] Susanne Albers and Jens Quedenfeld. Algorithms for right-sizing heterogeneous data centers. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'21)*, pages 48–58. ACM, 2021.

[AS17] Antonios Antoniadis and Kevin Schewior. A tight lower bound for online convex optimization with switching costs. In *International Workshop on Approximation and Online Algorithms*, pages 164–175. Springer, 2017.

[BAC+08] Peter Bodik, Michael Paul Armbrust, Kevin Canini, Armando Fox, Michael Jordan, and David A Patterson. A case for adaptive datacenters to conserve energy and improve reliability. *University of California at Berkeley, Tech. Rep. UCB/EECS-2008-127*, 2008.

[Bap06] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 364–367, 2006.

[Baw16] Tom Bawden. Global warming: Data centres to consume three times as much energy in next decade, experts warn, 2016. URL: http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html [last downloaded 2021-11-26].

[BBE+18] Nikhil Bansal, Martin Böhm, Marek Eliáš, Grigorios Koumoutsos, and Seeun William Umboh. Nested convex bodies are chaseable. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1253–1260. SIAM, 2018.

[BBMN15] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *Journal of the ACM (JACM)*, 62(5):1–49, 2015.

[BCD12] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial-time algorithms for minimum energy scheduling. *ACM Transactions on Algorithms (TALG)*, 8(3):1–29, 2012.

[BCL+18] Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, James R Lee, and Aleksander Mądry. K-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 3–16, 2018.

[BCLL21] Sébastien Bubeck, Michael B Cohen, James R Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. *SIAM Journal on Computing*, 50(3):909–923, 2021.

[BD97] Luca Benini and Giovanni DeMicheli. *Dynamic power management: design techniques and CAD tools*. Springer Science & Business Media, 1997.

[BDBK+94] Shai Ben-David, Allan Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[BGK+15] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Cliff Stein. A 2-competitive algorithm for online convex optimization with switching costs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *LIPIcs*, pages 96–109. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[BH07] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[BKL+20] Sébastien Bubeck, Bo'az Klartag, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Chasing nested convex bodies nearly optimally. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1496–1508. SIAM, 2020.

[BLLS19] Sébastien Bubeck, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. Competitively chasing convex bodies. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 861–868, New York, NY, USA, 2019. Association for Computing Machinery.

[BLMN03] Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric ramsey-type phenomena. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, pages 463–472, 2003.

[BLS92] Allan Borodin, Nathan Linial, and Michael E Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM (JACM)*, 39(4):745–763, 1992.

[BLW15] Masoud Badiei, Na Li, and Adam Wierman. Online convex optimization with ramp constraints. In *54th IEEE Conference on Decision and Control (CDC)*, pages 6730–6736. IEEE, 2015.

[Bri07] Kenneth G Brill. The invisible crisis in the data center: The economic meltdown of moore's law. *white paper, Uptime Institute*, pages 2–5, 2007.

[CAW+15] Niangjun Chen, Anish Agarwal, Adam Wierman, Siddharth Barman, and Lachlan LH Andrew. Online convex optimization using predictions. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):191–204, 2015.

[CCL+16] Niangjun Chen, Joshua Comden, Zhenhua Liu, Anshul Gandhi, and Adam Wierman. Using predictions in online optimization: Looking forward with an eye on the past. *ACM SIGMETRICS Performance Evaluation Review*, 44(1):193–206, 2016.

[CFH+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI 2005)*, pages 273–286. USENIX, 2005.

[CGW18] Niangjun Chen, Gautam Goel, and Adam Wierman. Smoothed online convex optimization in high dimensions via online balanced descent. *Proceedings of Machine Learning Research*, 75:1574–1594, 2018.

[Chu16] Sergei Chubanov. A polynomial-time descent method for separable convex optimization problems with linear constraints. *SIAM Journal on Optimization*, 26(1):856–889, 2016.

[Dam03] Peter Damaschke. Nearly optimal strategies for special cases of on-line capital investment. *Theoretical Computer Science*, 302(1-3):35–44, 2003.

[Das14] Puja Das. *Online convex optimization and its application to online portfolio selection*. PhD thesis, University of Minnesota, 2014.

[Dea14] Pierre Delforge and et al. Data center efficiency assessment, 2014. URL: https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf [last downloaded 2021-11-26].

[DGH+13] Erik D Demaine, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, Amin S Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. *Journal of Scheduling*, 16(2):151–160, 2013.

[DKP02] Etienne De Klerk and Dmitrii V Pasechnik. Approximation of the stability number of a graph via copositive programming. *SIAM Journal on Optimization*, 12(4):875–892, 2002.

[DWF16] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016.

[Enc21a] Wikipedia: The Free Encyclopedia. General-purpose computing on graphics processing units, 2021. URL: https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units [last downloaded 2021-11-26].

[Enc21b] Wikipedia: The Free Encyclopedia. Heterogeneous computing, 2021. URL: https://en.wikipedia.org/wiki/Heterogeneous_computing [last downloaded 2021-11-26].

[Ene20] Enerdata. Global energy statistical yearbook 2020, 2020. URL: https://yearbook.enerdata.net/electricity/electricity-domestic-consumption-data.html [last downloaded 2021-11-26].

[FI05] Hiroshi Fujiwara and Kazuo Iwama. Average-case competitive analyses for ski-rental problems. *Algorithmica*, 42(1):95–107, 2005.

[FKF16] Hiroshi Fujiwara, Takuma Kitano, and Toshihiro Fujito. On the best possible competitive ratio for the multislope ski-rental problem. *Journal of Combinatorial Optimization*, 31(2):463–490, 2016.

[FL93] Joel Friedman and Nathan Linial. On convex body chasing. *Discrete & Computational Geometry*, 9(1):293–321, 1993.

[GCW17] Gautam Goel, Niangjun Chen, and Adam Wierman. Thinking fast and slow: Optimization decomposition across timescales. In *IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1291–1298. IEEE, 2017.

[GHB11] Anshul Gandhi and Mor Harchol-Balter. How data center size impacts the effectiveness of dynamic power management. In *49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1164–1169. IEEE, 2011.

[GHBA10] Anshul Gandhi, Mor Harchol-Balter, and Ivo Adan. Server farms with setup costs. *Performance Evaluation*, 67(11):1123–1138, 2010.

[GJ79] Michael R Garey and David S Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GLLK79] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[GLSW19] Gautam Goel, Yiheng Lin, Haoyuan Sun, and Adam Wierman. Beyond online balanced descent: An optimal algorithm for smoothed online optimization. *Advances in Neural Information Processing Systems*, 32:1875–1885, 2019.

[GP13] Hadi Goudarzi and Massoud Pedram. Geographical load balancing for online service applications in distributed datacenters. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 351–358. IEEE, 2013.

[GW19] Gautam Goel and Adam Wierman. An online algorithm for smoothed regression and lqr control. *Proceedings of Machine Learning Research*, 89:2504–2513, 2019.

[GZ16] Yuxiang Gao and Peng Zhang. A survey of homogeneous and heterogeneous system architectures in high performance computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 170–175. IEEE, 2016.

[Ham08] James 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/ [last downloaded 2021-11-26].

[Hüb21] Jonas Hübotter. Implementation of algorithms for right-sizing data centers. Bachelor's thesis, Technical University of Munich, 2021.

[IM20] Leila Ismail and Huned Materwala. Computing server power modeling in a data center: Survey, taxonomy, and performance evaluation. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020.

[IP05] Sandy Irani and Kirk R Pruhs. Algorithmic problems in power management. *ACM Sigact News*, 36(2):63–76, 2005.

[ISG03] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):325–346, 2003.

[ISG07] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms (TALG)*, 3(4):41–es, 2007.

[JDV12] Vinay Joseph and Gustavo De Veciana. Jointly optimizing multi-user rate adaptation for video transport over wireless systems: Mean-fairness-variability tradeoffs. In *Proceedings of the IEEE INFOCOM 2012*, pages 567–575. IEEE, 2012.

[Kar92] Richard M Karp. On-line algorithms versus off-line algorithms: How much. In *Algorithms, Software, Architecture: Information Processing 92: Proceedings of the IFIP 12th World Computer Congress, Madrid, Spain, 7-11 September 1992*, volume 1, page 416. North-Holland, 1992.

[KG14] Seung-Jun Kim and Geogios B Giannakis. Real-time electricity pricing for demand response using online convex optimization. In *IEEE PES*

Innovative Smart Grid Technologies Conference, ISGT 2014, pages 1–5. IEEE, 2014.

[KKR01] Anna R Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgement and other stories about e/(e-1). In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pages 502–509, 2001.

[KLS10] Samir Khuller, Jian Li, and Barna Saha. Energy efficient scheduling via partial shutdown. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1360–1372. SIAM, 2010.

[KMRS88] Anna R Karlin, Mark S Manasse, Larry Rudolph, and Daniel D Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[KP77] Wook Hyun Kwon and Allan E Pearson. A modified quadratic cost problem and feedback stabilization of linear discrete time systems. *IEEE Transactions on Automatic Control*, 22(5):838–842, 1977.

[KP95] Elias Koutsoupias and Christos H Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.

[KT04] Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theoretical Computer Science*, 324(2-3):347–359, 2004.

[KYTM15] Taehwan Kim, Yisong Yue, Sarah Taylor, and Iain Matthews. A decision tree framework for spatiotemporal sequence prediction. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 577–586. ACM, 2015.

[LCA13] Tan Lu, Minghua Chen, and Lachlan LH Andrew. Simple and effective dynamic provisioning for power-proportional data centers. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1161–1171, 2013.

[Lee18] James R Lee. Fusible hsts and the randomized k-server conjecture. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 438–449. IEEE, 2018.

[LGW20] Yiheng Lin, Gautam Goel, and Adam Wierman. Online optimization with predictions and non-convex losses. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(1):1–32, 2020.

[LK11] Jian Li and Samir Khuller. Generalized machine activation problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 80–94. SIAM, 2011.

[LLC21] Google LLC. Google scholar, 2021. URL: https://scholar.google.de [last downloaded 2021-11-10].

[LLW+11] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H Low, and Lachlan LH Andrew. Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 233–244. ACM, 2011.

[LLW+15] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven Low, and Lachlan LH Andrew. Greening geographical load balancing. *IEEE/ACM Transactions on Networking*, 23(2):657–671, 2015.

[LLWA12] Minghong Lin, Zhenhua Liu, Adam Wierman, and Lachlan LH Andrew. Online algorithms for geographical load balancing. In *International Green Computing Conference (IGCC)*, pages 1–10. IEEE, 2012.

[LPSR12] Zvi Lotker, Boaz Patt-Shamir, and Dror Rawitz. Rent, lease, or buy: Randomized algorithms for multislope ski rental. *SIAM Journal on Discrete Mathematics*, 26(2):718–736, 2012.

[LQL18] Yingying Li, Guannan Qu, and Na Li. Using predictions in online optimization with switching costs: A fast algorithm and a fundamental limit. In *2018 Annual American Control Conference (ACC)*, pages 3008–3013. IEEE, 2018.

[LWAT11a] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. Dynamic right-sizing for power-proportional data centers. In *30th IEEE International Conference on Computer Communications (INFOCOM'11)*, pages 1098–1106. IEEE, 2011.

[LWAT11b] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. Online dynamic capacity provisioning in data centers. In *49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1159–1163. IEEE, 2011.

[LWAT13] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking*, 21(5):1378–1391, 2013.

[Mey05] Adam Meyerson. The parking permit problem. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 274–282. IEEE, 2005.

[Mit14] Sparsh Mittal. Power management techniques for data centers: A survey. Technical report, Future Technologies Group, Oak Ridge National Laboratory, 2014.

[MM88] David Q Mayne and Hannah Michalska. Receding horizon control of nonlinear systems. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 464–465. IEEE, 1988.

[MMS88] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 322–333. ACM, 1988.

[MMS90] Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[MSD+20] Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. Top500, November 2020. URL: https://top500.org/lists/top500/2020/11/ [last downloaded 2021-11-26].

[MV15] Sparsh Mittal and Jeffrey S Vetter. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, 47(4):1–35, 2015.

[Nes03] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[PBBDM98] Giuseppe A Paleologo, Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. Policy optimization for dynamic power management. In *Proceedings 1998 Design and Automation Conference. 35th DAC.(Cat. No. 98CH36175)*, pages 182–187. IEEE, 1998.

[Pru21] Kirk Pruhs. Errata, 2021. URL: http://people.cs.pitt.edu/~kirk/Errata.html [last downloaded 2021-11-26].

[PST04] Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.

[Rit20] Hannah Ritchie. Electricity mix, 2020. URL: https://ourworldindata.org/electricity-mix [last downloaded 2021-11-26].

[RPSS20] Aasheesh Raizada, Kishan Pal Singh, and Mohammad Sajid. Worldwide energy consumption of hyperscale data centers: A survey. *International Research Journal on Advanced Science Hub*, 2:8–15, 2020.

[RSR+07] Suzanne Rivoire, Mehul A Shah, Parthasarathy Ranganathan, Christos Kozyrakis, and Justin Meza. Models and metrics to enable energy-efficiency optimizations. *Computer*, 40(12):39–48, 2007.

[SBM+14] Junaid Shuja, Kashif Bilal, Sajjad A Madani, Mazliza Othman, Rajiv Ranjan, Pavan Balaji, and Samee U Khan. Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Systems Journal*, 10(2):507–519, 2014.

[Sch18] Leander Schnaars. Approximation algorithms for dynamic power management. Bachelor's thesis, Technical University of Munich, 2018.

[Sel20] Mark Sellke. Chasing convex bodies optimally. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1509–1518. SIAM, 2020.

[Sit14] René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM Journal on Computing*, 43(1):96–125, 2014.

[SR09] Patrick Schmid and Achim Roos. Overclocking core i7: Power versus performance, 2009. URL: http://www.tomshardware.com/reviews/overclock-core-i7,2268-10.html [last downloaded 2021-11-26].

[SSS+16] Arman Shehabi, Sarah Smith, Dale Sartor, Richard E Brown, Magnus Herrlin, Jonathan G Koomey, Eric R Masanet, Nathaniel Horner, Ines Lima Azevedo, and William Lintner. United states data center energy usage report. Technical Report LBNL-1005775, Lawrence Berkeley National Laboratory, California, 2016.

[UF21] Inc. UEFI Forum. Advanced configuration and power interface (acpi) specification, 2021. Version 6.4. URL: https://uefi.org/sites/default/files/resources/ACPI_Spec_6_4_Jan22.pdf [last downloaded 2021-11-26].

[Vaz13] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[Vég16] László A Végh. A strongly polynomial algorithm for a class of minimum-cost flow problems with separable convex objectives. *SIAM Journal on Computing*, 45(5):1729–1761, 2016.

[VHLL+14] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. Trends in worldwide ict electricity consumption from 2007 to 2012. *Computer Communications*, 50:64–76, 2014.

[WAT09] Adam Wierman, Lachlan LH Andrew, and Ao Tang. Power-aware speed scaling in processor sharing systems. In *Proceedings of the IEEE INFOCOM 2009*, pages 2007–2015. IEEE, 2009.

[WLC+15] Kai Wang, Minghong Lin, Florin Ciucu, Adam Wierman, and Chuang Lin. Characterizing the impact of the workload on the value of dynamic resizing in data centers. *Performance Evaluation*, 85:1–18, 2015.

[YDS95] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th annual foundations of computer science*, pages 374–382. IEEE, 1995.

[ZZS18] Ming Zhang, Zizhan Zheng, and Ness B Shroff. An online algorithm for power-proportional data centers with switching cost. In *IEEE Conference on Decision and Control (CDC)*, pages 6025–6032. IEEE, 2018.

Appendix A

Variables

The following table gives an overview of the variables defined in this work.

A.1 Latin lowercase letters

Variable	Description
$c(\mathcal{I})$	Constant that depends on the problem instance. Formally, $c(\mathcal{I}) \coloneqq \sum_{j=1}^{d} \max_{t\in[T]} \frac{f_{t,j}(0)}{\beta_j}$.
d	Number of server types.
$f(z)$	Operating cost in the restricted model for a single server in a homogeneous data center running with load $z \in [0,1]$ for a single time slot.
$f_j(z)$	Operating cost for a single server of type j running with load $z \in [0, z_j^{\max}]$ for one time slot.
$f_{t,j}(z)$	Operating cost for a single server of type j running with load $z \in [0, z_j^{\max}]$ during time slot t.
$g_t(\boldsymbol{x}_t)$	Operating cost for the server configuration $\boldsymbol{x}_t$ during time slot t.
$\bar{g}_t(x)$	Continuous extension of g_t, see equation (2.1).
$g_{t,j}(x,q)$	Operating cost of x servers of type j processing the fraction q of the incoming job volume at time t.
m	Total number of servers, $m \coloneqq \sum_{j=1}^{d} m_j$.
$\boldsymbol{m}$	Number of servers for each type. Formally, $\boldsymbol{m} = (m_1, \dots, m_d)$.
m_j	Number of servers of type j.
$m_{t,j}$	Number of available server of type j at time t.
$\tilde{n}_t$	Number of sub time slots for time slot t in algorithm $\mathcal{E}$.
$q_{t,j}$	Job fraction assigned to server type j at time slot t.
r_j	Constant operating cost of server type j.
$\bar{t}_j, \bar{t}_{t,j}$	Running time of a single server of type j (at time t). The precise definition depends on the particular algorithm.
$w_{t,j}$	Number of servers of type j that were powered up at time slot t.
$x_t, x_t^*, \hat{x}_t^k, \bar{x}_t, x_t^{\Xi}$	Number of active servers in a homogeneous data center at time t in the schedule X, X^*, $\hat{X}^k$, $\bar{X}$ or X^{Ξ} with $\Xi \in \{\mathrm{LCP}, \mathcal{R}\}$, respectively.
$\boldsymbol{x}_t$	Server configuration at time slot t. Formally, $\boldsymbol{x}_t = (x_{t,1}, \dots, x_{t,d})$.
x_τ^L, x_τ^U	Last state of X_τ^L and X_τ^U, respectively. Formally, $x_\tau^L \coloneqq x_{\tau,\tau}^L$ and $x_\tau^U \coloneqq x_{\tau,\tau}^U$.

Variable	Description
$x_{t,j}, x^*_{t,j}, \hat{x}^u_{t,j}, x^\Xi_{t,j}$	Number of active servers of type j at time t in the schedule X, X^*, $\hat{X}^u$ or X^Ξ with $\Xi \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}\}$, respectively.
$y_{t,k}, y^{\mathcal{A}}_{t,k}, \hat{y}^u_{t,k}$	Server type used in the k-th lane at time t in the schedule X, $X^{\mathcal{A}}$ or $\hat{X}^u$, respectively.
$\tilde{y}^u_{t,k}$	Largest server type used in lane k by the schedule $\hat{X}^{t'}$ at time slot t' for $t' \in [t:u]$. Formally, $\tilde{y}^u_{t,k} \coloneqq \max_{t' \in [t:u]} \hat{y}^{t'}_{t',k}$.
$z^{\max}_j$	Maximal amount of jobs that a single server of type j can process per time slot.

A.2 Latin uppercase letters

Variable	Description
$A_{j,i}$	The i-th block for server type j in the schedule $X^{\mathcal{C}}$ or $X^{\mathcal{D}}$.
$A_{t,k}$	Block at time slot t in lane k of the schedule $X^{\mathcal{A}}$ or $X^{\mathcal{B}}$.
$C(X)$	Total cost of the schedule X, see equation (1.1).
$C^{\mathcal{J}}(X)$	Total cost of the schedule X regarding the problem instance $\mathcal{J}$.
$C_{t,k}(X)$	Switching and operating cost of the schedule X at time t in lane k.
$G, G(\mathcal{I})$	Graph representing the problem instance $\mathcal{I}$.
$\mathcal{G}$	Operating cost function sequence. Formally, $\mathcal{G} = (g_1, \ldots, g_T)$.
$\mathcal{I}$	Problem instance.
$\mathcal{I}{=}(T,m,\beta,\mathcal{G})$	Problem instance of a homogeneous data center in the general model.
$\mathcal{I}{=}(T,m,\beta,f,\Lambda)$	Problem instance of a homogeneous data center in the restricted model.
$\mathcal{I}{=}(T,d,\boldsymbol{m},\boldsymbol{\beta},\mathcal{G})$	Problem instance of a heterogeneous data center with arbitrary convex operating cost functions g_t.
$\mathcal{I}{=}(T,d,\boldsymbol{m},\boldsymbol{\beta},\boldsymbol{r},\Lambda)$	Problem instance of a heterogeneous data center with constant operating costs $\boldsymbol{r} = (r_1, \ldots, r_d)$.
$\mathcal{I}{=}(T,d,\boldsymbol{m},\boldsymbol{\beta},F,\Lambda)$	Problem instance of a heterogeneous data center with operating cost functions $F = (f_{1,1}, \ldots, f_{T,d})$.
$\bar{\mathcal{I}}$	Continuous extension of the problem instance $\mathcal{I}$.
$\mathcal{I}^k$	Problem instance solved in iteration k of GraphBasedBinarySearch. The number of active servers must be a multiple of 2^k.
$\mathcal{I}^t$	Problem instance that ends at time slot t.
K	Index of the first iteration in GraphBasedBinarySearch. The total number of iterations is given by $K+1$. Formally, $K \coloneqq \log_2 m - 2$.
M^γ_j	Set of values that the number of active servers of type j can take on in the approximation algorithm with parameter γ.
M^k	Considered states in iteration k of GraphBasedBinarySearch. Formally, $M^k \coloneqq \{n \in [m]_0 \mid n \mod 2^k = 0\}$.
$L_{t,j}(X)$	Load-dependent operating cost for server type j at time t in the schedule X, see equation (3.6).
$\mathcal{Q}$	Set of all possible job assignments. Formally, $\mathcal{Q} \coloneqq \{(q_1, \ldots, q_d) \in [0,1]^d \mid \sum_{j=1}^d q_j\}$.

Variable	Description
$R(X), R^{\mathcal{J}}(X)$	Operating cost of the schedule X (regarding the problem instance $\mathcal{J}$).
$S(X), S^{\mathcal{J}}(X)$	Switching cost of the schedule X (regarding the problem instance $\mathcal{J}$).
T	Total number of time slots.
W_t	Servers that were powered up at time slot $u \in W_t$ in algorithm $\mathcal{D}$ will be powered down at time slot t.
X	An arbitrary schedule. Formally, $X = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ and $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,d})$.
X^*	An optimal schedule.
X^{Ξ}	Schedule calculated by algorithm $\Xi \in \{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \text{LCP}, \mathcal{R}\}$.
$\hat{X}^k$	Optimal schedule calculated in iteration k in GraphBasedBinarySearch. The number of active servers are always multiples of 2^k.
X^L_τ, X^U_τ	Lower and upper bound used for the LCP algorithm. Formally, $X^L_\tau = (x^L_{\tau,1}, \ldots, x^L_{\tau,\tau})$ and $X^U_\tau = (x^U_{\tau,1}, \ldots, x^U_{\tau,\tau})$.
$\hat{X}^t$	Optimal schedule for the problem instance $\mathcal{I}^t$ that ends at time t.
$\mathcal{X}$	Set of all possible server configurations. Formally, $\mathcal{X} \coloneqq \bigtimes_{j=1}^{d} [m_j]_0$.
$\mathcal{X}^\gamma$	Set of considered server configurations in the approximation algorithm with parameter γ.

A.3 Greek letters

Variable	Description
β	Switching cost of a single server in a homogeneous data center.
$\boldsymbol{\beta}$	Switching cost vector. Formally, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)$.
β_j	Switching cost of server type j.
γ	Parameter of the approximation algorithm. The algorithm calculates a $(2\gamma - 1)$-approximation.
λ_t	Job volume that arrives at time slot t.
Λ	Sequence of job volumes. Formally, $\Lambda = (\lambda_1, \ldots, \lambda_T)$.
$\Omega(\mathcal{J})$	The set of all optimal schedules for the problem instance $\mathcal{J}$.

Appendix B

Optimal Algorithms for Right-Sizing Data Centers, SPAA 2018

This chapter has been published as **peer-reviewed conference paper** [AQ18a].

> Susanne Albers and Jens Quedenfeld. Optimal algorithms for right-sizing data centers. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA'18)*, pages 363–372. ACM, 2018.

An extended version that contains all proofs and some additional results was submitted to the ACM journal *Transactions on Parallel Computing* on 16th March 2021, but the reviewing process has not yet been finished. A preprint of the extended version can be found on arXiv [AQ18b], see `https://arxiv.org/abs/1807.05112`.

Synopsis. We analyze the right-sizing problem of homogeneous data centers in the discrete setting where the number of active servers must be integral. First, we present an offline algorithm that calculates the optimal schedule in polynomial time. Afterwards, we investigate the online setting and show that the LCP algorithm is 3-competitive in the discrete setting. We prove that no deterministic online algorithm can achieve a competitive ratio smaller than 3, so LCP is optimal. Furthermore, we show a lower bound of 2 for the fractional setting. Finally, we prove that all lower bounds still hold for a more restricted model introduced by Lin et al. [LWAT11a].

Contributions of thesis author. The thesis author developed the algorithms and lower bounds including all proofs. He wrote the whole manuscript except for the abstract and the introduction.

Optimal Algorithms for Right-Sizing Data Centers*

Susanne Albers
Technical University of Munich
albers@in.tum.de

Jens Quedenfeld
Technical University of Munich
jens.quedenfeld@in.tum.de

ABSTRACT

Electricity cost is a dominant and rapidly growing expense in data centers. Unfortunately, much of the consumed energy is wasted because servers are idle for extended periods of time. We study a capacity management problem that dynamically right-sizes a data center, matching the number of active servers with the varying demand for computing capacity. We resort to a data-center optimization problem introduced by Lin, Wierman, Andrew and Thereska [17, 19] that, over a time horizon, minimizes a combined objective function consisting of operating cost, modeled by a sequence of convex functions, and server switching cost. All prior work addresses a continuous setting in which the number of active servers, at any time, may take a fractional value.

In this paper, we investigate for the first time the discrete data-center optimization problem where the number of active servers, at any time, must be integer valued. Thereby we seek truly feasible solutions. First, we show that the offline problem can be solved in polynomial time. Our algorithm relies on a new, yet intuitive graph theoretic model of the optimization problem and performs binary search in a layered graph. Second, we study the online problem and extend the algorithm *Lazy Capacity Provisioning* (LCP) by Lin et al. [17, 19] to the discrete setting. We prove that LCP is 3-competitive. Moreover, we show that no deterministic online algorithm can achieve a competitive ratio smaller than 3. Hence, while LCP does not attain an optimal competitiveness in the continuous setting, it does so in the discrete problem examined here. We prove that the lower bound of 3 also holds in a problem variant with more restricted operating cost functions, introduced by Lin et al. [17].

Finally, we address the continuous setting and give a lower bound of 2 on the best competitiveness of online algorithms. This matches an upper bound by Bansal et al. [5]. A lower bound of 2 was also recently shown by Antoniadis and Schewior [3]. We develop an independent proof that extends to the scenario with more restricted operating cost.

KEYWORDS

Homogeneous servers; polynomial-time offline algorithm; online algorithm; lower bounds; discrete setting.

*Work supported by the European Research Council, Grant Agreement No. 691672.

ACM ISBN 978-1-4503-5799-9/18/07.
https://doi.org/10.1145/3210377.3210385

ACM Reference Format:
Susanne Albers and Jens Quedenfeld. 2018. Optimal Algorithms for Right-Sizing Data Centers. In *SPAA '18: SPAA '18: 30th ACM Symposium on Parallelism in Algorithms and Architectures, July 16–18, 2018, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3210377.3210385

1 INTRODUCTION

Energy conservation in data centers is a major concern for both operators and the environment. In the U.S., about 1.8% of the total electricity consumption is attributed to data centers [22]. In 2015, more than 416 TWh (terawatt hours) were used by the world's data centers, which exceeds the total power consumption in the UK [7]. Electricity cost is a significant expense in data centers [9]; about 18–28% of their budget is invested in power [8, 13]. Remarkably, the servers of a data center are only utilized 20–40% of the time on average [4, 6]. Even worse, when idle and in active mode, they consume about half of their peak power [21]. Hence, a promising approach for energy conservation and capacity management is to transition idle servers into low-power sleep states. However, state transitions, and in particular power-up operations, also incur energy/cost. Therefore, dynamically matching the number of active servers with the varying demand for computing capacity is a challenging optimization problem. In essence, the goal is to right-size a data center over time so as to minimize energy and operation costs.

Problem Formulation. We investigate a basic algorithmic problem with the objective of dynamically resizing a data center. Specifically, we resort to a framework that was introduced by Lin, Wierman, Andrew and Thereska [17, 19] and further explored, for instance, in [1–3, 5, 18, 20, 23].

Consider a data center with m homogeneous servers, each of which has an active state and a sleep state. An optimization is performed over a discrete, finite time horizon consisting of time steps $t = 1, \ldots, T$. At any time t, $1 \leq t \leq T$, a non-negative convex cost function $f_t(\cdot)$ models the operating cost of the data center. More precisely, $f_t(x_t)$ is the incurred cost if x_t servers are in the active state at time t, where $0 \leq x_t \leq m$. This operating cost captures, e.g., energy cost and service delay, for an incoming workload, depending on the number of active servers. Furthermore, at any time t there is a switching cost, taking into account that the data center may be resized by changing the number of active servers. This switching cost is equal to $\beta(x_t - x_{t-1})^+$, where β is a positive real constant and $(x)^+ = \max(0, x)$. Here we assume that transition cost is incurred when servers are powered up from the sleep state to the active state. A cost of powering down servers may be folded into this cost. The constant β incorporates, e.g., the energy needed to transition a server from the sleep state to the active state, as well as delays resulting from a migration of data and connections. We assume that at the beginning and at the end of the time horizon all servers are in the sleep state, i.e. $x_0 = x_{T+1} = 0$. The goal is to determine a vector $X = (x_1, \ldots, x_T)$ called *schedule*,

specifying at any time the number of active servers, that minimizes

$$\sum_{t=1}^{T} f_t(x_t) + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+. \quad (1)$$

In the offline version of this data-center optimization problem, the convex functions f_t, $1 \le t \le T$, are known in advance. In the online version, the f_t arrive over time. At time t, function f_t is presented. Recall that the operating cost at time t depends for instance on the incoming workload, which becomes known only at time t.

All previous work on the data-center optimization problem assumes that the server numbers x_t, $1 \le t \le T$, may take fractional values. That is, x_t may be an arbitrary real number in the range $[0, m]$. From a practical point of view this is acceptable because a data center has a large number of machines. Nonetheless, from an algorithmic and optimization perspective, the proposed algorithms do not compute feasible solutions. Important questions remain if the x_t are indeed integer valued: (1) Can optimal solutions be computed in polynomial time? (2) What is the best competitive ratio achievable by online algorithms? In this paper, we present the first study of the data-center optimization problem assuming that the x_t take integer values and, in particular, settle questions (1) and (2).

Previous Work. As indicated above, all prior work on the data-center optimization problem assumes that the x_t, $1 \le t \le T$, may take fractional values in $[0, m]$. First, Lin et al. [19] consider the offline problem. They develop an algorithm based on a convex program that computes optimal solutions. Second, Lin et al. [19] study the online problem. They devise a deterministic algorithm called *Lazy Capacity Provisioning (LCP)* and prove that it achieves a competitive ratio of exactly 3. Algorithm LCP, at any time t, computes a lower bound and an upper bound on the number of active servers by considering two scenarios in which the switching cost β is charged, either when a server is powered up or when it is powered down. The LCP algorithm lazily stays within these two bounds. The tight bound of 3 on the competitiveness of LCP also holds if the algorithm has a finite prediction window w, i.e. a time t it knows the current as well as the next w arriving functions $f_t, \ldots, f_{t+w}$. Furthermore, Lin et al. [19] perform an experimental study with two real-world traces evaluating the savings resulting from right-sizing in data centers.

Bansal et al. [5] presented a 2-competitive online algorithm and showed that no deterministic or randomized online strategy can attain a competitiveness smaller than 1.86. Recently, Antoniadis and Schewior [3] improved the lower bound to 2. Bansal et al. [5] also gave a 3-competitive memoryless algorithm and showed that this is the best competitive factor achievable by a deterministic memoryless algorithm. The data-center optimization problem is an online convex optimization problem with switching costs. Andrew et al. [1] showed that there is an algorithm with sublinear regret but that $O(1)$-competitiveness and sublinear regret cannot be achieved simultaneously. Antoniadis et al. [2] examine generalized online convex optimization, where the values x_t selected by an algorithm may be points in a metric space, and relate it to convex body chasing.

Further work on energy conservation in data center includes, for instance, [14, 15]. Khuller et al. [14] introduce a machine activation problem. There exists an activation cost budget and jobs have to be scheduled on the selected, activated machines so as to minimize the makespan. They present algorithms that simultaneously approximate the budget and the makespan. A second paper by Li and Khuller [15] considers a generalization where the activation cost of a machine is a non-decreasing function of the load. In the more applied computer science literature, power management strategies and the value of sleep states have been studied extensively. The papers focus mostly on experimental evaluations. Articles that also present analytic results include [10–12].

Our Contribution. We conduct the first investigation of the *discrete* data-center optimization problem, where the values x_t, specifying the number of active servers at any time $t \in \{1, \ldots, T\}$, must be integer valued. Thereby, we seek truly feasible solutions.

First, in Section 2 we study the offline algorithm. We show that optimal solutions can be computed in polynomial time. Our algorithm is different from the convex optimization approach by Lin et al. [19]. We propose a new, yet natural graph-based representation of the discrete data-center optimization problem. We construct a grid-structured graph containing a vertex $v_{t,j}$, for each $t \in \{1, \ldots, T\}$ and $j \in \{0, \ldots, m\}$. Edges represent right-sizing operations, i.e. changes in the number of active servers, and are labeled with operating and switching costs. An optimal solution could be determined by a shortest path computation. However, the resulting algorithm would have a pseudo-polynomial running time. Instead, we devise an algorithm that improves solutions iteratively using binary search. In each iteration the algorithm uses only a constant number of graph layers. The resulting running time is $O(T \log m)$.

The remaining paper focuses on the online problem and develops tight bounds on the competitiveness. In Section 3 we adapt the LCP algorithm by Lin et al. [19] to the discrete data-center optimization problem. We prove that LCP is 3-competitive, as in the continuous setting. We remark that our analysis is different from that by Lin et al. [19]. Specifically, our analysis resorts to the discrete structure of the problem and identifies respective properties. The analysis by Lin et al. [19] relates to their convex optimization approach that characterizes optimal solutions in the continuous setting.

In Section 4 we devise lower bounds. We prove that no deterministic online algorithm can obtain a competitive ratio smaller than 3. Hence, LCP achieves an optimal competitive factor. Interestingly, while LCP does not attain an optimal competitiveness in the continuous data-center optimization problem (where the x_t may take fractional values), it does so in the discrete problem. We prove that the lower bound of 3 on the best possible competitive ratio also holds for a more restricted setting, originally introduced by Lin et al. [17] in the conference publication of their paper. Specifically, the problem is to find a vector $X = (x_1, \ldots, x_T)$ that minimizes

$$\sum_{t=1}^{T} x_t f(\lambda_t / x_t) + \beta \sum_{t=1}^{T} (x_t - x_{t-1})^+, \quad (2)$$

subject to $x_t \ge \lambda_t$, for $t \in \{1, \ldots, T\}$. Here λ_t is the incoming workload at time t and $f(z)$ is a non-negative convex function representing the operating cost of a single server running with load $z \in [0, 1]$. Since f is convex, it is optimal to distribute the jobs equally to all active servers, so that the operating cost at time t is $x_t f(\lambda_t / x_t)$. This problem setting is more restricted in that there is only a single function f modeling operating cost over the

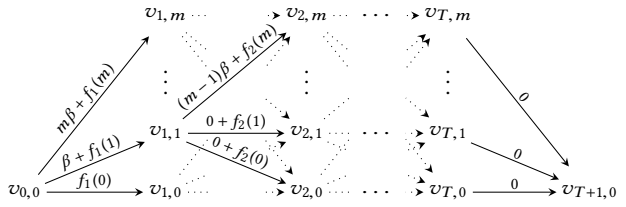

Figure 1: Construction of the graph.

time horizon. Nonetheless it is well motivated by real data center environments.

Furthermore, in Section 4 we address the continuous data-center optimization problem and prove that no deterministic online algorithm can achieve a competitive ratio smaller than 2. The same result was shown by Antoniadis and Schewior [3]. We develop an independent proof that can again be extended to the more restricted optimization problem stated in (2), i.e. the lower bound of 2 on the best competitiveness holds in this setting as well.

Finally, in Section 4 we analyze online algorithms with a finite prediction window, i.e. at time t an online algorithm knows the current as well as the next w arriving functions $f_t, \dots, f_{t+w}$. We show that all our lower bounds, for both settings (continuous and discrete) and both models (general and restricted), still hold.

2 AN OPTIMAL OFFLINE ALGORITHM

In this section we study the offline version of the discrete data-center optimization problem. We develop an algorithm that computes optimal solutions in $O(T \log m)$ time.

2.1 Graph-based approach

Our algorithm works with an underlying directed, weighted graph $G = (V, E)$ that we describe first. Let $[k] := \{1, 2, \dots, k\}$ and $[k]_0 := \{0, 1, \dots, k\}$ with $k \in \mathbb{N}$. For each $t \in [T]$ and each $j \in [m]_0$, there is a vertex $v_{t,j}$, representing the state that exactly j servers are active at time t. Furthermore, there are two vertices $v_{0,0}$ and $v_{T+1,0}$ for the initial and final states $x_0 = 0$ and $x_{T+1} = 0$. For each $t \in \{2, \dots, T\}$ and each pair $j, j' \in [m]_0$, there is a directed edge from $v_{t-1,j}$ to $v_{t,j'}$ having weight $\beta(j'-j)^+ + f_t(j')$. This edge weight corresponds to the switching cost when changing the number of servers between time $t-1$ and t and to the operating cost incurred at time t. Similarly, for $t = 1$ and each $j' \in [m]_0$, there is a directed edge from $v_{0,0}$ to $v_{1,j'}$ with weight $f_1(j') + \beta(j')^+$. Finally, for $t = T$ and each $j \in [m]_0$, there is a directed edge from $v_{T,j}$ to $v_{T+1,0}$ of weight 0. The structure of G is depicted in Figure 1.

In the following, for each $j \in [m]_0$, vertex set $R_j = \{v_{t,j} \mid t \in [T]\}$ is called *row j*. For each $t \in [T]$, vertex set $C_t = \{v_{t,j} \mid j \in [m]_0\}$ is called *column t*.

A path between v_0 and v_{T+1} represents a schedule. If the path visits $v_{t,j}$, then $x_t = j$ servers are active at time t. Note that a path visits exactly one vertex in each column C_t, $1 \le t \le T$, because the directed edges connect adjacent columns. The total length (weight) of a path is equal to the cost of the corresponding schedule. An optimal schedule can be determined using a shortest path computation, which takes $O(Tm)$ time in the particular graph G. However, this running time is not polynomial because the encoding length of an input instance is linear in T and $\log m$, in addition to the encoding of the functions f_t.

In the following, we present a polynomial time algorithm that improves an initial schedule iteratively using binary search. In each iteration the algorithm constructs and uses only a constant number of rows of G.

2.2 Polynomial time algorithm

An instance of the data-center optimization problem is defined by the tuple $\mathcal{P} = (T, m, \beta, F)$ with $F = (f_1, \dots, f_T)$. We assume that m is a power of two. If this is not the case we can transform the given problem instance $\mathcal{P} = (T, m, \beta, F)$ to $\mathcal{P}' = (T, m', \beta, F')$ with $m' = 2^{\lceil \log m \rceil}$ and

$$f'_t(x) = \begin{cases} f_t(x) & x \le m \\ x \cdot (f_t(m) + \epsilon) & \text{otherwise} \end{cases}$$

with $\epsilon > 0$. The term $x \cdot f_t(m)$ ensures that $f'_t(x)$ is a convex function, since the greatest slope of f_t is $f_t(m) - f_t(m-1) \le f_t(m)$. The inequality holds because $f_t(x) \ge 0$ for all $x \in [m]_0$. The additional term $x \cdot \epsilon$ ensures that it is adverse to use a state $x > m$, because the cost of $f_t(m)$ is always smaller.

Our algorithm uses $\log m - 1$ iterations denoted reversely by $k = K := \log m - 2$ for the first iteration and $k = 0$ for the last iteration. The states used in iteration k are always multiples of 2^k. For the first iteration we use the rows $R_0, R_{m/4}, R_{m/2}, R_{3m/4}, R_m$, so that the graph of the first iteration contains the vertices

$$V^K := \{v_{0,0}, v_{T+1,0}\} \cup \left\{v_{t,\xi \cdot m/4} \mid t \in [T], \xi \in \{0, 1, 2, 3, 4\}\right\}.$$

The optimal schedule for this simplified problem instance can be calculated in $O(T)$ time, since each column contains only five states. Given an optimal schedule $\hat{X}^k = (\hat{x}^k_1, \dots, \hat{x}^k_T)$ of iteration k, let

$$V^{k-1}_t := \left\{\hat{x}^k_t + \xi \cdot 2^{k-1} \mid \xi \in \{-2, -1, 0, 1, 2\}\right\} \cap [m]_0$$

be the states used in the t-th column of the next iteration $k-1$. Thus the iteration $k-1$ uses the vertex set

$$V^{k-1} := \{v_{0,0}, v_{T+1,0}\} \cup \left\{v_{t,j} \mid t \in [T], j \in V^{k-1}_t\right\}.$$

Note that the states with $\xi \in \{-2, 0, 2\}$ were already used in iteration k and we just insert the intermediate states $\xi = -1$ and $\xi = 1$. If $\hat{x}^k_t = 0$ (or $\hat{x}^k_t = m$), then $\xi \in \{-2, -1\}$ (or $\xi \in \{1, 2\}$) leads to negative states (or to states larger than m), thus the set V^{k-1}_t is cut with $[m]_0$ to ensure that we only use valid states.

The last iteration ($k = 0$) provides an optimal schedule for the original problem instance as shown in the next section. The runtime of the algorithm is $O(T \cdot \log m)$ and thus polynomial.

2.3 Correctness

To prove the correctness of the algorithm described in the previous section we have to introduce some definitions:

Given the original problem instance $\mathcal{P} = (T, m, \beta, F)$, we define $\mathcal{P}_k$ (with $k \in [K]_0 := [\log m - 2]_0$) as the data-center optimization problem where we are only allowed to use the states that are multiples of 2^k. Let $M_k := \{n \in [m]_0 \mid n \bmod 2^k = 0\}$, so X is a feasible schedule for $\mathcal{P}_k$ if $x_t \in M_k$ holds for all $t \in [T]$. To express $\mathcal{P}_k$ as a tuple, we need another tuple element called M which

describes the allowed states, i.e. $x_t \in M$ for all $t \in [T]$. The original problem instance can be written as $\mathcal{P} = (T, m, \beta, F, [m]_0)$ and $\mathcal{P}_k = (T, m, \beta, F, M_k)$. Note that $\mathcal{P}_0 = \mathcal{P}$. Let $\hat{X}^k = (\hat{x}_1^k, \dots, \hat{x}_T^k)$ denote an optimal schedule for $\mathcal{P}_k$. In general, for any given problem instance $Q = (T, m, \beta, F, M)$, let $\Phi_k(Q) \coloneqq (T, m, \beta, F, M \cap \{i \cdot 2^k \mid i \in \mathbb{N}\})$, so $\Phi_k(\mathcal{P}) = \mathcal{P}_k$.

Instead of using only states that are multiple of 2^k we can also scale a given problem instance $Q = (T, m, \beta, F, M)$ as follows. Let

$$\Psi_l(Q) \coloneqq (T, m/2^l, \beta \cdot 2^l, F', M')$$

with $M' \coloneqq \{x/2^l \mid x \in M\}$, $F' = (f_1', \dots, f_T')$ and $f'(x)_t \coloneqq f_t(x \cdot 2^l)$. Given a schedule $X = (x_1, \dots, x_T)$ for Q with cost $C^Q(X)$, the corresponding schedule $X' = (x_1/2^l, \dots, x_T/2^l)$ for $\Psi_l(Q)$ has exactly the same cost, i.e. $C^Q(X) = C^{\Psi_l(Q)}(X')$. Note that the problem instance $\Psi_k(\mathcal{P}_k)$ uses all integral states less than or equal to $m/2^l$, so there are no gaps.

Furthermore, we introduce a continuous version of any given problem instance Q where fractional schedules are allowed. Let $\bar{Q} = (T, m, \beta, \bar{F}, [0, m])$ with $\bar{F} = (\bar{f}_1, \dots, \bar{f}_T)$ be the continuous extension of the problem instance $Q = (T, m, \beta, F, M)$, where $x_t \in [0, m]$, $\bar{f}_t : [0, m] \to \mathbb{R}_{\geq 0}$ and

$$\bar{f}_t(x) \coloneqq \begin{cases} f_t(x) & \text{if } x \in M \\ (\lceil x \rceil - x) f_t(\lfloor x \rfloor) + (x - \lfloor x \rfloor) f_t(\lceil x \rceil) & \text{else.} \end{cases}$$

The operating cost of the fractional states is linearly interpolated, thus $\bar{f}_t$ is convex for all $t \in [T]$. Let $X^* = (x_1^*, \dots, x_T^*) \in [0, m]^T$ be an optimal schedule for $\bar{\mathcal{P}}$.

The set of all optimal schedules for a given problem instance Q is denoted by $\Omega(Q)$. Let $C^Q_{[a,b]}(X) \coloneqq \sum_{t=a}^{b} f_t(x_t) + \sum_{t=a+1}^{b} \beta(x_t - x_{t-1})^+$ be the cost during the time interval $\{a, a+1, \dots, b\}$. We define $f_0(x) \coloneqq 0$, so $C^Q_{[0,T]}(X) = C^Q(X)$.

Now, we are able to prove the correctness of our algorithm. We begin with a simple lemma showing the relationship between the functions Φ and Ψ.

Lemma 2.1. *The problem instances $\Phi_{k-l}(\Psi_l(\mathcal{P}_l))$ and $\Psi_l(\mathcal{P}_k)$ are equivalent.*

The lemma can be proven by simple calculations using the definitions of Φ and Ψ as shown in the full version of this paper. The next technical lemma will be needed later. Informally, it demonstrates that optimal solutions of the reduced discrete problem and the above continuous problem behave similarly.

Lemma 2.2. *Let $Y \in \Omega(\mathcal{P}_k)$ be an optimal schedule for $\mathcal{P}_k$ with $k \in [K]_0$. There exists an optimal solution $X^* \in \Omega(\bar{\mathcal{P}})$ such that*

$$(y_t - y_{t-1}) \cdot (x_t^* - x_{t-1}^*) \geq 0 \tag{3}$$

holds for all $t \in [T]$ with $|y_t - x_t^*| \geq 2^k$ or $|y_{t-1} - x_{t-1}^*| \geq 2^k$.

The proof (see full paper) consists of a careful case distinction according to the relations of y_{t-1}, y_t, x_{t-1} and x_t. By using this lemma, we can show that an optimal solution for a discrete problem instance $\mathcal{P}_k$ cannot be very far from an optimal solution of the continuous problem instance $\bar{\mathcal{P}}$.

Lemma 2.3. *Let $\hat{X}^k \in \Omega(\mathcal{P}_k)$ be an arbitrary optimal schedule for $\mathcal{P}_k$ with $k \in [K]_0$. There exists an optimal schedule $X^* \in \Omega(\bar{\mathcal{P}})$ for $\bar{\mathcal{P}}$ such that $|\hat{x}_t^k - x_t^*| < 2^k$ holds for all $t \in [T]$. Formally,*

$$\forall k \in [K]_0 : \forall \hat{X}^k \in \Omega(\mathcal{P}_k) : \exists X^* \in \Omega(\bar{\mathcal{P}}) : \forall t \in [T] : |\hat{x}_t^k - x_t^*| < 2^k.$$

Proof. To get a contradiction, we assume that there exists a $\hat{X}^k \in \Omega(\mathcal{P}_k)$ with $k \in [K]_0$ such that for all optimal schedules $X^* \in \Omega(\bar{\mathcal{P}})$ there is at least one $t \in [T]$ with $|\hat{x}_t^k - x_t^*| \geq 2^k$. Let $X^* \in \Omega(\bar{\mathcal{P}})$ be arbitrary. Given the schedule $\hat{X}^k$, let $J_1, \dots, J_l \subseteq [T]$ be the inclusion maximal time intervals such that $|\hat{x}_t^k - x_t^*| \geq 2^k$ holds for all $t \in J_j$ and the sign of $\hat{x}_t^k - x_t^*$ remains the same during J_j. The set of all J_j with $j \in [l]$ is denoted by $\mathcal{J}$. If $\mathcal{J}$ is empty, then the condition $|\hat{x}_t^k - x_t^*| < 2^k$ is fulfilled for all $t \in [T]$. We divide $\mathcal{J}$ into the disjunct sets $\mathcal{J}^+$ and $\mathcal{J}^-$ such that $\mathcal{J}^+$ contains the intervals where $\hat{x}_t^k - x_t^*$ is positive and $\mathcal{J}^-$ the others.

Given a schedule X, the corresponding interval set is denoted by $\mathcal{J}(X)$, the set of all time slots by $S(X) \coloneqq \{t \in J \mid J \in \mathcal{J}(X)\}$, and the number of time slots in $\mathcal{J}$ by $L(X) \coloneqq |S(\mathcal{J}(X))| = \sum_{J \in \mathcal{J}} |J|$.

We will use a recursive transformation ϕ that reduces $L(X)$ at least by one for each step, while the cost of X is not increased. Formally, we have to show that $L(\phi(X)) \leq L(X) - 1$ and $C^{\bar{\mathcal{P}}}(\phi(X)) \leq C^{\bar{\mathcal{P}}}(X)$ holds. The first inequality ensures that the recursive procedure will terminate. The transformation described below will produce fractional schedules, however for each $t \in [T] \setminus S(X)$ it is ensured that $x_t \in M_k$. Therefore, if $L(X) = 0$, the corresponding schedule fulfills $|x_t - x_t^*| < 2^k$ and $x_t \in M_k$ for all $t \in [T]$.

To describe the transformation, we will use the following notation: A given schedule $Y = (y_1, \dots, y_T)$ with $L(Y) > 0$ is transformed to $Z = \phi(Y) = (z_1, \dots, z_T)$.

Let $J \coloneqq (t_i + 1, \dots, t_{i+1} - 1) \in \mathcal{J}(Y)$. We differ between two cases, in case 1 we handle the intervals in $\mathcal{J}^+$, i.e. $y_t > x_t^* + 2^k$ holds for all $t \in J$ and in case 2 we handle the intervals in $\mathcal{J}^-$, i.e. $y_t < x_t^* - 2^k$. We will handle case 1 first.

Let $\lceil x \rceil_n \coloneqq n \cdot \lceil x/n \rceil$ with $x \in \mathbb{R}$ and $n \in \mathbb{N}$ be the smallest value that is divisible by n and greater than or equal to x. The schedule Y is transformed to Z with

$$z_t \coloneqq \begin{cases} y_t & \text{if } t \notin J \\ \lambda \cdot y_t + (1 - \lambda) \cdot x_t^* & \text{if } t \in J \end{cases}$$

where $\lambda \in [0, 1]$ is as small as possible such that $z_t \geq \lceil x_t^* \rceil_{2^k}$ holds for all $t \in J$, so at least one time slot $t_= \in J$ satisfies this condition with equality. This transformation ensures that $L(Z) \leq L(Y) - 1$ holds, because the interval J is split into at least two intervals and one time slot ($t_=$) between them that fulfills $|z_{t_=} - x_{t_=}^*| < 2^k$.

We still have to show that the total cost is not increased by this operation. The total cost can be written as

$$\begin{aligned} C^{\bar{\mathcal{P}}}(X) = C^{\bar{\mathcal{P}}}_{[0,t_i]}(X) + \beta(x_{t_i+1} - x_{t_i})^+ + C^{\bar{\mathcal{P}}}_{[t_i+1, t_{i+1}-1]}(X) \\ + \beta(x_{t_{i+1}} - x_{t_{i+1}-1})^+ + C^{\bar{\mathcal{P}}}_{[t_{i+1}, T]}(X). \end{aligned} \tag{4}$$

We have $C^{\bar{\mathcal{P}}}_{[0,t_i]}(Y) = C^{\bar{\mathcal{P}}}_{[0,t_i]}(Z)$ and $C^{\bar{\mathcal{P}}}_{[t_{i+1},T]}(Y) = C^{\bar{\mathcal{P}}}_{[t_{i+1},T]}(Z)$.

Consider the time slot t_i. By the definition of the interval J, the condition $|y_{t_i+1} - x^*_{t_i+1}| \geq 2^k$ is fulfilled. Thus we can apply Lemma 2.2 which says that the terms $(y_{t_i+1} - y_{t_i})$ and $(x^*_{t_i+1} - x^*_{t_i})$ are both either non-negative or non-positive, so in Equation (4) the

term $\beta(x_{t_i+1} - x_{t_i})^+$ can be replaced by $\beta(x_{t_i+1} - x_{t_i})$ or zero, respectively. Analogously, for the time slot t_{i+1}, the condition $|y_{t_{i+1}-1} - x^*_{t_{i+1}-1}| \geq 2^k$ is fulfilled, so by Lemma 2.2 the term $\beta(x_{t_{i+1}} - x_{t_{i+1}-1})^+$ in Equation (4) can be replaced by $\beta(x_{t_{i+1}} - x_{t_{i+1}-1})$ or zero. In the former cases, the cost function is

$$C^{\bar{\mathcal{P}}}(X) = C^{\bar{\mathcal{P}}}_{[0,t_i]}(X) + \beta x_{t_i+1} - \beta x_{t_i} + C^{\bar{\mathcal{P}}}_{[t_i+1,t_{i+1}-1]}(X) + \beta x_{t_{i+1}} - \beta x_{t_{i+1}-1} + C^{\bar{\mathcal{P}}}_{[t_{i+1},T]}(X).$$

Given a schedule $X = (x_1, \ldots, x_t)$, we define $X_{[a,b]} \coloneqq (x_a, \ldots, x_b)$ and $X_J = X_{[t_i+1,t_{i+1}-1]}$. Since there is no summand that contains both x_{t_i} and x_{t_i+1}, the function

$$D_{X^*}((x'_{t_i+1}, \ldots, x'_{t_{i+1}-1})) \coloneqq C^{\bar{\mathcal{P}}}_{[0,t_i]}(X^*) - \beta x^*_{t_i} + \beta x'_{t_i+1} + C^{\bar{\mathcal{P}}}_{[t_i+1,t_{i+1}-1]}(X') + \beta x'_{t_{i+1}} - \beta x^*_{t_{i+1}-1} + C^{\bar{\mathcal{P}}}_{[t_{i+1},T]}(X^*)$$

with $x'_{t_i+1} \geq x^*_{t_i+1}$ and $x'_{t_{i+1}-1} \geq x^*_{t_{i+1}-1}$ is convex and has a minimum at $X^{\min}_J \coloneqq (x^*_{t_i+1}, \ldots, x^*_{t_{i+1}-1})$.

Due to convexity, $D_{X^*}(Y_J) \geq D_{X^*}(Z_J) \geq D_{X^*}(X^{\min}_J)$, because $Z_J = \lambda Y_J + (1-\lambda)X^{\min}_J$. Therefore $C^{\bar{\mathcal{P}}}(Z) \leq C^{\bar{\mathcal{P}}}(Y)$ holds. If $\beta(x_{t_i+1} - x_{t_i})^+ = 0$ or $\beta(x_{t_{i+1}} - x_{t_{i+1}-1})^+ = 0$ we can use the same argument.

We still have to handle the second case, i.e. $y_t < x^*_t$. The proof works almost analogously, the difference is that we choose λ as small as possible such that $z_t \leq \lfloor x^*_t \rfloor_{2^k}$ (where $\lfloor x \rfloor_n \coloneqq n \cdot \lfloor x/n \rfloor$). Then we have a time slot $t_=$ with $z_{t_=} = \lfloor x^*_{t_=} \rfloor_{2^k}$, so $L(Z) \leq L(Y) - 1$. The proof that shows $C^{\bar{\mathcal{P}}}(Z) \leq C^{\bar{\mathcal{P}}}(Y)$ holds for both cases.

We use the transformation ϕ until $L(Z) = 0$. Then $\mathcal{J}(Z)$ is empty, so all states of Z are multiples of 2^k, i.e. $z_t \in M_k$ for all $t \in [T]$. Since $\hat{X}^k$ was defined to be optimal, $C^{\bar{\mathcal{P}}}(\hat{X}^k) = C^{\bar{\mathcal{P}}}(Z)$ holds. By our assumption, $Z \neq \hat{X}^k$ holds (because otherwise $|\hat{x}^k_t - x^*_t| < 2^k$ would be fulfilled for all $t \in [T]$), so there was a transformation with $\lambda < 1$. Thus we moved towards the optimal schedule, however by $C^{\bar{\mathcal{P}}}(\hat{X}^k) = C^{\bar{\mathcal{P}}}(Z)$, the cost does not change. As $D_{X^*}(X')$ is a convex function, $C^{\bar{\mathcal{P}}}(\hat{X}^k) = C^{\bar{\mathcal{P}}}(Z)$ implies that $C^{\bar{\mathcal{P}}}(Z) = C^{\bar{\mathcal{P}}}(X^*)$, because X^* minimizes $C^{\bar{\mathcal{P}}}$. In this case $\hat{X}^k$ is also optimal for $\bar{\mathcal{P}}$, so the condition $|\hat{x}^k_t - x^*_t| < 2^k$ is already fulfilled.

In all cases we get a contradiction, so our assumption was wrong and the lemma is proven. □

The next lemma shows how an optimal fractional schedule can be rounded to an integral schedule such that it is still optimal.

Lemma 2.4. *Let $X^* \in \Omega(\bar{\mathcal{P}})$. The schedules $\lfloor X^* \rfloor \coloneqq (\lfloor x^*_1 \rfloor, \ldots, \lfloor x^*_T \rfloor)$ and $\lceil X^* \rceil \coloneqq (\lceil x^*_1 \rceil, \ldots, \lceil x^*_T \rceil)$ are optimal too, i.e. $\lfloor X^* \rfloor, \lceil X^* \rceil \in \Omega(\bar{\mathcal{P}})$.*

Proof. Let $X^* \in \Omega(\bar{\mathcal{P}})$ be arbitrary. Let $\mathcal{I}(X^*) = \{I_1, \ldots, I_l\}$ be the set of time intervals such that for each $I_i \coloneqq \{a_i, a_i+1, \ldots, b_i\}$ with $i \in [l]$ the following conditions are fulfilled

(1) All states of X^* have the same value during I_i, i.e. $x^*_t = v_i$ for all $t \in I_i$.
(2) The value is fractional, i.e. $v_i \notin \mathbb{N}$.
(3) Each I_i is inclusion maximal, i.e. $x^*_{a_i-1} \neq v_i$ and $x^*_{b_i+1} \neq v_i$.
(4) The intervals are sorted, i.e. $b_i < a_{i+1}$ for all $i \in [l-1]$.

If $\mathcal{I}(X^*) = \emptyset$, then X^* is an integral schedule, so $\lfloor X^* \rfloor = X^* = \lceil X^* \rceil$. Otherwise let $I_i \in \mathcal{I}(X^*)$ be an arbitrary interval. We will transform X^* to X' by changing the states at I_i such that $|\mathcal{I}(X')| < |\mathcal{I}(X^*)|$ and $\lfloor x^*_t \rfloor \leq x'_t \leq \lceil x^*_t \rceil$ for all $t \in I_i$. Let $g(x) \coloneqq \sum_{t=a_i}^{b_i} \bar{f}_t(x)$. Since each $\bar{f}_t(x)$ is linear for $x \in [\lfloor v_i \rfloor, \lceil v_i \rceil]$, the slope of $g(x)$ is constant for $x \in [\lfloor v_i \rfloor, \lceil v_i \rceil]$ and denoted by $g'(v_i)$. According to I_i, we have to differ between different cases:

(1) $x^*_{a_i-1} < v_i < x^*_{b_i+1}$
Let $\tilde{x}^*_{a_i-1} \coloneqq \max\{x^*_{a_i-1}, \lfloor v_i \rfloor\}$ and $\tilde{x}^*_{b_i+1} \coloneqq \min\{x^*_{b_i+1}, \lceil v_i \rceil\}$. By using any schedule with $x'_{a_i} = x'_{a_i+1} = \cdots = x'_{b_i} \in [\tilde{x}^*_{a_i-1}, \tilde{x}^*_{b_i+1}]$ (and $x'_t = x^*_t$ otherwise), the switching cost is unchanged. Since I_i is inclusion maximal and X^* is optimal, we can conclude that $g'(v_i) = 0$, so $C(X') = C(X^*)$. To show that $\lfloor X^* \rfloor$ is optimal, we set $x'_t = \tilde{x}^*_{b_i+1}$ for all $t \in I_i$. To show that $\lceil X^* \rceil$ is optimal, we set $x'_t = \tilde{x}^*_{a_i-1}$ for all $t \in I_i$.
(2) $x^*_{a_i-1} > v_i > x^*_{b_i+1}$
This case works analogously to the first case
(3) $x^*_{a_i-1} > v_i < x^*_{b_i+1}$
Let $v^+ = \min\{x^*_{a_i-1}, x^*_{b_i+1}, \lceil v_i \rceil\}$. Let $v'_i \in [\lfloor v_i \rfloor, v^+]$. By using the schedule $x'_{a_i} = x'_{a_i+1} = \cdots = x'_{b_i} = v'_i$ for all $t \in I_i$, the switching cost is increased by $\beta(v_i - v'_i)$, but the operating cost is reduced by $g'(v_i) \cdot (v_i - v'_i)$. Since v_i is fractional, $v_i \notin \{x^*_{a_i-1}, x^*_{b_i+1}\}$. As X^* is optimal, we can conclude that $g'(v_i) = \beta$, so the total cost of X' does not change for $v'_i \in [\lfloor v_i \rfloor, v^+]$. To show that $\lfloor X^* \rfloor$ is optimal, we set $x'_t = \lfloor v_i \rfloor$ for all $t \in I_i$. To show that $\lceil X^* \rceil$ is optimal, we set $x'_t = v^+$ for all $t \in I_i$.
(4) $x^*_{a_i-1} < v_i > x^*_{b_i+1}$
This case works analogously to the third case, but $\lfloor x \rfloor$ and $\lceil x \rceil$ are swapped as well as min and max. Furthermore, $g'(v_i) = -\beta$ and we replace $(v_i - v'_i)$ with $(v'_i - v_i)$.

By using the transformation described above, we can reduce the number $|\mathcal{I}|$ of fractional intervals is at least reduced by 1. By applying the transformation several times until $|\mathcal{I}| = 0$, we receive $\lfloor X^* \rfloor$ or $\lceil X^* \rceil$. The total cost is not increased by the operations. □

So far, we have shown in Lemma 2.3 that for each optimal solution of the discrete problem instance $\mathcal{P}_k$ there is an optimal solution of the continuous problem instance $\bar{\mathcal{P}}$ that is not far away. In the following lemma we expand this statement: Given an optimal solution for $\mathcal{P}_k$, there is not only a fractional solution for $\bar{\mathcal{P}}$ that is not far away, but also an optimal solution of the discrete problem instance $\mathcal{P}_l$ for the subsequent iterations $l < k$.

Lemma 2.5. *Let $k > l$ with $k, l \in [K]_0$. Let $\hat{X}^k \in \Omega(\mathcal{P}_k)$ be an arbitrary optimal schedule for $\mathcal{P}_k$ with $k \in [K]_0$. There exists an optimal schedule $\hat{X}^l \in \Omega(\mathcal{P}^l)$ for $\mathcal{P}^l$ such that $|\hat{x}^k_t - \hat{x}^l_t| \leq 2^k$ for all $t \in [T]$. Formally, $\forall k \in [K]_0 : \forall l \in [k-1] : \forall \hat{X}^k \in \Omega(\mathcal{P}_k) : \exists \hat{X}^l \in \Omega(\mathcal{P}_l) : \forall t \in [T] : |\hat{x}^k_t - \hat{x}^l_t| \leq 2^k$.*

Proof. Consider the reduced problem instance $\mathcal{Q} \coloneqq \Psi_l(\mathcal{P}_l)$ as well as the problem instance $\mathcal{Q}_{k-l} \coloneqq \Phi_{k-l}(\Psi_l(\mathcal{P}_l))$ which is equivalent to $\Psi_l(\mathcal{P}_k))$ due to Lemma 2.1. Let $\hat{X}^{k-l}_{\mathcal{Q}} = (\hat{x}^k_1/2^l, \ldots, \hat{x}^k_T/2^l)$ be an optimal schedule for $\mathcal{Q}_{k-l}$. We apply Lemma 2.3, but we

use $\hat{X}^{k-l}_{\mathcal{Q}}$ and $\mathcal{Q}$ instead of $\hat{X}^k$ and $\mathcal{P}$. By Lemma 2.3, there exists an optimal fractional schedule $X^*_{\mathcal{Q}} = (x^*_1, \dots, x^*_T)$ for $\bar{\mathcal{Q}}$ such that $|\hat{x}^k_t/2^l - x^*_t| \leq 2^{k-l}$. By Lemma 2.4, $\lfloor X^*_{\mathcal{Q}} \rfloor$ is also an optimal schedule for $\bar{\mathcal{Q}}$ and therefore it is also optimal for $\mathcal{Q}$. The inequality $|\hat{x}^k_t/2^l - \lfloor x^*_t \rfloor| \leq 2^{k-l}$ still holds, because the terms $\hat{x}^k_t/2^l$ and 2^{k-l} are integral and therefore adding a value less than 1 to the left side cannot invalidate the inequality. Let $\hat{X}^l := (\lfloor x^*_1 \rfloor \cdot 2^l, \dots, \lfloor x^*_T \rfloor \cdot 2^l)$. As $\lfloor X^*_{\mathcal{Q}} \rfloor$ is optimal for $\mathcal{Q}$, $\hat{X}^l$ must be optimal for $\mathcal{P}_l$. Furthermore $\lfloor x^*_t \rfloor = \hat{x}^l_t/2^l$ holds, so we can insert it into the above inequality and get $|\hat{x}^k_t/2^l - \hat{x}^l_t/2^l| \leq 2^{k-l}$ which is equivalent to $|\hat{x}^k_t - \hat{x}^l_t| \leq 2^k$. □

Now, we have proven all parts to show the correctness of our polynomial-time algorithm:

Theorem 2.6. *The algorithm described in Section 2.2 is correct.*

Proof. We will show the correctness by induction. In the first iteration, the algorithm finds an optimal schedule for $\mathcal{P}_K$, because all states of M_K are considered.

Given an optimal schedule $\hat{X}^k$, in the next iteration the algorithm only considers the states x_t with $|\hat{x}^k_t - x_t| \leq 2^k$. By Lemma 2.5, there exists an optimal schedule $\hat{X}^l$ with $l = k-1$ such that $|\hat{x}^k_t - \hat{x}^l_t| \leq 2^k$ holds. Therefore the schedule found in iteration $k-1$ must be optimal for $\mathcal{P}_{k-1}$ (although some states are ignored by the algorithm). Thus, by induction, the algorithm will find an optimal schedule for $\mathcal{P}_0 = \mathcal{P}$ in the last iteration. □

3 AN OPTIMAL ONLINE ALGORITHM

Lin et. al. [17, 19] developed an algorithm called *Lazy Capacity Provisioning* (LCP) that achieves a competitive ratio of 3 for the continuous setting (i.e. $x_t \in \mathbb{R}$). In this section we adapt LCP to the discrete data-center optimization problem and prove the algorithm is 3-competitive for this problem as well.

The general approach of our proof is similar to the proof of the continuous setting in [17]. Some lemmas (e.g., Lemma 3.1 and 3.6) were adopted, however, their proofs are completely different. Lin et. al. use the properties of the convex program, especially duality and the complementary slackness conditions. This approach cannot be adapted to the discrete setting.

3.1 Algorithm

First, we will define lower and upper bounds for the optimal offline solution that can be calculated online. For a given time slot τ let $X^L_\tau := (x^L_{\tau,1}, \dots, x^L_{\tau,\tau})$ be the vector that minimizes

$$C^L_\tau(X) = \sum_{t=1}^{\tau} f_t(x_t) + \beta \sum_{t=1}^{\tau} (x_t - x_{t-1})^+ \tag{5}$$

with $X = (x_1, \dots, x_\tau)$. This term describes the cost of a workload that ends at $\tau \leq T$. For $\tau = T$ this equation is equivalent to (1) . Let $x^L_\tau := x^L_{\tau,\tau}$ be the last state for this truncated workload. If there is more than one vector that minimizes (5), then x^L_τ is defined as the smallest possible value.

Similarly, let $X^U_\tau := (x^U_{\tau,1}, \dots, x^U_{\tau,\tau})$ be the vector that minimizes

$$C^U_\tau(X) = \sum_{t=1}^{\tau} f_t(x_t) + \beta \sum_{t=1}^{\tau} (x_{t-1} - x_t)^+. \tag{6}$$

The difference to the equation (5) is that we pay the switching cost for powering down. Powering up does not cost anything. The last state is denoted by $x^U_\tau := x^U_{\tau,\tau}$. If there is more than one vector that minimizes (6), then x^U_τ is the largest possible value.

Define $[x]^b_a := \max\{a, \min\{b, x\}\}$ as the projection of x into the interval $[a, b]$. The LCP algorithm is defined as follows:

$$x^{\text{LCP}}_\tau := \begin{cases} 0, & \tau = 0 \\ [x^{\text{LCP}}_{\tau-1}]^{x^U_\tau}_{x^L_\tau}, & \tau \geq 1 \end{cases} \tag{7}$$

Before we can prove that this algorithm is 3-competitive, we have to introduce some notation.

3.2 Notation

Let $X^* = (x^*_1, \dots, x^*_T)$ be an optimal offline solution that minimizes equation (1) (i.e. the whole workload). Note that $C^L_\tau(X^*)$ indicates the cost of the optimal solution until τ.

Let $R_\tau(X) := \sum_{t=1}^{\tau} f_t(x_t)$ with $X = (x_1, \dots, x_\tau)$ denote the operating cost until τ, let $S^L_\tau(X) := \beta \sum_{t=1}^{\tau} (x_t - x_{t-1})^+$ denote the switching cost in $C^L_\tau(X)$ and let $S^U_\tau(X) := \beta \sum_{t=1}^{\tau} (x_{t-1} - x_t)^+$ denote the switching cost in $C^U_\tau(X)$. Note that $C^L_\tau(X) = R_\tau(X) + S^L_\tau(X)$ and $C^U_\tau(X) = R_\tau(X) + S^U_\tau(X)$. Furthermore,

$$S^L_\tau(X) = S^U_\tau(X) + \beta x_\tau \tag{8}$$

as well as $C^L_\tau(X) = C^U_\tau(X) + \beta x_\tau$ holds, because in C^L_τ we have to pay the missing switching cost to reach the final state x_τ. Note that βx_τ equals the cost for powering up in C^L_τ minus the cost for powering down in C^U_τ.

Given an arbitrary function $f : [m] \to \mathbb{R}$, we define

$$\Delta f(x) := f(x) - f(x-1)$$

as the slope of f at x. Let

$$\hat{C}^Y_\tau(x) := \min_{x_1, \dots, x_{\tau-1}} C^Y_\tau((x_1, \dots, x_{\tau-1}, x))$$

with $Y \in \{L, U\}$ be the minimal cost achievable with $x_\tau = x$.

3.3 Competitive ratio

In this section we prove that the LCP algorithm described by equation (7) achieves a competitive ratio of 3. First, we show that the optimal solution is bounded by the upper and lower bounds defined in the previous section.

Lemma 3.1. *For all τ, $x^L_\tau \leq x^*_\tau \leq x^U_\tau$ holds.*

Proof. We prove both parts of the inequality by contradiction:

Part 1 ($x^L_\tau \leq x^*_\tau$): Assume that $x^L_\tau > x^*_\tau$. By the definition of the lower bound, $C^L_\tau(X^L_\tau) < C^L_\tau(X^*)$ holds and we can replace $(x^*_1, \dots, x^*_\tau)$ by $(x^L_{\tau,1}, \dots x^L_{\tau,\tau})$. This reduces the total cost of x^*, because the cost up to τ is reduced and for $\tau + 1$ there are no additional switching costs because $x^L_\tau > x^*_\tau$ holds. Thus x^* would not be an optimal solution which is a contradiction, so $x^L_\tau \leq x^*_\tau$ must be fulfilled.

Part 2 ($x^*_\tau \leq x^U_\tau$): Assume that $x^*_\tau > x^U_\tau$. By definition of the upper bound, $C^U_\tau(X^U_\tau) < C^U_\tau(X^*)$ and, thus,

$$R_\tau(X^U_\tau) + S^U_\tau(X^U) < R_\tau(X^*) + S^U_\tau(X^*) \tag{9}$$

holds. The cost of the optimal solution until τ is $R_\tau(X^*)+S^L_\tau(X^*)$. If the states $(x^*_1,\dots,x^*_\tau)$ are replaced by X^U_τ and afterwards $x^*_\tau - x^U_{\tau,\tau}$ servers are powered up (to ensure that we end in the same state), then the cost is $R_\tau(X^U_\tau)+S^L_\tau(X^U)+\beta(x^*_\tau - x^U_{\tau,\tau})$. This cost must be greater than or equal to the cost of the optimal solution, so

$$R_\tau(X^U_\tau)+S^L_\tau(X^U)+\beta(x^*_\tau - x^U_{\tau,\tau}) \geq R_\tau(X^*)+S^L_\tau(X^*)$$

holds. By using equation (8), we get

$$R_\tau(X^U_\tau)+S^U_\tau(X^U)+\beta x^U_{\tau,\tau}+\beta(x^*_\tau - x^U_{\tau,\tau}) \geq R_\tau(X^*)+S^U_\tau(X^*)+\beta x^*_\tau.$$

We eliminate identical terms and get

$$R_\tau(X^U_\tau)+S^U_\tau(X^U) \geq R_\tau(X^*)+S^U_\tau(X^*)$$

which is a contradiction to inequality (9). Therefore our assumption was wrong, so $x^*_\tau \leq x^U_\tau$ must be fulfilled. □

The following four lemmas show important properties of $\hat{C}^L_\tau(x)$. First, we prove that the relation between $C^L_\tau(X)$ and $C^U_\tau(X)$ described by equation (8) still holds for $\hat{C}^L_\tau(x)$ and $\hat{C}^U_\tau(x)$.

Lemma 3.2. *For all τ, $\hat{C}^L_\tau(x) = \hat{C}^U_\tau(x) + \beta x$ holds.*

Proof. Let X^L be a corresponding solution for $\hat{C}^L_\tau(x)$ such that $C^L_\tau(X^L) = \hat{C}^L_\tau(x)$ and let X^U be a corresponding solution for $\hat{C}^U_\tau(x)$ such that $C^U_\tau(X^U) = \hat{C}^U_\tau(x)$. Note that the last state of X^L and X^U is x. Since X^U is optimal for C^U_τ, the inequality $C^U_\tau(X^U) \leq C^U_\tau(X)$ holds for all $X = (x_1,\dots,x_{\tau-1},x)$. By equation (8), we get

$$C^L_\tau(X^U) - \beta x \leq C^L_\tau(X) - \beta x$$

which is equivalent to $C^L_\tau(X^U) \leq C^L_\tau(X)$. With $X := X^L$ we get $C^L_\tau(X^U) \leq C^L_\tau(X^L)$. Since X^L is optimal for C^L_τ, X^U must be optimal too, so $C^L_\tau(X^U) = C^L_\tau(X^L)$ holds. All in all, we get

$$\hat{C}^L_\tau(x) = C^L_\tau(X^L) = C^L_\tau(X^U) = C^U_\tau(X^U) + \beta x = \hat{C}^U_\tau(x) + \beta x. \quad \square$$

Obviously, the cost functions $C^L_\tau(X)$ and $C^U_\tau(X)$ are convex, since convexity is closed under addition. The following lemma shows that also $\hat{C}^L_\tau(x)$ and $\hat{C}^U_\tau(x)$ are convex.

Lemma 3.3. *For all τ and $Y \in \{L,U\}$, $\hat{C}^Y_\tau(x)$ is a convex function.*

Lemma 3.4. *The slope of $\hat{C}^L_\tau(x)$ is at most β for $x \leq x^U_\tau$ and at least β for $x > x^U_\tau$, i.e. $\Delta\hat{C}^L_\tau(x^U_\tau) \leq \beta$ and $\Delta\hat{C}^L_\tau(x^U_\tau+1) \geq \beta$*

The proof of Lemma 3.3 and 3.4 can be found in the full paper.

Lemma 3.5. *For $x \leq x^U_\tau$, the slope of $\hat{C}^L_\tau(x)$ is at most β, i.e. $\Delta\hat{C}^L_\tau(x) \leq \beta$ holds.*

Proof. By Lemma 3.4 $\Delta\hat{C}^L_\tau(x^U_\tau) \leq \beta$ holds and by Lemma 3.3 $\hat{C}^L_\tau$ is convex, so $\Delta\hat{C}^L_\tau(x) \leq \beta$ for $x \leq x^U_\tau$. □

The next lemma characterizes the behavior of the optimal solution backwards in time.

Lemma 3.6. *A solution vector $(\hat{x}_1,\dots,\hat{x}_T)$ that fulfills the following recursive equality for all $t \in \{1,\dots,T\}$ is optimal:*

$$\hat{x}_t := \begin{cases} 0, & t = T+1 \\ [\hat{x}_{t+1}]^{x^U_t}_{x^L_t}, & t \leq T \end{cases}$$

Proof. We will prove the lemma by induction in reverse time. Powering down does not cost anything, so setting $\hat{x}_{T+1} = 0$, does not produce any additional costs. Assume that $(\hat{x}_{\tau+1},\dots,\hat{x}_T)$ can lead to an optimal solution, i.e. there exists an optimal solution X^* with $x^*_t = \hat{x}_t$ for $t \geq \tau+1$. We will show that the vector $(\hat{x}_\tau,\dots,\hat{x}_T)$ can still lead to an optimal solution.

We have to examine three cases:

Case 1: If $\hat{x}_{\tau+1} < x^L_\tau$, then $\hat{x}_\tau = x^L_\tau$. By Lemma 3.1, $x^*_\tau \geq x^L_\tau$ holds. Since X^L_τ minimizes C^L_τ, we know that $C^L_\tau(X^L_\tau) \leq C^L_\tau(X)$ for all $X = (x_1,\dots,x_\tau)$. Thus there is no benefit to use a state $x' \geq x^L_\tau$, because afterwards we have to power down some servers to reach $\hat{x}_{\tau+1}$. Therefore $\hat{x}_\tau = x^L_\tau$ can still lead to an optimal solution.

Case 2: If $\hat{x}_{\tau+1} > x^U_\tau$, then $\hat{x}_\tau = x^U_\tau$. By Lemma 3.1, $x^*_\tau \leq x^U_\tau$ holds. Since X^U_τ minimizes C^U_τ, we know that $C^U_\tau(X^U_\tau) \leq C^U_\tau(X)$ for all X. By using the solution X^U_τ and then switching to state $\hat{x}_{\tau+1}$, the resulting cost is

$$\begin{aligned} & C^L_\tau(X^U_\tau) + \beta(\hat{x}_{\tau+1} - x^U_\tau) + f_{\tau+1}(\hat{x}_{\tau+1}) \\ = {} & C^U_\tau(X^U_\tau) + \beta\hat{x}_{\tau+1} + f_{\tau+1}(\hat{x}_{\tau+1}) \\ \leq {} & C^U_\tau(X) + \beta\hat{x}_{\tau+1} + f_{\tau+1}(\hat{x}_{\tau+1}) \\ = {} & C^L_\tau(X) + \beta(\hat{x}_{\tau+1} - x_\tau) + f_{\tau+1}(\hat{x}_{\tau+1}). \end{aligned}$$

The last line describes the cost until $\tau+1$ by using the vector $X = (x_1,\dots,x_\tau)$ with $x_\tau \leq x^U_\tau$ instead of X^U_τ. The cost is not reduced by using X, so $\hat{x}_\tau = x^U_\tau$ can still lead to an optimal solution.

Case 3: If $x^L_\tau \leq \hat{x}_{\tau+1} \leq x^U_\tau$, then $\hat{x}_\tau = \hat{x}_{\tau+1}$. Assume that there is a better state $\hat{x}^-_\tau < \hat{x}_\tau$ with $\hat{C}^L_\tau(\hat{x}^-_\tau) < \hat{C}^L_\tau(\hat{x}_\tau)$. Since $\hat{x}_{\tau+1}$ leads to an optimal solution, after the time slot τ we have to power up $\hat{x}_\tau - \hat{x}^-_\tau$ servers, so even $\hat{C}^L_\tau(\hat{x}^-_\tau) + \beta(\hat{x}_\tau - \hat{x}^-_\tau) < \hat{C}^L_\tau(\hat{x}_\tau)$ holds. By Lemma 3.5, we know that the slope of $\hat{C}^L_\tau(x)$ is at most β for $x \leq x^U_\tau$. This leads to the contradiction $\hat{C}^L_\tau(\hat{x}_\tau) \leq \hat{C}^L_\tau(\hat{x}^-_\tau) + \beta(\hat{x}^-_\tau - \hat{x}_\tau)$. Therefore there is no $\hat{x}^-_\tau$ with the desired properties.

The other direction is more simple: Assume that there is a better state $\hat{x}^+_\tau > \hat{x}_\tau$ with $\hat{C}^L_\tau(\hat{x}^+_\tau) < \hat{C}^L_\tau(\hat{x}_\tau)$, then x^L_τ (which minimizes $\hat{C}^L_\tau$) must be greater than $\hat{x}_\tau$, because by Lemma 3.3, $\hat{C}^L_\tau$ is a convex function. However, this is a contradiction to $x^L_\tau \leq \hat{x}_{\tau+1} = \hat{x}_\tau$. □

In the following $X^* = (x^*_1,\dots,x^*_T)$ denotes an optimal solution that fulfills the recursive equality of Lemma 3.6. The next lemma describes time slots where X^{LCP} and X^* are in same state. Informally, the lemma says that if the LCP curve cuts the optimal solution, then there is one time slot τ where both solutions are in the same state.

Lemma 3.7. *If $x^{LCP}_{\tau-1} < x^*_{\tau-1}$ and $x^{LCP}_\tau \geq x^*_\tau$, then $x^{LCP}_\tau = x^*_\tau$.*
If $x^{LCP}_{\tau-1} > x^*_{\tau-1}$ and $x^{LCP}_\tau \leq x^*_\tau$, then $x^{LCP}_\tau = x^*_\tau$.

Proof. We will only show the first statement of the lemma, since the other one works exactly analogously. Assume that $x^{\text{LCP}}_{\tau-1} < x^*_{\tau-1}$ and $x^{\text{LCP}}_\tau \geq x^*_\tau$ holds. We differ between two cases.

Case 1: If $x^{\text{LCP}}_{\tau-1} < x^{\text{LCP}}_\tau$, then $x^{\text{LCP}}_\tau = x^L_\tau$ (by the definition of the LCP algorithm). By $x^{\text{LCP}}_\tau \geq x^*_\tau$ and Lemma 3.1 (which says that $x^L_\tau \leq x^*_\tau$), we get $x^{\text{LCP}}_\tau = x^*_\tau$.

Case 2: If $x^{\text{LCP}}_{\tau-1} \geq x^{\text{LCP}}_\tau$, then $x^*_{\tau-1} > x^*_\tau$. By Lemma 3.6, $x^*_{\tau-1} = x^L_{\tau-1}$ holds which is a contradiction to $x^*_{\tau-1} > x^{\text{LCP}}_{\tau-1} \geq x^L_{\tau-1}$. □

The time slots where $x^{\text{LCP}}_t = x^*_t$ are denoted by $0 = t_0 < t_1 < \cdots < t_\kappa$. Between these time slots it is not possible that X^{LCP}

powers one or more servers down and X^* powers servers up or vice versa. In the following $[a : b]$ with $a, b \in \mathbb{N}$ denotes the set $\{a, a+1, \ldots, b\}$. Analogously, we define $[a : b[:= \{a, a+1 \ldots, b-1\}$, $]a : b] := \{a+1, a+2, \ldots, b\}$ and $]a : b[:= \{a+1, a+2, \ldots b-1\}$.

Lemma 3.8. *For all time intervals $]t_i : t_{i+1}[$ with $i \geq 0$, either*

(i) $x_\tau^{LCP} > x_\tau^*$ and both x_τ^{LCP} and x_τ^* are non-increasing for all $\tau \in]t_i : t_{i+1}[$, or
(ii) $x_\tau^{LCP} < x_\tau^*$ and both x_τ^{LCP} and x_τ^* are non-decreasing for all $\tau \in]t_i : t_{i+1}[$.

Proof. First, we consider the case (i), i.e. $x_\tau^{\text{LCP}} > x_\tau^*$.

If $x_{\tau+1}^{\text{LCP}} > x_\tau^{\text{LCP}}$, then $x_{\tau+1}^L = x_{\tau+1}^{\text{LCP}}$ by the LCP algorithm and $x_{\tau+1}^* \geq x_{\tau+1}^L$ by Lemma 3.1. By Lemma 3.6, we get $x_\tau^U = x_\tau^*$ which leads to the contradiction $x_\tau^U = x_\tau^* < x_\tau^{\text{LCP}} \leq x_\tau^U$ (the last inequality uses the definition of the LCP algorithm). Thus x_τ^{LCP} is non-increasing for all $\tau \in]t_i : t_{i+1}[$.

If $x_{\tau+1}^* > x_\tau^*$, then $x_\tau^* = x_\tau^U$ by Lemma 3.6 which is a contradiction to $x_\tau^U \geq x_\tau^{\text{LCP}} > x_t^*$, so x_τ^* is also non-increasing for all $\tau \in]t_i : t_{i+1}[$. By Lemma 3.7 the inequality $x_\tau^{\text{LCP}} > x_\tau^*$ is fulfilled for all $\tau \in]t_i : t_{i+1}[$.

Case (ii) works analogously. □

Now we can calculate the switching cost of the LCP algorithm.

Lemma 3.9. $S_T^L(X^{LCP}) \leq S_T^L(X^*)$

Proof. By Lemma 3.8, both x_τ^{LCP} and x_τ^* are either non-increasing or non-decreasing until there is a time slot t with $x_t^{\text{LCP}} = x_t^*$. Therefore, the switching cost during the time interval $[t_i : t_{i+1}]$ is $\beta(x_{t_i}^* - x_{t_{i-1}}^*)^+$ for both X^{LCP} and X^*. At the end of the workload, X^{LCP} and X^* are maybe in different states. If $x_T^{\text{LCP}} < x_T^*$, then x_τ^{LCP} and x_τ^* are non-decreasing in the corresponding time interval $[t_\kappa : T]$, so the switching cost of LCP is less than the switching cost of the optimal solution. If the $x_T^{\text{LCP}} > x_T^*$, both x_τ^{LCP} and x_τ^* are non-increasing, so there are no switching costs for this time interval. All in all, we get $S_T^L(X^{\text{LCP}}) \leq S_T^L(X^*)$. □

Lemma 3.8 divides the intervals $[t_i : t_{i+1}[$ into two sets: Intervals of case (i) are called *decreasing* intervals, the set of those intervals is denoted by $\mathcal{T}^-$. Intervals of case (ii) are called *increasing* intervals and the set is denoted by $\mathcal{T}^+$. The following lemma is needed to estimate the operating cost of the LCP algorithm.

Lemma 3.10. *For all $\tau \in [t_i : t_{i+1}[\in \mathcal{T}^+$,*

$$\hat{C}_\tau^L(x_\tau^{LCP}) + f_{\tau+1}(x_{\tau+1}^{LCP}) \leq \hat{C}_{\tau+1}^L(x_{\tau+1}^{LCP}). \tag{10}$$

Analogously, for all $\tau \in [t_i : t_{i+1}[\in \mathcal{T}^-$,

$$\hat{C}_\tau^U(x_\tau^{LCP}) + f_{\tau+1}(x_{\tau+1}^{LCP}) \leq \hat{C}_{\tau+1}^U(x_{\tau+1}^{LCP}). \tag{11}$$

The lemma can be proven by using the properties of $\hat{C}_\tau^L$ and $\hat{C}_\tau^U$ shown in Lemma 3.3 and 3.5, the detailed proof can be found in the full version of this paper. We can use Lemma 3.10 to estimate the operating cost of the LCP algorithm.

Lemma 3.11. $R_T(X^{LCP}) \leq R_T(X^*) + \beta \sum_{t=1}^T |x_t^* - x_{t-1}^*|$

Proof. Consider the time interval $[t_i : t_{i+1}[\in \mathcal{T}^+$. By adding the inequalities of Lemma 3.10 for $\tau \in [t_i : t_{i+1}[$, we get

$$\sum_{t=t_i}^{t_{i+1}-1} \hat{C}_t^L(x_t^{\text{LCP}}) + \sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^{\text{LCP}}) \leq \sum_{t=t_i}^{t_{i+1}-1} \hat{C}_{t+1}^L(x_{t+1}^{\text{LCP}})$$

Subtracting the first sum gives

$$\sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^{\text{LCP}}) \leq \hat{C}_{t_{i+1}}^L(x_{t_{i+1}}^{\text{LCP}}) - \hat{C}_{t_i}^L(x_{t_i}^{\text{LCP}})$$

$$= \hat{C}_{t_{i+1}}^L(x_{t_{i+1}}^*) - \hat{C}_{t_i}^L(x_{t_i}^*) = \sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^*) + \beta(x_{t_{i+1}}^* - x_{t_i}^*) \tag{12}$$

The first equality holds because $x_{t_i}^{\text{LCP}} = x_{t_i}^*$ and $x_{t_{i+1}}^{\text{LCP}} = x_{t_{i+1}}^*$. Considering the time interval $[t_i : t_{i+1}[\in \mathcal{T}^-$ yields to the following inequality:

$$\sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^{\text{LCP}}) \leq \hat{C}_{t_{i+1}}^U(x_{t_{i+1}}^{\text{LCP}}) - \hat{C}_{t_i}^U(x_{t_i}^{\text{LCP}})$$

$$= \hat{C}_{t_{i+1}}^U(x_{t_{i+1}}^*) - \hat{C}_{t_i}^U(x_{t_i}^*) = \sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^*) + \beta(x_{t_i}^* - x_{t_{i+1}}^*) \tag{13}$$

In both (12) and (13) the factor after β is positive, so we can write

$$\sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^{\text{LCP}}) \leq \sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^*) + \beta|x_{t_{i+1}}^* - x_{t_i}^*|$$

$$= \sum_{t=t_i}^{t_{i+1}-1} f_{t+1}(x_{t+1}^*) + \beta \sum_{t=t_i}^{t_{i+1}-1} |x_{t+1}^* - x_t^*|$$

By adding all intervals in $\mathcal{T}^+ \cup \mathcal{T}^-$ we get

$$\sum_{t=1}^T f_t(x_t^{\text{LCP}}) \leq \sum_{t=1}^T f_t(x_t^*) + \beta \sum_{t=1}^T |x_t^* - x_{t-1}^*| \qquad \square$$

The term $\beta \sum_{t=1}^T |x_t^* - x_{t-1}^*|$ in Lemma 3.11 is upper bounded by twice the switching cost of the optimal schedule:

Lemma 3.12. $\beta \sum_{t=1}^T |x_t^* - x_{t-1}^*| \leq 2 \cdot S_T^L(X^*)$

This lemma can be proven with simple calculations (see full paper). Now, we are able to show that LCP is 3-competitive.

Theorem 3.13. *The LCP algorithm is 3-competitive.*

Proof. By using Lemma 3.9, 3.11 and 3.12 we get

$$\begin{aligned} C_T^L(X^{\text{LCP}}) &= R_T(X^{\text{LCP}}) + S_T^L(X^{\text{LCP}}) \\ &\leq R_T(X^*) + \beta \sum_{t=1}^T |x_t^* - x_{t-1}^*| + S_T^L(X^*) \\ &\leq R_T(X^*) + 3 \cdot S_T^L(X^*) \leq C_T^L(X^*). \end{aligned}$$ □

4 LOWER BOUNDS

In this section we will show lower bounds for both the discrete and continuous data-center optimization problem. First, we prove that there is no deterministic online algorithm that achieves a competitive ratio better than 3 for the discrete problem. This lower bound demonstrates that the LCP algorithm analyzed in the previous section is optimal. Afterwards, we show that this lower bound

also holds for Lin's model (introduced in [17]) that is a bit more restricted than the general model investigated in the previous sections. A formal definition of Lin's model is given in Section 4.2. Moreover, we give a lower bound for the continuous setting and show that this lower bound holds again for Lin's model. A lower bound of 2 for the general continuous setting was independently shown by Antoniadis et. al. [3]. Finally, we extend our lower bounds to the scenario that an online algorithm has a finite prediction window.

To simplify the analysis, the switching costs are paid for both powering up and powering down. At the end of the workload all servers have to be powered down. This ensures that the total cost remains the same. We will set $\beta = 2$, so changing a server's state will cost $\beta/2 = 1$. Thus the cost of a schedule is defined by

$$C(X) \coloneqq \sum_{t=1}^{T} f_t(x_t) + \sum_{t=1}^{T+1} |x_t - x_{t-1}|$$

with $x_0 \coloneqq x_{T+1} \coloneqq 0$.

4.1 Discrete setting

First, we analyze the discrete setting.

Theorem 4.1. *There is no deterministic online algorithm that achieves a competitive ratio of $c < 3$ for the discrete data-center optimization problem.*

Proof. Assume that there is a deterministic algorithm $\mathcal{A}$ that is $(3-\delta)$-competitive with $\delta > 0$. The adversary will use the functions $\varphi_0(x) = \epsilon|x|$ and $\varphi_1(x) = \epsilon|x-1|$ with $\epsilon \to 0$, so we only need the states 0 and 1, there is no benefit to use other states. If $\mathcal{A}$ is in state 0 or 1, the adversary will send φ_1 or φ_0, respectively.

Let S be the number of time slots where algorithm $\mathcal{A}$ changes the state of a server, i.e. S is the switching cost of $\mathcal{A}$. Let T be length of the whole workload (we will define T later), so for $T - S$ time slots the operating costs of $\mathcal{A}$ are ϵ. Thus, the total cost of $\mathcal{A}$ is

$$C(\mathcal{A}) = (T - S)\epsilon + S.$$

The cost of the optimal offline schedule can be bounded by the minimum of the following two strategies. The first strategy is to stay at one state for the whole workload. If φ_0 is sent more often than φ_1, then this is state 0, else it is state 1. The operating cost is at most $T\epsilon/2$, the switching cost is at most 2, because if we use state 1, we have to switch the state at the beginning and end of the workload. The second strategy is to always switch the state, such that there are no operating costs. In this case the switching cost is at most $S + 2$, because we switch the state after each time $\mathcal{A}$ switches its state as well as possibly at the beginning and the end of the workload. Thus, the cost of the optimal offline schedule is

$$C(X^*) \leq \min(T\epsilon/2 + 2, S + 2). \tag{14}$$

We want to find a lower bound for the competitive ratio $\frac{C(\mathcal{A})}{C(X^*)}$. We distinguish between $S \geq T\epsilon/2$ (case 1) and $S < T\epsilon/2$ (case 2).

In **case 1** the competitive ratio of $\mathcal{A}$ is

$$\frac{C(\mathcal{A})}{C(X^*)} \overset{(14)}{\geq} \frac{(T-S)\epsilon + S}{T\epsilon/2 + 2} = 2 + \frac{S(1-\epsilon) - 4}{T\epsilon/2 + 2}$$
$$\geq 2 + \frac{(T\epsilon/2)(1-\epsilon) - 4}{T\epsilon/2 + 2} = 2 + (1-\epsilon) - \frac{2(1-\epsilon) + 4}{T\epsilon/2 + 2}$$

The last inequality uses $S \geq T\epsilon/2$ that holds for case 1. By setting $T \geq \frac{1}{\epsilon^2}$, we get $\lim_{\epsilon\to 0} T\epsilon = \infty$ and thus $\lim_{\epsilon\to 0} \frac{C(\mathcal{A})}{C(X^*)} = 3$.

In **case 2**, we get

$$\frac{C(\mathcal{A})}{C(X^*)} \overset{(14)}{\geq} \frac{(T-S)\epsilon + S}{S + 2} = (1-\epsilon) + \frac{T\epsilon - 2(1-\epsilon)}{S + 2}$$
$$\geq (1-\epsilon) + \frac{T\epsilon - 2(1-\epsilon)}{T\epsilon/2 + 2} = 3 - \epsilon - \frac{2(1-\epsilon) + 4}{T\epsilon/2 + 2}$$

Again, we set $T \geq \frac{1}{\epsilon^2}$ and get $\lim_{\epsilon\to 0} \frac{C(\mathcal{A})}{C(X^*)} = 3$.

Therefore there is no algorithm with a competitive ratio that is less than 3. We can set T to an arbitrarily large value, so the total cost of $\mathcal{A}$ converges to infinity. □

4.2 Discrete setting: Lin's model

Lin et. al. [17] introduced a more restricted setting as described by equation (2). In this section we show that the lower bound of 3 still holds for this model. The essential differences of Lin's model to the general model are: (1) There is only one convex function for the whole problem instance and (2) there is the additional condition that $x_t \geq \lambda_t$. The different definition of the switching cost does not influence the total cost as already mentioned in the beginning of Section 4.

Theorem 4.2. *There is no deterministic online algorithm for the discrete setting of Lin's model with a competitive ratio of $c < 3$.*

Proof. The general model (examined in the previous sections) is denoted by $\mathcal{G}$ and Lin's model is denoted by $\mathcal{L}$. The states of the model $\mathcal{X} \in \{\mathcal{G}, \mathcal{L}\}$ are indicated by $x_t^{\mathcal{X}}$. We will use the same idea as in the proof of Theorem 4.1, but we have to modify it such that it fits for Lin's model.

We use 2 servers, so the states are $x_t^{\mathcal{L}} \in \{0, 1, 2\}$. Instead of switching between the states 0 and 1 in $\mathcal{G}$, we will switch between 1 and 2 in $\mathcal{L}$, so for $t \in \{1, \ldots, T\}$ we have $x_t^{\mathcal{L}} = x_t^{\mathcal{G}} + 1$. In $\mathcal{L}$ the state 0 is only used at the beginning ($t = 0$) of the workload. This leads to additional switching costs of 1 for both the optimal offline solution and the online algorithm. However, for a sufficiently long workload the total cost converges to infinity, so the constant extra cost does not influence the competitive ratio.

We will apply the same adversary strategy used in the proof of Theorem 4.1. Let $f(z) \coloneqq \epsilon|1 - 2z|$ with $\epsilon \to 0$, let $\beta = 2$. If the adversary in $\mathcal{G}$ sends $\varphi_0(x) = \epsilon|x|$ as function, then we will use $\lambda_t = l_0 \coloneqq 0.5$ which leads to operating cost of

$$x_t^{\mathcal{L}} f\left(l_0/x_t^{\mathcal{L}}\right) = x_t^{\mathcal{L}} \cdot \epsilon \left|1 - \frac{1}{x_t^{\mathcal{L}}}\right| = \epsilon \left|x_t^{\mathcal{L}} - 1\right| = \epsilon \left|x_t^{\mathcal{G}}\right|$$

If the adversary sends $\varphi_1(x) = \epsilon|1-x|$, then we will use $\lambda_t = l_1 = 1$ which leads to operating cost of

$$x_t^{\mathcal{L}} f\left(l_1/x_t^{\mathcal{L}}\right) = x_t^{\mathcal{L}} \cdot \epsilon \left|1 - \frac{2}{x_t^{\mathcal{L}}}\right| = \epsilon \left|x_t^{\mathcal{L}} - 2\right| = \epsilon \left|1 - x_t^{\mathcal{G}}\right|$$

Thus the difference (1) between both models is solved.

For $t \geq 1$ it is not allowed to use the state $x_t^{\mathcal{L}} = 0$, because both l_0 and l_1 are greater than 0. For $x_t^{\mathcal{L}} \in \{1, 2\}$ the inequality $x_t \geq \lambda_t$ is always fulfilled, so the difference (2) is solved too. □

4.3 Continuous setting

In this section we analyze the continuous setting of the data-center optimization problem.

Theorem 4.3. *There is no deterministic online algorithm for the continuous data-center optimization problem that achieves a competitive ratio that is less than 2.*

The proof consists of two parts. First we will construct an algorithm $\mathcal{B}$ whose competitive ratio is greater than $2-\delta$ for an arbitrary small δ. Then we will show that the competitive ratio of any deterministic algorithm that differs from $\mathcal{B}$ is greater than 2.

For the first part we use an adversary that uses $\varphi_0(x) = \epsilon|x|$ and $\varphi_1(x) = \epsilon|1-x|$ as functions where $\epsilon \to 0$. Let b_t be the state of $\mathcal{B}$ at time t. If the function φ_0 arrives, then the next state b_{t+1} is $\max\{b_t - \epsilon/2, 0\}$. If φ_1 arrives, the next state is $b_{t+1} := \min\{b_t + \epsilon/2, 1\}$. The algorithm starts at $b_0 = 0$, so $b_t \in [0,1]$ is fulfilled for all t. Note that algorithm $\mathcal{B}$ is equivalent to Bansal's algorithm given in [5] for the special case of φ_0 and φ_1 functions. As shown in the full version of this paper, the competitive ratio of $\mathcal{B}$ is at least 2.

For the second part we use the following adversary strategy. Let a_t be the state of $\mathcal{A}$ at time t. If $a_t < b_t$ or $a_t = 0$, the adversary sends φ_1 as next function. If $a_t > b_t$ or $a_t = 1$, the adversary sends φ_0. For the other cases the adversary can choose an arbitrary function. By doing this, it can be shown that the cost of $\mathcal{A}$ is always greater than or equal to the cost of $\mathcal{B}$, so the competitive ratio of $\mathcal{A}$ is at least 2. The detailed analysis can be found in the full paper.

4.4 Continuous setting: Lin's model

Analogously to the discrete setting, in this section we want to show that the lower bound of 2 for the continuous data-center optimization problem still holds for Lin's model described in Section 4.3.

Theorem 4.4. *There is no deterministic online algorithm for the continuous setting of Lin's model with a competitive ratio of $c < 2$.*

To prove this theorem we will apply the same adversary strategy used in the previous section. We set $f(z) := \epsilon|1-kz|$ with $\epsilon \to 0$ and $k \to \infty$. If the adversary in the general model sends $\varphi_0(x) = \epsilon|x|$ as function, then we will use $\lambda_t = 0$, if it sends $\varphi_1(x) = \epsilon|1-x|$, then we will use $\lambda_t = 1/k$. As shown in the full paper, this leads to the same operating cost functions as in the general model.

4.5 Online algorithms with prediction window

So far, we have considered online algorithms that at time t only know the arriving function f_t in determining the next state. In contrast, an offline algorithm knows the whole function sequence F. There are models between these edge cases. An online algorithm with a *prediction window* of length w, at any time t, can not only use the function f_t but the function set $\{f_t, \dots, f_{t+w}\}$ to choose the state x_t. This problem extension was also defined by Lin et al. [17, 19]. If w has a constant size (i.e. w is independent of T), then the lower bounds developed in the previous sections still holds as the following theorem shows. We develop the lower bounds for Lin's model so that they hold for the general model as well.

Theorem 4.5. *Let $w \in \mathbb{N}$ and $\delta > 0$ be arbitrary constants. There is no deterministic online algorithm with a prediction window of length w that achieves a competitive ratio of $3-\delta$ in the discrete setting or $2-\delta$ in the continuous setting for Lin's model.*

The basic idea for proving this theorem is to use a worst case workload for online algorithms without prediction window and send a scaled version of each function several times (dependent on w), such that there is no advantage in knowing the next w functions. Details can be found in the full version of this paper.

REFERENCES

[1] L.L.H. Andrew, S. Barman, K. Ligett, M. Lin, A. Meyerson, A. Roytman and A. Wierman. A tale of two metrics: Simultaneous bounds on competitiveness and regret. *Proc. 26th Annual Conference on Learning Theory (COLT'13)*, 741–763, 2013.

[2] A. Antoniadis, N. Barcelo, M. Nugent, K. Pruhs, K. Schewior and M. Scquizzato. Chasing convex bodies and functions. *Proc. 12th Latin American Symposium on Theoretical Informatics (LATIN'16)*, 68–81, 2016.

[3] A. Antoniadis and K. Schewior. A tight lower bound for online convex optimization with switching costs. Announced at the *15th Workshop on Approximation and Online Algorithms (WAOA'17)*, 2017.

[4] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report No. UCB/EECS-2009-28. EECS Department, University of California, Berkeley, 2009.

[5] N. Bansal, A. Gupta, R. Krishnaswamy, K. Pruhs, K. Schewior and C. Stein. A 2-competitive algorithm for online convex optimization with switching costs. *Proc. 18th. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX15)*, 96–109, 2015.

[6] L.A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[7] T. Bawden. Global warming: Data centres to consume three times as much energy in next decade, experts warn. `http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html`, 2016.

[8] K.G. Brill. The Invisible Crisis in the Data Center: The Economic Meltdown of Moore's Law. *white paper, Uptime Institute*, 2–5, 2007.

[9] M. Dayarathna, Y. Wen and R. Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys and Tutorials*, 18(1):732–794, 2016.

[10] A. Gandhi and M. Harchol-Balter. How data center size impacts the effectiveness of dynamic power management. *49th Annual Allerton Conference*, 1164–1169, 2011.

[11] A. Gandhi, M. Harchol-Balter and I.J.B.F. Adan. Server farms with setup costs. *Perform. Eval.*, 67(11):1123–1138, 2010.

[12] Z.J. Haas and S. Gu. On power management policies for data centers. *Proc. IEEE Int. Conf. on Data Science and Data Intensive Systems (DSDIS)*, 404–411, 2015.

[13] J. Hamilton. Cost of Power in Large-Scale Data Centers. `http://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/`

[14] S. Khuller, J. Li and B. Saha. Energy efficient scheduling via partial shutdown. *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, 1360–1372, 2010.

[15] J. Li and S. Khuller. Generalized machine activation problems. *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 80–94, 2011.

[16] M. Lin, Z. Liu, A. Wierman and L.L.H. Andrew. Online algorithms for geographical load balancing. *Proc. International Green Computing Conference (IGCC'12)*, 1–10, 2012.

[17] M. Lin, A. Wierman, L.L.H. Andrew and E. Thereska. Dynamic right-sizing for power-proportional data centers. *Proc. 30th IEEE International Conference on Computer Communications (INFOCOM'11)*, 1098–1106, 2011.

[18] M. Lin, A. Wierman, L.L.H. Andrew and E. Thereska. Online dynamic capacity provisioning in data centers. *Proc. 49th Annual Allerton Conference on Communication, Control, and Computing*, 1159–1163, 2011.

[19] M. Lin, A. Wierman, L.L.H. Andrew and E. Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Trans. Netw.*, 21(5):1378–1391, 2013.

[20] Z. Liu, M. Lin, A. Wierman, S.H. Low, L.L.H. Andrew. Greening geographical load balancing. *IEEE/ACM Trans. Netw.*, 23(2):657–671, 2015.

[21] P. Schmid and A. Roos. Overclocking Core i7: Power Versus Performance. Idle/Peak Power Consumption Analysis. `http://www.tomshardware.com/reviews/overclock-core-i7,2268-10.html`, 2009.

[22] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo and W. Lintner. United States data center energy usage report. Lawrence Berkeley National Laboratory, LBNL-1005775, 2016.

[23] K. Wang, M. Lin, F. Ciucu, A. Wierman and C. Lin. Characterizing the impact of the workload on the value of dynamic resizing in data centers. *Perform. Eval.*, 85–86:1–18, 2015.

Appendix C

Algorithms for Energy Conservation in Heterogeneous Data Centers, CIAC 2021

This chapter has been published as **peer-reviewed conference paper** [AQ21c].

> Susanne Albers and Jens Quedenfeld. Algorithms for energy conservation in heterogeneous data centers. In *Algorithms and Complexity - 12th International Conference, CIAC 2021*. Springer, 2021.

A full version of this paper containing all proofs was recently published in a special issue of the journal *Theoretical Computer Science* [AQ21a]. A freely accessible preprint of the extended version can be found on arXiv [AQ21b], see `https://arxiv.org/abs/2107.14672`.

Synopsis. We analyze the right-sizing problem of heterogeneous data centers with d different server types. We consider the discrete setting, i.e., the number of active servers must be integral. The operating cost of a server is load- and time-independent, so it only depends on the server type. Each server can process one job per time slot. We develop a $2d$-competitive deterministic online algorithm for this problem and show that no deterministic online algorithm can achieve a better competitive ratio. Furthermore, we devise a randomized version of our algorithm that attains a competitive ratio of $\frac{e}{e-1}d \approx 1.582d$ against an oblivious adversary.

Contributions of thesis author. The thesis author developed the algorithms and the lower bound including all proofs. Furthermore, he wrote the manuscript.

Algorithms for Energy Conservation in Heterogeneous Data Centers

Susanne Albers and Jens Quedenfeld(✉)

Technical University of Munich, 85748 Garching, Germany
{albers,jens.quedenfeld}@in.tum.de

Abstract. Power consumption is the major cost factor in data centers. It can be reduced by dynamically right-sizing the data center according to the currently arriving jobs. If there is a long period with low load, servers can be powered down to save energy. For identical machines, the problem has already been solved optimally by [25] and [1].

In this paper, we study how a data-center with heterogeneous servers can dynamically be right-sized to minimize the energy consumption. There are d different server types with various operating and switching costs. We present a deterministic online algorithm that achieves a competitive ratio of $2d$ as well as a randomized version that is $1.58d$-competitive. Furthermore, we show that there is no deterministic online algorithm that attains a competitive ratio smaller than $2d$. Hence our deterministic algorithm is optimal. In contrast to related problems like convex body chasing and convex function chasing [17,30], we investigate the discrete setting where the number of active servers must be an integral, so we gain truly feasible solutions.

1 Introduction

Energy management is an important issue in data centers. A huge amount of a data center's financial budget is spent on electricity that is needed to operate the servers as well as to cool them [12,20]. However, server utilization is typically low. In fact there are data centers where the average server utilization is as low as 12% [16]; only for a few days a year is full processing power needed. Unfortunately, idle servers still consume about half of their peak power [29]. Therefore, right-sizing a data center by powering down idle servers can save a significant amount of energy. However, shutting down a server and powering it up immediately afterwards incurs much more cost than holding the server in the active state during this time period. The cost for powering up and down does not only contain the increased energy consumption but also, for example, wear-and-tear costs or the risk that the server does not work properly after restarting [26]. Consequently, algorithms are needed that manage the number of active servers to minimize the total cost, without knowing when new jobs will arrive in the future. Since about 3% of the global electricity production

Work supported by the European Research Council, Grant Agreement No. 691672.

T. Calamoneri and F. Corò (Eds.): CIAC 2021, LNCS 12701, pp. 75–89, 2021.
https://doi.org/10.1007/978-3-030-75242-2_5

is consumed by data centers [11], a reduction of their energy consumption can also decrease greenhouse emissions. Thus, right-sizing data centers is not only important for economical but also for ecological reasons.

Modern data centers usually contain heterogeneous servers. If the capacity of a data center is no longer sufficient, it is extended by including new servers. The old servers are still used however. Hence, there are different server types with various operating and switching costs in a data center. Heterogeneous data centers may also include different processing architectures. There can be servers that use GPUs to perform massive parallel calculations. However, GPUs are not suitable for all jobs. For example, tasks with many branches can be computed much faster on common CPUs than on GPUs [31].

Problem Formulation. We consider a data center with d different server types. There are m_j servers of type j. Each server has an active state where it is able to process jobs, and an inactive state where no energy is consumed. Powering up a server of type j (i.e., switching from the inactive into the active state) incurs a cost of β_j (called *switching* cost); powering down does not cost anything. We consider a finite time horizon consisting of the time slots $\{1, \dots, T\}$. For each time slot $t \in \{1, \dots, T\}$, jobs of total volume $\lambda_t \in \mathbb{N}_0$ arrive and have to be processed during the time slot. There must be at least λ_t active servers to process the arriving jobs. We consider a basic setting where the operating cost of a server of type j is load and time independent and denoted by $l_j \in \mathbb{R}_{\geq 0}$. Hence, an active server incurs a constant but type-dependent operating cost per time slot.

A schedule X is a sequence $\boldsymbol{x}_1, \dots, \boldsymbol{x}_T$ with $\boldsymbol{x}_t = (x_{t,1}, \dots, x_{t,d})$ where each $x_{t,j}$ indicates the number of active servers of type j during time slot t. At the beginning and the end of the considered time horizon all servers are shut down, i.e., $\boldsymbol{x}_0 = \boldsymbol{x}_{T+1} = (0, \dots, 0)$. A schedule is called *feasible* if there are enough active servers to process the arriving jobs and if there are not more active servers than available, i.e., $\sum_{j=1}^{d} x_{t,j} \geq \lambda_t$ and $x_{t,j} \in \{0, 1, \dots, m_j\}$ for all $t \in \{1, \dots, T\}$ and $j \in \{1, \dots, d\}$. The cost of a feasible schedule is defined by

$$C(X) \coloneqq \sum_{t=1}^{T} \left(\sum_{j=1}^{d} l_j x_{t,j} + \sum_{j=1}^{d} \beta_j (x_{t,j} - x_{t-1,j})^+ \right) \tag{1}$$

where $(x)^+ \coloneqq \max(x, 0)$. The switching cost is only paid for powering up. However, this is not a restriction, since all servers are inactive at the beginning and end of the workload. Thus the cost of powering down can be folded into the cost of powering up. A problem instance is specified by the tuple $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{l}, \Lambda)$ where $\boldsymbol{m} = (m_1, \dots, m_d)$, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_d)$, $\boldsymbol{l} = (l_1, \dots, l_d)$ and $\Lambda = (\lambda_1, \dots, \lambda_T)$. The task is to find a schedule with minimum cost.

We focus on the central case without *inefficient* server types. A server type j is called *inefficient* if there is another server type $j' \neq j$ with both smaller (or equal) operating and switching costs, i.e., $l_j \geq l_{j'}$ and $\beta_j \geq \beta_{j'}$. This assumption is natural because a better server type with a lower operating cost usually has a higher switching cost. An inefficient server of type j is only powered up, if all servers of all types j' with $\beta_{j'} \leq \beta_j$ and $l_{j'} \leq l_j$ are already running. Therefore,

excluding inefficient servers is not a relevant restriction in practice. In related work, Augustine et al. [6] exclude inefficient states when operating a single server.

Our Contribution. We analyze the online setting of this problem where the job volumes λ_t arrive one-by-one. The vector of the active servers $\boldsymbol{x}_t$ has to be determined without knowledge of future jobs $\lambda_{t'}$ with $t' > t$. A main contribution of our work, compared to previous results, is that we investigate heterogeneous data centers and examine the online setting when truly feasible (integral) solutions are sought.

In Sect. 2, we present a $2d$-competitive deterministic online algorithm, i.e., the total cost of the schedule calculated by our algorithm is at most $2d$ times larger than the cost of an optimal offline solution. Roughly, our algorithm works as follows. It calculates an optimal schedule for the jobs received so far and ensures that the operating cost of the active servers is at most as large as the operating cost of the active servers in the optimal schedule. If this is not the case, servers with high operating cost are replaced by servers with low operating cost. If a server is not used for a specific duration depending on its switching and operating costs, it is shut down.

In Sect. 3, we devise a randomized version of our algorithm achieving a competitive ratio of $\frac{e}{e-1}d \approx 1.582d$ against an oblivious adversary.

In Sect. 4, we show that there is no deterministic online algorithm that achieves a competitive ratio smaller than $2d$. Therefore, our algorithm is optimal. Additionally, for a data center that contains m unique servers (that is $m_j = 1$ for all $j \in \{1, \ldots, d\}$), we show that the best achievable competitive ratio is $2m$.

Related Work. The design of energy-efficient algorithms has received quite some research interest over the last years, see e.g. [3,10,21] and references therein. Specifically, data center right-sizing has attracted considerable attention lately. Lin and Wierman [25,26] analyzed the data-center right-sizing problem for data centers with identical servers ($d = 1$). The operating cost is load dependent and modeled by a convex function. In contrast to our setting, continuous solutions are allowed, i.e., the number of active server x_t can be fractional. This allows for other techniques in the design and analysis of an algorithm, but the created schedules cannot be used directly in practice. They gave a 3-competitive deterministic online algorithm for this problem. Bansal et al. [9] improved this result by randomization and developed a 2-competitive online algorithm. In our previous paper [1] we showed that 2 is a lower bound for randomized algorithms in the continuous setting; this result was independently shown by [4]. Furthermore, we analyzed the discrete setting of the problem where the number of active servers is integral ($x_t \in \mathbb{N}_0$). We presented a 3-competitive deterministic and a 2-competitive randomized online algorithm. Moreover, we proved that these competitive ratios are optimal.

Data-center right-sizing of heterogeneous data centers is related to convex function chasing, which is also known as smoothed online convex optimization [15]. At each time slot t, a convex function f_t arrives. The algorithm then has to choose a point $\boldsymbol{x}_t$ and pay the cost $f_t(\boldsymbol{x}_t)$ as well as the movement cost $\|\boldsymbol{x}_t - \boldsymbol{x}_{t-1}\|$ where $\|\cdot\|$ is any metric. The problem described by Eq. (1) is a

special case of convex function chasing if fractional schedules are allowed, i.e., $x_{t,j} \in [0, m_j]$ instead of $x_{t,j} \in \{0, \dots, m_j\}$. The operating cost $\sum_{j=1}^{d} l_j x_{t,j}$ in Eq. (1) together with the feasibility requirements can be modeled as a convex function that is infinite for $\sum_{j=1}^{d} x_{t,j} < \lambda_t$ and $x_{t,j} \notin [0, m_j]$. The switching cost equals the Manhattan metric if the number of servers is scaled appropriately. Sellke [30] gave a $(d+1)$-competitive algorithm for convex function chasing. A similar result was found by Argue et al. [5].

In the discrete setting, convex function chasing has at least an exponential competitive ratio, as the following setting shows. Let $m_j = 1$ and $\beta_j = 1$ for all $j \in \{1, \dots, d\}$, so the possible server configurations are $\{0,1\}^d$. The arriving convex functions f_t are infinite for the current position $\boldsymbol{x}_{t-1}$ of the online algorithm and 0 for all other positions $\{0,1\}^d \backslash \{\boldsymbol{x}_{t-1}\}$. After $T := 2^d - 1$ functions arrived, the switching cost paid by the algorithm is at least $2^d - 1$ (otherwise it has to pay infinite operating costs), whereas the offline schedule can go directly to a position without any operating cost and only pays a switching cost of at most d.

Already for the 1-dimensional case (i.e. identical machines), it is not trivial to round a fractional schedule without increasing the competitive ratio (see [26] and [2]). In d-dimensional space, it is completely unclear, if continuous solutions can be rounded without arbitrarily increasing the total cost. Simply rounding up can lead to arbitrarily large switching costs, for example if the fractional solution rapidly switches between 1 and $1 + \epsilon$. Using a randomized rounding scheme like in [2] (that was used for homogeneous data centers) independently for each dimension can result in an infeasible schedule (for example, if $\lambda_t = 1$ and $\boldsymbol{x}_t = (1/d, \dots, 1/d)$ is rounded down to $(0, \dots, 0)$). Therefore, Sellke's result does not help us for analyzing the discrete setting. Other publications handling convex function chasing or convex body chasing are [8,13,17].

Goel and Wierman [19] developed a $(3 + \mathcal{O}(1/\mu))$-competitive algorithm called Online Balanced Descent (OBD) for convex function chasing, where the arriving functions were required to be μ-strongly convex. We remark that the operating cost defined by Eq. (1) is not strongly convex, i.e., $\mu = 0$. Hence their result cannot be used for our problem. A similar result is given by Chen et al. [15] who showed that OBD is $(3 + \mathcal{O}(1/\alpha))$-competitive if the arriving functions are locally α-polyhedral. In our case, $\alpha = \min_{j \in \{1,\dots,d\}} l_j/\beta_j$, so α can be arbitrarily small depending on the problem instance.

Another similar problem is the Parking Permit Problem by Meyerson [28]. There are d different permits which can be purchased for β_j dollars and have a duration of D_j days. Certain days are driving days where at least one parking permit is needed ($\lambda_t \in \{0,1\}$). The permit cost corresponds to our switching cost. However, the duration of the permit is fixed to D_j, whereas in our problem the online algorithm can choose for each time slot if it wants to power down a server. Furthermore, there is no operating cost. Even if each server type is replaced by an infinite number of permits with the duration t and the cost $\beta_j + l_j \cdot t$, it is still a different problem, because the algorithm has to choose the time slot for powering down in advance (when the server is powered up).

Data-center right-sizing of heterogeneous data centers is related to geographical load balancing analyzed in [24] and [27]. Other applications are shown in [7,14,18,22,23,32,33].

2 Deterministic Online Algorithm

In this section we present a deterministic $2d$-competitive online algorithm for the problem described in the preceding section. The basic idea of our algorithm is to calculate an optimal schedule for the problem instance that ends at the current time slot. Based on this schedule, we decide when a server is powered up. If a server is idle for a specific time, it is powered down.

Formally, given the original problem instance $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{l}, \Lambda)$, the shortened problem instance $\mathcal{I}^t$ is defined by $\mathcal{I}^t := (t, d, \boldsymbol{m}, \boldsymbol{\beta}, \boldsymbol{l}, \Lambda^t)$ with $\Lambda^t = (\lambda_1, \ldots, \lambda_t)$. Let $\hat{X}^t$ denote an optimal schedule for $\mathcal{I}^t$ and let $X^{\mathcal{A}}$ be the schedule calculated by our algorithm $\mathcal{A}$.

W.l.o.g. there are no server types with the same operating and switching costs, i.e., $\beta_j = \beta_{j'}$ and $l_j = l_{j'}$ implies $j = j'$. Furthermore, let $l_1 > \cdots > l_d$, i.e., the server types are sorted by their operating costs. Since inefficient server types are excluded, this implies that $\beta_1 < \cdots < \beta_d$.

Let $[n] := \{1, \ldots, n\}$ where $n \in \mathbb{N}$. We separate a problem instance into $m := \sum_{j=1}^{d} m_j$ lanes. At time slot t, there is a single job in lane $k \in [m]$, if and only if $k \leq \lambda_t$. We can assume that $\lambda_t \leq m$ holds for all $t \in [T]$, because otherwise there is no feasible schedule for the problem instance. Let X be an arbitrary feasible schedule with $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,d})$. We define

$$y_{t,k} := \begin{cases} \max\{j \in [d] \mid \sum_{j'=j}^{d} x_{t,j'} \geq k\} & \text{if } k \in \left[\sum_{j=1}^{d} x_{t,j}\right] \\ 0 & \text{else} \end{cases} \tag{2}$$

to be the server type that handles the k-th lane during time slot t. If $y_{t,k} = 0$, then there is no active server in lane k at time slot t. By definition, the values $y_{t,1}, \ldots, y_{t,m}$ are sorted in descending order, i.e., $y_{t,k} \geq y_{t,k'}$ for $k < k'$. Note that $y_{t,k} = 0$ implies $\lambda_t < k$, because otherwise there are not enough active servers to handle the jobs at time t. For the schedule $\hat{X}^t$, the server type used in lane k at time slot t' is denoted by $\hat{y}^t_{t',k}$. Our algorithm calculates $y^{\mathcal{A}}_{t,k}$ directly, the corresponding variables $x^{\mathcal{A}}_{t,j}$ can be determined by $x^{\mathcal{A}}_{t,j} = |\{k \in [m] \mid y^{\mathcal{A}}_{t,k} = j\}|$.

Our algorithm works as follows: First, an optimal solution $\hat{X}^t$ is calculated. If there are several optimal schedules, we choose a schedule that fulfills the inequality $\hat{y}^t_{t',k} \geq \hat{y}^{t-1}_{t',k}$ for all time slots $t' \in [t]$ and lanes $k \in [m]$, so $\hat{X}^t$ never uses smaller server types than the previous schedule $\hat{X}^{t-1}$. We will see in Lemma 2 that such a schedule exists and how to construct it.

If there is a server type j with $l_j = 0$, then in an optimal schedule such a server can be powered up before it is needed, although $\lambda_t = 0$ holds for this time slot. Similarly, such a server can run for more time slots than necessary. W.l.o.g. let $\hat{X}^t$ be a schedule where servers are powered up as late as possible and powered down as early as possible.

Beginning from the lowest lane ($k = 1$), it is ensured that $\mathcal{A}$ uses a server type that is not smaller than the server type used by $\hat{X}^t$, i.e., $y^{\mathcal{A}}_{t,k} \geq \hat{y}^t_{t,k}$ must be fulfilled. If the server type $y^{\mathcal{A}}_{t-1,k}$ used in the previous time slot is smaller than $\hat{y}^t_{t,k}$, it is powered down and server type $\hat{y}^t_{t,k}$ is powered up. A server of type j that is not replaced by a greater server type stays active for $\bar{t}_j := \lfloor \beta_j / l_j \rfloor$ time slots. If $\hat{X}^t$ uses a smaller server type $j' \leq j$ in the meantime, then server type j will run for at least $\bar{t}_{j'}$ further time slots (including time slot t). Formally, a server of type j in lane k is powered down at time slot t, if $\hat{y}^{t'}_{t',k} \neq j'$ holds for all server types $j' \leq j$ and time slots $t' \in [t - \bar{t}_{j'} + 1 : t]$ with $[a : b] := \{a, a+1, \ldots, b\}$.

The pseudocode below clarifies how algorithm $\mathcal{A}$ works. The variables e_k for $k \in [m]$ store the time slot when the server in the corresponding lane will be powered down.

Algorithm 1. Algorithm $\mathcal{A}$

1: **for** $t := 1$ **to** T **do**
2: Calculate $\hat{X}^t$ such that $\hat{y}^t_{t',k} \geq \hat{y}^{t-1}_{t',k}$ for all $t' \in [t]$ and $k \in [m]$
3: **for** $k := 1$ **to** m **do**
4: **if** $y^{\mathcal{A}}_{t-1,k} < \hat{y}^t_{t,k}$ **or** $t \geq e_k$ **then**
5: $y^{\mathcal{A}}_{t,k} := \hat{y}^t_{t,k}$
6: $e_k := t + \bar{t}_{y^{\mathcal{A}}_{t,k}}$
7: **else**
8: $y^{\mathcal{A}}_{t,k} := y^{\mathcal{A}}_{t-1,k}$
9: $e_k := \max\{e_k, t + \bar{t}_{\hat{y}^t_{t,k}}\}$ where $\bar{t}_0 := 0$

Structure of Optimal Schedules. Before we can analyze the competitiveness of algorithm $\mathcal{A}$, we have to show that an optimal schedule with the desired properties required by line 2 actually exists. First, we will investigate basic properties of optimal schedules. In an optimal schedule $\hat{X}$, a server of type j that runs in lane k does not change the lane while running. Formally, if $\hat{y}_{t-1,k} = j$ and $\hat{y}_{t,k} \neq j$, then there exists no other lane $k' \neq k$ with $\hat{y}_{t-1,k'} \neq j$ and $\hat{y}_{t,k'} = j$. Furthermore, a server is only powered up or powered down if the number of jobs is increased or decreased, respectively. Finally, in a given lane k, the server type does not change immediately, i.e., there must be at least one time slot, where no server is running in lane k. These properties are proven in the full version of this paper.

Given the optimal schedules $\hat{X}^u$ and $\hat{X}^v$ with $u < v$, we construct a *minimum* schedule $X^{\min(u,v)}$ with $y^{\min(u,v)}_{t,k} := \min\{\hat{y}^u_{t,k}, \hat{y}^v_{t,k}\}$. Furthermore, we construct a *maximum* schedule $X^{\max(u,v)}$ as follows. Let $z_l(t,k)$ be the last time slot $t' < t$ with $\hat{y}^u_{t',k} = \hat{y}^v_{t',k} = 0$ (no active servers in both schedules) and let $z_r(t,k)$ be the first time slot $t' > t$ with $\hat{y}^u_{t',k} = \hat{y}^v_{t',k} = 0$. The schedule $X^{\max(u,v)}$ is defined by

$$y^{\max(u,v)}_{t,k} := \max_{t' \in [z_l(t,k)+1 : z_r(t,k)-1]} \{\hat{y}^u_{t',k}, \hat{y}^v_{t',k}\}. \tag{3}$$

Another way to construct $X^{\max(u,v)}$ is as follows. First, we take the maximum of both schedules (analogously to $X^{\min(u,v)}$). However, this can lead to situations where the server type changes immediately, so the necessary condition for optimal schedules would not be fulfilled. Therefore, we replace the lower server type by the greater one until there are no more immediate server changes. This construction is equivalent to Eq. (3).

We will see in Lemma 2 that the *maximum* schedule is an optimal schedule for $\mathcal{I}^v$ and fulfills the property required by algorithm $\mathcal{A}$ in line 2, which says that the server type used in lane k at time t never decreases when the considered problem instance is expanded. To prove this property, first we have to show that $X^{\min(u,v)}$ and $X^{\max(u,v)}$ are feasible schedules for the problem instances $\mathcal{I}^u$ and $\mathcal{I}^v$, respectively.

Lemma 1. *$X^{min(u,v)}$ and $X^{max(u,v)}$ are feasible for $\mathcal{I}^u$ and $\mathcal{I}^v$, respectively.*

The proof can be found in the full version of this paper. Now, we are able to show that the maximum schedule is optimal for the problem instance $\mathcal{I}^v$.

Lemma 2. *Let $u, v \in [T]$ with $u < v$. $X^{max(u,v)}$ is optimal for $\mathcal{I}^v$.*

The works roughly as follows (the complete proof can be found in the full paper). First, we prove that the sum of the operating costs of $\hat{X}^u$ and $\hat{X}^v$ is greater than or equal to the sum of the operating cost of $X^{\min(u,v)}$ and $X^{\max(u,v)}$. Each server activation in $X^{\min(u,v)}$ and $X^{\max(u,v)}$ can be mapped to exactly one server activation in $\hat{X}^u$ and $\hat{X}^v$ with the same or a greater server type. Therefore, $C(X^{\min(u,v)}) + C(X^{\max(u,v)}) \leq C(\hat{X}^u) + C(\hat{X}^v)$ holds and by using Lemma 1, it is shown that $X^{\max(u,v)}$ is optimal for $\mathcal{I}^v$.

Feasibility. In the following, let $\{\hat{X}^1, \ldots, \hat{X}^T\}$ be optimal schedules that fulfill the inequality $\hat{y}^t_{t',k} \geq \hat{y}^{t-1}_{t',k}$ for all $t, t' \in [T]$ and $k \in [m]$ as required by algorithm $\mathcal{A}$. Lemma 2 ensures that such a schedule sequence exists (and also shows how to construct it). Before we can prove that algorithm $\mathcal{A}$ is $2d$-competitive, we have to show that the computed schedule $X^{\mathcal{A}}$ is feasible. In an optimal schedule $\hat{X}^t$, the values $\hat{y}^t_{t',1}, \ldots, \hat{y}^t_{t',m}$ are sorted in descending order by definition. This also holds for schedule calculated by our algorithm.

Lemma 3. *For all time slots $t \in [T]$, the values $y^{\mathcal{A}}_{t,1}, \ldots, y^{\mathcal{A}}_{t,m}$ are sorted in descending order, i.e., $y^{\mathcal{A}}_{t,k} \geq y^{\mathcal{A}}_{t,k'}$ for $k < k'$.*

The proof uses the fact that the running times $\bar{t}_j$ are sorted in ascending order, i.e., $\bar{t}_1 \leq \cdots \leq \bar{t}_d$, because $l_1 > \cdots > l_d$ and $\beta_1 < \cdots < \beta_d$. In other words, the higher the server type is, the longer it stays in the active state. See the full paper for more details. By means of Lemma 3, we are able to prove the feasibility of $X^{\mathcal{A}}$.

Lemma 4. *The schedule $X^{\mathcal{A}}$ is feasible.*

Proof Idea. A schedule is feasible, if (1) there are enough active servers to handle the incoming jobs (i.e., $\sum_{j=1}^{d} x^{\mathcal{A}}_{t,j} \geq \lambda_t$) and (2) there are not more active servers

than available (i.e., $x^{\mathcal{A}}_{t,j} \leq m_j$). The first property directly follows from the definition of algorithm $\mathcal{A}$, since $\sum_{j=1}^{d} x^{\mathcal{A}}_{t,j} \geq \sum_{j=1}^{d} \hat{x}^t_{t,j} \geq \lambda_t$. Lemma 3 is used to prove that $x^{\mathcal{A}}_{t,j} \leq m_j$ is always fulfilled after setting $y^{\mathcal{A}}_{t,k}$ in line 5 or 8. The complete proof is presented in the full paper. □

Competitiveness. To show the competitiveness of $\mathcal{A}$, we divide the schedule $X^{\mathcal{A}}$ into blocks $A_{t,k}$ with $t \in [T]$ and $k \in [m]$. Each block $A_{t,k}$ is described by its creation time t, its start time $s_{t,k}$, its end time $e_{t,k}$, the used server type $j_{t,k}$ and the corresponding lane k. The start time is the time slot when $j_{t,k}$ is powered up and the end time is the first time slot, when $j_{t,k}$ is inactive, i.e., during the time interval $[s_{t,k} : e_{t,k} - 1]$ the server of type $j_{t,k}$ is in the active state.

There are two types of blocks: *new* blocks and *extended* blocks. A *new* block starts when a new server is powered up, i.e., lines 5 and 6 of algorithm $\mathcal{A}$ are executed because $y^{\mathcal{A}}_{t-1,k} < \hat{y}^t_{t,k}$ or $t \geq e_k \wedge y^{\mathcal{A}}_{t-1,k} > \hat{y}^t_{t,k} \wedge \hat{y}^t_{t,k} > 0$ (in words: the previous block ends and $\hat{X}^t$ has an active server in lane k, but the server type is smaller than the server type used by $\mathcal{A}$ in the previous time slot). It ends after $\bar{t}_{y^{\mathcal{A}}_{t,k}}$ time slots. Thus $s_{t,k} \coloneqq t$ and $e_{t,k} \coloneqq t + \bar{t}_{y^{\mathcal{A}}_{t,k}}$ (i.e., $e_{t,k}$ equals e_k after executing line 6).

An *extended* block is created when the running time of a server is extended, i.e., the value of e_k is updated, but the server type remains the same (that is $y^{\mathcal{A}}_{t-1,k} = y^{\mathcal{A}}_{t,k}$). We have $e_{t,k} \coloneqq t + \bar{t}_{\hat{y}^t_{t,k}}$ (i.e., the value of e_k after executing line 9 or 6) and $s_{t,k} \coloneqq e_{t',k}$, where $A_{t',k}$ is the previous block in the same lane. Note that an *extended* block can be created not only in line 9, but also in line 6, if $t = e_k$ and $y^{\mathcal{A}}_{t-1,k} = \hat{y}^t_{t,k}$. If line 8 and 9 are executed, but the value of e_k does not change (because $t + \bar{t}_{\hat{y}^t_{t,k}}$ is smaller than or equal to the previous value of e_k), then the block $A_{t,k}$ does not exist.

Let $d_{t,k} \coloneqq e_{t,k} - s_{t,k}$ be the duration of the block $A_{t,k}$ and let $C(A_{t,k})$ be the cost caused by $A_{t,k}$ if the block $A_{t,k}$ exists or 0 otherwise. The next lemma describes how the cost of a block can be estimated.

Lemma 5. *The cost of the block $A_{t,k}$ is upper bounded by*

$$C(A_{t,k}) \leq \begin{cases} 2\beta_{j_{t,k}} & \text{if } A_{t,k} \text{ is a new block} \\ l_{j_{t,k}} d_{t,k} & \text{if } A_{t,k} \text{ is an extended block} \end{cases} \tag{4}$$

The lemma follows from the definition of $\bar{t}_j$ (see the full paper for more details). To show the competitiveness of algorithm $\mathcal{A}$, we introduce another variable that will be used in Lemmas 7 and 8. Let

$$\tilde{y}^u_{t,k} \coloneqq \max_{t' \in [t:u]} \hat{y}^{t'}_{t',k}$$

be the largest server type used in lane k by the schedule $\hat{X}^{t'}$ at time slot t' for $t' \in [t : u]$. The next lemma shows that $\tilde{y}^u_{t,k}$ is monotonically decreasing with respect to t as well as k and increasing with respect to u.

Lemma 6. *Let $u' \geq u$, $t' \leq t$ and $k' \leq k$. It is $\tilde{y}^u_{t,k} \leq \tilde{y}^{u'}_{t',k'}$.*

This lemma follows from the definition of $\tilde{y}^u_{t,k}$. A proof can be found in the full paper. The cost of schedule X in lane k during time slot t is denoted by

$$C_{t,k}(X) := \begin{cases} l_{y_{t,k}} + \beta_{y_{t,k}} & \text{if } y_{t-1,k} \neq y_{t,k} > 0 \\ l_{y_{t,k}} & \text{if } y_{t-1,k} = y_{t,k} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The total cost of X can be written as $C(X) = \sum_{t=1}^{T} \sum_{k=1}^{m} C_{t,k}(X)$. The technical lemma below will be needed for our induction proof in Theorem 1. Given the optimal schedules $\hat{X}^u$ and $\hat{X}^v$ with $u < v$, the inequality $\sum_{k=1}^{m} \sum_{t=1}^{u} C_{t,k}(\hat{X}^u) \leq \sum_{k=1}^{m} \sum_{t=1}^{u} C_{t,k}(\hat{X}^v)$ is obviously fulfilled (because $\hat{X}^u$ is an optimal schedule for $\mathcal{I}^u$, so $\hat{X}^v$ cannot be better). The lemma below shows that this inequality still holds if the cost $C_{t,k}(\cdot)$ is scaled by $\tilde{y}^u_{t,k}$.

Lemma 7. *Let $u, v \in [T]$ with $u < v$. It holds that*

$$\sum_{k=1}^{m} \sum_{t=1}^{u} \tilde{y}^u_{t,k} C_{t,k}(\hat{X}^u) \leq \sum_{k=1}^{m} \sum_{t=1}^{u} \tilde{y}^u_{t,k} C_{t,k}(\hat{X}^v). \tag{6}$$

The proof is shown in the full paper. The next lemma shows how the cost of a single block $A_{v,k}$ can be folded into the term $2\sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^v)$ which is the right hand side of Eq. (6) given in the previous lemma with $u = v - 1$.

Lemma 8. *For all lanes $k \in [m]$ and time slots $v \in [T]$, it is*

$$2\sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^v) + C(A_{v,k}) \leq 2\sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v). \tag{7}$$

Proof. If the block $A_{v,k}$ does not exists, Eq. (7) holds by Lemma 6 and $C(A_{v,k}) = 0$.

If $A_{v,k}$ is a *new* block, then $C(A_{v,k}) \leq 2\beta_j$ with $j := j_{v,k} = \hat{y}^v_{v,k}$ by Lemma 5. Since $A_{v,k}$ is a *new* block, server type j was not used in the last time slot of the last $\bar{t}_j$ schedules, i.e., $\hat{y}^t_{t,k} \leq j - 1$ for $t \in [v - \bar{t}_j : v - 1]$. If $\hat{y}^{v-\bar{t}_j}_{v-\bar{t}_j,k} = j$ would hold, then $y^{\mathcal{A}}_{v-1,k} = j$ and there would be an *extended* block at time slot v. By using the facts above and the definition of $\tilde{t}^v_{t,k}$, for $t \in [v - \bar{t}_j : v - 1]$, we get

$$\tilde{y}^{v-1}_{t,k} = \max_{t' \in [t:v-1]} \hat{y}^{t'}_{t',k} \leq j - 1 = \hat{y}^v_{v,k} - 1 \leq \max_{t' \in [t:v]} \hat{y}^{t'}_{t',k} - 1 = \tilde{y}^v_{t,k} - 1. \tag{8}$$

By using Lemma 6 and Eq. (8), we can estimate the first sum in (7):

$$\begin{aligned} \sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^v) &\overset{L6,(8)}{\leq} \sum_{t=1}^{v-\bar{t}_j-1} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v) + \sum_{t=v-\bar{t}_j}^{v-1} (\tilde{y}^{v}_{t,k} - 1) C_{t,k}(\hat{X}^v) \\ &\leq \sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v) - \beta_j. \end{aligned} \tag{9}$$

For the second inequality, we add $(\tilde{y}^v_{v,k} - 1) \cdot C_{v,k}(\hat{X}^v) \geq 0$ and use $\sum_{t=v-\bar{t}_j}^{v} C_{t,k}(\hat{X}^v) \geq \beta_j$ which holds because either j was powered up in $\hat{X}^v$ during $[v - \bar{t}_j : v]$ (then there is the switching cost of β_j) or j runs for $\bar{t}_j + 1$ time slots resulting in an operating cost of $l_j \cdot (\bar{t}_j + 1) = l_j \cdot (\lfloor \beta_j / l_j \rfloor + 1) \geq \beta_j$. Altogether, we get (beginning from the left hand side of Eq. (7) that has to be shown)

$$\begin{aligned} 2\sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^v) + C(A_{v,k}) &\overset{(9),L5}{\leq} 2\sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v) - 2\beta_j + 2\beta_j \\ &\leq 2\sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v). \end{aligned}$$

If $A_{v,k}$ is an *extended* block, the proof of Eq. (7) is quite similar (see the full version of this paper for more details). □

Theorem 1. *Algorithm $\mathcal{A}$ is $2d$-competitive.*

Proof. The feasibility of $X^{\mathcal{A}}$ was already proven in Lemma 4, so we have to show that $C(X^{\mathcal{A}}) \leq 2d \cdot C(\hat{X}^T)$. Let $C_v(X^{\mathcal{A}}) := \sum_{t=1}^{v} \sum_{k=1}^{m} C(A_{t,k})$ denote the cost of algorithm $\mathcal{A}$ up to time slot v. We will show by induction that

$$C_v(X^{\mathcal{A}}) \leq 2\sum_{k=1}^{m}\sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v) \tag{10}$$

holds for all $v \in [T]_0$.

For $v = 0$, we have no costs for both $X^{\mathcal{A}}$ and $\hat{X}^v$, so inequality (10) is fulfilled. Assume that inequality (10) holds for $v - 1$. By using the induction hypothesis as well as Lemmas 7 and 8, we get

$$\begin{aligned} C_v(X^{\mathcal{A}}) &= C_{v-1}(X^{\mathcal{A}}) + \sum_{k=1}^{m} C(A_{v,k}) \\ &\overset{\text{I.H.}}{\leq} 2\sum_{k=1}^{m}\sum_{t=1}^{v-1} \tilde{y}^{v-1}_{t,k} C_{t,k}(\hat{X}^{v-1}) + \sum_{k=1}^{m} C(A_{v,k}) \\ &\overset{L7,L8}{\leq} 2\sum_{k=1}^{m}\sum_{t=1}^{v} \tilde{y}^{v}_{t,k} C_{t,k}(\hat{X}^v). \end{aligned} \tag{11}$$

Since $\tilde{y}^v_{t,k} \leq d$, we get

$$C_T(X^{\mathcal{A}}) \overset{(11)}{\leq} 2\sum_{k=1}^{m}\sum_{t=1}^{T} \tilde{y}^{T}_{t,k} C_{t,k}(\hat{X}^T) \leq 2d\sum_{k=1}^{m}\sum_{t=1}^{T} C_{t,k}(\hat{X}^T) \leq 2d \cdot C(\hat{X}^T).$$

The schedule $\hat{X}^T$ is optimal for the problem instance $\mathcal{I}$, so algorithm $\mathcal{A}$ is $2d$-competitive. □

3 Randomized Online Algorithm

The $2d$-competitive algorithm can be randomized to achieve a competitive ratio of $\frac{e}{e-1}d \approx 1.582d$ against an oblivious adversary. The randomized algorithm $\mathcal{B}$ chooses $\gamma \in [0,1]$ according to the probability density function $f_\gamma(x) = e^x/(e-1)$ for $x \in [0,1]$. The variables $\bar{t}_j$ are set to $\lfloor \gamma \cdot \beta_j/l_j \rfloor$, so the running time of a server is randomized. Then, algorithm $\mathcal{A}$ is executed. Note that γ is determined at the beginning of the algorithm and not for each block.

Theorem 2. *Algorithm $\mathcal{B}$ is $\frac{e}{e-1}d$-competitive against an oblivious adversary.*

The complete proof of this theorem is shown in the full paper. Most lemmas introduced in the previous section still hold, because they do not depend on the exact value of $\bar{t}_j$, only Lemmas 5 and 8 have to be adapted. For the proof of Theorem 2, we first give an upper bound for the expected cost of block $A_{t,k}$ (replacing Lemma 5). This bound is used to show that

$$\frac{e}{e-1} \cdot \sum_{t=1}^{v-1} \tilde{y}_{t,k}^{v-1} C_{t,k}(\hat{X}^v) + \mathbb{E}[C(A_{v,k})] \leq \frac{e}{e-1} \cdot \sum_{t=1}^{v} \tilde{y}_{t,k}^{v} C_{t,k}(\hat{X}^v)$$

holds for all lanes $k \in [m]$ and time slots $v \in [T]$ (similar to Lemma 8). Finally, Theorem 2 is proven by induction.

4 Lower Bound

In this section, we show that there is no deterministic online algorithm that achieves a competitive ratio that is better than $2d$.

We consider the following problem instance: Let $\beta_j := N^{2j}$ and $l_j := 1/N^{2j}$ where N is a sufficiently large number that depends on the number of servers types d. The value of N will be determined later. The adversary will send a job for the current time slot if and only if the online algorithm has no active server during the previous time slot. This implies that the online algorithm has to power up a server immediately after powering down any server. Note that $\lambda_t \in \{0,1\}$, i.e., it is never necessary to power up more than one server. The optimal schedule is denoted by X^*. Let $\mathcal{A}$ be an arbitrary deterministic online algorithm and let $X^{\mathcal{A}}$ be the schedule computed by $\mathcal{A}$.

W.l.o.g. in $X^{\mathcal{A}}$ there is no time slot with more than one active server. If this were not the case, we could easily convert the schedule into one where the assumption holds without increasing the cost. Assume that at time slot t a new server of type k is powered up such that there are (at least) two active servers at time t. If we power up the server at $t+1$, the schedule is still feasible, but the total costs are reduced by l_k. We can repeat this procedure until there is at most one active server for each time slot.

Lemma 9. *Let $k \in [d]$. If $X^{\mathcal{A}}$ only uses servers of type lower than or equal to k and if the cost of $\mathcal{A}$ is at least $C(X^{\mathcal{A}}) \geq N\beta_k$, then the cost of $\mathcal{A}$ is at least*

$$C(X^{\mathcal{A}}) \geq (2k - \epsilon_k) \cdot C(X^*) \tag{12}$$

with $\epsilon_k = 9k^2/N$ *and* $N \geq 6k$.

Proof Idea. We will prove the lemma by induction. The base case $k = 1$ is shown in the full version of this paper, so we assume that Lemma 9 holds for $k - 1$.

We divide the schedule $X^{\mathcal{A}}$ into phases $L_0, K_1, L_1, K_2, \ldots, L_n$ such that in the phases $K_1, \ldots, K_n$ server type k is used exactly once, while in the intermediate phases $L_0, \ldots, L_n$ the other server types $1, \ldots, k-1$ are used. A phase K_i begins when a server of type k is powered up and ends when it is powered down. The phases L_i can have zero length (if the server type k is powered up immediately after it is powered down, so between K_i and K_{i+1} an empty phase L_i is inserted).

The operating cost during phase K_i is denoted by $\delta_i \beta_k$. The operating and switching costs during phase L_i are denoted by $p_i \beta_k$. We divide the intermediate phases L_i into long phases where $p_i > 1/N$ holds and short phases where $p_i \leq 1/N$. Note that we can use the induction hypothesis only for long phases. The index sets of the long and short phases are denoted by $\mathcal{L}$ and $\mathcal{S}$, respectively.

To estimate the cost of an optimal schedule we consider two strategies: In the first strategy, a server of type k is powered up at the first time slot and runs for the whole time except for phases K_i with $\delta_i > 1$, then powering down and powering up are cheaper than keeping the server in the active state (β_k vs. $\delta_i \beta_k$). The operating cost for the phases K_i is $\delta_i^* \beta_k$ with $\delta_i^* := \min\{1, \delta_i\}$ and the operating cost for the phases L_i is at most $\frac{1}{N^2} p_i \beta_k$, because algorithm $\mathcal{A}$ uses servers whose types are lower than k and therefore the operating cost of $\mathcal{A}$ is at least N^2 times larger. Thus, the total cost of this strategy is at most

$$\beta_k \left(1 + \sum_{i=1}^{n} \delta_i^* + \sum_{i \in \mathcal{L} \cup \mathcal{S}} \frac{1}{N^2} p_i \right) \geq C(X^*).$$

In the second strategy, for the long phases L we use the strategy given by our induction hypothesis, while for the short phases S we behave like algorithm $\mathcal{A}$ and in the phases K_i we run the server type 1 for exactly one time slot (note that in K_i we only have $\lambda_t = 1$ in the first time slot of the phase). Therefore the total cost is upper bounded by

$$\beta_k \left(\sum_{i \in \mathcal{L}} \frac{1}{\alpha} p_i + \sum_{i \in \mathcal{S}} p_i + 2n\beta_1/\beta_k \right) \geq C(X^*)$$

with $\alpha := 2k - 2 - \epsilon_{k-1}$.

The total cost of $\mathcal{A}$ is equal to $\beta_k \left(\sum_{i=1}^{n} (1 + \delta_i) + \sum_{i \in \mathcal{L} \cup \mathcal{S}} p_i \right)$, so the competitive ratio is given by

$$\frac{C(X^{\mathcal{A}})}{C(X^*)} \geq \frac{\sum_{i=1}^{n} (1 + \delta_i) + \sum_{i \in \mathcal{L} \cup \mathcal{S}} p_i}{C(X^*)/\beta_k}.$$

By cleverly separating the nominator into two terms and by estimating $C(X^*)$ with strategy 1 and 2, respectively, it can be shown that $\frac{C(X^{\mathcal{A}})}{C(X^*)} \geq 2 + \alpha - \frac{16k}{N} \geq 2k - \epsilon_k$. The complete calculation including all intermediate steps is shown in the full paper. □

Theorem 3. *There is no deterministic online algorithm for the data-center optimization problem with heterogeneous servers and time and load independent operating costs whose competitive ratio is smaller than* $2d$.

Proof Idea. Assume that there is an $(2d - \epsilon)$-competitive deterministic online algorithm $\mathcal{A}$. We construct a workload as described at the beginning of this section until the cost of $\mathcal{A}$ is greater than $N\beta_d$ (note that $l_j > 0$ for all $j \in [d]$, so the cost of $\mathcal{A}$ can be arbitrarily large). By using Lemma 9 with $k = d$ and $N := \max\{6d, \lceil 9k^2/\epsilon + 1 \rceil\}$, we get $C(X^{\mathcal{A}}) > (2d - \epsilon) \cdot C(X^*)$ which is a contradiction to our assumption. See the full paper for more details. □

The schedule constructed for the lower bound only uses at most one job in each time slot, so there is no reason for an online algorithm to utilize more than one server of a specific type. Thus, for a data center with m unique servers (i.e. $m_j = 1$ for all $j \in [d]$), the best achievable competitive ratio is $2d = 2m$.

References

1. Albers, S., Quedenfeld, J.: Optimal algorithms for right-sizing data centers. In: Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, pp. 363–372. ACM (2018)
2. Albers, S., Quedenfeld, J.: Optimal algorithms for right-sizing data centers–extended version. arXiv preprint arXiv:1807.05112 (2018)
3. Antoniadis, A., Garg, N., Kumar, G., Kumar, N.: Parallel machine scheduling to minimize energy consumption. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 2758–2769. SIAM (2020)
4. Antoniadis, A., Schewior, K.: A tight lower bound for online convex optimization with switching costs. In: Solis-Oba, R., Fleischer, R. (eds.) WAOA 2017. LNCS, vol. 10787, pp. 164–175. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89441-6_13
5. Argue, C., Gupta, A., Guruganesh, G., Tang, Z.: Chasing convex bodies with linear competitive ratio. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1519–1524. SIAM (2020)
6. Augustine, J., Irani, S., Swamy, C.: Optimal power-down strategies. SIAM J. Comput. **37**(5), 1499–1516 (2008)
7. Badiei, M., Li, N., Wierman, A.: Online convex optimization with ramp constraints. In: 2015 54th IEEE Conference on Decision and Control (CDC), pp. 6730–6736. IEEE (2015)
8. Bansal, N., Böhm, M., Eliáš, M., Koumoutsos, G., Umboh, S.W.: Nested convex bodies are chaseable. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1253–1260. SIAM (2018)

9. Bansal, N., Gupta, A., Krishnaswamy, R., Pruhs, K., Schewior, K., Stein, C.: A 2-competitive algorithm for online convex optimization with switching costs. In: LIPIcs-Leibniz International Proceedings in Informatics, vol. 40. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2015)
10. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. J. ACM (JACM) **54**(1), 1–39 (2007)
11. Bawden, T.: Global warming: data centres to consume three times as much energy in next decade, experts warn (2016). http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html
12. Brill, K.G.: The invisible crisis in the data center: the economic meltdown of Moore's law. White paper, Uptime Institute, pp. 2–5 (2007)
13. Bubeck, S., Klartag, B., Lee, Y.T., Li, Y., Sellke, M.: Chasing nested convex bodies nearly optimally. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1496–1508. SIAM (2020)
14. Chen, N., Agarwal, A., Wierman, A., Barman, S., Andrew, L.L.: Online convex optimization using predictions. In: ACM SIGMETRICS Performance Evaluation Review, vol. 43, pp. 191–204. ACM (2015)
15. Chen, N., Goel, G., Wierman, A.: Smoothed online convex optimization in high dimensions via online balanced descent. Proc. Mach. Learn. Res. **75**, 1574–1594 (2018)
16. Delforge, P., et al.: Data center efficiency assessment (2014). https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf
17. Friedman, J., Linial, N.: On convex body chasing. Discrete Comput. Geom. **9**(1), 293–321 (1993). https://doi.org/10.1007/BF02189324
18. Goel, G., Chen, N., Wierman, A.: Thinking fast and slow: optimization decomposition across timescales. In: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), pp. 1291–1298. IEEE (2017)
19. Goel, G., Wierman, A.: An online algorithm for smoothed regression and LQR control. Proc. Mach. Learn. Res. **89**, 2504–2513 (2019)
20. Hamilton, J.: Cost of power in large-scale data centers (2008). http://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/
21. Irani, S., Pruhs, K.R.: Algorithmic problems in power management. ACM SIGACT News **36**(2), 63–76 (2005)
22. Kim, S.J., Giannakis, G.B.: Real-time electricity pricing for demand response using online convex optimization. In: ISGT 2014, pp. 1–5. IEEE (2014)
23. Kim, T., Yue, Y., Taylor, S., Matthews, I.: A decision tree framework for spatiotemporal sequence prediction. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 577–586. ACM (2015)
24. Lin, M., Liu, Z., Wierman, A., Andrew, L.L.: Online algorithms for geographical load balancing. In: Green Computing Conference (IGCC), pp. 1–10. IEEE (2012)
25. Lin, M., Wierman, A., Andrew, L.L., Thereska, E.: Dynamic right-sizing for power-proportional data centers. IEEE/ACM Trans. Netw. (TON) **21**(5), 1378–1391 (2013)
26. Lin, M., Wierman, A., Andrew, L.L., Thereska, E.: Dynamic right-sizing for power-proportional data centers – extended version (2013)
27. Liu, Z., Lin, M., Wierman, A., Low, S.H., Andrew, L.L.: Greening geographical load balancing. In: Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, pp. 233–244. ACM (2011)

28. Meyerson, A.: The parking permit problem. In: 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), pp. 274–282. IEEE (2005)
29. Schmid, P., Roos, A.: Overclocking core i7: power versus performance (2009). www.tomshardware.com/reviews/overclock-core-i7,2268-10.html
30. Sellke, M.: Chasing convex bodies optimally. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1509–1518. SIAM (2020)
31. Shan, A.: Heterogeneous processing: a strategy for augmenting Moore's law. Linux J. **2006**(142), 7 (2006)
32. Wang, H., Huang, J., Lin, X., Mohsenian-Rad, H.: Exploring smart grid and data center interactions for electric power load balancing. ACM SIGMETRICS Perform. Eval. Rev. **41**(3), 89–94 (2014)
33. Zhang, M., Zheng, Z., Shroff, N.B.: An online algorithm for power-proportional data centers with switching cost. In: 2018 IEEE Conference on Decision and Control (CDC), pp. 6025–6032. IEEE (2018)

Appendix D

Algorithms for Right-Sizing Heterogeneous Data Centers, SPAA 2021

This chapter has been published as **peer-reviewed conference paper** [AQ21e].

> Susanne Albers and Jens Quedenfeld. Algorithms for right-sizing heterogeneous data centers. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'21)*, pages 48–58. ACM, 2021.

A full version of this paper that contains all proofs was submitted to a special issue of the ACM journal *Transactions on Parallel Computing* on 26th November 2021. A preprint of this version was uploaded on arXiv [AQ21d], see https://arxiv.org/abs/2107.14692.

Synopsis. We analyze the right-sizing problem of heterogeneous data centers with d different server types. The operating cost depends on the load of the server and is modeled by an increasing convex function. We consider the discrete setting where the number of active server must be integral. First, we present a $(2d+1)$-competitive deterministic online algorithm for time-independent operating cost functions. Afterwards, we show how our algorithm can be modified to handle time-dependent operating cost functions. The modified algorithm achieves a competitive ratio of $2d+1+\epsilon$ for any $\epsilon > 0$. Finally, we address the offline problem and give a $(1+\epsilon)$-approximation algorithm that is able to handle time-variable data-center sizes.

Contributions of thesis author. The thesis author developed the algorithms of this publication including all proofs. Furthermore, he wrote the manuscript.

Algorithms for Right-Sizing Heterogeneous Data Centers*

Susanne Albers
Technical University of Munich
Garching, Germany
albers@in.tum.de

Jens Quedenfeld
Technical University of Munich
Garching, Germany
jens.quedenfeld@in.tum.de

ABSTRACT

Power consumption is a dominant and still growing cost factor in data centers. In time periods with low load, the energy consumption can be reduced by powering down unused servers. We resort to a model introduced by Lin, Wierman, Andrew and Thereska [23, 24] that considers data centers with identical machines, and generalize it to heterogeneous data centers with d different server types. The operating cost of a server depends on its load and is modeled by an increasing, convex function for each server type. In contrast to earlier work, we consider the discrete setting, where the number of active servers must be integral. Thereby, we seek truly feasible solutions. For homogeneous data centers ($d = 1$), both the offline and the online problem were solved optimally in [3, 4].

In this paper, we study heterogeneous data centers with general time-dependent operating cost functions. We develop an online algorithm based on a work function approach which achieves a competitive ratio of $2d + 1 + \epsilon$ for any $\epsilon > 0$. For time-independent operating cost functions, the competitive ratio can be reduced to $2d + 1$. There is a lower bound of $2d$ shown in [5], so our algorithm is nearly optimal. For the offline version, we give a graph-based $(1+\epsilon)$-approximation algorithm. Additionally, our offline algorithm is able to handle time-variable data-center sizes.

CCS CONCEPTS

• **Theory of computation → Online algorithms**; **Approximation algorithms analysis**; *Scheduling algorithms*; *Discrete optimization*; • **Hardware → Enterprise level and data centers power issues**; *Switching devices power issues.*

KEYWORDS

Heterogeneous machines; energy conservation; online algorithm; competitive analysis; approximation algorithm; discrete setting.

ACM Reference Format:
Susanne Albers and Jens Quedenfeld. 2021. Algorithms for Right-Sizing Heterogeneous Data Centers. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21), July 6–8, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3409964.3461789

*Work supported by the European Research Council, Grant Agreement No. 691672.

SPAA '21, July 6–8, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8070-6/21/07.
https://doi.org/10.1145/3409964.3461789

1 INTRODUCTION

Energy conservation in data centers is important for both economical and ecological reasons [14]. A huge amount of the energy consumed in data centers is wasted because many servers run idle for long time periods, while still consuming half of their peak power [18, 28]. The power consumption can be reduced by powering down servers that are currently not needed. However, a power-up operation of a server causes increased energy consumption. Hence, holding an idle server in active mode for a short period of time is cheaper than powering it down and up again shortly after. Furthermore, power-up and -down operations generate delay and wear-and-tear costs [24]. Therefore, algorithms are needed that dynamically right-size a data center depending on incoming jobs so as to minimize the energy consumption.

In this paper, we consider data centers with heterogeneous servers. This can be different architectures, for example, servers that use the GPU to perform massive parallel calculations. However, tasks that contain many branches are not suitable for GPUs and can be processed much faster on a common CPU [30]. Heterogeneity may also result from old and new servers. It is a common practice that a data center is extended by new servers while the old ones are kept in use.

In practice, the energy consumption of a server is not constant but increases with load [6]. If a machine is idle, the CPU frequency is lowered in modern hardware to save energy [27]. For high frequencies, the CPU voltage has to be raised, which results in a superlinear increase in power consumption [32]. Therefore, in our model, the energy consumption of each server type j is modeled by an increasing convex function f_j of the load z. The operating cost of an idle server is given by $f_j(0)$. By setting the value of f_j to infinity for large load values z, it is possible to model different server capacities. For example, there may be a slow server type with a maximum load of 1 and a fast server type with a maximum load of 4 that can process four times as many jobs as the slow server.

Our model described below is a generalization of the model presented by Lin, Wierman, Andrew and Thereska [23, 24] for homogeneous data centers where all servers are identical.

Problem formulation. We consider a data center with d different server types and m_j servers of type j. The servers have two states, an active one where they are able to process jobs and an inactive one without energy consumption. Powering up a server of type j, i.e., switching it from the inactive to the active state, produces cost of β_j (called *switching* cost). Power-down operations do not incur any cost. We consider a time horizon consisting of the time slots $\{1, \dots, T\}$. For each time slot $t \in \{1, \dots, T\}$, a job volume of λ_t arrives and has to be processed during the time slot. The jobs can be arbitrarily distributed to the servers. Let $z_j^{\max}$ denote the maximum job volume that can be processed by one server of

type j during a single time slot. If a server of type j works with load $z \in [0, z_j^{\max}]$, it causes cost in the amount of $f_{t,j}(z)$ where $f_{t,j}(z)$ is a convex increasing non-negative function. Since $f_{t,j}$ is convex, the cost is minimized if each active server of type j runs with the same load (see Lemma 2.2 for a formal proof). Therefore, the *operating* cost for server type j during time slot t is given by

$$g_{t,j}(x,z) \coloneqq \begin{cases} x f_{t,j}\left(\frac{\lambda_t z}{x}\right) & \text{if } x > 0 \\ \infty & \text{if } x = 0 \text{ and } \lambda_t z > 0 \\ 0 & \text{if } x = 0 \text{ and } \lambda_t z = 0 \end{cases}$$

where x is the number of active servers of type j and z is the fraction of the job volume λ_t that is assigned to server type j. The total operating cost during time slot t is denoted by

$$g_t(x_1, \dots, x_d) \coloneqq \min_{(z_1,\dots,z_d)\in\mathcal{Z}} \sum_{j=1}^{d} g_{t,j}(x_j, z_j) \tag{1}$$

where $\mathcal{Z} \coloneqq \{(z_1, \dots, z_d) \in [0,1]^d \mid \sum_{j=1}^{d} z_j = 1\}$ is the set of all possible job assignments.

A schedule X is a sequence $\boldsymbol{x}_1, \dots, \boldsymbol{x}_T$ with $\boldsymbol{x}_t = (x_{t,1}, \dots, x_{t,d})$ where each $x_{t,j} \in \{0, 1, \dots, m_j\}$ indicates the number of active servers of type j during time slot t. We assume that at the beginning and end of the considered time horizon, all servers are in the inactive state, i.e., $\boldsymbol{x}_0 = \boldsymbol{x}_{T+1} = (0, \dots, 0)$. A schedule is called *feasible*, if there are not more active servers than available and if the maximum load of the active servers is not exceeded, i.e., $x_{t,j} \in \{0, 1, \dots, m_j\}$ and $\sum_{j=1}^{d} x_{t,j} z_j^{\max} \geq \lambda_t$ holds for all $t \in \{1, \dots, T\}$ and $j \in \{1, \dots, d\}$. The total cost of a schedule is defined by

$$C(X) \coloneqq \sum_{t=1}^{T} \left(g_t(x_{t,1}, \dots, x_{t,d}) + \sum_{j=1}^{d} \beta_j (x_{t,j} - x_{t-1,j})^+ \right) \tag{2}$$

where $(x)^+ \coloneqq \max(x, 0)$. Note that the switching cost is only paid for powering up. However, this is not a restriction, since all servers are inactive at the beginning and end of the workload. Thus, the cost for powering down can be folded into the cost for powering up.

A problem instance is specified by the tuple $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$ with $\boldsymbol{m} = (m_1, \dots, m_d)$, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_d)$, $F = (f_{1,1}, \dots, f_{T,d})$ and $\Lambda = (\lambda_1, \dots, \lambda_T)$. The task is to find a schedule with minimal cost.

In the online version of this problem, the job volumes λ_t and the operating cost functions $f_{t,j}$ arrive one-by-one, so $\boldsymbol{x}_t$ has to be determined without the knowledge of future jobs $\lambda_{t'}$ and functions $f_{t',j}$ with $t' > t$.

Our contribution. We investigate both the online and the offline version of this problem. In contrast to previous results, we consider the discrete setting where the number of active servers $x_{t,j}$ has to be integral. Thereby, we obtain truly feasible solutions.

For the online problem, we first examine a simplified version where the operating cost functions $f_{t,j}$ are time-independent (i.e., $f_{t,j} = f_j$ for all $t \in \{1, \dots, T\}$) and present a $(2d+1)$-competitive deterministic online algorithm (Section 2). The basic idea is to calculate an optimal schedule for the problem instance that ends at the current time slot. For each server type, the algorithm ensures that the number of active servers is at least as large as the number of active servers in the optimal schedule. A server is powered down if its accumulated idle operating cost $f_j(0)$ exceeds its switching cost β_j. Since the operating cost is time-independent, the runtime of a server can be determined in advance.

In Section 3, we demonstrate how our algorithm can be modified to handle time-dependent operating cost functions $f_{t,j}$. We achieve a competitive ratio of $2d + 1 + \epsilon$ for any $\epsilon > 0$. The basic idea of the algorithm is unchanged. However, in contrast to the previous section, the runtime of a server now depends on the time slot when it is powered up, since the idle operating cost $f_{t,j}(0)$ varies over time. Thus, the runtime of a server is not known in advance any more. The analysis results in a competitive ratio of $2d + 1 + c(\mathcal{I})$ where $c(\mathcal{I})$ is a constant that depends on the switching and operating costs of the problem instance $\mathcal{I}$. By allowing state changes at any time during a time slot and repairing the resulting schedule afterward (such that there are no intermediate state changes any more), we are able to make the constant $c(\mathcal{I})$ arbitrarily small.

In Section 4, we consider the offline version of the problem and present a $(1+\epsilon)$-approximation algorithm that runs in polynomial time if d is a constant. First, we present an optimal algorithm that uses a natural graph representation. The graph is structured in a $(d+1)$-dimensional grid and contains a vertex $v_{t,\boldsymbol{x}}$ for each time slot $t \in \{1, \dots, T\}$ and server configuration $\boldsymbol{x}$. The vertices are connected with weighted edges that represent the switching and operating costs. By calculating a shortest path, we obtain an optimal schedule. For our approximation algorithm, we only use a small polynomial-sized subset of all vertices depending on the desired approximation factor. Our $(1+\epsilon)$-approximation algorithm runs in $O\left(T \cdot \epsilon^{-d} \cdot \prod_{j=1}^{d} \log m_j\right)$ time. At the end of Section 4, we show that our algorithm still works if the total number of servers varies over time, i.e., m_j is time-dependent.

Related work. In recent years, energy conservation in data centers has received much attention, see for example [1, 8, 33] and references therein.

Regarding the online version, Lin et al. [23, 24] analyzed the problem described above for homogeneous data centers where all servers are identical, i.e., $d = 1$. The minimum function in equation (1) disappears, so the operating cost at time slot t is given by $g_t(x) = x f(\lambda_t / x)$, which makes the problem much easier. They presented a 3-competitive online algorithm for the fractional setting where the number of active servers does not need to be integral. This result was improved by Bansal et al. [13] who developed a 2-competitive algorithm. In our previous paper [3, 4], we analyzed the discrete setting for homogeneous data centers. We developed a 3-competitive deterministic and 2-competitive randomized online algorithm and showed that these algorithms are optimal (i.e., there is no algorithm that achieves a better competitive ratio). Furthermore, we proved that 2 is a lower bound for the fractional setting (this result was independently found in [9]).

The data-center right-sizing problem on *heterogeneous* data centers is related to convex function chasing, also known as smoothed online convex optimization [17]. At each time slot, a convex function g_t arrives and the algorithm has to choose a point $\boldsymbol{x}_t \in \mathbb{R}^d$. The cost at time slot t is given by $g_t(\boldsymbol{x}_t)$ plus the movement cost $\|\boldsymbol{x}_t - \boldsymbol{x}_{t-1}\|$ where $\|\cdot\|$ is any metric. Data-center right-sizing in the fractional setting (i.e., the number of active servers can be any real number) is a special case of convex function chasing where

$\|\cdot\|$ is a scaled Manhattan metric and the convex functions have the form given in equation (1).

Goel and Wierman [20] developed a $(3 + O(1/\mu))$-competitive algorithm called Online Balanced Descent (OBD) where the arriving functions are μ-strongly convex. Chen et al. [17] showed that OBD achieves a competitive ratio of $3 + O(1/\alpha)$ if the arriving functions are locally α-polyhedral. However, if the operating cost functions $f_{t,j}$ are load-independent, i.e., $f_{t,j}(z) = \text{const}$, then g_t is neither strongly convex nor locally polyhedral, so $\mu = 0$ and $\alpha = 0$. Hence, their results cannot be used for our problem.

Sellke [29] developed a $(d + 1)$-competitive online algorithm for convex function chasing without any restrictions. A similar result was found by Argue et al. [10]. The general convex function chasing problem in the *discrete* setting where g_t can be any convex function has (at least) an exponential competitive ratio as the following example shows. For all $j \in \{1, \dots, d\}$, let $m_j = 1$ and $\beta_j = 1$, so the feasible server configurations are $\{0, 1\}^d$. The arriving functions g_t are infinite for the current position $\boldsymbol{x}_{t-1}$ of the online algorithm and zero for all other positions $\{0, 1\}^d \setminus \{\boldsymbol{x}_{t-1}\}$. The online algorithm always has to change its position to avoid the infinite operating cost (otherwise the online algorithm is not competitive at all). Therefore, after $T := 2^d - 1$ time slots, the switching cost of the online algorithm is at least $2^d - 1$. The offline schedule can go directly to a position in $\{0, 1\}^d \setminus \bigcup_{t=1}^{T} \{\boldsymbol{x}_{t-1}\}$ where no operating cost occurs paying a switching cost of at most d. Thus, the competitive ratio for general convex function chasing is at least $\Omega(2^d/d)$. To gain a competitive ratio with more practical relevance, we focus on operating cost functions described by equation (1).

It is an open problem how fractional solutions can be rounded to achieve an integral schedule without significantly increasing the total cost. If the number of active servers is simply rounded up, the total switching cost can get arbitrarily large, for example if the fractional schedule switches permanently between 1 and $1 + \epsilon$. For homogeneous data centers, a randomized rounding scheme achieving a competitive ratio of 2 was presented in [4]. However, using this method for heterogeneous data centers independently for each server type can lead to an infeasible schedule (e.g., if $\lambda_t = 1$ and $\boldsymbol{x}_t = (1/d, \dots, 1/d)$ is rounded down to $(0, \dots, 0)$). Thus, Sellke's result does not help us in our analysis of the discrete setting. Further publications examining the convex body or function chasing problem are [7, 12, 15].

In [5], we analyzed the discrete setting for heterogeneous data centers where the operating cost does neither depend on the load nor on time , i.e., $f_{t,j}(z) = l_j = \text{const}$. In this case, the total operating cost at time t is given by $g_t(x_1, \dots, x_d) = \sum_{j=1}^{d} l_j x_j$ which is much simpler than the general expression given in equation (1). In addition, we assumed that there are no inefficient servers, i.e., a server with a higher switching cost always has a lower operating cost. We presented a $2d$-competitive algorithm for this special problem. Moreover, we gave a lower bound of $2d$, which also holds for the general problem that we consider in this paper. Thus, our online algorithms presented in Sections 2 and 3 of this paper are nearly optimal. If the operating cost functions are constant (i.e., $f_{t,j}(z) = \text{const}$), we achieve the optimal competitive ratio of $2d$.

The offline version of the discrete data-center right-sizing problem for *homogeneous* data centers can be solved in polynomial time [3]. It is an open question whether the problem on *heterogeneous* data centers is NP-hard or not. For the special case of load-independent operating costs (i.e., $f_{t,j}(z) = l_j = \text{const}$), a polynomial-time algorithm based on a minimum-cost flow computation was shown in [1, 2]. However, the flow representation of the problem cannot be generalized for load-dependent operating costs.

Right-sizing of heterogeneous data centers is related to geographical load balancing examined in [26] and [22]. For more works handling related problems, refer to [11, 16, 19, 21, 25, 31, 33].

NOTATION

Let $[k] := \{1, 2, \dots k\}$, $[k]_0 := \{0, 1, \dots k\}$ and $[k : l] := \{k, k + 1, \dots, l\}$ where $k, l \in \mathbb{N}$.

2 ONLINE ALGORITHM FOR TIME-*INDEPENDENT* OPERATING COST FUNCTIONS

In this section we present a $(2d + 1)$-competitive deterministic online algorithm for time-independent operating cost functions, i.e., $f_{t,j} = f_j$ for all time slots $t \in [T]$. Roughly, our algorithm works as follows. For each time slot, it calculates an optimal schedule for the job volumes received so far. Servers are powered up such that the number of active servers of each type is at least as large as the number of active servers of the same type in the optimal schedule. A server runs for exactly $\lceil \beta_j / f_j(0) \rceil$ time slots, then it is powered down, regardless of whether or not it was used. This is similar to the well-known ski rental problem where it is optimal to buy the skis once the total renting cost would exceed the buy price.

Formally, given the problem instance $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$, the shortened problem instance $\mathcal{I}^t$ is defined by $\mathcal{I}^t := (t, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda^t)$ with $\Lambda^t = (\lambda_1, \dots, \lambda_t)$. Let $\hat{X}^t$ denote an optimal schedule for this problem instance and let $X^{\mathcal{A}}$ be the schedule calculated by our algorithm $\mathcal{A}$.

Our algorithm works as follows: After calculating $\hat{X}^t$, the algorithm ensures that the number of active servers of each type $j \in [d]$ is greater than or equal to the number of active servers of type j in the last time slot of $\hat{X}^t$. That is, in each time slot $(\hat{x}^t_{t,j} - x^{\mathcal{A}}_{t-1,j})^+$, servers of type j are powered up such that the inequality $x^{\mathcal{A}}_{t,j} \geq \hat{x}^t_{t,j}$ is satisfied. A server of type j is powered down after $\bar{t}_j = \left\lceil \frac{\beta_j}{f_j(0)} \right\rceil$ time slots. Note that $f_j(0)$ is the operating cost of a server being idle. It does not matter if the server was used or not.

The pseudocode below clarifies how algorithm $\mathcal{A}$ works. The schedule $\hat{X}^t$ can be calculated with the optimal offline algorithm

Algorithm 1 Algorithm $\mathcal{A}$

```
w_{t,j} := 0 for all t ∈ ℤ and j ∈ [d]
for t := 1 to T do
    Calculate X̂^t
    for j := 1 to d do
        x^A_{t,j} := x^A_{t,j} − w_{t−t̄_j,j}
        if x^A_{t,j} ≤ x̂^t_{t,j} then
            w_{t,j} := x̂^t_{t,j} − x^A_{t,j}
            x^A_{t,j} := x̂^t_{t,j}
```

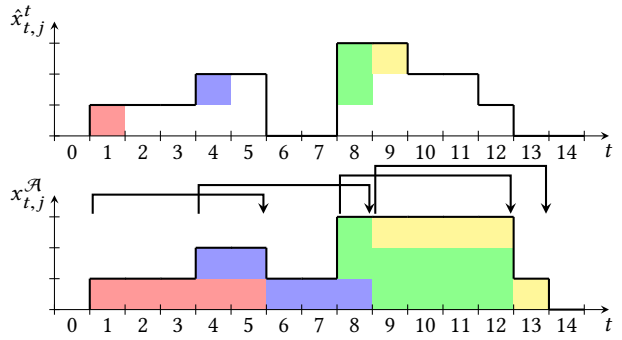

Figure 1: (This figure is colored) **Visualization of algorithm $\mathcal{A}$ for one specific server type j with $\bar{t}_j = 5$. The upper plot shows $\hat{x}^t_{t,j}$, while the lower plot shows the resulting values $x^{\mathcal{A}}_{t,j}$. Note that the upper plot is not an optimal schedule, but the last state of each optimal schedule $\hat{X}^1, \hat{X}^2, \ldots, \hat{X}^T$. The algorithm ensures that $x^{\mathcal{A}}_{t,j} \geq \hat{x}^t_{t,j}$ is always satisfied which is visualized by the colors: Each colored square in the upper plot causes a server to be powered up. The runtime of this server is drawn in the same color in the lower plot. Additionally, the arrows indicate the time slot when a server is powered down (e.g., at time slot 1, a server is powered up, and $\bar{t}_j = 5$ time slots later, it is powered down).**

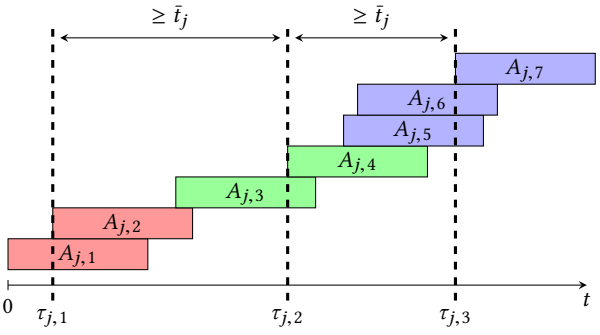

Figure 2: (This figure is colored) **Example of the blocks $A_{j,i}$ (rectangles) and the corresponding special time slots $\tau_{j,k}$ (dashed vertical lines) for one specific server type j. The distance between two consecutive special time slots is always greater than or equal to $\bar{t}_j$. The index block sets $B_{j,k}$ defined in the proof of Lemma 2.6 are $B_{j,1} = \{1, 2\}$ (marked in red), $B_{j,2} = \{3, 4\}$ (green), $B_{j,3} = \{5, 6, 7\}$ (blue).**

presented in Section 4.1. The variables $w_{t,j}$ store how many servers of type j were powered up at time slot t. A visualization of our algorithm is shown in Figure 1.

2.1 Feasibility

Before we determine the competitive ratio of our algorithm, we have to show that the calculated schedule is feasible.

Lemma 2.1. *The schedule $X^{\mathcal{A}}$ is feasible.*

Proof. A schedule is feasible, if (1) $\sum_{j=1}^{d} x_{t,j} z_j^{\max} \geq \lambda_t$ and (2) $x_{t,j} \in [m_j]_0$ holds for all $t \in [T]$ and $j \in [d]$. It is always ensured that $x^{\mathcal{A}}_{t,j} \geq \hat{x}^t_{t,j}$ holds, so condition (1) is satisfied, since $\hat{X}^t$ is a feasible schedule: $\sum_{j=1}^{d} x^{\mathcal{A}}_{t,j} z_j^{\max} \geq \sum_{j=1}^{d} \hat{x}^t_{t,j} z_j^{\max} \geq \lambda_t$.

Servers are powered up only in line 8. Since $\hat{X}^t$ is feasible, $x^{\mathcal{A}}_{t,j} \leq m_j$ is always satisfied. Servers are powered down only in line 5. Each variable $w_{t,j}$ is accessed exactly once, so $x^{\mathcal{A}}_{t,j}$ never gets negative. Therefore, condition (2) is satisfied. □

2.2 Competitiveness

In this section, we will show that algorithm $\mathcal{A}$ is $(2d+1)$-competitive.

For our analysis, we split the operating cost into an *idle* and a *load-dependent* part. The *idle* operating cost of an active server of type j for a single time slot is $f_j(0)$, i.e., it does not depend on the load. The *load-dependent* operating cost of all active servers of type j at time slot t is defined by

$$L_{t,j}(X) \coloneqq x_{t,j} \left(f_j\left(\frac{\lambda_t z_{t,j}}{x_{t,j}}\right) - f_j(0) \right) \tag{3}$$

where $z_{t,j}$ are the values z_j that minimize the right term in equation (1). Formally,

$$(z_{t,1}, \ldots, z_{t,d}) \coloneqq \operatorname*{argmin}_{(z_1,\ldots,z_d)\in\mathcal{Z}} \sum_{j=1}^{d} g_{t,j}(x_{t,j}, z_j).$$

Since f_j is an increasing function, $L_{t,j}(X)$ cannot be negative.

Let $s_{j,1} \leq \cdots \leq s_{j,n_j}$ denote the time slots when in $X^{\mathcal{A}}$ a server of type j is powered up. If n servers of type j are powered up at the same time slot, there are n equal values in the sequence. The time interval $A_{j,i} \coloneqq [s_{j,i} : s_{j,i} + \bar{t}_j - 1]$ is called *block* and contains the time slots when the server is in the active state. The switching and idle operating cost of a block $A_{j,i}$ is at most[1]

$$H_{j,i} \coloneqq \beta_j + \bar{t}_j \cdot f_j(0). \tag{4}$$

For each server type $j \in [d]$ we define special time slots $\tau_{j,1}, \ldots, \tau_{j,n'_j}$ that are constructed in reverse time as follows. τ_{j,n'_j} is defined as the last time slot when a server of type j is powered up in $X^{\mathcal{A}}$, i.e., $\tau_{j,n'_j} \coloneqq s_{j,n_j}$. Given $\tau_{j,k}$, the previous time slot $\tau_{j,k-1}$ is the last powering up of a server of type j before time slot $\tau_{j,k} - \bar{t}_j$. Formally, for $k < n'_j$, $\tau_{j,k}$ is defined by $\tau_{j,k} \coloneqq \max\{s_{j,i} \mid i \in [n_j], s_{j,i} \leq \tau_{j,k+1} - \bar{t}_j\}$. Figure 2 visualizes the definition of $\tau_{j,k}$. Since the runtime of a single server is exactly $\bar{t}_j$, this definition ensures that each block $A_{j,i}$ contains exactly one time slot $\tau_{j,k}$, $k \in [n'_j]$.

As already mentioned in the problem description section, the operating costs of all active servers of type j are minimized if the jobs assigned to type j are equally distributed to the servers of type j. This is formally stated in the lemma below.

[1] If there are two consecutive blocks without a gap between them, there is no switching cost for the second block, so $H_{j,i}$ gives an upper bound for the switching and idle operating cost of $A_{j,i}$.

Lemma 2.2. *Let f be a convex function, $x \in \mathbb{N}$, $\lambda, z \in \mathbb{R}$ and let $\sum_{i=1}^{x} a_i = 1$ with $a_i \geq 0$ for all $i \in [x]$. It holds*

$$x f(\lambda z/x) \leq \sum_{i=1}^{x} f(\lambda z a_i).$$

The proof uses Jensen's inequality and is shown in full version of this paper. The following lemma states that the load-dependent operating cost of $X^{\mathcal{A}}$ at time t is less than or equal to that of $\hat{X}^t$.

Lemma 2.3. *For all $t \in [T]$ and $j \in [d]$, it holds*

$$L_{t,j}(X^{\mathcal{A}}) \leq L_{t,j}(\hat{X}^t).$$

Proof. For $i \in [x_{t,j}^{\mathcal{A}}]$, let

$$a_i := \begin{cases} 1/\hat{x}_{t,j}^t & \text{if } i \leq \hat{x}_{t,j}^t \\ 0 & \text{otherwise.} \end{cases}$$

By using the definition of $L_{t,j}$ (equation (3)) and Lemma 2.2, we get

$$\begin{aligned} L_{t,j}(X^{\mathcal{A}}) &= x_{t,j}^{\mathcal{A}} f_j(\lambda_t z_{t,j}/x_{t,j}^{\mathcal{A}}) - x_{t,j}^{\mathcal{A}} f_j(0) \\ &\overset{L2.2}{\leq} \sum_{i=1}^{x_{t,j}^{\mathcal{A}}} f_j(\lambda_t z_{t,j} a_i) - x_{t,j}^{\mathcal{A}} f_j(0) \\ &= \sum_{i=1}^{\hat{x}_{t,j}^t} f_j(\lambda_t z_{t,j}/\hat{x}_{t,j}^t) + \sum_{i=\hat{x}_{t,j}^t+1}^{x_{t,j}^{\mathcal{A}}} f_j(0) - x_{t,j}^{\mathcal{A}} f_j(0) \\ &= \hat{x}_{t,j}^t f_j(\lambda_t z_{t,j}/\hat{x}_{t,j}^t) - \hat{x}_{t,j}^t f_j(0) \\ &= L_{t,j}(\hat{X}^t). \end{aligned}$$

In the third step, we simply use the definition of a_i and split the sum into two parts. The second sum is equal to $(x_{t,j}^{\mathcal{A}} - \hat{x}_{t,j}^t) f_j(0)$. At the end, we use the definition of $L_{t,j}$, again. □

By using Lemma 2.3, we can show that the load-dependent operating cost of $X^{\mathcal{A}}$ is at most as large as the total cost of the optimal schedule.

Lemma 2.4. *It holds*

$$\sum_{t=1}^{T} \sum_{j=1}^{d} L_{t,j}(X^{\mathcal{A}}) \leq C(\hat{X}^T).$$

Proof. We will prove the inequality $\sum_{t'=1}^{t} \sum_{j=1}^{d} L_{t',j}(X^{\mathcal{A}}) \leq C(\hat{X}^t)$ by induction. For $t = 0$, both terms are zero. Assume that $\sum_{t'=1}^{t-1} \sum_{j=1}^{d} L_{t',j}(X^{\mathcal{A}}) \leq C(\hat{X}^{t-1})$ holds. Let

$$C_M(X) := \sum_{t \in M} \left(g_t(x_{t,1}, \ldots, x_{t,d}) + \sum_{j=1}^{d} \beta_j (x_{t,j} - x_{t-1,j})^+ \right)$$

be the switching and operating cost of X for all time slots $t \in M$. Note that the total cost of a schedule is given by $C_{[1:T]}(X) = C(X)$.

Since $\hat{X}^{t-1}$ is an optimal schedule for $\mathcal{I}^{t-1}$, the cost of $\hat{X}^t$ up to the time slot $t-1$ is greater than or equal to $C(\hat{X}^{t-1})$, i.e., $C(\hat{X}^{t-1}) \leq C_{[1:t-1]}(\hat{X}^t)$. By using this fact as well as the induction hypothesis and Lemma 2.3, we get

$$\begin{aligned} \sum_{t'=1}^{t} \sum_{j=1}^{d} L_{t',j}(X^{\mathcal{A}}) &\overset{I.H.}{\leq} C(\hat{X}^{t-1}) + \sum_{j=1}^{d} L_{t,j}(X^{\mathcal{A}}) \\ &\overset{L2.3}{\leq} C_{[1:t-1]}(\hat{X}^t) + \sum_{j=1}^{d} L_{t,j}(\hat{X}^t) \\ &\leq C_{[1:t-1]}(\hat{X}^t) + C_{\{t\}}(\hat{X}^t) \\ &\leq C(\hat{X}^t). \end{aligned}$$ □

So far, we found an upper bound for the load-dependent operating cost of $X^{\mathcal{A}}$. The following lemma is needed to estimate the switching and idle operating cost of $X^{\mathcal{A}}$ in Lemma 2.6.

Lemma 2.5. *The switching and idle operating cost of the block $A_{j,i}$ is bounded by*

$$H_{j,i} \leq 2 \min\{\beta_j + f_j(0),\ \bar{t}_j \cdot f_j(0)\}.$$

The inequality directly follows from equation (4). The complete calculation is shown in the full paper. The next lemma shows that the switching and idle operating cost of all servers of type j in $X^{\mathcal{A}}$ is at most two times the total cost of the optimal schedule.

Lemma 2.6. *For all $j \in [d]$, it following inequality holds*

$$\sum_{i=1}^{n_j} H_{j,i} \leq 2 \cdot C(\hat{X}^T). \tag{5}$$

Proof. Let $B_{j,k} := \{i \in [n_j] \mid A_{j,i} \ni \tau_{j,k}\}$ with $k \in [n_j']$ be the indices of the blocks containing the time slot $\tau_{j,k}$ (see Figure 2). As already mentioned, each block $A_{j,i}$ contains exactly one time slot $\tau_{j,k}$, so $\bigcup_{k \in [n_j']} B_{j,k} = [n_j]$ and $B_{j,k} \cap B_{j,k'} = \emptyset$ for $k \neq k'$. Therefore, $\sum_{i=1}^{n_j} H_{j,i} = \sum_{k=1}^{n_j'} \sum_{i \in B_{j,k}} H_{j,i}$.

We will prove equation (5) by induction. To simplify the notation, let $\tau_{j,0} := 0$. We will show that

$$\sum_{k=1}^{n} \sum_{i \in B_{j,k}} H_{j,i} \leq 2C(\hat{X}^{\tau_{j,n}}) \tag{6}$$

holds for all $n \in [n_j']_0$. For $n = 0$, the inequality is obviously satisfied (the sum is empty and $\hat{X}^0$ is an empty schedule with zero cost). Assume that inequality (6) holds for $n-1$, i.e., $\sum_{k=1}^{n-1} \sum_{i \in B_{j,k}} H_{j,i} \leq 2C(\hat{X}^{\tau_{j,n-1}})$.

For $I \subseteq [T]$, let

$$C_I(X) := \sum_{t \in I} \left(g_t(\boldsymbol{x}_t) + \sum_{j=1}^{d} \beta_j (x_{t,j} - x_{t-1,j})^+ \right)$$

denote the cost of X during the time interval I. We begin from the left-hand side of equation (6), use our induction hypothesis and get

$$\begin{aligned} \sum_{k=1}^{n} \sum_{i \in B_{j,k}} H_{j,i} &\overset{IH}{\leq} 2C(\hat{X}^{\tau_{j,n-1}}) + \sum_{i \in B_{j,n}} H_{j,i} \\ &\leq 2C_{[1:\tau_{j,n-1}]}(\hat{X}^{\tau_{j,n}}) + \sum_{i \in B_{j,n}} H_{j,i} \end{aligned} \tag{7}$$

The last inequality holds because $\hat{X}^{\tau_{j,n-1}}$ is an optimal schedule for $\mathcal{I}^{\tau_{j,n-1}}$, so $C(\hat{X}^{\tau_{j,n-1}}) \leq C_{[1:\tau_{j,n-1}]}(\hat{X}^{\tau_{j,n}})$.

By the definition of $\tau_{j,k}$, at time $t := \tau_{j,n}$ at least one server of type j is powered up, so

$$\hat{x}^t_{t,j} = x^{\mathcal{A}}_{t,j} = |B_{j,n}|. \tag{8}$$

Furthermore, the cost of $\hat{X}^t$ during the time interval $I := [\tau_{j,n-1}+1 : \tau_{j,n}]$ is at least

$$C_I(\hat{X}^{\tau_{j,n}}) \geq \hat{x}^t_{t,j} \cdot \min\{\beta_j + f_j(0), f_j(0) \cdot \bar{t}_j\} \tag{9}$$

because each server of type j that is active at time slot t was powered up during the time interval I (so there is the switching cost β_j as well as the operating cost for at least one time slot) or it was powered up before I, so it was active for $|I| = \tau_{j,n} - \tau_{j,n-1} \geq \bar{t}_j$ time slots. Since f_j is an increasing function, the operating cost is at least $f_j(0)$ for each time slot.

By using Lemma 2.5 and the equations (8) and (9), we can transform the term (7) to

$$\begin{aligned}
&2C_{[1:\tau_{j,n-1}]}(\hat{X}^{\tau_{j,n}}) + \sum_{i \in B_{j,n}} H_{j,i} \\
&\overset{L2.5}{\leq} 2C_{[1:\tau_{j,n-1}]}(\hat{X}^{\tau_{j,n}}) + |B_{j,n}| \cdot 2\min\{\beta_j + f_j(0), f_j(0) \cdot \bar{t}_j\} \\
&\overset{(8)}{\leq} 2C_{[1:\tau_{j,n-1}]}(\hat{X}^{\tau_{j,n}}) + 2\hat{x}^t_{t,j} \min\{\beta_j + f_j(0), f_j(0) \cdot \bar{t}_j\} \\
&\overset{(9)}{\leq} 2C_{[1:\tau_{j,n-1}]}(\hat{X}^{\tau_{j,n}}) + 2C_{[\tau_{j,n-1}+1:\tau_{j,n}]}(\hat{X}^{\tau_{j,n}}) \\
&\leq 2C(\hat{X}^{\tau_{j,n}}).
\end{aligned}$$

Therefore, equation (6) is satisfied for all $n \in [n'_j]_0$. For $n = n'_j$, we get

$$\sum_{i=1}^{n_j} H_{j,i} = \sum_{k=1}^{n'_j} \sum_{i \in B_{j,k}} H_{j,i} \leq 2C(\hat{X}^{\tau_{j,n'_j}}) \leq 2C(\hat{X}^T). \qquad \square$$

Now, we are able to prove the competitive ratio of algorithm $\mathcal{A}$.

Theorem 2.7. *Algorithm $\mathcal{A}$ is $(2d+1)$-competitive.*

Proof. The total cost of $X^{\mathcal{A}}$ is the switching and idle operating cost given by $\sum_{j=1}^{d}\sum_{i=1}^{n_j} H_{j,i}$ plus the load-dependent operating cost given by $\sum_{t=1}^{T}\sum_{j=1}^{d} L_{t,j}(X^{\mathcal{A}})$. By using Lemmas 2.6 and 2.4, we get

$$\begin{aligned}
C(X^{\mathcal{A}}) &= \sum_{j=1}^{d}\sum_{i=1}^{n_j} H_{j,i} + \sum_{t=1}^{T}\sum_{j=1}^{d} L_{t,j}(X^{\mathcal{A}}) \\
&\overset{\substack{L2.6\\L2.4}}{\leq} \sum_{j=1}^{d} 2 \cdot C(\hat{X}^T) + C(\hat{X}^T) \\
&= (2d+1) \cdot C(\hat{X}^T).
\end{aligned}$$

The schedule $\hat{X}^T$ is optimal for the problem instance $\mathcal{I}$, so algorithm $\mathcal{A}$ is $(2d+1)$-competitive. $\square$

If the operating costs are load independent, i.e., $f_j(z) = l_j = \text{const}$ for all $j \in [d]$, then the load-dependent operating cost $L_{t,j}(X^{\mathcal{A}})$ is always zero. Thus, the competitive ratio of algorithm $\mathcal{A}$ is $2d$, so it matches the lower bound given in [5]. In contrast to the deterministic $2d$-competitive online algorithm presented in [5], our algorithm can handle inefficient server types, which were excluded in [5].

Corollary 2.8. *If the operating cost functions are load- and time-independent, algorithm $\mathcal{A}$ achieves an optimal competitive ratio of $2d$.*

3 ONLINE ALGORITHM FOR TIME-*DEPENDENT* OPERATING COST FUNCTIONS

In this section, we present a modified version of algorithm $\mathcal{A}$ that is able to handle time-dependent operating cost functions $f_{t,j}$ and achieves a competitive ratio of $2d+1+\epsilon$ for any $\epsilon > 0$. The proof is divided into two parts. First, as an intermediate result we introduce algorithm $\mathcal{B}$ that is $\left(2d+1+\sum_{j=1}^{d}\max_{t\in[T]}\frac{f_{t,j}(0)}{\beta_j}\right)$-competitive. Then, in Subsection 3.2 we show how the given problem instance $\mathcal{I}$ can be modified to make the constant $c(\mathcal{I}) := \sum_{j=1}^{d}\max_{t\in[T]}\frac{f_{t,j}(0)}{\beta_j}$ arbitrarily small. Finally, the resulting schedule is adapted to the original problem instance without increasing its cost.

3.1 Obtaining a competitive ratio of $2d+1+c(\mathcal{I})$

To handle time-dependent operating cost functions, algorithm $\mathcal{A}$ has to be modified, as the idle operating cost $f_{t,j}(0)$ is no longer constant over time. Similar to algorithm $\mathcal{A}$, in algorithm $\mathcal{B}$ a server is powered down when its accumulated idle operating cost $f_{t,j}(0)$ exceeds its switching cost. Formally, let $l_{t,j} := f_{t,j}(0)$ be the idle operating cost of server type j during time slot t and let

$$\bar{t}_{t,j} := \max\left\{\bar{t} \in [T-t] \mid \sum_{u=t+1}^{t+\bar{t}} l_{u,j} \leq \beta_j\right\}$$

be the maximal number of time slots such that the sum of the idle operating costs beginning from time slot $t+1$ is smaller than or equal to β_j. A server that is powered up at time slot t runs for $\bar{t}_{t,j}$ *further* time slots, i.e., it is powered down at time slot $t + \bar{t}_{t,j}$. This definition differs from $\bar{t}_j$ in algorithm $\mathcal{A}$ where a server is powered down at $t + \bar{t}_j - 1$. Note that the idle operating cost at time slot t does not influence the runtime of a server. The power-up policy of algorithm $\mathcal{B}$ is the same as in algorithm $\mathcal{A}$, i.e., it is always ensured that the number of active servers of type j is at least as large as the corresponding number in an optimal schedule for the problem instance that ends at the current time slot. Formally, $x^{\mathcal{B}}_{t,j} \geq \hat{x}^t_{t,j}$ holds for all $t \in [T]$ and $j \in [d]$.

In contrast to algorithm $\mathcal{A}$, the runtime of a server is not known when it is powered up, because the future operating cost functions did not arrive yet, so $\bar{t}_{t,j}$ cannot be calculated at this time. However, the runtime is known at the time slot when the server must be powered down, so $\mathcal{B}$ is a valid online algorithm. The pseudocode below clarifies how algorithm $\mathcal{B}$ works. Note that only lines 5 and 6 change in comparison to algorithm $\mathcal{A}$. The set W_t defined in line 5 contains all time slots u with $u + \bar{t}_{u,j} + 1 = t$. Servers that were powered up at time slot u are shut down at time slot t. Figure 3 visualizes the definition of $\bar{t}_{t,j}$ and W_t and shows an example of how algorithm $\mathcal{B}$ operates.

Before we analyze the competitive ratio of algorithm $\mathcal{B}$, we have to prove that the calculated schedule $X^{\mathcal{B}}$ is feasible.

Lemma 3.1. *The schedule $X^{\mathcal{B}}$ is feasible.*

The proof is similar to the feasibility proof of algorithm $\mathcal{A}$. In fact, the argumentations for $\sum_{j=1}^{d} x_{t,j} z^{\max}_j \geq \lambda_t$ and $x_{t,j} \leq m_j$ are

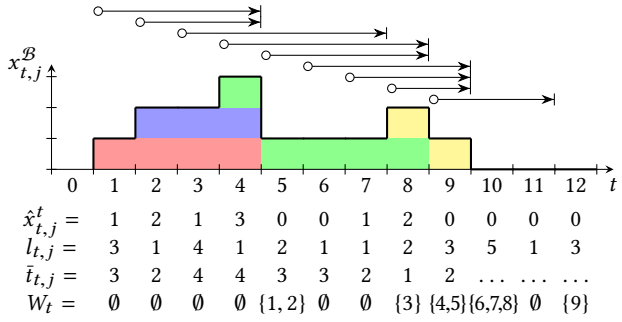

Figure 3: (This figure is colored) **Visualization of algorithm $\mathcal{B}$ for one specific server type j with $\beta_j = 6$. The plot shows the number of active servers $x^{\mathcal{B}}_{t,j}$ of algorithm $\mathcal{B}$. The colors indicate the running time of each server. The values $\hat{x}^t_{t,j}$ (that are needed to determine when a server has to be powered up) and the idle operating costs $l_{t,j}$ as well as the resulting values of $\bar{t}_{t,j}$ and W_t are shown below the plot. The running time $\bar{t}_{t,j}$ of a server that is powered up a time slot t is indicated by the arrows, e.g., a server that is powered up at time slot $t = 2$ runs for $\bar{t}_{2,j} = 2$ *additional* time slots, so it is powered down at the end of time slot $t + \bar{t}_{t,j} = 4$. The values $\bar{t}_{t,j}$ are the maximal number of time slots *after* t such that the idle operating costs do not exceed β_j, e.g., $\bar{t}_{2,j} = 2$, because $l_{3,j} + l_{4,j} = 4 + 1 = 5 \leq \beta_j = 6$, but $l_{3,j} + l_{4,j} + l_{5,j} = 7 > \beta_j$. At time slot t, the servers that were powered up at time $u \in W_t$ are shut down, e.g., $W_5 = \{1, 2\}$, so both the red and the blue server are powered down at time slot 5. For $t \geq 10$, the values of $\bar{t}_{t,j}$ are not known yet, because they depend on $l_{13,j}$.**

Algorithm 2 Algorithm $\mathcal{B}$

```
w_{t,j} := 0 for all t ∈ ℤ and j ∈ [d]
for t := 1 to T do
    Calculate X̂^t
    for j := 1 to d do
        W_t := { u ∈ [t−1] | Σ_{v=u+1}^{t−1} l_{v,j} ≤ β_j < Σ_{v=u+1}^{t} l_{v,j} }
        x^B_{t,j} := x^B_{t,j} − Σ_{u∈W_t} w_{u,j}
        if x^B_{t,j} ≤ x̂^t_{t,j} then
            w_{t,j} := x̂^t_{t,j} − x^B_{t,j}
            x^B_{t,j} := x̂^t_{t,j}
```

the same. To verify that $x_{t,j} \geq 0$, we show that the sets W_t are disjoint. This implies that each $w_{t,j}$ is accessed at most once. The complete proof can be found in the full version of this paper.

The analysis of the competitive ratio of algorithm $\mathcal{B}$ is quite similar to that of algorithm $\mathcal{A}$. Let $L_{t,j}(X) := x_{t,j}\left(f_{t,j}\left(\frac{\lambda_t z_{t,j}}{x_{t,j}}\right) - l_{t,j}\right)$ denote the load-dependent operating cost of X. Lemmas 2.2 and 2.3 still hold, since in their proofs we can simply replace f_j with $f_{t,j}$. Lemma 2.4 directly follows from Lemma 2.3, so it also remains applicable.

The schedule $X^{\mathcal{B}}$ is divided into blocks $A_{j,i} := [s_{j,i} : s_{j,i} + \bar{t}_{t,j}]$ (the definition of $s_{j,i}$ remains the same). The switching and idle operating cost of a block $A_{j,i}$ is at most

$$H_{j,i} := \beta_j + \sum_{u=s}^{s+\bar{t}_{s,j}} l_{u,j} \tag{10}$$

with $s = s_{j,i}$. The special time slots $\tau_{j,k}$ are defined in the same way as in the previous section. Formally, they are given by $\tau_{j,n'_j} := s_{j,n_j}$ and $\tau_{j,k} := \max\{s_{j,i} \mid i \in [n_j], s_{j,i} + \bar{t}_{s_{j,i},j} < \tau_{j,k+1}\}$ for $1 \leq k < n'_j$ as well as $\tau_{j,0} := 0$. The definition of the index sets $B_{j,k} = \{i \in [n_j] \mid A_{j,i} \ni \tau_{j,k}\}$ do not change. The following lemma replaces Lemma 2.5 and gives an upper bound for $H_{j,i}$.

Lemma 3.2. *The switching and idle operating cost of $A_{j,i}$ is at most*

$$H_{j,i} \leq 2\beta_j + \max_{t\in[T]} l_{t,j}.$$

Proof. Let $s := s_{j,i}$. By the definition of $\bar{t}_{s,j}$, we know that $\sum_{u=s+1}^{s+\bar{t}_{s,j}} \leq \beta_j$. We use this inequality in equation 10 and get $H_{j,i} = \beta_j + \sum_{u=s}^{s+\bar{t}_{s,j}} l_{u,j} \leq 2\beta_j + l_{s,j} \leq 2\beta_j + \max_{t\in[T]} l_{t,j}$. □

The next lemma replaces Lemma 2.6 and shows that the switching and idle operating costs caused by server type j are at most $2 + \max_{t\in[T]} l_{t,j}/\beta_j$ times larger than the total cost of an optimal schedule.

Lemma 3.3. *For all $j \in [d]$, it holds*

$$\sum_{i=1}^{n_j} H_{j,i} \leq \left(2 + \max_{t\in[T]} \frac{l_{t,j}}{\beta_j}\right) \cdot C(\hat{X}^T). \tag{11}$$

The proof uses Lemma 3.2 and is very similar to that of Lemma 2.6. See the full version of this paper for more details. Now, we are able to prove the competitive ratio of algorithm $\mathcal{B}$.

Theorem 3.4. *Algorithm $\mathcal{B}$ is $(2d + 1 + c(\mathcal{I}))$-competitive with $c(\mathcal{I}) = \sum_{j=1}^{d} \max_{t\in[T]} \frac{l_{t,j}}{\beta_j}$.*

The proof uses Lemmas 2.4 and 3.3 and is analogous to that of Theorem 2.7. See the full version of this paper for more details.

3.2 Improving the competitive ratio to $2d + 1 + \epsilon$

In the following, we show how the competitive ratio can be reduced to $2d + 1 + \epsilon$ for any $\epsilon > 0$. Given the original problem instance $\mathcal{I} = (T, d, \boldsymbol{m}, \boldsymbol{\beta}, F, \Lambda)$, we consider the modified problem instance $\tilde{\mathcal{I}} = (\tilde{T}, d, \boldsymbol{m}, \boldsymbol{\beta}, \tilde{F}, \tilde{\Lambda})$ where each time slot t of the original problem instance is divided into $\tilde{n}_t$ equal sub time slots. The values $\tilde{n}_t \in \mathbb{N}$ are defined later. The total number of time slots is given by $\tilde{T} := \sum_{t=1}^{T} \tilde{n}_t$. In the following, time slots in the original problem instance $\mathcal{I}$ are denoted by t, whereas time slots in the modified problem instance $\tilde{\mathcal{I}}$ are denoted by u. Let $U(t)$ be the set of time slots in the modified problem instance $\tilde{\mathcal{I}}$ that corresponds to the time slot $t \in [T]$ in the original problem instance $\mathcal{I}$. Formally, $U(t) := [u + 1 : u + \tilde{n}_t]$ with $u = \sum_{t'=1}^{t-1} \tilde{n}_{t'}$. Furthermore, we define $U^{-1}(u)$ with $u \in [\tilde{T}]$ to be the time slot $t \in [T]$ such that $u \in U(t)$. The operating cost functions of $\tilde{\mathcal{I}}$ are defined as

$$\tilde{f}_{u,j}(z) := \frac{1}{\tilde{n}_t} f_{t,j}(z)$$

with $t = U^{-1}(u)$ for all $u \in [\tilde{T}]$ and $j \in [d]$, so the operating cost during time slot t is divided into $\tilde{n}_t$ equal parts. The idle operating

cost is denoted by $\tilde{l}_{u,j} \coloneqq \tilde{f}_{u,j}(0)$ for $u \in [\tilde{T}]$ and $j \in [d]$. The job volumes do not change, so $\lambda_u \coloneqq \lambda_{U^{-1}(u)}$ for all $u \in [\tilde{T}]$. In other words, $\tilde{\mathcal{I}}$ matches the problem instance $\mathcal{I}$ where intermediate state changes are allowed.

Let $n \in \mathbb{N}$. We set $\tilde{n}_t = n \cdot \max_{j\in[d]} \frac{l_{t,j}}{\beta_j}$ and apply algorithm $\mathcal{B}$ on the corresponding problem instance $\tilde{\mathcal{I}}$. Therefore, we get

$$c(\tilde{\mathcal{I}}) = \sum_{j=1}^{d} \max_{u\in\tilde{T}} \frac{\tilde{l}_{u,j}}{\beta_j} = \sum_{j=1}^{d} \max_{t\in T} \frac{l_{t,j}}{\tilde{n}_t \beta_j} \le \sum_{j=1}^{d} \max_{t\in\tilde{T}} \frac{1}{n} = \frac{d}{n}. \quad (12)$$

In the second step, we use that $\tilde{l}_{u,j} = l_{t,j}/\tilde{n}_t$ with $t = U^{-1}(u)$. The inequality holds because $\tilde{n}_t \ge n \cdot \frac{l_{t,j}}{\beta_j}$ for all $j \in [d]$. To achieve a competitive ratio of $2d + 1 + \epsilon$, we set $n = d/\epsilon$. For $n \to \infty$, the competitive ratio converges to $2d + 1$.

We still have to show how the resulting $(2d + 1 + \epsilon)$-competitive schedule for the modified problem instance $\tilde{\mathcal{I}}$ can be transformed into a feasible schedule for the original problem instance $\mathcal{I}$. Let $X^{\mathcal{B}}$ be the schedule created by $\mathcal{B}$ and let X^C be the final schedule for $\mathcal{I}$. For each original time slot $t \in [T]$, let $\boldsymbol{x}_t^C \coloneqq \boldsymbol{x}_{\mu(t)}^{\mathcal{B}}$ with $\mu(t) \coloneqq \operatorname{argmin}_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_u^{\mathcal{B}})$ be the server configuration that minimizes the operating cost during the time interval $U(t)$.

The pseudocode below shows how the schedule X^C is calculated. For each arriving operating cost function $f_{t,j}$, the next $\tilde{n}_t$ time slot of $\tilde{\mathcal{I}}$ are created and passed to algorithm $\mathcal{B}$. Afterward, the next server configuration $\boldsymbol{x}_t^C$ is determined. The whole schedule $X^{\mathcal{B}}$ cannot be calculated at once, because the state $\boldsymbol{x}_t^C$ must be fixed before the next function $f_{t+1,j}$ can be processed.

Algorithm 3 Algorithm C

1: Initialize algorithm $\mathcal{B}$
2: **for** $t \coloneqq 1$ **to** T **do**
3: Create the next $\tilde{n}_t$ time slots of the modified problem instance $\tilde{\mathcal{I}}$ with $\tilde{n}_t \coloneqq d/\epsilon \cdot \max_{j\in[d]} l_{t,j}/\beta_j$
4: Execute $\tilde{n}_t$ time slots in algorithm $\mathcal{B}$
5: $\boldsymbol{x}_t^C \coloneqq \boldsymbol{x}_{\mu(t)}^{\mathcal{B}}$ with $\mu(t) \coloneqq \operatorname{argmin}_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_u^{\mathcal{B}})$

The following lemma shows that this procedure does not increase the cost of the schedule.

Lemma 3.5. *The total cost of X^C regarding the problem instance $\mathcal{I}$ is smaller than or equal to the total cost of $X^{\mathcal{B}}$ regarding the modified problem instance $\tilde{\mathcal{I}}$.*

Proof. Let $C_{\text{op}}^{\mathcal{J}}(X)$ be the operating cost of the schedule X regarding the problem instance $\mathcal{J} \in \{\mathcal{I}, \tilde{\mathcal{I}}\}$ and let $C_{\text{sw}}^{\mathcal{J}}(X)$ denote its switching cost.

First, we will compare the operating cost of both schedules. The operating cost of $X^{\mathcal{B}}$ is given by

$$C_{\text{op}}^{\tilde{\mathcal{I}}}(X^{\mathcal{B}}) = \sum_{u=1}^{\tilde{T}} \tilde{g}_u(\boldsymbol{x}_u^{\mathcal{B}}) = \sum_{t=1}^{T} \sum_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_u^{\mathcal{B}}) \ge \sum_{t=1}^{T} \tilde{n}_t \cdot \min_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_u^{\mathcal{B}}).$$

For the last inequality, we estimate each summand by the minimum of all summands. By using the definition of $\boldsymbol{x}_t^C$, we get

$$\sum_{t=1}^{T} \tilde{n}_t \cdot \min_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_u^{\mathcal{B}}) = \sum_{t=1}^{T} \tilde{n}_t \cdot \min_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_t^C).$$

The definition of $\tilde{f}_{u,j}$ implies that $\tilde{g}_u(\boldsymbol{x}) = \frac{1}{\tilde{n}_t} g_t(\boldsymbol{x})$ with $t = U^{-1}(u)$, so

$$\sum_{t=1}^{T} \tilde{n}_t \cdot \min_{u\in U(t)} \tilde{g}_u(\boldsymbol{x}_t^C) = \sum_{t=1}^{T} g_t(\boldsymbol{x}_t^C) = C_{\text{op}}^{\mathcal{I}}(X^C).$$

Altogether we have shown that $C_{\text{op}}^{\tilde{\mathcal{I}}}(X^{\mathcal{B}}) \ge C_{\text{op}}^{\mathcal{I}}(X^C)$.

Next, we will compare the switching cost. To simplify the notation, let $S(\boldsymbol{x}, \boldsymbol{x}') \coloneqq \sum_{j=1}^{d} \beta_j (x'_j - x_j)^+$ be the switching cost from the state $\boldsymbol{x}$ to $\boldsymbol{x}'$. The total switching cost of $X^{\mathcal{B}}$ is given by

$$C_{\text{sw}}^{\tilde{\mathcal{I}}}(X^{\mathcal{B}}) = \sum_{u=1}^{\tilde{T}} S(\boldsymbol{x}_{u-1}^{\mathcal{B}}, \boldsymbol{x}_u^{\mathcal{B}}) = \sum_{t=1}^{T+1} \sum_{u=\mu(t-1)+1}^{\mu(t)} S(\boldsymbol{x}_{u-1}^{\mathcal{B}}, \boldsymbol{x}_u^{\mathcal{B}})$$

with $\mu(0) \coloneqq 0$ and $\mu(T + 1) \coloneqq \tilde{T} + 1$. In the last step, the interval $[\tilde{T}]$ is partitioned into the sub-intervals $[1 : \mu(1)], [\mu(1) + 1 : \mu(2)], \ldots, [\mu(T) + 1 : \tilde{T} + 1]$ (note that the switching cost from time slot $\tilde{T}$ to $\tilde{T} + 1$ is always 0, since $\boldsymbol{x}_{\tilde{T}+1}^{\mathcal{B}} = 0$ by definition). The switching cost during each interval is at least as large as the switching cost for jumping directly to the last state of the interval. Therefore,

$$\sum_{t=1}^{T+1} \sum_{u=\mu(t-1)+1}^{\mu(t)} S(\boldsymbol{x}_{u-1}^{\mathcal{B}}, \boldsymbol{x}_u^{\mathcal{B}}) \le \sum_{t=1}^{T+1} S(\boldsymbol{x}_{\mu(t-1)}^{\mathcal{B}}, \boldsymbol{x}_{\mu(t)}^{\mathcal{B}}).$$

By using the definition of $\boldsymbol{x}_t^C$, we get

$$\sum_{t=1}^{T+1} S(\boldsymbol{x}_{\mu(t-1)}^{\mathcal{B}}, \boldsymbol{x}_{\mu(t)}^{\mathcal{B}}) = \sum_{t=1}^{T+1} S(\boldsymbol{x}_{t-1}^C, \boldsymbol{x}_t^C) = C_{\text{sw}}^{\mathcal{I}}(X^C),$$

so $C_{\text{sw}}^{\tilde{\mathcal{I}}}(X^{\mathcal{B}}) \ge C_{\text{sw}}^{\mathcal{I}}(X^C)$. □

Now, we can prove that algorithm C is $(2d + 1 + \epsilon)$-competitive.

Theorem 3.6. *For any $\epsilon > 0$, there is a $(2d + 1 + \epsilon)$-competitive algorithm for the data-center right-sizing problem with heterogeneous servers and time-dependent operating cost functions.*

Proof. Let $X^*_{\mathcal{J}}$ be an optimal schedule for the problem instance $\mathcal{J} \in \{\mathcal{I}, \tilde{\mathcal{I}}\}$ and let $C^{\mathcal{J}}(X)$ denote the total cost of X with respect to $\mathcal{J}$. We have to show that $C^{\mathcal{I}}(X^C) \le (2d + 1 + \epsilon) \cdot C^{\mathcal{I}}(X^*_{\mathcal{I}})$. By using Lemma 3.5, Theorem 3.4 and the competitive ratio of algorithm $\mathcal{B}$ given by equation (12), we get

$$\begin{aligned} C^{\mathcal{I}}(X^C) &\overset{L3.5}{\le} C^{\tilde{\mathcal{I}}}(X^{\mathcal{B}}) \\ &\overset{T3.4}{\le} \left(2d + 1 + c(\tilde{\mathcal{I}})\right) \cdot C^{\tilde{\mathcal{I}}}(X^*_{\tilde{\mathcal{I}}}) \\ &\overset{(12)}{\le} (2d + 1 + \epsilon) \cdot C^{\tilde{\mathcal{I}}}(X^*_{\tilde{\mathcal{I}}}) \le C^{\mathcal{I}}(X^*_{\mathcal{I}}). \end{aligned}$$

The last inequality holds because each feasible schedule X for the problem instance $\mathcal{I}$ can be converted into a feasible schedule $\tilde{X}$ for the modified problem instance $\tilde{\mathcal{I}}$ without increasing the cost.

Formally, the definition $\tilde{\boldsymbol{x}}_u := \boldsymbol{x}_{U^{-1}(u)}$ for all $u \in [\tilde{T}]$ implies $C^{\tilde{I}}(\tilde{X}) = C^{I}(X)$. Therefore, an optimal schedule for $\mathcal{I}$ cannot have a lower cost than an optimal schedule for $\tilde{\mathcal{I}}$. □

4 APPROXIMATION ALGORITHM

In this section, we consider the offline version of the data-center right-sizing problem and present a $(1+\epsilon)$-approximation algorithm that runs in $O\left(T \cdot \epsilon^{-d} \cdot \prod_{j=1}^{d} \log m_j\right)$ time, which is polynomial if d is a constant. It is based on an optimal, graph-based algorithm that is presented in the following subsection. Afterward, in Section 4.2, we show how the optimal algorithm can be modified to obtain a $(1+\epsilon)$-approximation in polynomial time.

To simplify the following calculations we introduce some notations. Let $M_j := [m_j]_0$ and $\mathcal{M} := \bigtimes_{j=1}^{d} M_j$ be the set of all possible server configurations. The operating cost is denoted by $C_{\text{op}}(X) := \sum_{t=1}^{T} g_t(\boldsymbol{x}_t)$ and the switching cost is denoted by $C_{\text{sw}}(X) := \sum_{t=1}^{T} \sum_{j=1}^{d} \beta_j (x_{t,j} - x_{t-1,j})^+$.

4.1 Optimal offline algorithm

An optimal schedule can be found by converting the problem instance $\mathcal{I}$ to a graph and finding the shortest path.

The graph $G(\mathcal{I})$ (or simply denoted by G) contains $2T \cdot \prod_{j=1}^{d}(m_j+1)$ vertices arranged in a $(d+1)$-dimensional grid (where the first dimension has $2T$ layers). For each time slot $t \in [T]$ and each server configuration $\boldsymbol{x} = (x_1, \dots, x_d) \in \mathcal{M}$, there are two vertices in the graph denoted by $v^{\uparrow}_{t,\boldsymbol{x}}$ and $v^{\downarrow}_{t,\boldsymbol{x}}$. There is an edge $e^{\text{op}}_{t,\boldsymbol{x}}$ from $v^{\uparrow}_{t,\boldsymbol{x}}$ to $v^{\downarrow}_{t,\boldsymbol{x}}$ with weight $g_t(\boldsymbol{x})$ representing the operating cost during time slot t. For each $j \in [d]$ and for each

$$\boldsymbol{x} = (x_1, \dots, x_d) \in M_1 \times \cdots \times (M_j \setminus \{m_j\}) \times \cdots \times M_d$$

(note that $x_j = m_j$ is excluded), let $\boldsymbol{x}' := (x_1, \dots, x_j + 1, \dots, x_d)$. There is an edge $e^{\uparrow}_{t,\boldsymbol{x},j}$ from $v^{\uparrow}_{t,\boldsymbol{x}}$ to $v^{\uparrow}_{t,\boldsymbol{x}'}$ with weight β_j (a server of type j is powered up) and another edge $e^{\downarrow}_{t,\boldsymbol{x}',j}$from $v^{\downarrow}_{t,\boldsymbol{x}'}$ to $v^{\downarrow}_{t,\boldsymbol{x}}$ with weight 0 (a server of type j is powered down). Furthermore, for each $t \in [T-1]$ we need an edge $e^{\rightarrow}_{t,\boldsymbol{x}}$ from $v^{\uparrow}_{t,\boldsymbol{x}}$ to $v^{\uparrow}_{t+1,\boldsymbol{x}}$ with weight 0 to switch to the next time slot.

Let $\mathbf{0} := (0, \dots, 0) \in \mathcal{M}$. Each schedule X for the problem instance $\mathcal{I}$ can be represented by a path P_X between $v^{\uparrow}_{1,\mathbf{0}}$ and $v^{\downarrow}_{T,\mathbf{0}}$. For each $t \in [T]$, the path uses the edge $e^{\text{op}}_{t,\boldsymbol{x}_t}$. The vertices $v^{\downarrow}_{t,\boldsymbol{x}_t}$ and $v^{\uparrow}_{t+1,\boldsymbol{x}_{t+1}}$ (for $t \in [T-1]$) are connected by an arbitrary shortest path between them. The same is done for the start and the end point. Note that the sum of the weights of the path's edges is equal to the cost of the schedule.

On the other hand, a given path P between $v^{\uparrow}_{1,\mathbf{0}}$ and $v^{\downarrow}_{T,\mathbf{0}}$ represents a schedule X^P. If the path uses the edge $e^{\text{op}}_{t,\boldsymbol{x}}$, then the corresponding schedule uses the server configuration $\boldsymbol{x}$ during time slot t. If P does not use a shortest path between $v^{\downarrow}_{t,\boldsymbol{x}_t}$ and $v^{\uparrow}_{t+1,\boldsymbol{x}_{t+1}}$ (for $t \in [T-1]$), then the sum of the weights of the path's edges are greater than the cost of the corresponding schedule. However, by replacing the path's vertices between $v^{\downarrow}_{t,\boldsymbol{x}_t}$ and $v^{\uparrow}_{t+1,\boldsymbol{x}_{t+1}}$ for all $t \in [T-1]$ by a shortest sub-path, both values are equal.

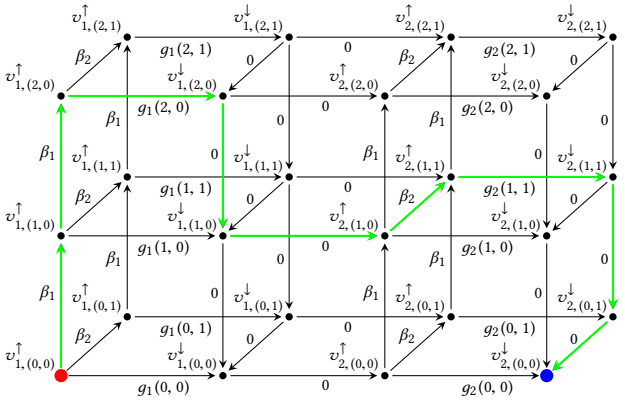

Figure 4: (This figure is colored) **Visualization of the graph representation. This example shows two server types ($d = 2$) and two time slots ($T = 2$). There are $m_1 = 2$ servers of type 1 and $m_2 = 1$ server of type 2. The algorithm calculates a shortest path from $v^{\uparrow}_{1,(0,0)}$ (red dot) to $v^{\downarrow}_{2,(0,0)}$ (blue dot). The shortest path is drawn in green and corresponds to the optimal schedule $\boldsymbol{x}_1 = (2,0)$ and $\boldsymbol{x}_2 = (1,1)$.**

A shortest path between $v^{\uparrow}_{1,\mathbf{0}}$ and $v^{\downarrow}_{T,\mathbf{0}}$ corresponds to an optimal schedule. Owing to the graph structure, a shortest path can be calculated with dynamic programming in $O(T \cdot \prod_{j=1}^{d} m_j)$ time. Note that this runtime is not polynomial (even if d is a constant), because the encoding length of the problem instance is $O(T + \sum_{j=1}^{d} \log m_j)$. The graph structure and the relation between a shortest path and an optimal schedule are visualized in Figure 4.

4.2 $(1+\epsilon)$-approximation

In this section, we develop a $(1+\epsilon)$-approximation which has a polynomial runtime, if d and ϵ are constants. The basic idea is to reduce the number of possible values for $x_{t,j}$, that is, we will calculate an optimal solution where the number of active servers can only take specific values. Broadly speaking, the number of active servers are powers of a constant $\gamma > 1$. For example, we will see that using the values $x_{t,j} \in \{0, 1, 2, 4, 8, \dots, m_j\}$ (i.e., each power of two up to m_j as well as m_j and 0) would result in a 3-approximation. The set of values that will be used for the number of active servers of type j is

$$\begin{aligned} M_j^{\gamma} &:= \{0, m_j\} \cup \{\lfloor \gamma^k \rfloor \in M_j \mid k \in \mathbb{N}\} \cup \{\lceil \gamma^k \rceil \in M_j \mid k \in \mathbb{N}\} \\ &= \{0, 1, \lfloor \gamma^1 \rfloor, \lceil \gamma^1 \rceil, \lfloor \gamma^2 \rfloor, \lceil \gamma^2 \rceil, \dots, m_j\}. \end{aligned}$$

Using both the rounded down and rounded up values of γ^k ensures that the ratio between two consecutive values is not larger than γ. Note that $|M_j^{\gamma}| \in O(\log_{\gamma} m_j)$. Furthermore, we define $\mathcal{M}^{\gamma} := \bigtimes_{j=1}^{d} M_j^{\gamma}$ as the set of server configurations that will be used in our algorithm. For a given value $x_j < m_j$, let $N_j(x_j)$ be the next greater value of x_j in M_j^{γ}, i.e., $N_j(x_j) := \min\{x \in M_j^{\gamma} \mid x > x_j\}$.

The reduced graph G^{γ} contains the vertices $v^{s}_{t,\boldsymbol{x}}$ with $s \in \{\uparrow, \downarrow\}$, $t \in [T]$ and $\boldsymbol{x} \in \mathcal{M}^{\gamma}$. Similar to G there is an edge from $v^{\uparrow}_{t,\boldsymbol{x}}$ to

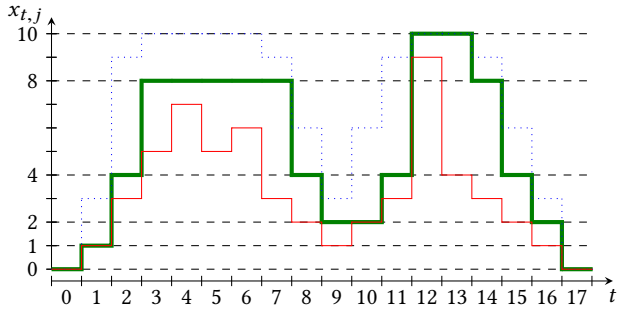

Figure 5: (This figure is colored) **Visualization of the construction of X' (shown in green) for one specific server type j. In this example, we have $\gamma = 2$ and $m_j = 10$, so the allowed states for X' are $M_j^\gamma = \{0, 1, 2, 4, 8, 10\}$ (dashed horizontal lines). The optimal schedule X^* is shown in red. The dotted blue line shows the value of $\min(m_j, (2\gamma-1)x^*_{t,j})$. Note that the schedule X' always stays between the red and the blue line and only changes the number of active servers to ensure the invariant.**

$v^{\downarrow}_{t,\boldsymbol{x}}$ with weight $g_t(\boldsymbol{x})$ (for all $t \in [T]$ and $\boldsymbol{x} \in \mathcal{M}^\gamma$) and an edge from $v^{\downarrow}_{t,\boldsymbol{x}}$ to $v^{\uparrow}_{t+1,\boldsymbol{x}}$ with cost 0 (for all $t \in [T-1]$ and $\boldsymbol{x} \in \mathcal{M}^\gamma$). For each $j \in [d]$ and for each

$$\boldsymbol{x} = (x_1, \dots, x_d) \in M_1^\gamma \times \cdots \times (M_j^\gamma \setminus \{m_j\}) \times \cdots \times M_d^\gamma,$$

let $\boldsymbol{x}' := (x_1, \dots, N_j(x_j), \dots, x_d)$. There is an edge from $v^{\uparrow}_{t,\boldsymbol{x}}$ to $v^{\uparrow}_{t,\boldsymbol{x}'}$ with weight $\beta_j(N_j(x_j) - x_j)$ and an edge from $v^{\downarrow}_{t,\boldsymbol{x}'}$ to $v^{\downarrow}_{t,\boldsymbol{x}}$ with weight 0.

Theorem 4.1. *Let P^γ be a shortest path in G^γ and X^γ the corresponding schedule. Let X^* be an optimal schedule for the original problem instance. Then, the inequality*

$$C(X^\gamma) \le (2\gamma - 1) \cdot C(X^*) \tag{13}$$

is satisfied, i.e., X^γ is a $(2\gamma-1)$-approximation.

To prove this theorem, we construct a path P' in G^γ with the corresponding schedule X' that is not necessarily a shortest path, however, it will satisfy the inequality $C(X') \le (2\gamma-1) \cdot C(X^*)$. The cost of X^γ can only be smaller, because the corresponding path P^γ is a shortest path in G^γ, so if X' is a $(2\gamma-1)$-approximation, then X^γ is a $(2\gamma-1)$-approximation too.

Given the optimal solution X^* the states of X' are defined by

$$x'_{t,j} = \begin{cases} x_{\min} & \text{if } x'_{t-1,j} \le x^*_{t,j} \\ x'_{t-1,j} & \text{if } x^*_{t,j} < x'_{t-1,j} \le (2\gamma-1) \cdot x^*_{t,j} \\ x_{\max} & \text{if } (2\gamma-1) \cdot x^*_{t,j} < x'_{t-1,j} \end{cases} \tag{14}$$

with $x_{\min} = \min\{x \in M_j^\gamma \mid x \ge x^*_{t,j}\}$ and $x_{\max} = \max\{x \in M_j^\gamma \mid x \le (2\gamma-1) \cdot x^*_{t,j}\}$ for all $t \in [T]$ and $j \in [d]$ (with $\boldsymbol{x}'_0 = \boldsymbol{0}$). Note that the invariant

$$x^*_{t,j} \le x'_{t,j} \le (2\gamma-1) \cdot x^*_{t,j} \tag{15}$$

is always satisfied. The construction of X' is visualized in Figure 5.

The complete proof of Theorem 4.1 is shown in the full version of this paper. Roughly, it works as follows. First, we show that the operating cost of X' is a $(2\gamma-1)$-approximation of the operating cost of X^*, i.e., $C_{\text{op}}(X') \le (2\gamma-1) \cdot C_{\text{op}}(X^*)$. The proof uses the definition of X', especially the invariant $x'_{t,j} \le (2\gamma-1) \cdot x^*_{t,j}$, as well as the fact that $f_{t,j}$ are non-negative increasing functions. Afterward, it is shown that the switching cost of X' is $(2\gamma-1)$-approximation of the switching cost of X^*. For this, the schedule X' is divided into phases such that in each phase servers are either powered up or down. By using the invariant (15) and the fact that the relative distance between two states in X' is at most γ, it is shown that each phase is a $(2\gamma-1)$-approximation according to the switching cost. Finally, both parts combined with the inequality $C(X^\gamma) \le C(X')$ lead to $C(X^\gamma) \le (2\gamma-1) \cdot C(X^*)$.

If we set $\gamma = (1+\epsilon/2)$, then $2\gamma - 1 = (1+\epsilon)$, so we have a $(1+\epsilon)$-approximation. For server type j, there are $|M_j^\gamma| \in O(\log_\gamma m_j) = O(\log_{1+\epsilon} m_j)$ different values that are used by our graph-based algorithm. Thus the graph consists of $O\left(T \cdot \prod_{j=1}^d \log_{1+\epsilon} m_j\right)$ vertices which is also the algorithm's runtime. For $\epsilon < 1$ (usually we are not interested in ϵ-values that are bigger than 1) the term $\frac{1}{\log(1+\epsilon)}$ can be written as $1/\epsilon + O(1)$, so the runtime is $O\left(T \cdot \epsilon^{-d} \cdot \prod_{j=1}^d \log m_j\right)$. We summarize our results in the following theorem:

Theorem 4.2. *Given the problem instance $\mathcal{I}$, a $(1+\epsilon)$-approximation can be calculated in $O\left(T \cdot \epsilon^{-d} \cdot \prod_{j=1}^d \log m_j\right)$ time.*

4.3 Time-dependent data-center size

In practice, the size of a data center can change over time. If a data center is extended with new servers of type j, then m_j increases. If parts of the data center are shut down for maintenance, m_j decreases temporarily. Let $m_{t,j}$ denote the total number of servers of type j at time slot t. In the following, we will show that the approximation algorithm still works in this setting.

Let $M_{t,j} := [m_{t,j}]_0$ and $\mathcal{M}_t = \bigtimes_{j=1}^d M_{t,j}$ be the allowed server configurations at time slot t. The vertices in G that represent unavailable server configurations are removed along with the incident edges. The shortest path in the new graph represents an optimal schedule. For the approximation, let

$$M^\gamma_{t,j} := \{0, m_{t,j}\} \cup \{\lfloor \gamma^k \rfloor \in M_{t,j} \mid k \in \mathbb{N}\} \cup \{\lceil \gamma^k \rceil \in M_{t,j} \mid k \in \mathbb{N}\}$$

and let $\mathcal{M}^\gamma_t := \bigtimes_{j=1}^d M^\gamma_{t,j}$ be the considered server configurations. The resulting graph is denoted by $\bar{G}^\gamma$. Theorem 4.1 still hold for the modified graph, i.e., the schedule that corresponds to the shortest path in $\bar{G}^\gamma$ is a $(2\gamma-1)$-approximation. The following theorem shows that a $(1+\epsilon)$-approximation can still be calculated in polynomial time (if d is a constant). The proof is analogous to that of Theorem 4.2.

Theorem 4.3. *Given the problem instance $\bar{\mathcal{I}}$ where the total number of available servers depends on time, a $(1+\epsilon)$-approximation can be calculated in*

$$O\left(\epsilon^{-d} \cdot \sum_{t=1}^T \prod_{j=1}^d \log m_{t,j}\right) \subseteq O\left(T \cdot \epsilon^{-d} \cdot \prod_{j=1}^d \log \max_{t \in [T]} m_{t,j}\right)$$

time.

REFERENCES

[1] Susanne Albers. 2017. On Energy Conservation in Data Centers. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. ACM, 35–44.

[2] Susanne Albers. 2019. On energy conservation in data centers. *ACM Transactions on Parallel Computing (TOPC)* 6, 3 (2019), 1–26.

[3] Susanne Albers and Jens Quedenfeld. 2018. Optimal algorithms for right-sizing data centers. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*. ACM, 363–372.

[4] Susanne Albers and Jens Quedenfeld. 2018. Optimal Algorithms for Right-Sizing Data Centers—Extended Version. *arXiv preprint arXiv:1807.05112* (2018).

[5] Susanne Albers and Jens Quedenfeld. 2021. Algorithms for Energy Conservation in Heterogeneous Data Centers. In *Algorithms and Complexity - 11th International Conference, CIAC 2021*. Springer.

[6] Lachlan LH Andrew, Minghong Lin, and Adam Wierman. 2010. Optimality, fairness, and robustness in speed scaling designs. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. 37–48.

[7] Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, Kevin Schewior, and Michele Scquizzato. 2016. Chasing convex bodies and functions. In *Proc. 12th Latin American Symposium on Theoretical Informatics (LATIN'16)*. Springer, 68–81.

[8] Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. 2020. Parallel Machine Scheduling to Minimize Energy Consumption. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2758–2769.

[9] Antonios Antoniadis and Kevin Schewior. 2017. A tight lower bound for online convex optimization with switching costs. In *International Workshop on Approximation and Online Algorithms*. Springer, 164–175.

[10] CJ Argue, Anupam Gupta, Guru Guruganesh, and Ziye Tang. 2020. Chasing convex bodies with linear competitive ratio. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1519–1524.

[11] Masoud Badiei, Na Li, and Adam Wierman. 2015. Online convex optimization with ramp constraints. In *54th IEEE Conference on Decision and Control (CDC)*. IEEE, 6730–6736.

[12] Nikhil Bansal, Martin Böhm, Marek Eliáš, Grigorios Koumoutsos, and Seeun William Umboh. 2018. Nested Convex Bodies are Chaseable. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1253–1260.

[13] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Cliff Stein. 2015. A 2-competitive algorithm for online convex optimization with switching costs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015) (LIPIcs, Vol. 40)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 96–109.

[14] Tom Bawden. 2016. Global warming: Data centres to consume three times as much energy in next decade, experts warn. http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html

[15] Sébastien Bubeck, Bo'az Klartag, Yin Tat Lee, Yuanzhi Li, and Mark Sellke. 2020. Chasing nested convex bodies nearly optimally. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1496–1508.

[16] Niangjun Chen, Anish Agarwal, Adam Wierman, Siddharth Barman, and Lachlan LH Andrew. 2015. Online convex optimization using predictions. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 43. ACM, 191–204.

[17] Niangjun Chen, Gautam Goel, and Adam Wierman. 2018. Smoothed Online Convex Optimization in High Dimensions via Online Balanced Descent. *Proceedings of Machine Learning Research* 75 (2018), 1574–1594.

[18] Pierre Delforge and et al. 2014. Data Center Efficiency Assessment. https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf

[19] Gautam Goel, Niangjun Chen, and Adam Wierman. 2017. Thinking fast and slow: Optimization decomposition across timescales. In *IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 1291–1298.

[20] Gautam Goel and Adam Wierman. 2019. An Online Algorithm for Smoothed Regression and LQR Control. *Proceedings of Machine Learning Research* 89 (2019), 2504–2513.

[21] Seung-Jun Kim and Geogios B Giannakis. 2014. Real-time electricity pricing for demand response using online convex optimization. In *ISGT 2014*. IEEE, 1–5.

[22] Minghong Lin, Zhenhua Liu, Adam Wierman, and Lachlan LH Andrew. 2012. Online algorithms for geographical load balancing. In *Green Computing Conference (IGCC)*. IEEE, 1–10.

[23] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. 2013. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking* 21, 5 (2013), 1378–1391.

[24] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. 2013. Dynamic right-sizing for power-proportional data centers — Extended Version.

[25] Yiheng Lin, Gautam Goel, and Adam Wierman. 2020. Online Optimization with Predictions and Non-convex Losses. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 1 (2020), 1–32.

[26] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H Low, and Lachlan LH Andrew. 2011. Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. ACM, 233–244.

[27] Sparsh Mittal. 2014. *Power Management Techniques for Data Centers: A Survey*. Technical Report. Future Technologies Group, Oak Ridge National Laboratory.

[28] Patrick Schmid and Achim Roos. 2009. Overclocking Core i7: Power Versus Performance. http://www.tomshardware.com/reviews/overclock-core-i7,2268.html

[29] Mark Sellke. 2020. Chasing convex bodies optimally. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1509–1518.

[30] Amar Shan. 2006. Heterogeneous processing: a strategy for augmenting moore's law. *Linux Journal* 2006, 142 (2006), 7.

[31] Hao Wang, Jianwei Huang, Xiaojun Lin, and Hamed Mohsenian-Rad. 2014. Exploring smart grid and data center interactions for electric power load balancing. *ACM SIGMETRICS Performance Evaluation Review* 41, 3 (2014), 89–94.

[32] Adam Wierman, Lachlan LH Andrew, and Ao Tang. 2009. Power-aware speed scaling in processor sharing systems. In *IEEE INFOCOM 2009*. IEEE, 2007–2015.

[33] Ming Zhang, Zizhan Zheng, and Ness B Shroff. 2018. An online algorithm for power-proportional data centers with switching cost. In *IEEE Conference on Decision and Control (CDC)*. IEEE, 6025–6032.