Renting Servers in the Cloud

Mahtab Masoori

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Computer Science) at

Concordia University

Montréal, Québec, Canada

March 2025

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mahtab Masoori**

Entitled: **Renting Servers in the Cloud**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Emre Erkmen

_____ External Examiner
Dr. Spyros Angelopoulos

_____ Examiner
Dr. Jeremy Clark

_____ Examiner
Dr. Jaroslav Opatrny

_____ Examiner
Dr. Hovhannes Harutyunyan

_____ Supervisor
Dr. Denis Pankratov

_____ Co-supervisor
Dr. Lata Narayanan

Approved by _____
Joey Paquet, Chair
Department of Computer Science and Software Engineering

_____ 2025

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Renting Servers in the Cloud

Mahtab Masoori, Ph.D.

Concordia University, 2025

We study the Renting Servers in the Cloud problem (*RSiC*), which addresses the need for efficient job allocation in cloud computing applications. Jobs arrive in an online manner and need to be assigned to servers. The size of each job is known at the time of arrival. If the duration of the jobs is also known, the scenario is termed clairvoyant; if the duration is unknown, it is non-clairvoyant. An infinite supply of identical servers is available, each providing one unit of computational capacity per unit of time across all dimensions. A server can be rented at any time and remains rented until all assigned jobs are completed. The cost of an assignment is the sum of the rental periods of all servers. The objective is to allocate jobs to servers in a way that minimizes the overall cost while adhering to server capacity constraints.

We first focus on a scenario where all jobs have equal durations and analyze the performance of two natural algorithms, *NextFit* and *FirstFit*. We establish a tight bound of 2 on the competitive ratio of *NextFit*. For *FirstFit*, we also prove bounds under several restrictions. Next, we conduct a parameterized analysis of *FirstFit*, examining inputs where it utilizes at most k servers simultaneously. We support the theoretical analysis with extensive experimental studies. Then, we show for a large class of well-known algorithms for *RSiC*, none of them always outperforms the others. We study the multi-dimensional *RSiC* setting, where jobs/servers have multi-parameter resource demands/capacities (e.g., cores, memory). We demonstrate a direct sum property of *RSiC*. We also propose a novel clairvoyant algorithm called *Greedy*; our experiments demonstrate its superiority over existing algorithms in most scenarios. We introduce and analyze performance of a new subfamily of *AnyFit* algorithms termed monotone *AnyFit*, which includes *Greedy*, *FirstFit*, *LastFit*, and

MoveToFront. Finally we evaluate both clairvoyant and non-clairvoyant algorithms for *RSiC* on real-world data using the Azure dataset. The results are sometimes different from those previously obtained on synthetic data. We also proposed some new algorithms that are combinations of known algorithms and outperform all existing algorithms in our experiments.

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Dr. Denis Pankratov and Dr. Lata Narayanan, for their continuous support throughout my Ph.D. Their patience, motivation, and kindness have been invaluable. I have learned so much from them—not only about science and academia but also about personal growth.

I am also sincerely grateful to Dr. Yaqiao Li and Dr. Shahin Kamali for their support and contributions to my thesis. Their insights and encouragement have played a significant role in shaping my research.

I extend my thanks to my committee members: Dr. Jaroslav Opatrny, Dr. Hovhannes Harutyunyan, and Dr. Jeremy Clark; as well as my external committee member, Dr. Spyros Angelopoulos, for their time and effort in reviewing my thesis. I deeply appreciate their constructive feedback which have helped refine my work.

Finally, I am profoundly thankful to my family for their unwavering support on this journey. Your belief in me has been a constant source of strength, and this achievement would not have been possible without you.

Dedication

To my parents

Contents

List of Figures x

List of Tables xiv

1 Introduction 1

1.1 Motivation . . . 1

1.2 Notation and Problem Statement . . . 3

1.2.1 Known Algorithms for *RSiC* . . . 5

1.3 Competitive Analysis . . . 6

1.3.1 Lower Bounds on OPT . . . 7

1.4 Thesis Contributions and Outline . . . 9

2 Literature Review 13

2.1 The 1-Dimensional Setting ($d = 1$) . . . 13

2.1.1 Classical Bin Packing . . . 14

2.1.2 Dynamic Bin Packing . . . 17

2.1.3 Renting Servers in the Cloud . . . 19

2.2 The d-Dimensional Setting ($d > 1$) . . . 21

2.2.1 d-Dimensional Bin Packing . . . 21

2.2.2 d-Dimensional Dynamic Bin Packing . . . 22

2.2.3 d-Dimensional Renting Servers in the Cloud . . . 22

3 The Case of Equal Duration Jobs **24**

3.1 Notation and Preliminaries . . . 25

3.2 Tight Competitive Ratio 2 for *NextFit* . . . 26

3.3 *FirstFit* . . . 29

3.3.1 Lower Bound for Jobs with Two Arrival Times . . . 29

3.3.2 Upper Bound for Jobs with Two Arrival Times . . . 33

3.3.3 Strict Upper Bound: Equal Duration 2 and Arrival Times 0 and 1 . . . 41

3.4 Summary and Discussion . . . 50

4 Equal Duration and Midterm Arrivals **52**

4.1 Long-Running Uniform Servers . . . 53

4.1.1 Upper Bound for Long-Running Uniform Servers . . . 53

4.1.2 Lower Bound for Long-Running Uniform Servers . . . 57

4.2 Inputs for Dual-Core Servers . . . 58

4.2.1 Upper Bound . . . 58

4.2.2 Lower Bound . . . 64

4.2.3 Lower Bound on Advice Complexity . . . 65

4.3 Summary and Discussion . . . 70

5 Restricted Number of Servers **71**

5.1 Theoretical Analysis of *FirstFit* for $k = 2, 3, 4$ . . . 72

5.2 Experimental Results . . . 79

5.2.1 The Effect of the Number of Jobs . . . 81

5.2.2 The Effect of the Job Arrival Rate . . . 82

5.2.3 The Effect of Job Sizes . . . 84

5.3 Summary and Discussion . . . 86

6 Instance Incomparability of Some *RSiC* Algorithms **88**

6.1 Instance Dominance and Instance Incomparability . . . 88

6.2 Catalog of Instances . . . 90

6.3 Summary and Discussion . . . 116

7 The Case of Multi-Parameter Jobs **117**

7.1 Preliminaries . . . 117

7.2 A $\Theta(d\sqrt{\log \mu})$ Algorithm via a Direct-Sum Property of *RSiC* . . . 118

7.3 Lower Bound for Randomized Algorithms for $d = 1$ . . . 120

7.4 *Greedy* and the Class of Monotone *AnyFit* Algorithms . . . 121

7.5 Experiments . . . 126

7.5.1 Experimental Setup . . . 127

7.5.2 Implemented Algorithms . . . 127

7.5.3 Experimental Results . . . 127

7.6 Summary and Discussion . . . 132

8 Experiments on Real-World Data **133**

8.1 Description of the Dataset . . . 133

8.1.1 Analysis of the Dataset . . . 134

8.2 Experimental Setup . . . 137

8.3 Experiments on the Full Dataset . . . 139

8.4 Experiments on μ-Filtered Data . . . 141

8.5 New Combined Algorithms . . . 143

8.5.1 Combined Clairvoyant Algorithms . . . 143

8.5.2 Combined Weakly Clairvoyant Algorithms . . . 147

8.6 Wasted-Space and Duration of Servers . . . 148

8.6.1 Clairvoyant Algorithms . . . 148

8.6.2 Non-clairvoyant Algorithms . . . 150

8.7 Summary and Discussion . . . 153

9 Conclusion **154**

References . . . 157

List of Figures

Figure 1.1 Visual representation of server S states and transitions: circles represent possible server statuses (Open and Active, Closed and Active, Closed and Inactive) and arrows indicate actions. Status definitions: Open: server accepts new requests if capacity allows; Closed: server does not accept new requests; Active: server contains at least one ongoing job; Inactive: all jobs on the server have completed. . 4

Figure 1.2 Online assignment of jobs into servers described in Example 1.2.1. *FirstFit* on (a) the first dimension, (b) the second dimension. 5

Figure 1.3 OPT assignment of jobs into servers described in Example 1.2.1. OPT on (a) the first dimension, (b) the second dimension. 6

Figure 2.1 Illustration of the example instance: (a) The optimal solution for static bin packing. (b) The optimal solution for dynamic bin packing. 18

Figure 3.1 B_1 must have a job that arrived in $(t-1, t]$. Server B_i was closed at time $t_i \in [t-1, t]$ by a job that was placed in B_{i+1}. 28

Figure 3.2 An illustration of schedules for the adversarial instance from Theorem 3.3.1 constructed by (a) *FirstFit*. (b) OPT. 32

Figure 4.1 The bound in Corollary 4.1.3 is shown in a tight example. 57

Figure 4.2 The change in number of active chains for (a) OPT. (b) ALG. The labels e and o on edges refer to even and odd n_i respectively. For OPT, the states correspond to the number of active chains, and for ALG, they correspond to the parity of the number of active chains c_i. 59

Figure 4.3 The assignment of ALG on the *ASC* input with cardinality sequence 121121211 61

Figure 4.4 (a) ALG. (b) OPT. In this figures, the shaded areas indicate the wasted space within each server. 65
Figure 4.5 (a) ALG. (b) OPT. In the figures, the shaded areas indicate the wasted space within each server. 65
Figure 5.1 Schedule of *FirstFit* on the adversarial input demonstrating $\rho_2(FirstFit) \geq 2$. 73
Figure 5.2 Schedule of OPT on the adversarial input demonstrating $\rho_2(FirstFit) \geq 2$. 73
Figure 5.3 Schedule of *FirstFit* on the adversarial input demonstrating $\rho_3(FirstFit) \geq 3$. 75
Figure 5.4 Schedule of OPT on the adversarial input demonstrating $\rho_3(FirstFit) \geq 3$. 75
Figure 5.5 The effect of number of jobs on the performance of *FirstFit*. The performance of *FirstFit* is seen to be stable with at least 500 jobs for different choices of λ. The results are averaged over 100 trials. 79
Figure 5.6 The effect of the job arrival rate on the performance of *FirstFit*. The total number of accepted jobs is 500. The results are averaged over 100 trials. 82
Figure 5.7 The effect of the job arrival rate on average size of accepted and rejected jobs. The total number of accepted jobs is 500. The results are averaged over 100 trials. 83
Figure 5.8 The effect of the job arrival rate on effective rate of accepted jobs and effective rate of rejected jobs. The total number of accepted jobs is 500. The results are averaged over 100 trials. 84
Figure 5.9 The effect of the interval $(\ell, u]$ from which job size is sampled on the performance metric of *FirstFit*. The total number of accepted jobs is 500. The results are averaged over 100 trials. 85
Figure 6.1 Example A and the assignment of (a) *FirstFit*, *BestFit* and Greedy. (b) *NextFit*, *MoveToFront*, *LastFit*, and *WorstFit*. 92
Figure 6.2 Instance B and the assignment of (a) *FirstFit* and *BestFit*. (b) *Greedy*, *MoveToFront*, *LastFit*, *WorstFit*, and *NextFit*. 93
Figure 6.3 Instance D and the assignment of (a) *NextFit*. (b) All *AnyFit* algorithms. . . 97

Figure 6.4 Instance E and the assignment of (a) *LastFit* and *BestFit*. (b) *FirstFit*, *MoveToFront*, *WorstFit*, *Greedy*. The assignment of *NextFit* is similar to part(b) with one extra server for all the items in the second stage. 99

Figure 6.5 Instance F and the assignment of (a) *LastFit* and *WorstFit*. (b) *MoveToFront*, *FirstFit*, *BestFit*, and *Greedy*. 101

Figure 6.6 Instance G and the assignment of (a) *NextFit*. (b) All *AnyFit* algorithms. . . 103

Figure 6.7 Instance H and the assignment of *WorstFit*. 105

Figure 6.8 Instance H and the assignment of *BestFit*, *FirstFit*, and *Greedy*. Note that *LastFit*, *MTF*, and *NextFit* follow a similar structure; the key difference is that they keep the last server open to accommodate jobs from stage 2 and beyond. 106

Figure 6.9 Instance I and the assignment of *BestFit*. 108

Figure 6.10 Instance I and the assignment of *FirstFit*. The *Greedy*, *LastFit*, *WorstFit*, *MoveToFront*, and *NextFit* algorithms also produce the same assignment. However, these algorithms differ by extending the last server rather than the first server during the assignment process. 109

Figure 6.11 Instance J and the assignment of (a) *Greedy*, *LastFit*, *WorstFit*, *MoveToFront*. (b) *BestFit* and *FirstFit*. 111

Figure 6.12 Instance K and the assignment of (a) *Greedy*, *FirstFit*, and *BestFit*. (b) *MoveToFront*, *NextFit*, *LastFit*, and *WorstFit*. 112

Figure 6.13 Instance L and the assignment of (a) *Greedy*, *LastFit*, *MoveToFront*, and *BestFit*. (b) *FirstFit*, and *WorstFit*. 114

Figure 7.1 Configuration of servers in interval $[t - 2\mu, t]$. Note that the A_i servers are ordered according to the ordering of ALG at time $t - \mu$ while the B_i servers are ordered by opening time. 124

Figure 8.1 Computational resources demand for all dimensions for machine 0. Note that since all type 0 machines are identical, their computing capacities are the same such as memory, cores, and other resources. All values are represented as integers and we scale all resource capacities to 1000 in order to standardize the analysis. . . 135

Figure 8.2 Sum of the computational resources demand for each dimensions at each specific time for machine 0. Note that all values are represented as integers. 136
Figure 8.3 The frequency of each job duration is shown for all jobs. Note that all values are represented as integers. 137
Figure 8.4 Performance of the new combined clairvoyant algorithms for $\tau \in [10^3, 10^{10}]$. 144
Figure 8.5 Distribution of jobs compared to the threshold. Note that 99.76% of jobs fall below threshold 10^9, while only 0.24% exceed it. 146
Figure 8.6 Comparison between the competitive ratio of the new clairvoyant combined algorithms. 146
Figure 8.7 The performance of the weakly clairvoyant combined algorithms over different values of τ. 148
Figure 8.8 Duration of servers for clairvoyant algorithms. 149
Figure 8.9 Wasted-space of servers for clairvoyant algorithms. 151
Figure 8.10 Duration of servers for non-clairvoyant algorithms. Note that due to overlapping values, *MFF* hides *FirstFit*, and *NextFit* similarly hides *MNF*. 152
Figure 8.11 Wasted-space of servers for non-clairvoyant algorithms. Note that due to overlapping values, *MFF* hides *FirstFit*, and *NextFit* similarly hides *MNF*. 152

List of Tables

Table 1.1 An overview of the various *RSiC* settings studied in each chapter of this thesis. By midterm arrival pattern, we mean that the arrival times of jobs are multiples of half of the job durations, under the assumption that all jobs have equal durations. The rest of the settings are self explanatory. 9

Table 2.1 Summary of the result for the online *RSiC*. Bold text highlights our results. Note that when $d = 1$, the *Hybrid Algorithm*$^{\oplus d}$ algorithm is identical to the *Hybrid Algorithm* algorithm. 23

Table 3.1 Overview of the settings for *RSiC* with uniform duration of jobs. 25

Table 3.2 Different types of servers in category D 37

Table 4.1 Overview of the settings for *RSiC* with equal duration and midterm arrivals. . 53

Table 5.1 Overview of the settings for *RSiC* with equal duration and midterm arrivals. . 71

Table 5.2 A sequence of indices and the corresponding n_i, m_i pairs that illustrating i_0, i_1, i_2. 78

Table 5.3 Values of λ that result in worst performance for various values of k. 82

Table 6.1 A summary of the *RSiC* setting for the instance incomparability of algorithms. 88

Table 6.2 Summary of instance characteristics including number of sizes, distinct arrival times, and duration counts. 115

Table 6.3 Theoretical comparison of algorithms. (A, k) in the table entry (i, j) indicate that algorithm j is better than algorithm i on instance A by a factor k. 115

Table 7.1 A summary of the *RSiC* setting for multi-parameter jobs. 117

Table 7.2 Implemented Algorithms. Similar to [45], we adopt the parameters for *ModifiedNextFit* and *ModifiedFirstFit* as $E^d/(\mu+1)$ and $E^d/(\mu+7)$, respectively. This choice of values is designed to optimize the competitive ratio of these algorithms, as indicated in [45, 52]. 128
Table 7.3 Average competitive ratio results for the *RSiC* problem when $d = 1$. 129
Table 7.4 Average competitive ratio results for the *RSiC* problem when $d = 2$. 129
Table 7.5 Average competitive ratio results for the *RSiC* problem when $d = 4$. 130
Table 7.6 Average competitive ratio results for the *RSiC* problem when $d = 5$. 130
Table 8.1 Number of jobs and the ratio μ for each dataset. 138
Table 8.2 Competitive ratio results for the *RSiC* problem on real data. Algorithms are listed in decreasing order of competitive ratio for Machine 0, Priority 0. 139
Table 8.3 Competitive ratio results for the *RSiC* problem on machine 0 for jobs with priority 0 after filtering the jobs based on μ. 142
Table 8.4 Job distribution for different thresholds on job durations. 145
Table 8.5 Number of servers that each clairvoyant algorithm used on the full dataset. . 149
Table 8.6 Number of servers that each non-clairvoyant algorithm used on the full dataset. 150

Chapter 1

Introduction

1.1 Motivation

Cloud computing [31] has become a key technology, fundamentally changing how computing services are provided and used. By delivering resources such as servers, storage, databases, and software over the internet, cloud computing allows users to access powerful infrastructure on a pay-as-you-go basis. This model eliminates the need for large investments in physical hardware and software, reducing capital expenses and offering cost savings, as users pay only for the resources they actually use. The growing importance of cloud computing is driven by the need for flexibility and scalability in today's dynamic business world. Companies can quickly adjust their IT resources to match changing demands, allowing them to respond swiftly to market shifts and stay competitive. This adaptability is crucial as organizations undergo digital transformation, using cloud services to implement advanced technologies like artificial intelligence, machine learning, and big data analytics.

Cloud service providers offer services and resources, hosted on computing or storage servers to their customers over the internet. Customers submit *jobs* to the system, which arrive at different times, have different durations, and have different resource requirements. The service provider must assign these jobs to its servers as they arrive, while considering the available capacity of each of its servers. For example, users request virtual machines (VMs) with certain resource requirements, and a cloud service provider such as Microsoft Azure must place the VMs on its physical servers [36].

The total time that the physical servers are active directly contributes to power and other costs for the cloud service provider. Poor placement decisions can result in fragmentation and over-provisioning of physical resources. Indeed, Hadary et al. [36] claim that even a 1% reduction in fragmentation can lead to cost savings of the order of $100M a year for Microsoft.

As another application, cloud gaming companies such as GaiKai, OnLive, and StreamMyGame rent servers from public cloud companies and are charged using a pay-as-you-go model. A customer's request to play a game is assigned to one of the rented servers that has enough capacity to serve the request. The rental cost paid by the gaming company is directly proportional to the duration of time that the servers are rented. Both the above situations could be modeled by the *Renting Servers in the Cloud (RSiC)* problem which is the focus of this thesis.

The *RSiC* problem was first introduced by Li et al. [52]. In this problem, jobs with resource requirements across multiple parameters (e.g., number of GPUs or CPUs, memory, network bandwidth, etc.) arrive at the system and must be assigned to servers with fixed capacities along each of these dimensions. This is referred to as the d-dimensional *RSiC* problem. Each job's resource needs are represented as a vector, where each component corresponds to a different type of resource. If a job requires only a single type of resource, the problem is referred to as the 1-dimensional *RSiC*.

RSiC is naturally an *online* problem, where jobs arrive and leave the system in sequence over time. In an online setting, the service provider must make irrevocable assignment decisions immediately upon each job's arrival, without any prior knowledge of future jobs. In contrast, an offline approach would involve having access to the entire sequence of jobs in advance. The online nature of this problem highlights the challenge of making decisions with only partial and evolving information. Naturally the size of a job is known at the time of arrival but its duration may or may not be. The problem has been studied both in the clairvoyant setting, where the job duration is also known at the arrival time, and in the non-clairvoyant setting, where the job duration is known only when the job departs. In this problem, job migration from one server to another is assumed to be prohibitively expensive; therefore, the assignment decisions are irrevocable. The performance of an online algorithm is usually measured by its *competitive ratio*, which is the worst-case ratio (over the inputs) between the cost achieved by the online algorithm and the cost achieved by an offline optimal solution that knows the entire input instance in advance. One distinguishes two notions of

competitive ratio: *strict*, which applies to all inputs, and *asymptotic*, which applies only to large enough inputs.

RSiC is a generalization of the classical *bin packing* problem, which corresponds to the input scenario for *RSiC* when all jobs arrive at the same time and have equal duration. *RSiC* introduces a temporal component to the classical problem. Another related problem is the *dynamic bin packing* problem, introduced by Coffman et al. [20], which however has a different objective – the goal is to minimize the maximum *number* of servers rented at any point of time during the operation of the system. The objective function of *RSiC* is also known as *Min Usage Time*. In recent years, interest in *RSiC* has increased due to the proliferation of cloud-computing services and the pay-as-you-go model, and the *Min Usage Time* objective is more suitable to model such services [52].

1.2 Notation and Problem Statement

In this section, we introduce the notation used throughout the thesis and present fundamental facts related to the *RSiC* problem. We begin by providing a general overview of the *RSiC* problem, including its input parameters and objective function.

The input to the d-dimensional *RSiC* problem is a sequence of n items $\sigma = (\sigma_1, \ldots, \sigma_n)$, where item i is a triple (a_i, f_i, s_i), denoting a job starting at time $a_i \in [0, \infty)$ and finishing at time $f_i > a_i$ where multi-dimensional resource demand $s_i \in (0, 1)^d$ such that s_i^j denotes the size of the job σ_i in the j^{th} dimension for $j \in [d]$. We assume that an algorithm for *RSiC* has access to a supply of identical servers of capacity 1 in each dimension, i.e, the size of each server is 1^d. Thus, for every time t the combined size of jobs in a particular dimension assigned to a particular server at time t must not exceed 1. For the same starting time a, jobs starting at time a are presented in an adversarial order. An online algorithm for *RSiC* receives one job σ_i at a time, and it must make an *irrevocable* decision on which server to schedule job i before job $i + 1$ arrives. The goal of the online algorithm for *RSiC* is to *minimize the total cost of all servers used*, where the cost of each server is equal to the duration for which it is rented/utilized.

The *duration* of a job i is $f_i - a_i$; we sometimes use $d(\sigma_i)$ to represent it.

An important parameter of the input sequence is $\mu(\sigma) := \frac{\max_i(f_i - s_i)}{\min_i(f_i - s_i)}$, which is the ratio of the

maximum duration of a job to the minimum duration of a job in the sequence.

We say that a job $\sigma_i = (a_i, f_i, s_i)$ is *active* or *alive* at time t if $t \in [a_i, f_i)$. Two jobs σ_i and σ_j are said to be *non-overlapping* if $[a_i, f_i) \cap [a_j, f_j) = \emptyset$, otherwise they are said to be *overlapping*. Note that representing active times of jobs by half-open intervals allows us to consider two jobs such that one starts at exactly the same time as the other one finishes as non-overlapping. Note that when the number of dimension $d = 1$, each item i has $s_i \in (0, 1]$, and servers possess capacity of 1. Jobs are presented in an order that respects their starting times, i.e., $a_1 \leq a_2 \leq \cdots \leq a_n$.

Opening a server signifies the start of its rental period, typically marked by the assignment of the first job to it. A server is considered *active* or *alive* at time t if it currently hosts at least one scheduled job that remains active at t. A server is considered *closed* when it will no longer receive any future job assignments; it is important to note that this status is determined by the algorithm rather than an inherent characteristic of the server itself. Once a server is closed it is no longer open. Despite being closed, a server may continue to be active until all its assigned jobs are completed. Once all jobs are finished, the server is *released*, and subsequently, it becomes *inactive*. See Figure 1.1 for the visual representation of a server. Additionally, we define the duration of each server B, denoted by $d(B)$, as the time interval from its opening until its last active job finishes. In this terminology, the goal of the *RSiC* problem is to minimize the total duration of all rented servers.

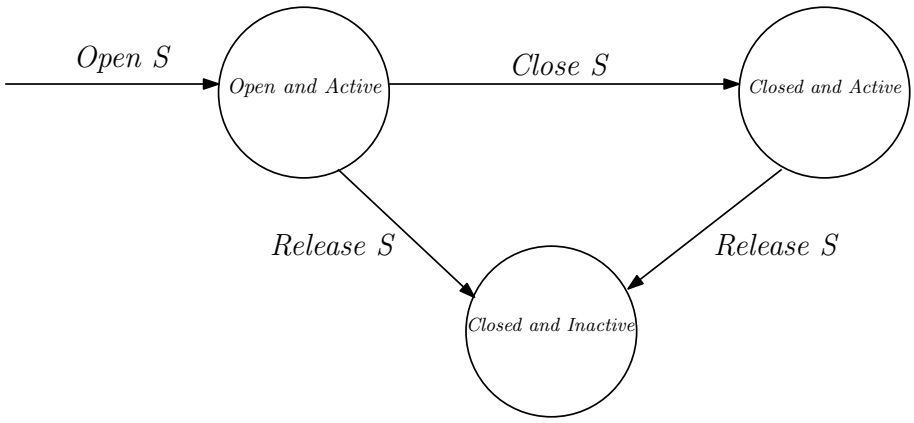

Figure 1.1: Visual representation of server S states and transitions: circles represent possible server statuses (Open and Active, Closed and Active, Closed and Inactive) and arrows indicate actions. Status definitions: Open: server accepts new requests if capacity allows; Closed: server does not accept new requests; Active: server contains at least one ongoing job; Inactive: all jobs on the server have completed.

1.2.1 Known Algorithms for *RSiC*

Several algorithms commonly used in the bin packing problem are also applicable to the *RSiC* problem, including *NextFit*, *WorstFit*, *BestFit*, and *FirstFit*. The *NextFit* algorithm allows only one server to be open at a time. If there is space available, the current job is assigned to the open server; if not, the server is closed and a new one is opened. The *WorstFit* algorithm, in contrast, opens a new server if no existing one can accommodate the job. If an open server can fit the job, *WorstFit* places it in the server with the most free space that can accommodate it. If multiple servers have equal space, *WorstFit* selects the server with the lowest index. *BestFit* works oppositely to *WorstFit*, placing the job in the server with the least available space that can still accommodate it. If no such server exists, *BestFit* opens a new one. The *FirstFit* algorithm maintains the order of server openings, placing the job in the first server that has enough space to fit it. If no such server exists, *FirstFit* opens a new one. An algorithm is categorized as part of the *AnyFit* family if it only opens a new server when no existing server can accommodate the job. Unlike *NextFit*, *AnyFit* algorithms ensure that servers remain open once they are created. *WorstFit*, *BestFit*, and *FirstFit* are all members of the *AnyFit* family.

The following example gives a sample input and solution for the d-dimensional *RSiC* problem.

Example 1.2.1. In this example we have $d = 2$. The input sequence σ consists of three jobs, $\sigma = \{\sigma_1 = (0, 6, s_1 = [0.5, 0.2]), \sigma_2 = (4, 8, s_2 = [0.2, 0.2]), \sigma_3 = (4, 8, s_3 = [0.6, 0.1])\}$. Figure **??** shows one possible assignment of these jobs to servers. *FirstFit* opens the first server for σ_1, and σ_2. However, upon the arrival of σ_3, a new server is opened since the first server lacks the capacity in the first dimension to accommodate it.

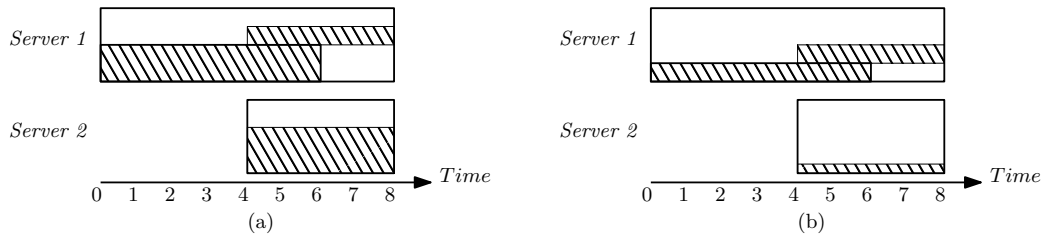

Figure 1.2: Online assignment of jobs into servers described in Example 1.2.1. *FirstFit* on (a) the first dimension, (b) the second dimension.

The cost of each server is determined by its opening and closing times. The first server's cost

is 8 (opening at 0 and finishing at 8), while the second server has a cost of 4. The total cost of the algorithm is the sum of the costs of all servers, resulting in a total cost of 12.

In this example, however, OPT assigns the first job to the first server and opens a new server to accommodate both the second and third jobs together, resulting in a total cost of 10. Figure 1.3 illustrates the assignment of OPT for this example.

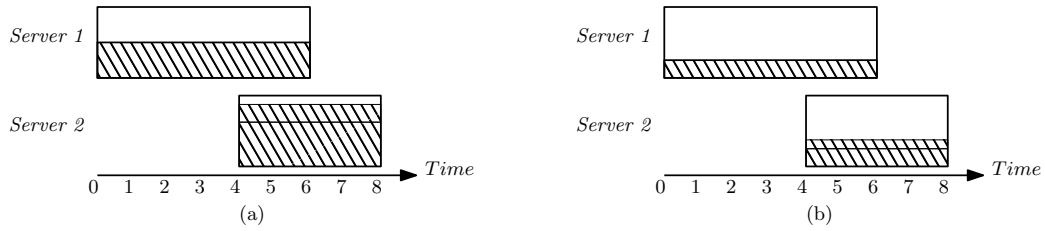

Figure 1.3: OPT assignment of jobs into servers described in Example 1.2.1. OPT on (a) the first dimension, (b) the second dimension.

1.3 Competitive Analysis

Online algorithms were initially analyzed under stochastic models, where the input sequence follows a random distribution. There are several difficulties with this approach: (1) it is not easy to estimate the correct distribution from real-life data (as real-life data is often rather noisy), (2) it lacks worst-case guarantees, (3) it is technically quite challenging to obtain tight bounds in stochastic settings. Competitive analysis emerged as an alternative framework that provides worst-case guarantees for online algorithms, does not require knowledge of distributions, and is often easier to handle than distributional analysis. Competitive analysis became the standard method following the work of Sleator and Tarjan [65]. This approach compares the performance of an online algorithm to that of an optimal offline algorithm in the worst case. In the following, we formally define competitive analysis and its key measures.

For an online algorithm ALG and an input sequence σ, let $ALG(\sigma)$ denote the value of the objective function achieved by ALG on σ. Similarly, let $OPT(\sigma)$ represent the cost of an optimal offline algorithm OPT for the same input σ. For a minimization problem[1], we say that ALG is

[1]For a maximization problem, we say ALG is ρ-competitive if: $ALG(\sigma) \geq \rho \cdot OPT(\sigma) - c$.

asymptotically ρ-competitive if there exists a constant $c > 0$ such that for all input sequences σ:

$$ALG(\sigma) \leq \rho \cdot OPT(\sigma) + c. \tag{1}$$

The infimum over all such ρ is denoted by $\rho(ALG)$ and is called the *competitive ratio* of ALG. If $c = 0$ then the algorithm is called *strictly* ρ-competitive.

Ironically, some of the strengths of worst-case analysis can also be its drawbacks. For example, while worst-case guarantees may be easier to establish than average-case guarantees and provide stronger benchmarks, they can be overly pessimistic and not reflect real-life performance. To address this concern, researchers have explored alternative frameworks to complement competitive analysis. For a comprehensive discussion, we refer readers to the survey by Hiller and Vredeveld [39]. In addition, as data becomes more abundant, it now becomes possible to get useful estimates of real-life distributions. Thus, distributional analysis is making a comeback. See, for example, the survey by Mehta [56] for the developments of distributional analysis in the context of bipartite matching. In this thesis, we analyze algorithm performance using the standard framework of competitive analysis (with an exception of one result on advice complexity).

1.3.1 Lower Bounds on OPT

Note that since *RSiC* is NP-hard, it cannot be solved optimally in polynomial time unless $P = NP$. Therefore, to establish bounds on the competitive ratio of ALG, we derive lower bounds on OPT. The remainder of this section presents these lower bounds on OPT.

The two key parameters of an instance σ, usually called the *span* denoted by $span(\sigma)$ and the *utilization* denoted by $util(\sigma)$ [52, 57], are defined as follows:

$$util(\sigma) = \sum_{i=1}^{n} (f_i - a_i) \cdot \|s_i\|_\infty \text{ and } span(\sigma) = \left| \bigcup_{i=1}^{n} [a_i, f_i] \right|.$$

Thus, the utilization can be thought of as the total volume (size times duration) of all jobs, and span is the total duration of all jobs ignoring overlaps. Note that in the case of $d = 1$, the utilization

of input sequence σ is defined as [52]:

$$util(\sigma) = \sum_{i=1}^{n} s_i(f_i - a_i)$$

Without loss of generality, we assume that $\bigcup_{i\in[n]}[a_i, f_i) = [0, T)$, that is, span arises from a single uninterrupted interval, and the first job arrives at time 0. It is easy to see that the span and utilization are lower bounds on the cost of OPT.

Proposition 1.3.1 ([57])**.** $\mathrm{OPT}(\sigma) \geq \max\{\mathrm{span}(\sigma), \mathrm{util}(\sigma)/d\}$.

We denote the sum of sizes of jobs active at time t by $s(\sigma, t)$, i.e., $s(\sigma, t) = \sum_{i:a_i\leq t<f_i} s_i$. For $t \in [0, \infty)$, we use $s(t)$ to denote the sum of all sizes of jobs that have start time t, i.e., $s(t) = \sum_{i\in[n]:a_i=t} s_i$. For a server B we use $s(B, t)$ to denote the sum of all sizes of jobs that are scheduled on B and active at time t. The value $s(B, t)$ is also called the *load* of server B at time t. As previously mentioned, the cost associated with each server corresponds to the total duration it remains open, and the overall cost of the algorithm is the sum of the costs of all servers.

For an algorithm ALG and $t \in [0, T)$ we use $\mathrm{ALG}(\sigma, t)$ to denote the number of servers opened by ALG that are active at time t. We use $\mathrm{ALG}(\sigma)$ to denote the total cost of ALG on input σ, i.e., the sum of durations of servers opened by ALG. Similar notation is used for the offline optimal solution OPT, where $\mathrm{OPT}(\sigma, t)$ refers to the number of active servers used by OPT at time t, and $\mathrm{OPT}(\sigma)$ represents the total cost of the optimal solution on input σ. Observe that:

Proposition 1.3.2. $\mathrm{OPT}(\sigma) = \int_0^T \mathrm{OPT}(\sigma, t)dt$, *and* $\mathrm{ALG}(\sigma) = \int_0^T \mathrm{ALG}(\sigma, t)dt$.

Lemma 1.3.3. $\int_0^T \lceil \|s(\sigma, t)\|_\infty \rceil dt \leq \mathrm{OPT}(\sigma)$.

Proof. As the capacity of each server is 1 for every dimension $j \in [d]$; any algorithm needs at least $\lceil \|s(\sigma, t)\|_\infty \rceil$ servers to pack the total load at any time t. Therefore, $\mathrm{OPT}(\sigma, t) \geq \lceil \|s(\sigma, t)\|_\infty \rceil$. Using Proposition 1.3.2, we can conclude:

$$\int_0^T \lceil \|s(\sigma, t)\|_\infty \rceil dt \leq \int_0^T \mathrm{OPT}(\sigma, t)dt = \mathrm{OPT}(\sigma). \qquad \square$$

For the 1-dimensional setting, the following proposition gives a lower bound on the optimal cost $\mathrm{OPT}(\sigma)$.

Proposition 1.3.4 ([45]). $\mathrm{OPT}(\sigma) \geq \lceil \sum_{i \in [n]} s(\sigma, t) \rceil$.

1.4 Thesis Contributions and Outline

In this thesis, we study the *RSiC* problem, which is a generalization of the bin packing problem. Bin packing and its variants have been widely studied by researchers. The literature on classical bin packing is extensive, spanning over 50 years of active research. Since bin packing is an NP-hard problem, many different variants have been explored. The *RSiC* problem introduces an additional component, making it even more challenging than the classical bin packing problem. Given its complexity, we investigate *RSiC* under various restrictions like equal duration of jobs, limited number of arrival times, etc. Table 1.1 summarizes the problem settings studied in this study.

Chapter	Job Duration	Arrival Pattern	# of Arrivals	# of Dimensions	# of Servers	Other
Chapter 3	*Equal Duration*	*Arbitrary*	*Arbitrary*	*1*	*Unlimited*	–
Chapter 3	*Equal Duration*	*Arbitrary*	*2*	*1*	*Unlimited*	–
Chapter 3	*Equal Duration*	*Midterm*	*2*	*1*	*Unlimited*	–
Chapter 4	*Equal Duration*	*Midterm*	*Arbitrary*	*1*	*Unlimited*	*Long Running*
Chapter 4	*Equal Duration*	*Midterm*	*Arbitrary*	*1*	*Unlimited*	*Dual-Core*
Chapter 5	*Equal Duration*	*Arbitrary*	*Arbitrary*	*1*	*Limited*	–
Chapter 6	*Arbitrary*	*Arbitrary*	*Arbitrary*	*1*	*Unlimited*	–
Chapter 7	*Arbitrary*	*Arbitrary*	*Arbitrary*	*d*	*Unlimited*	–

Table 1.1: An overview of the various *RSiC* settings studied in each chapter of this thesis. By midterm arrival pattern, we mean that the arrival times of jobs are multiples of half of the job durations, under the assumption that all jobs have equal durations. The rest of the settings are self explanatory.

It is easy to see that *RSiC* in its generality does not admit algorithms with a constant competitive ratio. Thus, the research so far has focused on the inputs parameterized by μ.

In Chapter 3, we consider the problem under the special case of $\mu = 1$, which we refer to as the equal duration setting. Additionally, we explore a specific instance of the *RSiC* problem where jobs have only two arrival times. Although this might seem like an artificial restriction, it reflects certain real-life scenarios. For example, in the Map-Reduce model [23], the input is divided into approximately equal blocks processed in two phases: Map and Reduce. This corresponds to input items arriving at two times in the *RSiC* model. During the map phase, the user's map function creates a set of key/value pairs, generating intermediate key/value pairs. In the reduce phase, these intermediate pairs are passed to a reduce function, merging values related to the same key. Map-Reduce is extensively used in cloud computing for automatic parallel processing and large-scale

data distribution. For this scenario, we establish a tight bound of 2 on *NextFit*; the previous best bound for equal duration jobs was $2\mu + 1 = 3$.

We also derive a lower bound of $\frac{34}{27}(1 + t)$ for $t \in (0, 1)$ on the competitive ratio for *FirstFit* when jobs have duration 1 and arrival times 0 and t for $t \in (0, 1)$. This surpasses the bin packing lower bound for $t > 0.35$. Using the weight function technique, we show an upper bound of $\frac{168}{131}(1 + t)$ on the asymptotic competitive ratio of *FirstFit* where jobs have duration 1 and arrival times 0 and t for $t \in [0.559, 1)$. To our knowledge, this is the first application of the weight function technique to the analysis of *RSiC*. When $t \to 1$, our results indicate that the competitive ratio of *FirstFit* ranges between 2.519 and 2.565. Furthermore, we prove a strict competitive ratio of 2 for *FirstFit* when each job has duration 1 and arrival times are either 0 or $1/2$.

In Chapter 4, we demonstrate a tight asymptotic bound of $2/3$ on the load of long-running uniform *FirstFit* servers when jobs have duration 2 and integer arrival times $0, 1, 2, \ldots$, implying a competitive ratio of approximately $3/2$ for *FirstFit* in this long-running uniform case. This suggests that challenging inputs for *FirstFit* are those where servers are short-lived. Then, we consider a simpler version of the problem when all jobs have the same size $1/2$. We prove that every online algorithm has competitive ratio at least $5/4$, and any *AnyFit* algorithm can achieve competitive ratio $5/4$. We also show that even in this rather restricted setting it is impossible to achieve competitive ratio better than $9/8$ with sub-linear advice[2].

Another setting explored in this thesis in Chapter 5 involves a limited number of servers. We conduct a parameterized analysis, examining input families where *FirstFit* requires at most k servers at any time, and provide tight bounds on *FirstFit*'s competitive ratio for such inputs. This parameterized version naturally models scenarios where the number of available servers is limited but sufficient to meet demand. It also offers insights into the original *RSiC* problem with no server number restriction at the number of servers. For $k = 2$, we establish a tight bound of 2 on *FirstFit*'s competitive ratio. For $k = 3, 4$, we establish a tight bound of 3. Our lower bound of 3 applies to the infinite server case and notably uses the time dimension, unlike previous constructions that generalized bad bin packing cases. To prove the upper bound for $k = 4$, we introduce a novel technique of partitioning the span into intervals based on the number of servers used by *FirstFit* and OPT,

[2]The definition of advice complexity can be found in Section 4.2.3.

showing that the total duration of intervals where *FirstFit* uses 4 servers while OPT uses only one server cannot be that large. This technique could be useful for analyzing higher values of k or the unrestricted server case. We also conduct an experimental study of *FirstFit* in the k-server setting to understand its average-case behavior and the interplay between k and other system parameters, such as job arrival rate and size range.

In Chapter 6, we construct specific instances to compare the theoretical performance of different algorithms. Our results demonstrate that, within the class of algorithms we have discussed, no single algorithm in the studied class consistently outperforms others. This highlights the inherent complexity of the *RSiC* problem, indicating that different algorithms may perform better or worse depending on the specific characteristics of the input instances.

In Chapter 7, we study both the clairvoyant and non-clairvoyant variants of the d-dimensional *RSiC* problem, where job sizes are represented as d-dimensional vectors. In this setup, each job's size in any dimension is normalized to fall within the range of 0 to 1, and the server capacity in each dimension is set to 1. For this variant, we propose a new clairvoyant algorithm called *Greedy*, which assigns a new job to the server with enough remaining capacity that would incur the *least additional rental cost*. Surprisingly, this natural algorithm has not been studied previously. We prove that *Greedy* has a competitive ratio of $3d\mu + 1$. Our analysis extends to a class of algorithms we call *monotone AnyFit*, which includes *FirstFit*, *LastFit*, and *MoveToFront*. This proof employs a new technique compared to those in existing literature [1, 45, 50, 59]. Notably, while the proof of a $6\mu + 8$ upper bound on the competitive ratio of *MoveToFront* in [45] also applies to *Greedy* for 1-dimensional *RSiC*, our technique achieves a tighter upper bound of $3\mu + 1$ for the 1-dimensional case.

Additionally, we demonstrate a direct sum property of the *RSiC* problem by showing how to transform a 1-dimensional algorithm ALG into a d-dimensional algorithm $\text{ALG}^{\oplus d}$, with the competitive ratio scaling *exactly* by a factor of d. Consequently, we derive another clairvoyant algorithm for d-dimensional *RSiC* with a competitive ratio of $\Theta(d\sqrt{\log \mu})$. For non-clairvoyant *RSiC* with $d = 1$, we show that no randomized algorithm can achieve a competitive ratio better than $\frac{1-e^{-1}}{2}\mu$.

We also conduct experiments in the average-case scenario, evaluating nearly all existing algorithms for *RSiC* using randomly generated synthetic data. Our findings show that the *Greedy* algorithm outperforms other algorithms, whether clairvoyant or non-clairvoyant, in the vast majority of cases. Specifically, it surpasses *MoveToFront*, the previous best algorithm, in these experiments.

In Chapter 8 we focus on a comprehensive performance evaluation of clairvoyant and non-clairvoyant algorithms for the *RSiC* problem using real-world data. Leveraging the Azure dataset [36], we rigorously test these algorithms in realistic scenarios. This approach not only provides deep insights into their practical effectiveness but also establishes a new benchmark to guide future research. Additionally, we introduce new algorithms, derived from combinations of existing algorithms, which outperform all the existing algorithms in our experimental evaluations.

Finally in Chapter 9, we conclude and discuss the possible future directions for the *RSiC* problem.

Part of the work presented in this thesis has already been appeared in the following papers:

- Mahtab Masoori, Lata Narayanan, Denis Pankratov. Renting servers in the Cloud: The case of equal duration jobs. *Discrete Applied Mathematics*, 362:82–99, 2025.
- Mahtab Masoori, Lata Narayanan, Denis Pankratov. Renting Servers in the Cloud: Parameterized Analysis of FirstFit. *In Proceedings of ICDCN 2024* — Best Paper Award.
- Yaqiao Li, Mahtab Masoori, Lata Narayanan, Denis Pankratov. Renting Servers for Multi-Parameter Jobs in the Cloud. *In Proceedings of ICDCN 2025* — Best Paper Award.
- Mahtab Masoori, Lata Narayanan, Denis Pankratov. Renting Servers in the Cloud: Empirical Study on Real-World Data. Submitted to *ACDA25*.

Chapter 2

Literature Review

In this chapter, we review the previous research on the bin packing problem studied in this thesis. We investigate this problem under 1-*dimensional* ($d = 1$) and d-*dimensional* ($d > 1$) settings. The bin packing problem has been a crucial key in the development of classical methods for analyzing the performance of approximation algorithms [42]. It has also significantly contributed to the concept of the competitive ratio, which is essential in the analysis of online algorithms [67, 44]. Yao's research on the bin packing problem established the first general lower bound for the competitiveness of online algorithms, marking a pivotal advancement in the field [74]. Furthermore, this problem has offered important perspectives on the average-case performance of approximation algorithms, as illustrated by Lueker [55].

In the rest of this chapter, we provide an overview of the most significant related work on *RSiC*, a variant of bin packing and dynamic bin packing, along with its connections to various extensions. We begin by reviewing the these problems in the one-dimensional setting and then broaden our discussion to include the d-dimensional case for $d > 1$.

2.1 The 1-Dimensional Setting ($d = 1$)

In the classical one-dimensional bin packing problem, we are given a list of real numbers in $(0, 1]$. The goal is to pack these numbers into bins, each with a capacity of 1, such that the total sum of the items in any bin does not exceed its capacity, while minimizing the total number of bins used.

By a reduction from the partition problem, it is known that bin packing is an *NP-hard* problem [35].

2.1.1 Classical Bin Packing

Online Bin Packing

We start by reviewing the online version of the bin packing problem. In this version, items in the input sequence σ are revealed sequentially. Each item σ_i has a size denoted by s_i. An algorithm for the online bin packing problem must place each item into a bin without any knowledge of the future items. Online processing is challenging due to the unpredictable sizes of items that may appear in the future. In fact, the performance of an online bin packing algorithm is significantly influenced by the order in which items are presented in the list. Many different algorithms have been proposed for the online bin packing problem. In this chapter, we aim to address the most important ones.

As discussed in Chapter 1, one of the simplest algorithms for the bin packing problem is *NextFit*. Johnson et al. [44] proved that the competitive ratio of *NextFit* is 2. Similarly, the competitive ratio of *WorstFit* is also 2, offering no improvement in terms of competitive ratio [42]. Ullman in [67] established the asymptotic bound on the competitive ratio of *FirstFit* of the form $1.7 \cdot OPT + 3$. The additive term 3 was improved multiple times [33, 34, 64, 73, 11] and finally Dosa and Sgall [26, 27] demonstrated the strict ratio 1.7. In general, algorithms in the *AnyFit* family have a tight competitive ratio of 2. This indicates that their competitive ratio does not exceed 2, and there exists at least one algorithm within the family, such as *WorstFit*, that achieves an exact competitive ratio of 2 .However, rather large sub-families of *AnyFit* achieve a tighter competitive ratio of 1.7 [41].

The *Harmonic* family of algorithms represents a class of bin packing algorithms that attempt to efficiently group items of similar sizes within the same bins. These algorithms maintain a maximum of k open bins at the same time, where k is a parameter. Lee and Lee [48] present the *Harmonic Fit* (HF_k) algorithm which partitions the interval $(0, 1]$ into k sub-intervals: $(1/(j+1), 1/j]$ (for $1 \leq j \leq k-1$), and $(0, 1/k]$. Items are then assigned to these intervals based on their sizes using the *NextFit* algorithm. When $k \to \infty$ the algorithm is denoted as HF_∞. In this case, the algorithm considers infinitely many intervals to classify item sizes. It is shown that the HF_∞ algorithm has competitive ratio 1.692 [48].

A significant restriction of the HF_∞ algorithm is its inefficiency in handling items slightly larger than $1/2$ as these items are packed alone in a bin, leaving the remaining space unused. To address this issue, Lee and Lee [48] introduced the *Refined Harmonic Fit* algorithm, which optimizes bin usage by sharing bins between items from the first two intervals. This refinement reduced the total number of bins required and improved the competitive ratio to 1.63596. Over time, several efforts have been made to further enhance the competitive ratio of the *Harmonic Fit* algorithm [58]. Richey [62] introduced the Harmonic+1 algorithm, claiming a competitive ratio of 1.58. However, in 2002, Sheiden demonstrated that the competitive ratio of *Harmonic+1* is at least 1.59217, disproving the earlier claim of 1.58. This discovery led to the development of the *Harmonic++* algorithm, which achieved a competitive ratio of at most 1.58889. Finally, Balogh et al. [5] proposed a new algorithm in 2017 with a competitive ratio of 1.57829, which remains the best-known algorithm in this family to date.

Regarding lower bounds, Yao [74] was the first to establish a lower bound of $3/2$ for online bin packing. His construction relied on three lists of items with equal sizes: $1/2 + \epsilon$, $1/2 + \epsilon$, and $1/6 - 2\epsilon$. Later, Brown [13] and Liang [53] independently improved this bound to 1.536345. Van [69] later conducted an exhaustive analysis of lower bound constructions using linear programming techniques, further raising the bound to 1.54015. Additionally, Faigle et al. [29] demonstrated that when only two item sizes are considered, no online algorithm can achieve a competitive ratio better than $4/3$.

The bin packing problem has also been extensively studied under the random order model, where the input sequence of items is selected by an adversary, but the order of arrival of these items is determined by a uniformly random permutation from all possible permutations of the items. Although this thesis does not focus on the random order model, but readers interested in further exploration are referred to the works by [17, 21, 38].

Offline Bin Packing

As previously mentioned, bin packing is *NP-hard* due to a reduction from the partition problem. In the partition problem, we are given n integers b_i for $i \in \{1, 2, \ldots, n\}$ with a total sum $B = \sum_{i=1}^{n} b_i$. The goal is to divide these numbers into two sets, S_1 and S_2, such that the sum of the

numbers in S_1 equals the sum of the numbers in S_2. By considering each item in the bin packing input sequence to have a size of $2b_i/B$, we can reduce the partition problem to the bin packing problem by checking if we can pack the items into two bins. It has been shown that the bin packing problem is *NP-hard* even for achieving an absolute approximation ratio better than $3/2$ [35].

In the offline version, the entire input sequence is available to the algorithm in advance for pre-processing. Johnson [41, 44] introduced two algorithms, *First Fit Decreasing (FFD)* and *Best Fit Decreasing (BFD)*. These algorithms first sort the items in the input sequence in non-increasing order of their sizes and then apply *FirstFit* and *BestFit*, respectively. It has been shown that these algorithms have an approximation ratio of $\frac{11}{9}OPT + 4$ [41]. This result has been improved several times, eventually reaching $\frac{11}{9}OPT + \frac{2}{3}$ [25] which is tight. A modified version of this algorithm, called *Modified First Fit Decreasing*, was introduced by Johnson and Garey [43] and has an improved approximation ratio of $\frac{71}{60}$. More generally, if an *Any Fit* algorithm is applied after sorting, the resulting algorithm has an approximation ratio of at least $\frac{11}{9}$ [41].

A problem admits a *Polynomial Time Approximation Scheme (PTAS)* if, for every constant $\epsilon > 0$, there is a polynomial-time algorithm (with respect to the size of the input n) that produces a solution with an approximation ratio of $1 + \epsilon$. The running time of this algorithm is $O(n^{f(1/\epsilon)})$, where f is a function that depends only on ϵ and may grow with $1/\epsilon$. If the running time of a $PTAS$ is polynomial in both n and $1/\epsilon$, the algorithm is called a *Fully Polynomial Time Approximation Scheme (FPTAS)*. A problem admits an *Asymptotic Polynomial Time Approximation Scheme (APTAS)* if, for every $\epsilon > 0$, there is a polynomial-time algorithm with an *asymptotic approximation ratio* of $1 + \epsilon$. This means the approximation ratio is guaranteed only when the size of the input or some problem parameter (e.g., OPT) becomes sufficiently large. If the running time of an $APTAS$ is polynomial in both n and $1/\epsilon$, it is called an *Asymptotic Fully Polynomial Time Approximation Scheme (AFPTAS)*.

The first *APTAS* for the bin packing problem was proposed by Fernández de la Vega and Lueker in 1981 [30]. This was improved by Karmarkar and Karp in 1982, who introduced an algorithm with a guarantee of $OPT + O(\log^2 OPT)$ [46]. Subsequently, Rothvoss in 2013 [63] improved the additive term to $OPT + O(\log OPT \cdot \log\log OPT)$, and Hoberg and Rothvoss in 2017 [40] further reduced it to $OPT + O(\log OPT)$. The problem of achieving $OPT + 1$ remains an open problem.

For more details, refer to the book by Williamson and Shmoys [70].

2.1.2 Dynamic Bin Packing

The dynamic bin packing problem, generalizing the vanilla bin packing, was initially introduced by Coffman et al. [20]. In this variation, items of varying sizes arrive and depart sequentially, with their sizes and arrival times known upon arrival, but their departure times remain unknown until they actually depart. The primary objective is to minimize the maximum number of bins used at any time during the execution. Coffman et al. considered a version where repacking of items is not allowed, establishing an initial lower bound of 2.38 for any online algorithm, which has since been improved to 2.66 [72]. Coffman et al. [20] proposed two types of optimal algorithms for evaluating online algorithms: OPT_R which allows repacking, and OPT_{NR} which repacking is not allowed. Recall that the algorithm ALG does not allow for repacking items. For OPT_R, the competitive ratio of any online algorithm ALG is at least 2.5. Within this setting, *FirstFit* achieves a competitive ratio between 2.75 and 2.89 while the *Modified FirstFit (MFF)* algorithm has a competitive ratio of 2.788. For OPT_{NR}, the lower bound is 2.38.

Static bin packing can be seen as a special case of dynamic bin packing, where all items arrive at the same time and remain in the system indefinitely. Dynamic bin packing allows for items to depart, potentially reducing the number of bins needed if managed optimally. For example, consider a sequence of items arriving and departing at different times. An optimal dynamic bin packing algorithm might use fewer bins than an algorithm for the static problem by taking advantage of the varying departure times. The main difference between these two cases would be clear with the following example:

Example 2.1.1. Suppose we are given an input sequence σ consisting of seven items. Let $\sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7\}$. For a small $\varepsilon > 0$, suppose $s_1 = s_2 = 1/2 - \varepsilon$, $s_3 = s_4 = 1/2 + \varepsilon$, $s_5 = 1/3$, $s_6 = 2/3$, and $s_7 = 1$. In static bin packing, all items arrive in the same time and stay in the system forever. In this case, any optimal algorithm uses 4 bins for packing the sequence σ. Now let us consider the dynamic version of bin packing. In this case suppose σ_7 arrives after σ_1 and σ_3 depart. To be more clear, suppose the input sequence is $\sigma = \{\sigma_1 = (1, 3, 1/2-\varepsilon), \sigma_2 = (1, 6, 1/2-$

$\varepsilon), \sigma_3 = (1, 3, 1/2 + \varepsilon), \sigma_4 = (1, 6, 1/2 + \varepsilon), \sigma_5 = (1, 6, 1/3), \sigma_6 = (2, 6, 2/3,), \sigma_7 = (3, 6, 1)\}$. As you can see, an optimal algorithm uses 3 bins for packing these items. See Figure 2.1.

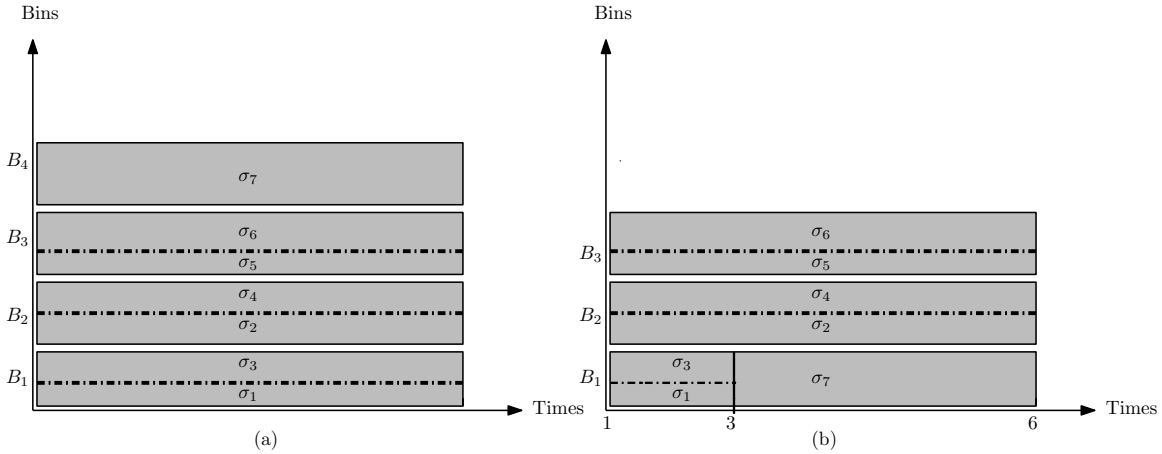

Figure 2.1: Illustration of the example instance: (a) The optimal solution for static bin packing. (b) The optimal solution for dynamic bin packing.

In the discrete version of the problem, each item has a size of $1/k$, where k is a positive integer. Chan et al. [16] proved a tight bound of 3 on the competitive ratio of the *BestFit* and *WorstFit*. They also showed that the *FirstFit* algorithm achieves a competitive ratio of 2.4942, which was later improved to 2.4842 by Han et al. [37]. Nevertheless, no online algorithm can achieve a competitive ratio better than 2.428 [16].

In the discrete problem, the competitive ratio of online algorithms is at most 2.48 [16]. However, the *Harmonic* family of algorithms, which perform well in classical bin packing, are less effective in dynamic settings. As these algorithms group items into categories based on their size and assign separate bins to each category. In dynamic settings, an adversary can exploit this approach by repeatedly adding and removing items of a single category, leading to substantial wasted space in the bins.

In summary, dynamic bin packing introduces complexity due to the sequential arrival and departure of items and the unknown duration of their presence. The development and analysis of algorithms for this problem reveal various competitive ratios and lower bounds, highlighting the challenge of optimizing bin usage in dynamic environments.

2.1.3 Renting Servers in the Cloud

The 1-dimensional *RSiC* problem were introduced by Li et al. [52] for the first time. Except [57], and this work, all existing literature on *RSiC* studies only dimension 1. For non-clairvoyant 1-dimensional *RSiC*, Li et al. [52] proved that no deterministic online algorithm can achieve a competitive ratio of less than μ. They also showed that no *AnyFit* algorithm can achieve a competitive ratio of less than $\mu + 1$. Finally they showed that the competitive ratio of any online algorithm for *RSiC* when the size of jobs are greater or equal to $1/k$, where k is a positive number, is at most k. Later, Kamali and Lopez-Oritz [45] showed that μ is in fact a lower bound for any deterministic algorithm. As we mentioned before, *AnyFit* is a family of algorithms for the bin packing problem. For *RSiC*, researchers have mostly studied *FirstFit* and *BestFit*. For the *FirstFit* algorithm when the size of jobs are at most $1/k$ then the competitive ratio is at most $\frac{k\mu}{k-1} + \frac{6k}{k-1} + 1$. In the case that k is equal to 1 (i.e. that there is no restriction on item sizes) then the competitive ratio of *FirstFit* is at most $2\mu + 13$ [51]. In the same work, Li et al. [51] introduced the *Modified FirstFit* algorithm, which employs a parameter K to divide jobs into two categories: small jobs, with sizes less than $1/K$, and large jobs, with sizes at least $1/K$. The *FirstFit* strategy is then applied independently to each category. This algorithm achieves a competitive ratio of at most $8/7\mu + 55/7$ when μ is not known in advance and a competitive ratio of at most $\mu + 8$ when the value of μ is known. In [61] the authors improved these results. In particular, they proved that *FirstFit* packing achieves a competitive ratio of $\mu + 3$, indicating that it is near-optimal for *RSiC* in the non-clairvoyant online setting. Li et al. [51], proved a surprising result about the competitive ratio of *BestFit* for renting servers in the cloud problem; they proved the *BestFit* algorithm does not have a bounded competitive ratio.

Kamali and Lopez-Oritz [45] considered the *NextFit* algorithm for *RSiC*. They showed that if every job in the input sequence σ has a size of at most $1/k$, where $k \geq 1$, then:

$$\rho(\textit{NextFit}) \leq \begin{cases} \dfrac{\mu}{1 - 1/k} & \text{if } k \geq 2, \\ 2\mu + 1 & \text{if } k < 2. \end{cases}$$

In the same paper, they proposed a modified version of *NextFit* for improved performance. This modified algorithm, known as *Modified NextFit* with parameter K, processes smaller jobs (less

than $1/K$) separately from larger jobs (at least $1/K$). The *NextFit* strategy is applied independently to each category. The competitive ratio for *Modified NextFit* with parameter K is at most: $K \times \max\left\{1, \frac{\mu}{k-1}\right\} + 1$. If the algorithm knows the value of μ, then K can be set to $\mu + 1$, making *Modified NextFit* at most $\mu + 2$-competitive. Kamali and Lopez-Oritz in the same paper, also introduced a new *AnyFit* algorithm called the *MoveToFront* algorithm, which places the next job in the *most recently used* bin. They proved that the competitive ratio of *MoveToFront* is at most $6\mu + 7$.

Although these algorithms have promising competitive ratios, practical applications require algorithms that also perform well on average. Kamali and Lopez-Oritz [45] and Ren et al. [60] also presented experiments on random inputs to compare different algorithms and show various non-trivial phenomena. In particular, Kamali and Lopez-Oritz [45] performed some experiments to compare the average-case performance of different algorithms for *RSiC* on randomly-generated sequences. In those experiments, they assumed that each bin had a integer capacity E. Moreover, the size and length of items had a uniform distribution in the ranges $[1, E]$ and $[1, \mu]$, respectively. Their experiments affirm that *MoveToFront* in general, performs the best among all known non-clairvoyant algorithms at dimension 1. The closest counterpart of *MoveToFront* regarding the average case performance is *BestFit* which does not have a bounded competitive ratio.

The clairvoyant setting has been studied in [1, 60]. In particular, Ren et al. [60] established a lower bound of $\frac{1+\sqrt{5}}{2}$ on the competitive ratio for any online packing algorithm. They also proposed two item classification strategies for online packing: one based on departure time and the other on duration. They applied these classification strategies to the classical *FirstFit* algorithm and analyzed their competitiveness. Azar et al. [1] proposed the combination of these strategies in the so-called *Hybrid Algorithm* which classifies jobs according to their length and their arrivals. Suppose the maximum duration of jobs in the input sequence is μ. Then all the jobs whose lengths are in range $[2^{i-1}, 2^i]$ for integer $1 \leq i \leq \lceil \log \mu \rceil + 1$ and whose arrival times are in time interval $[(c-1)2^i, c2^i)$ for an integer c are put into the same category. They proved a tight bound of $\Theta(\sqrt{\log \mu})$ on its competitive ratio.

In the offline setting, Ren et al. [60] introduced two approximation algorithms for the *RSiC* problem: the 5-approximation *Duration Descending FirstFit* algorithm and the 4-approximation

Dual Coloring algorithm. Additionally, Buchbinder et al. [14] proposed an offline algorithm with an approximation ratio of $2HF_\infty \approx 3.38$, which is the current best-known algorithm. To the best of our knowledge, there has not yet been the study of *RSiC* under advice model.

2.2 The d-Dimensional Setting ($d > 1$)

2.2.1 d-Dimensional Bin Packing

In this section, we will explore previous work on the bin packing problem in a multi-dimensional setting. This problem has been studied in two distinct scenarios: *Vector bin packing (VBP)* and *Geometric bin packing (GBP)*.

The *VBP* problem is a non-geometric generalization of the bin packing problem. Here we are given a sequence σ of n vectors $\{v_1, \cdots, v_n\}$, where each vector $v_i \in [0,1]^d$. The goal is to pack all vectors into the minimum number of unit bins such that the sum of the vectors in each dimension within a bin does not exceed the bin's capacity. In their paper, Fernández de la Vega and Lueker [30] proposed the first *APTAS* for bin packing, which also provides a $(d+\epsilon)$ approximation for *VBP*. Woeginger [71] proved that, unless *P = NP*, there is no *APTAS* even for $d = 2$ in the general case. Later, Chekuri and Khanna [18] demonstrated that *VBP* can be approximated to $O(\ln d)$ in polynomial time for a fixed d, and this result was improved by Bansal et al. [8] to $1 + \ln d$. Yao [74] showed that no algorithm running in $O(n \log n)$ time can achieve better than a d-approximation for VBP. Chekuri et al. [18] showed that for a fixed value of d, the *VBP* problem is hard to approximate within a $d^{1/2-\epsilon}$ factor for any $\epsilon > 0$. For $d = 2$, Kellerer and Kotov [47] presented an algorithm with an absolute approximation ratio of 2. Recall that, achieving an absolute approximation ratio better than $3/2$ for the bin packing problem is *NP-hard*. Bansal et al. [9] improved these results by providing a polynomial-time algorithm with an asymptotic approximation guarantee of $(1 + \ln(3/2) + \epsilon)$ for $d = 2$ and $\ln d + 0.807 + o_d(1) + \epsilon$ for $d > 2$. They also demonstrated that for any small constant $\epsilon > 0$, there is a polynomial-time algorithm with an almost tight absolute approximation ratio of $3/2 + \epsilon$ for $d = 2$.

For the online version of *VBP*, Garey et al. [34] extended the analysis of *FirstFit* to achieve a competitive ratio of $d + 7/10$. Galambos et al. [32] demonstrated that as $d \to \infty$, the competitive

ratio of online algorithms has a lower bound that approaches 2. Epstein [28] conjectured that the lower bound on the competitive ratio is super-constant but sublinear. Subsequently, Azar et al. [2] provided an $\Omega(d^{1-\epsilon})$ lower bound for the *VBP* problem. Special cases of the online version of VBP have been also studied, as detailed in [3, 4].

In the 2-dimensional *GBP*, the input consists of a sequence σ of n rectangular items, each with a width and height in the range $(0, 1]$. The objective is to pack these rectangles into the fewest possible unit square bins, ensuring that the sides of the items are parallel to the sides of the bins. This problem is generally studied in two variants: one where items cannot be rotated and another where items can be rotated by $90°$. This problem has been explored in both offline and online settings; for more details, refer to [54, 7, 22]. Other related problems to *VBP* and *GBP* include *Strip Packing* and *Geometric Knapsack*, which extend the concepts of bin packing into geometric forms. Additionally, *Vector Knapsack*, *Vector Scheduling*, and *Vector Bin Covering* generalize bin packing to vector spaces. For a detailed exploration of these topics that are beyond the scope of this thesis, readers are encouraged to see the survey by Christensen et al. [19].

2.2.2 d-Dimensional Dynamic Bin Packing

To the best of our knowledge, the d-dimensional dynamic *VBP* problem has not been extensively studied. In [68], the authors introduce a polynomial-time data reduction algorithm that can achieve $(1 + \epsilon)$-approximate solutions for any arbitrary $\epsilon > 0$.

2.2.3 d-Dimensional Renting Servers in the Cloud

Murhekar et al. [57] initiated the study of non-clairvoyant d-dimensional *RSiC*. They proved that *MoveToFront* has an upper bound $(2\mu + 1)d + 1$, which in particular improves the previous upper bound of *MoveToFront* in [45] for $d = 1$ to $2\mu + 2$. They also generalized various upper and lower bounds of algorithms such as *FirstFit* and *NextFit* from dimension 1 to dimension d. In particular, they showed a lower bound of $d(\mu + 1)$ for *AnyFit* algorithms. Similar to [45], Murhekar et al. also presented some experiments on the same setting as [45] and confirmed their results even in d-dimensional setting. The currently best known upper and lower bounds for both non-clairvoyant and clairvoyant *RSiC* problem under both 1-dimensional and d-dimensional settings are summarized in

Table 2.1.

	Setting	Algorithm	Lower bound $d = 1$	Upper bound $d = 1$	Lower Bound $d > 1$	Upper bound $d > 1$
Online	**Non-clairvoyant**	***AnyFit***	$\mu + 1$ [50]	∞	$(\mu + 1)d$ [57]	∞
		FirstFit	$\mu + 1$ [50, 61]	$\mu + 3$ [61]	$(\mu + 1)d$ [57]	$(\mu + 2)d + 1$ [57]
		NextFit	2μ [66]	$\mathbf{2\mu}$	$2\mu d$ [57]	$2\mu d + 1$ [57]
		BestFit	∞ [50]	∞	∞ [50]	∞
		MoveToFront	2μ [57]	$2\mu + 2$ [57]	$\max\{2\mu, (\mu + 1)d\}$ [57]	$(2\mu + 1)d + 1$ [57]
	Clairvoyant	***Hybrid Algorithm***	$\sqrt{\log \mu}$ [2]	$\sqrt{\log \mu}$ [2]	–	–
		Hybrid Algorithm$^{\oplus d}$	$\Omega(\sqrt{\log \mu})$ [2]	$O(\sqrt{\log \mu})$ [2]	$\boldsymbol{\Omega(d\sqrt{\log \mu})}$	$\boldsymbol{O(d\sqrt{\log \mu})}$
		Greedy	$\mu + 1$ [50]	$\mathbf{3\mu + 1}$	$\boldsymbol{(\mu + 1)d}$	$\mathbf{3d\mu + 1}$

Table 2.1: Summary of the result for the online *RSiC*. Bold text highlights our results. Note that when $d = 1$, the *Hybrid Algorithm*$^{\oplus d}$ algorithm is identical to the *Hybrid Algorithm* algorithm.

Chapter 3

The Case of Equal Duration Jobs

In this chapter, we study the 1-dimensional *RSiC* problem, for the situation in which all jobs have the same duration, that is the case where $\mu = 1$. As we mentioned in Section 2.1.3, the previous results for *RSiC*, which are based on span and utilization, give an upper bound $2\mu + 1$ for *NextFit* and $\mu + 3$ for *FirstFit*. There is a natural reason for the appearance of these additive terms and it has to do with the distinction between the asymptotic and strict competitive ratios. Suppose one proves that for all times t we have $ALG(t) \leq \rho \cdot OPT(t) + \delta$, which would be analogous to an "asymptotic bound" for a fixed time t, so to speak (note that this bound does not have to apply for each time t, and amortization tricks may be used to prove this bound on average, but we assume such a bound for each t for simplicity of the argument). The overall cost of the algorithm is $ALG = \int_0^\infty ALG(t)\ dt \leq \int_0^\infty \rho \cdot OPT(t) + \delta\ dt = \rho \cdot OPT + \delta \cdot span(\sigma) \leq (\rho + \delta)OPT$. Thus, an "asymptotic bound" for a fixed time t with competitive ratio ρ gets converted into the overall bound of $(\rho + \delta)$ for the problem. The "asymptotic bounds" for a fixed time t arise because often one needs to ignore one or two special servers to argue the desired inequality. Thus, one approach towards tightening these bounds is to attempt to prove "strict bounds" for a fixed time t, i.e., $ALG(t) \leq \rho' OPT(t)$. This way would avoid the appearance of the $span(\sigma)$ term. This is akin to the difference between proving the asymptotic competitive ratio of *FirstFit* and strict competitive ratio of *FirstFit* for the classical bin packing. We view this as the main obstacle to improving the known results.

We now describe one possible program to overcome the above-mentioned obstacle. First, generalize the weight function technique from the bin packing problem to the *RSiC* problem to prove asymptotic bounds in restricted cases. Second, optimize these bounds and make them strict in restricted cases. Third, generalize these techniques to the original problem.

In this chapter we make progress towards the above program, for the special case of $\mu = 1$, which we call equal duration. Specifically, we analyze two well-known algorithms, *NextFit* and *FirstFit*, under the assumption that all jobs have the same duration. As summarized in Table 3.1, we consider a scenario where jobs have a uniform duration but have different arrival pattern. We first consider *NextFit* where the number of arrivals is arbitrary. Then, *FirstFit* when we assume jobs have only two arrival times. We also consider *FirstFit* under a more simplified version, all jibs have a fixed duration of x and arrival times that are multiple of $x/2$. We refer to this setting as *midterm arrivals*. Without loss of generality, we assume $x = 2$, and we restrict the arrival times to be 0 and 1 in this case.

Algorithm	**Job Duration**	**Arrival Pattern**	**# of Arrivals**	**# of Dimensions**	**# of Servers**
NextFit	*Equal Duration*	*Arbitrary*	*Arbitrary*	*1*	*Unlimited*
FirstFit	*Equal Duration*	*Arbitrary*	*2*	*1*	*Unlimited*
FirstFit	*Equal Duration*	*Midterm*	*2*	*1*	*Unlimited*

Table 3.1: Overview of the settings for *RSiC* with uniform duration of jobs.

3.1 Notation and Preliminaries

We specify each job σ_i by its size and start time to simplify the notation, i.e., $\sigma_i = (a_i, s_i)$, since the finishing time f_i is a function of the start time, i.e., $f_i = a_i + 1$. We focus on the case of two arrival times 0 and $t \in (0, 1)$, where t is arbitrary, and jobs of duration 1. In some sections, we concentrate on arrival times 0 and 1 and jobs of duration 2, unless stated otherwise; by scaling, this setting is equivalent to arrival times 0 and $1/2$ and jobs of duration 1. Which of these notations is used in a particular section will be clear from the context.

Recall that for $t \in [0, \infty)$, $s(t)$ denotes the sum of all sizes of jobs that have start time t, i.e.,

$$s(t) = \sum_{i \in [n]: a_i = t} s_i.$$

For $t_1 < t_2$ we use $S(t_1, t_2)$ to denote the sum of all sizes of jobs that have start time in the interval $(t_1, t_2]$:

$$S(t_1, t_2) = \sum_{i \in [n]: s_i \in (t_1, t_2]} s_i$$

3.2 Tight Competitive Ratio 2 for *NextFit*

In this section, we assume $\mu = 1$ and jobs have arbitrary arrival times. Recall that the *NextFit* algorithm keeps at most one server open at any time, i.e., a server that can be assigned newly arriving jobs. If a given job does not fit in the open server, the algorithm closes the server (meaning it will not assign any new jobs to it from this point onward), opens a new server, and assigns the job to the newly opened server. Note that *NextFit* never reopens a closed server. Algorithm 1 presents the pseudocode for *NextFit*.

Algorithm 1 *NextFit* algorithm.

```
procedure NextFit(σ = (σ_1, ..., σ_n))
    B ← ⊥                                   ▷ Keeps track of an open server
    T ← −∞                                  ▷ Keeps track of when the open server terminates
    for i = 1 to n do
        new job σ_i = (a_i, f_i, s_i) arrives
        if a_i > T or s(B, a_i) + s_i > 1 then
            ▷ open server terminated prior to arrival of the new job or the new job does not fit into it
            B' ← a new open server
            Close B
            B ← B'
        end if
        Assign σ_i to B
        T ← max(T, f_i)
    end for
end procedure
```

The lower bound of 2 follows immediately from the lower bound on the competitive ratio of *NextFit* for the bin packing problem and the fact that *RSiC* generalizes bin packing. The upper bound of 2 on the competitive ratio of *NextFit* follows from the theorem below, which shows that

the upper bound holds not only on the total cost of the schedule of *NextFit*, but, in fact, it holds at each individual time t.

Theorem 3.2.1. *Suppose that the duration of each job in σ is exactly 1 while arrival times s_i are arbitrary. Then, for every time $t \in [0, \infty)$ we have:*

$$\textit{NextFit}(\sigma, t) \leq 2 \cdot OPT(\sigma, t).$$

Proof. If there are no active jobs at time t then $\textit{NextFit}(t) = 0$ and the claim trivially holds. If there are active jobs at time t and $\textit{NextFit}(t) \leq 2$ then we have $OPT(t) \geq 1$ and the claim also trivially holds.

It remains to show the claim for those t where $\textit{NextFit}(t) \geq 3$. Let $k = \textit{NextFit}(t)$ and let $B_1, \ldots, B_k$ be the servers of *NextFit* that are still active at time t and ordered in the non-decreasing order of their opening times. Observe that since *NextFit* maintains only one open server at a time, servers $B_1, \ldots, B_{k-1}$ must be closed at time t. We also use the following simple fact based on duration of each job being exactly 1: a job is active at time t if and only if its arrival time is in $(t-1, t]$. Therefore we have:

$$S(t-1, t) = \sum_{i=1}^{k} s(B_i, t). \tag{2}$$

Since B_1 is active at time t, it must have had a job scheduled on it in time interval $(t-1, t]$. Also, the job that closed B_1 must have arrived in $(t-1, t]$. We conclude that servers $B_2, B_3, \ldots, B_k$ were opened in $(t-1, t]$ and thus all jobs scheduled on these servers since their opening are still active at time t; see Figure 3.1.

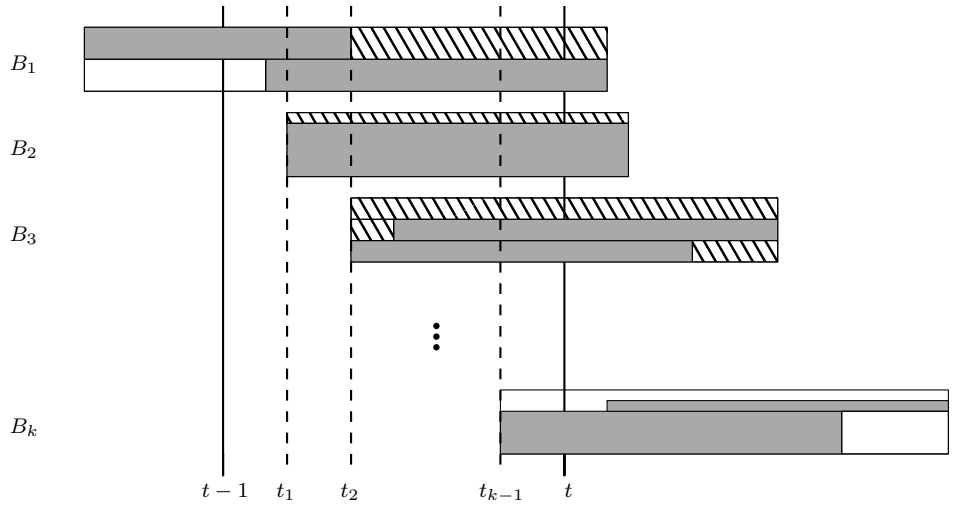

Figure 3.1: B_1 must have a job that arrived in $(t-1, t]$. Server B_i was closed at time $t_i \in [t-1, t]$ by a job that was placed in B_{i+1}.

Consider a server B_i that got closed at time $t_i \in (t-1, t]$ by a job σ_{z_i}. This means that job σ_{z_i} was scheduled on server B_{i+1} so we have the inequality:

$$s(B_i, t_i) + s(B_{i+1}, t_i) > 1 \quad \text{for } i \in \{2, \ldots, k-1\}.$$

Combining this with the fact that all jobs scheduled on servers $B_2, \ldots, B_k$ are still active at time t, we conclude:

$$s(B_i, t) + s(B_{i+1}, t) > 1 \quad \text{for } i \in \{2, \ldots, k-1\}. \tag{3}$$

Observe that the above inequality might fail for $i = 1$ since the jobs active at time t_1 in B_1 might finish before time t. Consider grouping servers in maximally many disjoint pairs $(2, 3)$, $(4, 5)$, ..., $(2j, 2j+1)$. The last pair must satisfy $2j + 1 \leq k$, which implies that $j = \lfloor \frac{k-1}{2} \rfloor$. Summing up equations (3) corresponding to this pairing we derive:

$$\sum_{q=1}^{j} (s(B_{2q}, t) + s(B_{2q+1}, t)) \geq j.$$

Combining this with equation (2) we conclude that $S(t-1, t) > j$. Rounding the left-hand side up we get $\lceil S(t-1, t) \rceil > j$, which implies that $\lceil S(t-1, t) \rceil \geq j + 1$ by integrality. Multiplying both

sides by 2 we get:

$$2\lceil S(t-1,t)\rceil \geq 2j+2 = 2(j+1) = 2\left(\lfloor\frac{k-1}{2}\rfloor+1\right) \geq 2((k/2-1)+1) = k.$$

To conclude, we have demonstrated that *NextFit*$(t) \leq 2\lceil S(t-1,t)\rceil$ and the equation $OPT(t) \geq \lceil S(t-1,t)\rceil$ is obvious. □

Corollary 3.2.2. *The competitive ratio of NextFit for the case equal duration and arbitrary arrival times is at most* 2*.*

3.3 *FirstFit*

In this section we consider the *FirstFit* algorithm, which unlike *NextFit* does not close servers unless they are no longer active. As already discussed in Section 1.1, in the *FirstFit* algorithm, a newly arriving job is assigned to the earliest (in the order of opening) active server that can accommodate it. Algorithm 2 provides the pseudocode.

3.3.1 Lower Bound for Jobs with Two Arrival Times

In this section, we present a parameterized lower bound on the competitive ratio of *FirstFit* for the case of equal duration and derive a couple of consequences of this parameterized lower bound. The adversarial instance consists of two sequences of jobs. In the first sequence all jobs have arrival time 0 and in the second sequence all jobs have arrival time $t \in (0,1)$, by which our lower bound is parameterized. All jobs have duration 1 in this instance. The first sequence is simply the nemesis instance for the bin packing problem due to [33]. Recall that this instance creates three groups of *FirstFit* servers: (1) those with the load roughly $5/6$, (2) those with the load roughly $2/3$, and (3) those with the load roughly $1/2$. Meanwhile, OPT is able to schedule all the jobs in the same instance into servers of duration 1 and load close to 1. Our second sequence consists of jobs that arrive at time t and are used to extend the duration of each existing *FirstFit* server by an additive $1-t$. Thus, the items used to extend group (1) servers have size $1/12$ each. Those items used to extend group (2) have size $1/6$ each. Lastly, those items used to extend group (3) have size $1/4$

Algorithm 2 *FirstFit* algorithm.

```
procedure FirstFit(σ = (σ_1, ..., σ_n))
    L ← ∅                                              ▷ List of active servers
    for i = 1 to n do
        new job σ_i = (a_i, f_i, s_i) arrives
        found ← false   ▷ Has an active server that can accommodate the new job been found?
        for j = 1 to L.size() do
            if a_i > L[j].T then          ▷ L[j].T is the current finishing time of server L[j]
                L.remove(L[j])
            else if s_i + s(L[j], a_i) ≤ 1 then                          ▷ The new job fits
                found ← true
                break
            end if
        end for
        if not found then
            Open a new server B at time a_i
            L.insert(B)
            j ← L.size()
        end if
        Assign σ_i to server L[j]
        L[j].T ← max(L[j].T, f_i)
    end for
end procedure
```

each. These items can be neatly combined by OPT into servers of duration 1 and load 1. The following theorem presents the parameterized lower bound. The proof is self-contained, since we reproduce the nemesis instance from [33] as part of the proof.

Theorem 3.3.1. *The asymptotic competitive ratio of FirstFit for the RSiC problem in the case where the duration of each job is* 1 *and jobs arrive at times* 0 *and* $t \in (0, 1)$ *is at least* $\frac{34}{27}(1 + t)$.

Proof. Fix a large positive integer k and take δ such that $0 < \delta \ll 18^{-k}$. For $i \in [k]$ we define $\delta_i = \delta \times 18^{k-i}$. The adversarial sequence σ of input items is a concatenation of two sequences. The jobs in the first sequence start at time zero, while the jobs in the second sequence start at time t. Furthermore, the first sequence is subdivided into three phases. The first phase consists of k groups, where each one consists of 10 jobs. Likewise, the second phase consists of k groups, where each group consists of 10 jobs. The third phase consists of $10k$ individual large jobs. Next, we describe the entire input with more details.

We begin by describing the first sequence. In the first phase, group $i \in [k]$ consists of items

$\sigma_{1,i} = (a_{1,i}, s_{1,i}), \sigma_{2,i} = (a_{2,i}, s_{2,i}), \ldots, \sigma_{10,i} = (a_{10,i}, s_{10,i})$ where $a_{1,i} = a_{2,i} = \cdots = a_{10,i} = 0$, and the sizes $s_{j,i}$ are defined as follows:

$$s_{1,i} = \frac{1}{6} + 33\delta_i, s_{2,i} = \frac{1}{6} - 3\delta_i, s_{3,i} = s_{4,i} = \frac{1}{6} - 7\delta_i, s_{5,i} = \frac{1}{6} - 13\delta_i, s_{6,i} = \frac{1}{6} + 9\delta_i,$$

$$s_{7,i} = s_{8,i} = s_{9,i} = s_{10,i} = \frac{1}{6} - 2\delta_i.$$

FirstFit puts $\sigma_{1,i}, \ldots, \sigma_{5,i}$ into server $2i-1$, which results in the server having the load $\frac{5}{6} + 3\delta_i$. None of the jobs $\sigma_{6,i}, \ldots, \sigma_{10,i}$ fit into this server. So, *FirstFit* assigns these jobs to server $2i$ with resulting load of $\frac{5}{6} + \delta_i$. Note that, none of the jobs in group i will fit in any of the previous servers from group $i-1$. The smallest load of a server corresponding to group $i-1$ jobs, which is server $2i-2$, is $\frac{5}{6} + \delta_{i-1} = \frac{5}{6} + 18\delta_i$, while the smallest job in group i is $\sigma_{5,i}$. Therefore, $\sigma_{5,i}$ cannot fit into server $2i-2$. Thus, *FirstFit* has to open 2 servers for each group i, where in total $2k$ servers are used for all of the jobs in the first phase.

The second phase has k groups, where each one consists of 10 jobs $\sigma'_{1,i} = (a'_{1,i}, s'_{1,i}), \sigma'_{2,i} = (a'_{2,i}, s'_{2,i}), \ldots, \sigma'_{10,i} = (a'_{10,i}, s'_{10,i})$. Each job has start time 0 and their sizes are given in order as follows:

$$s'_{1,i} = \frac{1}{3} + 46\delta_i, s'_{2,i} = \frac{1}{3} - 34\delta_i, s'_{3,i} = x'_{4,i} = \frac{1}{3} + 6\delta_i, s'_{5,i} = \frac{1}{3} + 12\delta_i, s'_{6,i} = \frac{1}{3} - 10\delta_i,$$

$$s'_{7,i} = s'_{8,i} = s'_{9,i} = s'_{10,i} = \frac{1}{3} + \delta_i.$$

According to *FirstFit*, jobs $\sigma'_{1,i}$ and $\sigma'_{2,i}$ are assigned to one server with the total size of $\frac{2}{3} + 12\delta_i$; and, jobs $\sigma'_{3,i}$ and $\sigma'_{4,i}$ are packed into one server with total load of $\frac{2}{3} + 12\delta_i$. It is clear that, the smallest job from the remaining items in this group cannot fit in any of the open servers. Thus, *FirstFit* has to place each pair $\sigma'_{5,i}$ and $\sigma'_{6,i}$; $\sigma'_{7,i}$ and $\sigma'_{8,i}$; $\sigma'_{9,i}$ and $\sigma'_{10,i}$ into different servers, each with a total load of $\frac{2}{3} + 2\delta_i$. Note that the smallest job in group i, which is $\sigma'_{2,i}$, cannot fit into any server from group $i-1$, as the load of all the servers is at least $\frac{2}{3} + 2\delta_{i-1} = \frac{2}{3} + 36\delta_i$. In other words, *FirstFit* has to open 5 servers for assigning jobs in each group i, employing $5k$ servers for

all the jobs in the second phase.

In the third phase, $10k$ jobs of size $\frac{1}{2} + \delta$ each are presented having start time 0. Since none of these jobs fit into the previously opened servers, *FirstFit* has to assign these jobs to new servers, resulting in opening $10k$ servers in this phase. This finishes the description of the first sequence.

In the second sequence, all jobs have start time t, and their sizes are as follows. The first $2k$ jobs are of size $\frac{1}{12}$ each, which are followed by $5k$ jobs of size $\frac{1}{6}$ each, followed by $10k$ jobs of size $\frac{1}{4}$ each. *FirstFit* assigns the first $2k$ jobs to the servers that were opened in the first phase (one job per server), the next $5k$ jobs to the servers that were opened in phase 2 (one job per server), and the last $10k$ jobs to the servers that were opened in phase 3 (one job per server).

In general, *FirstFit* uses $2k$ servers in the first phase, $5k$ bins in the second phase and $10k$ servers in the third phase. See Figure 3.2.a. The second sequence has the effect of extending the duration of each server (the period during which the server is active) to $1 + t$. Therefore, the cost of *FirstFit* is $17k(1 + t)$.

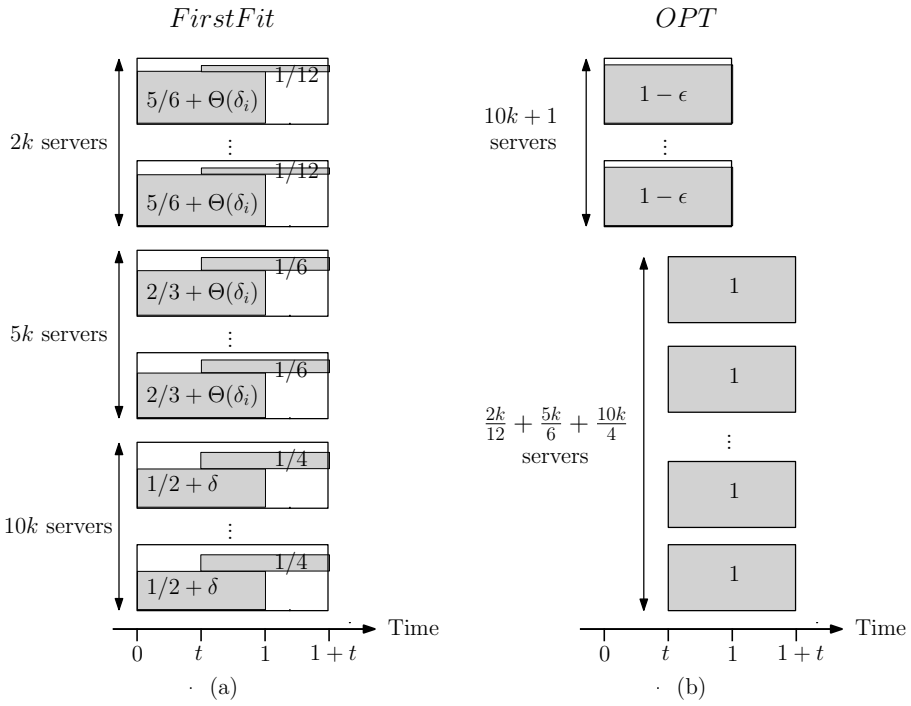

Figure 3.2: An illustration of schedules for the adversarial instance from Theorem 3.3.1 constructed by (a) *FirstFit*. (b) OPT.

The optimal offline algorithm, OPT, uses $10k$ servers for all the jobs from phase 3 (i.e., $10k$

jobs of size $\frac{1}{2}+\delta$). The remaining space in these bins will be filled with the following combinations of items from phases 1 and 2:

- for all jobs j; $j \geq 3$, in group i: OPT combines $\sigma_{j,i}$ with $\sigma'_{j,i}$.
- since $\sigma_{1,i}$ cannot be combined with $\sigma'_{1,i}$, OPT combines:

  (1) $\sigma_{1,i}$ with $\sigma'_{2,i}$ and,

  (2) $\sigma_{2,i}$ with $\sigma'_{1,i+1}$.

According to this packing scheme, two jobs $\sigma_{2,k}$ and $\sigma'_{1,1}$ remain unpacked. Therefore, OPT uses one extra server for packing these two jobs. Thus, OPT uses $10k+1$ servers for duration 1 to pack all jobs in the first 3 phases, i.e., the first sequence. For the jobs in the second sequence, OPT uses $\frac{k}{6}+\frac{5k}{6}+\frac{10k}{4}$ servers for jobs that are released at time t. In total, OPT uses $\frac{162k}{12}+1=\frac{27}{2}k+1$ servers, each of duration 1.

As a result, the competitive ratio of *FirstFit* for *RSiC* is at least $\frac{17k(1+t)}{27k/2+1} \rightarrow \frac{34}{27}(1+t)$ as $k \rightarrow \infty$. □

Substituting $t \rightarrow 1$ and $t = 1/2$ in Theorem 3.3.1, we obtain the following corollaries:

Corollary 3.3.2. *The competitive ratio of FirstFit for the case of equal duration and arbitrary arrival times is at least* $2.\overline{518}$.

Corollary 3.3.3. *The competitive ratio of FirstFit for the case of all jobs having duration* 2 *and arrival times* 0 *and* 1 *is at least* $1.\overline{8}$.

3.3.2 Upper Bound for Jobs with Two Arrival Times

In this section, we prove an upper bound for the competitive ratio of *FirstFit* for *RSiC* when all jobs have duration of precisely one and arrival times of 0 and t. Our result improves upon the previous best known upper bound of $\mu+3=4$. Our proof extends the weight function technique that was previously successfully applied to the bin packing problem to the *RSiC* problem. To the best of our knowledge, this is the first time the weight function technique has been used in the analysis of *RSiC*.

For the case of jobs of equal duration 1 and arrival times 0 and t, each *FirstFit* server falls in one of the following three categories:

Category C: starts at 0 and ends at 1,

Category D: starts at 0 and ends at $1 + t$,

Category E: starts at t and ends at $1 + t$.

In other words, servers in the category C have some items that arrived at time 0 and no items that arrived at time t. Servers in category D have some items that arrived at time 0 and some items that arrived at time t. Servers in category E have some items that arrived at time t and no items that arrived at time 0.

Let $C_1, C_2, \ldots, C_{k_1}$ be all the servers in category C listed in the order of their opening times. Let $D_1, D_2, \ldots, D_{k_2}$ be all servers in category D listed in the order of their opening times. Lastly, let $E_1, E_2, \ldots, E_{k_3}$ be all servers in category E listed in the order of their opening times.

Although we have ordered servers according to their opening times within each category, we will sometimes need to see how servers of different categories interact. Thus, we will make use of the following observations:

(1) C_i was opened before E_j for every C_i and E_j

(2) D_i was opened before E_j For every D_i and E_j

Thus, if we were to order C_i, D_j, E_k according to their opening times, then all the C_i would appear before any of the E_k and all the D_j would appear before any of the E_k; however, C_i and D_j could be interleaved.

Recall that for a server B opened by *FirstFit*, $s(B, 0)$ denotes the sum of all sizes of jobs assigned to B that were active at time 0. Since the only jobs active at time 0 are those with arrival time 0, we equivalently can say that $s(B, 0)$ is the sum of sizes of jobs assigned to B with arrival time 0. Similarly, note that at time 1 the only jobs that are active in B are those that arrived at time $t > 0$ and were assigned to B. Thus, $s(B, 1)$ denotes the sum of all sizes of jobs assigned to B with arrival time t. The sum of all the sizes of jobs assigned to B is called the total load on the server B and is denoted by $S(B)$, that is, $S(B) = s(B, 0) + s(B, 1)$. We now prove lower bounds on loads of different categories of servers.

Lemma 3.3.4. *For servers of category C or E used by FirstFit, at most one server can have a total load less than $1/2$, within each category. Moreover, if there are servers from both categories C and E used by FirstFit, then at most one server across both categories can have a load less than $1/2$ concurrently.*

Proof. Suppose, by contradiction, that we have two servers B_i and B_j with $i < j$ of category C, that have load less than $1/2$. This implies that $s(B_i, 0) + s(B_j, 0) \leq 1$. Therefore, *FirstFit* should not have opened server B_j because its items could have been accommodated by server B_i, contradicting the assumption. The same reasoning applies if both B_i and B_j belong to category E. Suppose we have a server B_i of category C and a server B_j of category E, each with a load less than $1/2$. According to *FirstFit*, items in server B_j should have been assigned to server B_i, since $s(B_i, 0) + s(B_j, t) \leq 1$. Therefore, *FirstFit* should not have opened server B_j, leading to a contradiction.

□

Lemma 3.3.5. *For servers of category D used by FirstFit, no more than two servers can have a total load less than $3/4$.*

Proof. Suppose, by contradiction, that we have three bins B_1, B_2, B_3 such that $S(B_i) < 3/4$ for $i = 1, 2, 3$. Without loss of generality, assume that the three bins were opened by *FirstFit* in the order B_1, B_2, B_3. Due to the definition of *FirstFit*, it follows that each item in B_2 and B_3 has size at least $1/4$ (otherwise, this item would have been placed into B_1). In particular, $s(B_2, 1), s(B_3, 1) > 1/4$. It follows that $s(B_2, 0), s(B_3, 0) < 3/4 - 1/4 = 1/2$. This is a contradiction since $s(B_3, 0)$ should have been placed into bin B_2 by the *FirstFit*. □

In light of the previous lemma, by ignoring at most 2 servers, we can assume that every server of category D in *FirstFit* has total load at least $3/4$. Inspired by the lower bound construction from Section 3.3.1, we note that there are certain thresholds for $s(B, 0)$ and $S(B)$ that are important for the proof. More specifically, $s(B, 0)$ thresholds are $5/6, 2/3, 1/2$; whereas $S(B)$ thresholds are $11/12, 5/6, 3/4$. For the sake of the analysis, we do not have to consider all pairs of thresholds. This follows from the following lemma:

Lemma 3.3.6. *Let $\alpha \in (0,1)$ and let $B_1, B_2, \ldots, B_k$ be the servers opened by FirstFit (in that order) that satisfy $s(B_i, 0) \geq \alpha$. Then at most one server B_i satisfies $S(B_i) < (1+\alpha)/2$.*

Proof. Consider two servers B_i, B_j with $i < j$ and assume, for contradiction, that $S(B_i), S(B_j) < (1+\alpha)/2$. This implies that $s(B_j, 1) = S(B_j) - S(B_j, 0) < (1+\alpha)/2 - \alpha = (1-\alpha)/2$, so the $s(B_j, 1)$ items can fit into B_i, since $S(B_i) + S(B_j, 1) < (1+\alpha)/2 + (1-\alpha)/2 = 1$. By the *FirstFit* rule these items should not have been placed in B_j. □

Thus, by ignoring at most three more servers, we may assume that all servers with $s(B, 0) \geq 5/6$ have $S(B) \geq 11/12$, all servers with $s(B, 0) \geq 2/3$ have $S(B) \geq 5/6$, and all servers with $s(B, 0) \geq 1/2$ have $S(B) \geq 3/4$.

Lemma 3.3.7. *For all but at most one server B with $1/2 < x(B, 0) \leq 2/3$, it holds that B contains a single job at time 0.*

Proof. Suppose for contradiction that there are two servers B_1 and B_2 containing at least two jobs at time 0 and satisfying $1/2 < s(B_i, 0) \leq 2/3$. Without loss of generality, assume that B_2 was opened later. Since $s(B_2, 0) \leq 2/3$, one of the jobs assigned to B_2 at time 0 must have size at most $1/3$. Then this job should have been placed in B_1 by *FirstFit*. □

For the purpose of analysis we divide servers in category D into three types as shown in Table 3.2.

Observe that by the previous discussion, a Type I server also satisfies $S(B) \geq 11/12$. Type II server is further subdivided into two subtypes: Type II(a) satisfies $S(B) \geq 11/12$, Type II(b) satisfies $5/6 \leq S(B) < 11/12$. Similarly Type III server is further subdivided into several subtypes: Type III(a) satisfies $S(B) \geq 11/12$, Type III(b) satisfies $5/6 \leq S(B) < 11/12$, and Type III(c) satisfies $3/4 \leq S(B) < 5/6$. By Lemma 3.3.7 each Type III server contains exactly one item at time 0.

We make some further observations. Consider a server of Type II (b). We let $\epsilon_1, \epsilon_2 > 0$ be such that $s(B, 0) = 5/6 - \epsilon_1$ and $S(B) = 11/12 - \epsilon_2$. It follows that $s(B, 1) = 1/12 + \epsilon_1 - \epsilon_2$. Also, since each job arriving at time t (except possibly for the first server of each type) has to have size

Table 3.2: Different types of servers in category D

Type	Subtype	Bounds on $s(B,0)$	Bounds on $S(B)$	Num of items at time 0
Type I		$5/6 \le s(B,0) \le 1$	$11/12 \le S(B) \le 1$	≥ 1
Type II		$2/3 \le s(B,0) < 5/6$		
	(a)		$11/12 \le S(B) \le 1$	≥ 1
	(b)		$5/6 \le S(B) < 11/12$	≥ 1
Type III		$1/2 < s(B,0) < 2/3$		
	(a)		$11/12 \le S(B) \le 1$	$= 1$
	(b)		$5/6 \le S(B) < 11/12$	$= 1$
	(c)		$3/4 \le S(B) < 5/6$	$= 1$

greater than $1/12$, it follows that $S(B) = s(B,0) + s(B,1) > 5/6 - \epsilon_1 + 1/12 = 11/12 - \epsilon_1$. It follows that $\epsilon_1 > \epsilon_2$. In addition, we observe that $\epsilon_1 \le 1/6$ and $\epsilon_2 \le 1/12$.

Similar observations hold for a server of Type III (c). More specifically, we have $s(B,0) = 2/3 - \epsilon_1$ and $s(B,1) = 1/6 + \epsilon_1 - \epsilon_2$. The choices of ϵ_1 and ϵ_2 depend on the server, of course; however, we always have $\epsilon_1 > \epsilon_2$ and $\epsilon_1 \le 1/6$ while $\epsilon_2 \le 1/12$.

With these notations, we are ready to establish an upper bound using the weighting technique. Define the following weight functions: w_1 for the items arriving at time 0 and w_2 for the items arriving at time t:

$$w_1(s) = \frac{156}{131}(1+t)s + \begin{cases} 0 & \text{if } s \le 1/2 \\ \frac{12}{131}(1+t) & \text{otherwise} \end{cases} \qquad \text{and} \qquad w_2(s) = \frac{168}{131}(1+t)s.$$

We extend the definitions of w_1 and w_2 to servers as follows: for a server S the notation $w_1(S)$ stands for the sum of $w_1(s)$ which ranges over sizes of items $\sigma = (s,0)$ that arrived and were placed into S at time 0. Similarly, $w_2(S)$ stands for $w_2(s(S,2))$.

By employing these two weight functions on the items packed into servers of categories C, D and E used by *FirstFit*, we can prove the following:

Lemma 3.3.8. *On any input σ with items arriving at two arrival times 0 and t, we have the following:*

(i) If $t \ge \frac{41}{90} \approx 0.456$ then $w_1(C_i) \ge 1 = d(C_i)$ for all but constantly many servers C_i in

category C.

(ii) For all but constantly many servers D_i in category D it holds that $w_1(D_i)+w_2(D_i) \geq 1+t = d(D_i)$.

(iii) If $t \geq \frac{47}{84} \approx 0.560$ then $w_2(E_i) \geq 1 = d(E_i)$ for all but constantly many servers E_i in category E.

(iv) If $t \geq \frac{1}{28} \approx 0.035$ then $w_1(S) + w_2(S) \leq \frac{168}{131}(1+t)d(S)$ for every server S of OPT.

Proof. (i) By Lemma 3.3.4 we may assume that $s(C_i, 0) > 1/2$. If $s(C_i, 0) \leq 2/3$ then by Lemma 3.3.7, we may assume that C_i contains a single item. Therefore, we have:

$$w_1(C_i) \geq \frac{156}{131}(1+t)\cdot s(C_i,0)+\frac{12}{131}(1+t) \geq \frac{156+156(41/90)}{131}\cdot\frac{1}{2}+\frac{12+12(41/90)}{131} = 1,$$

where we used $t \geq 41/90$ in the second inequality.

If $s(C_i, 0) > 2/3$, then we have:

$$w_1(C_i) \geq \frac{156}{131}(1+t)\cdot s(C_i,0) \geq \frac{156+156(41/90)}{131}\cdot\frac{2}{3} > 1.$$

(ii) Let D_i be a server in category D that is used by *FirstFit* such that $S(D_i) \geq 11/12$. Then we have:

$$w_1(D_i) + w_2(D_i) \geq \frac{156}{131}(1+t)S(D_i) \geq \frac{156}{131}(1+t)\frac{11}{12} > 1+t.$$

We handle servers of Type II (b) next. Let D_i be such a server and use the notation of ϵ_1 and

ϵ_2 introduced prior to the theorem statement. In this case, we have

$$\begin{aligned} w_1(D_i) + w_2(D_i) &= \frac{156}{131}(1+t)\left(\frac{5}{6} - \epsilon_1\right) + \frac{168}{131}(1+t)\left(\frac{1}{12} + \epsilon_1 - \epsilon_2\right) \\ &= (1+t)\left(\frac{156}{131}\cdot\frac{5}{6} + \frac{168}{131}\cdot\frac{1}{12} + \frac{12}{131}\epsilon_1 - \frac{168}{131}\epsilon_2\right) \\ &= (1+t)\left(\frac{130}{131} + \frac{14}{131} + \frac{12}{131}\epsilon_1 - \frac{168}{131}\epsilon_2\right) \\ &> (1+t)\left(\frac{130}{131} + \frac{14}{131} + \frac{12}{131}\epsilon_2 - \frac{168}{131}\epsilon_2\right) \\ &= (1+t)\left(\frac{130}{131} + \frac{14}{131} - \frac{156}{131}\epsilon_2\right) \\ &\geq (1+t)\left(\frac{130}{131} + \frac{14}{131} - \frac{13}{131}\right) = 1+t, \end{aligned}$$

where the first inequality follows from $\epsilon_1 > \epsilon_2$ and the second from $\epsilon_2 \leq 1/12$.

Next, we handle Type III (b) servers. Let D_i be such a server, and define ϵ to mean $s(D_i, 0) = 1/2 + \epsilon$. Since $S(D_i) \geq 5/6$, it follows that $s(D_i, 1) \geq 5/6 - (1/2 + \epsilon) = 1/3 - \epsilon$. Also observe that $\epsilon < 1/6$. Plugging these estimates into the weight function we obtain:

$$\begin{aligned} w_1(D_i) + w_2(D_i) &= \frac{156}{131}(1+t)\left(\frac{1}{2} + \epsilon\right) + \frac{12}{131}(1+t) + \frac{168}{131}(1+t)\left(\frac{1}{3} - \epsilon\right) \\ &= (1+t)\left(\frac{146}{131} - \frac{12}{131}\epsilon\right) > (1+t)\left(\frac{146}{131} - \frac{2}{131}\right) \geq 1+t, \end{aligned}$$

where we have used $\epsilon < 1/6$ in the penultimate step.

We are only left to check the servers of Type III (c). Let D_i be such a server and redefine ϵ_1 and ϵ_2 for this server. Then we have

$$\begin{aligned} w_1(D_i) + w_2(D_i) &= \frac{156}{131}(1+t)\left(\frac{2}{3} - \epsilon_1\right) + \frac{12}{131}(1+t) + \frac{168}{131}(1+t)\left(\frac{1}{6} + \epsilon_1 - \epsilon_2\right) \\ &= (1+t)\left(\frac{144}{131} + \frac{12}{131}\epsilon_1 - \frac{168}{131}\epsilon_2\right) \geq (1+t), \end{aligned}$$

Thus, we have proved that for all, but constantly c many, servers of type 2 used by *FirstFit* it

holds that $w_1(D_i) + w_2(D_i) \geq 1 + t$, considering $t > 0.3571$.

(iii) Let E_i be a server that starts at time t and finishes at time $t + 1$ which is of category E of servers. By Lemma 3.3.4 we can assume this server has total load $> 1/2$. Therefore, when we apply the weight function w_2, we get:

$$w_2(E_i) \geq \frac{168}{131}(1 + t) \cdot \frac{1}{2} \geq \frac{168(1 + (47/84))}{262} = 1.$$

(iv) Let S be an arbitrary server of OPT. If S contains only jobs that arrived at time 0 then $w_1(S) \leq \frac{156}{131}(1 + t) + \frac{12}{131}(1 + t) = \frac{168}{131}(1 + t) = \frac{168}{131}(1 + t)d(S)$, since $d(S) = 1$ in this case. Similarly, if S contains only jobs that arrived at time t, then $w_2(S) \leq \frac{168}{131}(1 + t) = \frac{168}{131}(1 + t)d(S)$, since again $d(S) = 1$. If S contains both jobs arriving at time 0 and time t then the weight function is maximized when there is one item at time 0 that barely exceeds $1/2$ (so that we collect the bonus for w_1) and the remaining items arrive at time t and add up to barely $1/2$ (since w_2 point-wise is at least w_1). Thus, we have:

$$\begin{aligned} w_1(S) + w_2(S) &\leq \frac{1}{2} \cdot \frac{156}{131}(1 + t) + \frac{12}{131}(1 + t) + \frac{1}{2} \cdot \frac{168}{131}(1 + t) \\ &= \frac{174}{131}(1 + t) \leq \frac{168}{131}(1 + t) \cdot (1 + t) \\ &= \frac{168}{131}(1 + t)d(S), \end{aligned}$$

where the last inequality follows from $t \geq \frac{1}{28}$, and the last equality follows from $d(S) = 1+t$ in this case. Thus, in all cases, we show that $w_1(S) + w_2(S) \leq \frac{168}{131}(1 + t)d(S)$, where $d(S)$ denotes the duration of the server S.

□

Theorem 3.3.9. *On input σ where each job has duration 1 and arrival time 0 and $t > \frac{47}{84} \approx 0.560$, FirstFit achieves competitive ratio at most* $\frac{168}{131}(1 + t)$.

Proof. Let $s_1, \ldots, s_n$ denote the sizes of all jobs arriving at time 0 and $s'_1, \ldots, s'_m$ denote sizes of all jobs arriving at time t. Let $B_1, \ldots, B_k$ denote *FirstFit* servers and $S_1, \ldots, S_p$ denote servers

of OPT. Since $t > \frac{47}{84}$, parts (i)-(iii) of Lemma 3.3.8 imply that $w_1(B_i) + w_2(B_i) \geq d(B_i)$ for all but some constant number c of servers B_i. Similarly, part (iv) of Lemma 3.3.8 implies that $w_1(S_i) + w_2(S_i) \leq \frac{168}{131}(1+t)d(S_i)$. Combining everything together we have:

$$\begin{aligned} \textit{FirstFit} - (1+t)c &\leq \sum_{i=1}^{k} d(B_i) \leq \sum_{i=1}^{k}(w_1(B_i) + w_2(B_i)) = \sum_{i=1}^{n} w_1(x_i) + \sum_{i=1}^{m} w_2(x_i') \\ &= \sum_{i=1}^{p}(w_1(S_i) + w_2(S_i)) \leq \frac{168}{131}(1+t)\sum_{i=1}^{p} d(S_i) = \frac{168}{131}(1+t)OPT. \end{aligned}$$

□

By taking $t \to 1$, we obtain the following corollary:

Corollary 3.3.10. *The asymptotic competitive ratio of FirstFit for the case of jobs of equal duration, two arbitrary arrival times* 0 *and* $t \in (\frac{47}{84}, 1]$ *is at most* 2.565.

Examining various conditions on the range of t in Lemma 3.3.8, we extend the range of t where Theorem 3.3.2 holds for inputs where *FirstFit* has specified behaviors.

Corollary 3.3.11. *On inputs* σ *where each job has duration* 1*, arrival time* 0 *and* $t \in (\frac{41}{90}, 1)$*, and no new servers are opened by FirstFit at time* t*, FirstFit achieves competitive ratio at most* $\frac{168}{131}(1+t)$.

Corollary 3.3.12. *On inputs* σ *where each job has duration* 1*, arrival time* 0 *and* $t \in [\frac{1}{28}.1)$*, and all FirstFit servers have duration* $1+t$*, FirstFit achieves competitive ratio at most* $\frac{168}{131}(1+t)$.

3.3.3 Strict Upper Bound: Equal Duration 2 and Arrival Times 0 and 1

In this section, we prove an upper bound of 2 on the *strict* competitive ratio of *FirstFit* for *RSiC* when all jobs have duration of precisely two and arrival times of 0 and 1. Clearly this is equivalent to the situation when all jobs have duration one, and arrival times of 0 or $1/2$. The main result of this section is stated in Theorem 3.3.18. The proof is done by a careful case analysis. Since there are two arrival times, we have a lot more cases to deal with than in the bin packing problem. We begin by setting up some notation that will be common to all lemmas in this section.

For the case of jobs of equal duration 2 and arrival times 0 and 1, we have the same categories of servers as mentioned in Section 3.3.2.

Recall that $s(B, j)$ denotes the total size of all items that are *active* at time j and were assigned to server B. If we wish to refer to items that arrived to server B at time 1, we can use the notation $s(B, 2)$ since the only jobs active at time 2 in B are those that arrived at time 1 (jobs that arrived at time 0 are active during the half-open interval $[0, 2)$).

We begin with a number of observations concerning the relationships between pairs of servers of the same type.

Lemma 3.3.13. *The following inequalities hold:*

$$s(C_i, 0) + s(C_{i+1}, 0) > 1 \quad \textit{for} \quad i \in [k_1 - 1] \tag{4}$$

$$s(D_i, 0) + s(D_{i+1}, 0) > 1 \quad \textit{for} \quad i \in [k_2 - 1] \tag{5}$$

$$s(E_i, 2) + s(E_{i+1}, 2) > 1 \quad \textit{for} \quad i \in [k_3 - 1] \tag{6}$$

$$s(C_1, 0) + s(C_{k_1}, 0) > 1 \quad \textit{if} \quad k_1 > 1 \tag{7}$$

$$s(D_1, 0) + s(D_{k_2}, 0) > 1 \quad \textit{if} \quad k_2 > 1 \tag{8}$$

$$s(E_1, 0) + s(E_{k_3}, 0) > 1 \quad \textit{if} \quad k_3 > 1 \tag{9}$$

$$s(D_i, 0) + s(D_i, 2) + s(D_{i+1}, 2) > 1 \quad \textit{for} \quad i \in [k_2 - 1] \tag{10}$$

Proof. All the above inequalities follow directly from the definition of *FirstFit* and the ordering of the servers. For example, C_{i+1} was opened by *FirstFit* because the first item to be placed in it did not fit in C_i, yielding Inequality (4). The same logic gives rise to a chain of inequalities (5) and (6) concerning the D_i and E_i servers respectively, as well as (7), (8), and (9). Note that the $s(D_{i+1}, 2)$ items didn't fit into D_i, which contained at most $s(D_i, 0) + s(D_i, 2)$, hence the chain of inequalities (10). □

The next lemma concerns relationships between pairs of servers of *different* types.

Lemma 3.3.14. *The following inequalities hold:*

$$s(D_{k_2}, 0) + s(D_{k_2}, 2) + s(E_{k_3}, 2) > 1 \tag{11}$$

$$s(D_{k_2}, 0) + s(D_{k_2}, 2) + s(E_1, 2) > 1 \tag{12}$$

$$s(C_1, 0) + s(D_{k_2}, 0) > 1 \tag{13}$$

$$s(C_i, 0) + s(E_j, 2) > 1 \quad \textit{for } i \in [k_3], j \in [k_2] \tag{14}$$

Proof. As observed earlier, every E_j server was opened at time 1, after *all* C_i and D_i servers were opened, and items were placed into an E_j server because they did not fit into D_i and C_i servers, leading to Inequalities (12),(11), and (14. Inequality (13) holds regardless of if C_1 was opened before or after D_{k_2}. □

Define A_1 to be the sum of all items that arrived at time 0 and were packed into C_i servers, A_2 to be the sum of all items that arrived at time 0 and were packed into D_i servers, and B_2 to be the sum of all items that arrived at time 1 and were packed into D_i servers, and finally B_3 to be the sum of all items that arrived at time 1 and were packed into E_i servers. More specifically, we have:

$$A_1 := \sum_{i=1}^{k_1} s(C_i, 0), \quad A_2 := \sum_{i=1}^{k_2} s(D_i, 0), \quad B_2 := \sum_{i=1}^{k_2} s(D_i, 2), \quad B_3 := \sum_{i=1}^{k_3} s(E_i, 2).$$

We also define $A := A_1 + A_2$ and $B = B_2 + B_3$

Since the duration of each item is 2, the following lower bound on the cost of OPT is immediate:

Lemma 3.3.15. $OPT \geq \lceil 2A + 2B \rceil$

Next we show some upper bounds on the number of different categories of servers, that hold when the number of servers of each category are large enough.

Lemma 3.3.16. *The following inequalities hold:*

$$2A_1 > k_1 \quad \text{if } k_1 > 1 \tag{15}$$

$$2A_2 > k_2 \quad \text{if } k_2 > 1 \tag{16}$$

$$2B_3 > k_3 \quad \text{if } k_3 > 1 \tag{17}$$

$$A_2 + 2B_2 > k_2 - 1 + s(D_{k_2}, 0) + s(D_1, 2) + s(D_{k_2}, 2) \quad \text{if } k_2 > 1 \tag{18}$$

$$2A_1 + 2B_3 > k_1 + k_3 \quad \text{if } k_1, k_3 \geq 1 \tag{19}$$

$$4A_2 + 4B_2 > 3k_2 - 1 \quad \text{if } k_2 > 2 \tag{20}$$

Proof. Adding up all the inequalities (4) and (7) gives us (15). Adding up all the inequalities (5) and (8) gives us (16); adding up all the inequalities (6) with (9) gives us (17), and adding all the inequalities (10) gives us (18).

It follows from (14) that $s(C_{k_1}, 0) + s(E_1, 2) > 1$ and $s(C_1, 0) + s(E_{k_3}, 2) > 1$. Adding these two inequalities to the two chains of inequalities (4) and (6), we obtain (19). (It can be verified that this is true even if one or both of k_1, k_2 equal 1.)

To show Inequality 20, we consider the following 2 cases:

Case 1: $s(D_1, 2) + s(D_{k_2}, 0) + s(D_{k_2}, 2) \geq 1/2$: Then we can write Inequality 18 as: $A_2 + 2B_2 > k_2 - 1/2$. Multiplying this inequality by two and adding to it Inequality 16, we obtain Inequality 20.

Case 2: $s(D_1, 2) + s(D_{k_2}, 0) + s(D_{k_2}, 2) < 1/2$. Let us denote $s(D_{k_2}, 0)$ by ϵ. By *FirstFit* rules we $s(D_i, 0) > 1 - \epsilon$ for $i \in \{1, \ldots, k_2 - 1\}$.

Adding all these inequalities together with $s(D_{k_2}, 0) = \epsilon$ we obtain:

$$A_2 > k_2 - 1 - (k_2 - 2)\epsilon. \tag{21}$$

From Inequality 18 it follows that:

$$A_2 + 2B_2 > k_2 - 1 + \epsilon. \tag{22}$$

If we multiply this inequality by 2 and sum it up with twice the Inequality (21) then we have:

$$\begin{aligned} 4A_2 + 4B_2 &> 4(k_2 - 1) - 2(k_2 - 2)\epsilon + 2\epsilon \\ &> 4(k_2 - 1) - 2(k_2 - 3)\epsilon \\ &> 4(k_2 - 1) - (k_2 - 3) \\ &= 4k_2 - 4 - k_2 + 3 = 3k_2 - 1, \end{aligned}$$

where the third inequality is due to $\epsilon < 1/2$. Observe that we assumed that $k_2 > 3$ in the above calculation. Observe that if $k_2 = 3$ we obtain $4A_2 + 4B_2 > 4(k_2 - 1) - 2(k_2 - 3)\epsilon = 4(k_2 - 1) = 4k_2 - 4$. However, for the case $k_2 = 3$ it holds that $4k_2 - 4 = 3k_2 - 1$, so we conclude that the inequality $4A_2 + 4B_2 \geq 3k_2 - 1$ holds for all $k_2 > 2$. □

The following lemma relates the cost of *FirstFit* to the cost of OPT and the number of category-D servers used by *FirstFit* by using a reduction to bin packing.

Lemma 3.3.17. *Let σ be an input on which FirstFit uses k_2 servers of type D. Then*

$$\textit{FirstFit}(\sigma) \leq 1.7\,\mathrm{OPT}(\sigma) + k_2$$

Proof. Define σ' to be an input derived from σ by shifting the arrival times of items arriving at time 1 to time 0, while preserving the original ordering of items. It is easy to see that for any solution for σ, we can 'slide' the items arriving at time 1 back to time 0, thus obtaining a valid solution for σ' without increasing the cost. Thus $OPT(\sigma') \leq OPT(\sigma)$. Furthermore, we have *FirstFit*$(\sigma') =$ *FirstFit*$(\sigma) - k_2$, since all the k_2 type-D *FirstFit* servers have duration 2 in the solution for σ' instead of 3 in the solution for σ. Lastly, since σ' is just an instance of regular bin packing, we have *FirstFit*$(\sigma') \leq 1.7 \cdot OPT(\sigma')$. Combining everything together, we get: *FirstFit*$(\sigma) =$ *FirstFit*$(\sigma') + k_2 \leq 1.7 \cdot OPT(\sigma') + k_2 \leq 1.7 \cdot OPT(\sigma) + k_2$ □

Now, we are ready to state the main result of this section.

Theorem 3.3.18. *Let σ be an input to the RSiC problem. If the duration of each job in σ is exactly 2 and arrival times are 0 and 1 then FirstFit$(\sigma) \leq 2 \cdot OPT(\sigma)$.*

Proof. We show that *FirstFit* $< 2 \cdot OPT + 1$ when *FirstFit* uses (a) exactly one type of server (Lemma 3.3.19) (b) exactly two types of servers (Lemma 3.3.20, 3.3.21, and 3.3.22) and (c) all three types of servers (Lemma 3.3.23). Then the result follows using integrality. □

We start with the case where only one type of server is used.

Lemma 3.3.19. *If exactly one of k_1, k_2, k_3 is non-zero, then FirstFit $< 2 \cdot OPT + 1$.*

Proof. If $k_2 = 0$, we have *FirstFit* $\leq 1.7\,\mathrm{OPT}$ by Lemma 3.3.17. So we assume below that $k_1 = k_3 = 0$ and $k_2 \geq 1$. First note that if $k_2 = 1$, then *FirstFit* is optimal. If $k_2 = 2$ then the cost of *FirstFit* is 6 (two servers of duration 3). Meanwhile, OPT has to open at least 2 servers at time 0; at time 1, OPT either opens a new bin with total cost 6 or extends at least one server with total cost ≥ 5. In both cases, the lemma follows. Therefore we assume $k_2 \geq 3$. We have $OPT \geq 2(A_2 + B_2)$. Observing that *FirstFit* $= 3k_2$ and using Equation (20), we conclude *FirstFit* $< 2 \cdot OPT + 1$. □

The next three lemmas consider the cases when exactly two types of servers are present.

Lemma 3.3.20. *If $k_1 = 0$ and $k_2, k_3 \geq 1$, then FirstFit $< 2 \cdot OPT + 1$.*

Proof. We consider the following cases:

$k_2 = 1$: Then there were items that arrived at time 0 as well as at time 1. At time 0, OPT had to open a server for a cost of 2. Since $k_3 \geq 1$, clearly the total size of items at time 1 exceeded 1, and OPT had to pay a minimum additional cost of 2. Thus $\mathrm{OPT} \geq 4$, which implies that $1.7\,\mathrm{OPT} + k_2 = 1.7\,\mathrm{OPT} + 1 \leq 2\,\mathrm{OPT}$. By Lemma 3.3.17, we have:

$$\textit{FirstFit} \leq 1.7OPT + k_2 \leq 2\,\mathrm{OPT}$$

$k_2 = 2$: If $k_3 = 1$, we have *FirstFit* $= 8$ and $\mathrm{OPT} \geq 5$, as at least two servers are needed at time 1 for a cost of 4, and OPT has to pay at least an additional cost of 1 for the items that arrive at time 1. Thus *FirstFit* $\leq 2\,\mathrm{OPT} + 1$. Therefore let $k_3 \geq 2$. In this case, we claim $\mathrm{OPT} \geq 7$.

To see this, observe that OPT must have opened at least two servers at time 0, and pay a cost of 2 between time 0 and 1. Next between time 1 and 2, since $s(D_1, 0) + s(D_2, 0) > 1$ and $s(E_1, 2) + s(E_2, 2) > 1$, and all these items are active in this time interval, OPT must pay a cost of at least 3. Finally between time 2 and 3, OPT must pay a cost of at least 2, for a total cost of at least 7. We conclude that $1.7\,\mathrm{OPT} + k_2 = 1.7\,\mathrm{OPT} + 2 \leq 2\,\mathrm{OPT}$. The result now follows by Lemma 3.3.17.

$k_2 > 2$: If $k_3 > 1$, we have *FirstFit* $= 3k_2 + 2k_3$ and $OPT \geq 2(A_2 + B_2 + B_3)$. Therefore

$$\textit{FirstFit} - 1 = 3k_2 - 1 + 2k_3 < 4A_2 + 4B_2 + 4B_3 \leq 2 \cdot OPT$$

where the second inequality is derived by adding (20) to twice (17).

If instead $k_3 = 1$, we have *FirstFit* $= 3k_2 + 2$.

Then we add the inequalities $s(D_2, 0) + s(D_3, 0) > 1, \ldots, s(D_{k_2-1}, 0) + s(D_{k_2}, 0) > 1$ and $s(D_2, 0) + s(D_{k_2}, 0) > 1$ to obtain:

$$2A_2 > k_2 - 1 + 2x(D_1, 0). \tag{23}$$

Adding (23) to twice Inequality (18) and to $4B_3 = 4s(E_1, 2)$ we obtain:

$$\begin{aligned} 4A + 4B &> 2k_2 - 2 + 2s(D_1, 2) + 2s(D_{k_2}, 0) + 2s(D_{k_2}, 2) + k_2 - 1 + 2s(D_1, 0) + 4s(E_1, 2) \\ &= 3k_2 - 3 + 2(s(D_1, 0) + s(D_1, 2) + s(E_1, 2)) + 2(s(D_{k_2}, 0) + s(D_{k_2}, 2) + s(E_1, 2)) \\ &> 3k_2 + 1 = \textit{FirstFit} - 1, \end{aligned}$$

where the last inequality follows from $s(D_1, 0) + s(D_1, 2) + s(E_1, 2) > 1$ and $s(D_{k_2}, 0) + s(D_{k_2}, 2) + s(E_1, 2) > 1$. The lemma follows from Lemma 3.3.15.

□

Lemma 3.3.21. *If $k_2 = 0$ and $k_1, k_3 \geq 1$ then FirstFit $< 2 \cdot OPT + 1$.*

Proof. Follows from Lemma 3.3.17. □

Lemma 3.3.22. *If $k_3 = 0$ and $k_1, k_2 \geq 1$, then FirstFit $< 2 \cdot OPT + 1$.*

Proof. We consider the following cases:

$k_2 = 1$: Observe that there must be items that arrived at time 0 as well as at time 1, since $k_2 \geq 1$. *FirstFit* opens at least two servers at time 0, so it must be that OPT has to open at least two servers at this time, for a cost of 4. Then OPT has to pay at least an additional cost of 1 for the items arriving at time 1. Thus $\mathrm{OPT} \geq 5$. The result now follows from Lemma 3.3.17 and the fact that $1.7\,\mathrm{OPT} + k_2 \leq 2\,\mathrm{OPT}$ for $\mathrm{OPT} \geq 5$.

$k_1 = 1; k_2 > 1$: We have *FirstFit* $= 3k_2 + 2$. By adding $s(D_{k_2}, 0) + s(C_1, 0) > 1$ to Inequality 18, we obtain

$$A_1 + A_2 + 2B_2 > k_2 \tag{24}$$

If $A_1 > 1/2$, we have $2A_1 > 1$, and adding this to twice the above inequality and to Inequality 16, we get $4A + 4B > 3k_2 + 1$. Applying Lemma 3.3.15 gives the desired bound. If instead $A_1 \leq 1/2$, we add the following series of inequalities:

$$s(D_i, 0) > 1 - A_1 \quad for \quad i \in [1, ..., k_2], and$$
$$s(C_1, 0) = A_1$$

to obtain:

$$A_2 + A_1 > k_2 - A_1(k_2 - 1) \tag{25}$$

By adding $A_1 + s(D_{k_2}, 0) > 1$ to Inequality (25) and Inequality (18) and multiplying by 2, we get:

$$\begin{aligned} 4A_1 + 4A_2 + 4B &> 4k_2 - 2A_1(k_2 - 1) + 2s(D_1, 2) + 2s(D_{k_2}, 2) \\ &> 3k_2 + 1 + 2s(D_1, 2) + 2s(D_{k_2}, 2) > 3k_2 + 1 \end{aligned}$$

where the second inequality follows from $A_1 < 1/2$. Once again, Lemma 3.3.15 gives the desired bound.

$k_1, k_2 > 1$: We know that $s(C_1, 0) + s(C_{k_1}, 0) > 1$, therefore at least one of them is greater than $1/2$. If $s(C_{k_1}, 0) > 1/2$, we add up Inequalities (4), (10) and (13 to obtain:

$$2A_1 - s(C_{k_1}, 0) + A_2 + 2B > k_1 + k_2 - 1 + s(D_1, 2) + s(D_{k_2}, 2)$$

Since $s(C_{k_1}, 0) > 1/2$ we have:

$$2A_1 + A_2 + 2B > k_1 + k_2 - 1/2. \tag{26}$$

The case $s(C_1, 0) > 1/2$ can be handled similarly to obtain (26), by using $s(C_{k_1}, 0) + s(D_{k_2}, 0) > 1$ in place of (13).

Adding Inequality (16) and twice Inequality (26) we obtain $2k_1 + 3k_2 - 1 < 4A + 4B$. Since $2k_1 + 3k_2 =$ *FirstFit*, using Lemma 3.3.15, we obtain *FirstFit* $< 2 \cdot OPT + 1$.

□

We end with the case when all three types of servers exist.

Lemma 3.3.23. *If $k_1, k_2, k_3 \geq 1$, then FirstFit $< 2 \cdot OPT$ + 1.*

Proof. We consider the following cases:

$k_2 = 1$: As in the proof of Lemma 3.3.22, the lemma follows from Lemma 3.3.17 and the fact that OPT ≥ 5.

$k_2 = 2$: If $k_1 = k_3 = 1$ then *FirstFit* $= 10$ and OPT ≥ 5 as it has to open at least 2 servers at time 0 for a cost of 4, and pay at least an additional cost of 1 to accommodate items arriving at time 1. Therefore *FirstFit* $<$ 2 OPT $+1$ in this case. Otherwise either $k_1 \geq 2$ or $k_2 \geq 2$. We claim that OPT ≥ 7 in both cases. If $k_1 \geq 2$, then since *FirstFit* opens at least 4 servers at time 0, it follows from the bin packing upper bound on *FirstFit* that OPT needs at least 3 servers at time 0 for a cost of 6; and then must pay at least an additional cost of 1 at time 1 to accommodate items arriving at time 1 for a total cost of at least 7. If $k_3 \geq 2$, then OPT needs to pay at least cost 2 between times 0 and 1, at least 3 between time 1 and 2, and at least

2 between time 2 and 3, for a total cost of at least 7. This completes the proof of the claim. Since $\mathrm{OPT} \geq 7$, by Lemma 3.3.17, we have *FirstFit* $\leq 1.7\,\mathrm{OPT} + 2 \leq 2\,\mathrm{OPT}$ as needed.

$k_2 > 2$: Observe that *FirstFit* $= 2k_1 + 3k_2 + 2k_3$. Adding Inequality (20 to twice (19), we obtain

$$\textit{FirstFit} - 1 = 2k_1 + 3k_2 + 2k_3 - 1 < 4A + 4B \leq 2\,\mathrm{OPT}$$

□

3.4 Summary and Discussion

In this chapter, we considered the *RSiC* problem under equal duration and two arrival times. For this scenario, we established a tight bound of 2 on *NextFit*. We also derived a lower bound on the competitive ratio for *FirstFit* when jobs have duration 1 and arrival times 0 and t for $t \in (0, 1)$. This surpasses the bin packing lower bound for $t > 0.35$. Using the weight function technique, we showed an upper bound of $\frac{168}{131}(1 + t)$ on the asymptotic competitive ratio of *FirstFit* where jobs have duration 1 and arrival times 0 and t for $t \in [0.559, 1)$.

Although the theoretical analysis established that the competitive ratios of *NextFit* and *FirstFit* are 2 and approximately 2.56, respectively, our experimental findings demonstrated that the competitive ratios of these algorithms are significantly better than the theoretical predictions. A detailed discussion of our experimental results can be found in Section 7.5.

As mentioned at the start of this chapter, the objective of examining *RSiC* with equal duration jobs was to first establish results for this simplified case and then work toward solving the problem in its general form. Thus, the initial focus was on determining asymptotic bounds for the restricted case, followed by refining these bounds to derive strict bounds, and ultimately extending these findings to the original problem. In this chapter, we made progress on the first two steps: we derived asymptotic bounds for the special case where jobs arrive at two distinct times, and then we derived new strict bounds (by other methods) for the case of two midterm arrivals. Our techniques do not immediately generalize to more general arrival patterns, as we discuss below.

The asymptotic result in this section relied on finding two distinct weight functions, one for each

arrival time. As the number of distinct arrival times increases, it becomes unclear how to define an appropriate set of weight functions. We might require more than two weight functions, but there is no clear pattern that allows one to determine the structure of the weight functions to accommodate additional arrival times. Also, we ignored a few servers to derive the bounds, as they incurred an additional cost of 1 at each time step. If we generalize these results, this constant cost would extend over the span in the general case, leading to a higher competitive ratio, which we aim to avoid.

For the strict result which dealt with midterm arrivals, we had to do a careful case analysis based on the number of servers opened at times 1 and 2 of duration 1, as well as the number of servers opened at time 1 of duration 2. While it may be feasible to extend this analysis to three arrival times, the number of cases to consider would increase greatly, the details would be tedious and unlikely to yield any fresh insights. Furthermore, it would be impossible to extend in a straightforward way to an arbitrary number of arrival times.

Owing to the reasons explained above, extending our results to the general case of non-uniform durations and more arrival times would seem to need a fresh approach and new techniques, and we leave it as an open problem for future research.

Chapter 4

Equal Duration and Midterm Arrivals

In the previous chapter, we analyzed the *RSiC* problem under the assumptions of equal job durations and two distinct arrival times. Another setting that we considered was *midterm arrivals* under restricted version where we only have two arrival times. In this chapter, we continue studying the midterm arrivals setting but without any restriction on the number of arrivals. Table 4.1 provides details on the settings considered in this chapter.

First we consider the so-called *uniform servers* scenario in which all *FirstFit* servers have the same start and finish times; we establish an upper bound of $3/2$ on the competitive ratio of *FirstFit* for this case. Next, we examine the problem under the Dual Core input setting, where each server has two cores and can accommodate at most two jobs simultaneously. Specifically, we consider the scenario where all jobs have a uniform duration of 2, and integer arrival times. Despite these constraints, we demonstrate that *RSiC* remains unsolvable optimally in an online setting. Specifically, we prove that any online algorithm for this scenario has a competitive ratio of at least $5/4$. Moreover, we demonstrate that achieving a competitive ratio better than $9/8$ is impossible, even when leveraging sub-linear advice. In terms of positive results, we show that a tight bound of $5/4$ on the competitive ratio of any *AnyFit* algorithm in this setting, and we show that *RSiC* can be solved optimally offline for such inputs.

Job Duration	Arrival Pattern	# of Arrivals	# of Dimensions	# of Servers	Extra
Equal Duration	*Midterm*	*Arbitrary*	*1*	*Unlimited*	*Long Running*
Equal Duration	*Midterm*	*Arbitrary*	*1*	*Unlimited*	*Dual-Core*

Table 4.1: Overview of the settings for *RSiC* with equal duration and midterm arrivals.

4.1 Long-Running Uniform Servers

In this section we consider the case when all jobs in the input sequence σ have duration 2 and arrival times $0, 1, 2, \ldots, \ell$, and *FirstFit* packs these items into servers starting at time 0 and finishing at time $\ell + 2$. Thus, as we let ℓ goes to infinity, this setting represents "long-running" uniform servers of *FirstFit*. We show that asymptotically (as $\ell \to \infty$) such *FirstFit* servers have amortized load of $2/3$ at all times. We also observe that this bound is tight, i.e., there are inputs on which long-running *FirstFit* servers have load of roughly $2/3$ at all times. Thus, long-running servers are beneficial for *FirstFit* since load of $2/3$ translates to competitive ratio of $3/2$ when *FirstFit* cost is compared to the cost of OPT. This suggests that worst-case adversarial instances for *FirstFit* on equal duration jobs should be such that *FirstFit* servers are short-lived.

4.1.1 Upper Bound for Long-Running Uniform Servers

We begin with a few basic observations about long running servers in the next lemma, before giving our main result in Lemma 4.1.2.

Lemma 4.1.1. *Fix $\ell \in \mathbb{N}$. Let σ be such that all jobs in σ have duration 2 and arrival times $0, 1, 2, \ldots, \ell$ and let $\mathcal{B}$ be a set of FirstFit servers of duration exactly $\ell + 2$ that were opened at time 0. We denote the servers in $\mathcal{B}$ by $B_1, B_2, \ldots, B_k$ (opened in this order), where $k = |\mathcal{B}|$.*

For $i \in \mathbb{Z}_{\geq 0}$ we define the layer i, denoted by L_i, to be the set of all items packed in $\mathcal{B}$ that arrived at time i. Let $s(L_i)$ denote the size of all items in layer i. Assume that $k \geq 2$. Then we have:

(1) The cost of FirstFit arising from servers in $\mathcal{B}$ is $k(\ell + 2)$.

(2) $s(L_0) > k/2$;

(3) $s(L_i) + s(L_{i-1})/2 > (k-1)/2$ for $i \in \{1, \ldots, \ell\}$

Proof. (1) Immediate from the definition of the cost and the fact that each server in $\mathcal{B}$ has duration exactly $\ell + 2$.

(2) This is a standard argument about *FirstFit*. Let $\gamma_i = s(L_0 \cap B_i)$ denote the total size of items in server B_i that arrived at time 0. We have $\gamma_i + \gamma_{i-1} > 1$ for $i \in \{2, 3, \ldots, k\}$ since otherwise items in server B_i would have been placed in server B_{i-1} instead. Adding up all these inequalities we get:

$$\sum_{i=2}^{k} (\gamma_{i-1} + \gamma_i) > k - 1.$$

By adding γ_1 and γ_k to both sides, we obtain $2\sum_{i=1}^{k} \gamma_i > k - 1 + \gamma_1 + \gamma_k$. Observe that $\gamma_1 + \gamma_k > 1$ by the same reasoning as before. Moreover, we have $\sum_{i=1}^{k} \gamma_i = x(L_0)$ by definition. Combining all these facts, establishes this part of the lemma.

(3) Fix $i \in \{1, \ldots, \ell\}$. Let $\gamma_j = s(L_{i-1} \cap B_j)$ denote the total size of items in server B_j that arrived at time $i - 1$. Similarly, let $\delta_j = s(L_i \cap B_j)$ denote the total size of items in server B_j that arrived at time i. For $j \in \{2, 3, \ldots, k\}$ we have:

$$\delta_j + \gamma_{j-1} + \delta_{j-1} > 1$$

otherwise items packed into server B_j at time i should have been placed into server $j - 1$ by *FirstFit*. Summing all these inequalities we obtain:

$$\sum_{j=2}^{k} (\delta_j + \delta_{j-1} + \gamma_{j-1}) > k - 1$$

By adding $\delta_1 + \delta_k + \gamma_k$ to both sides we obtain:

$$2\sum_{j=1}^{k} \delta_j + \sum_{j=1}^{k} \gamma_j > k - 1 + \delta_1 + \delta_k + \gamma_k.$$

We have $\sum_{j=1}^{k} \delta_j = s(L_i)$ and $\sum_{j=1}^{k} \gamma_j = s(L_{i-1})$. Therefore, we obtain:

$$2s(L_i) + s(L_{i-1}) > k - 1 + \delta_1 + \delta_k + \gamma_k.$$

□

Now from the results in Lemma 4.1.1, we can establish strong upper bounds on the ratio between utilization and *FirstFit* cost, as follows.

Lemma 4.1.2. *Let σ be the input such that each job has duration 2 and arrival times $0, 1, 2, \ldots, \ell$ and suppose FirstFit opens $k \geq 2$ servers and each server of FirstFit on σ starts at time 0 and finishes at time $\ell + 2$. Then we have:*

$$\frac{util(\sigma)}{\textit{FirstFit}(\sigma)} > \frac{2}{3} - \frac{2}{3k} - \frac{2}{3(\ell+2)}.$$

where $util(\sigma)$ is the utilization of the input sequence which is defined as the total volume (size times duration) of all jobs.

Proof. Fix $\ell \geq 2$. From the results in Lemma 4.1.1, we have:

$$IN_i : s(L_{\ell-i}) + s(L_{\ell-i-1})/2 > (k-1)/2 : \qquad i \in \{0, 1, \ldots, \ell-1\}$$
$$IN_\ell : s(L_0) > k/2 > (k-1)/2 : \qquad o.w$$

We will choose multipliers $f_0, f_1, \ldots, f_\ell$ such that the linear combination of inequalities $\sum_{i=0}^{\ell} f_i IN_i$ has $util(\sigma)/2$

on the left hand side. The multipliers f_i are defined recursively:

$$f_i = \begin{cases} 1 & \text{if } i = 0, \\ 1 - f_{i-1}/2 & \text{if } i \geq 1. \end{cases}$$

Observe that the multipliers satisfy $f_i + \frac{1}{2} f_{i-1} = 1$ for $i \geq 1$. Next, we verify that $\sum_{i=0}^{\ell} f_i IN_i$ has $\sum_{i=0}^{\ell} s(L_i)$ on the left-hand side. For that it is convenient to think of inequality IN_ℓ as $s(L_0) +$

$s(L_{-1})/2 > k/2$, where we define $s(L_{-1})/2 = 0$.

$$\begin{aligned}\sum_{i=0}^{\ell} f_i(s(L_{\ell-i}) + s(L_{\ell-i-1})/2) &= \sum_{i=0}^{\ell} f_i s(L_{\ell-i}) + \frac{1}{2}\sum_{i=0}^{\ell} f_i s(L_{\ell-i-1}) \\ &= f_0 s(L_\ell) + \sum_{i=1}^{\ell} f_i s(L_{\ell-i}) + \frac{1}{2}\sum_{i=1}^{\ell} f_{i-1} s(L_{\ell-i}) \\ &= s(L_\ell) + \sum_{i=1}^{\ell} s(L_{\ell-i})\,(f_i + f_{i-1}/2) = util(\sigma)/2,\end{aligned}$$

where the last equality is because the sum of sizes is half of utilization, since all items have duration 2.

Let $F = \sum_{i=0}^{\ell} f_i$. To compute a bound on F observe the following:

$$\begin{aligned}\ell + 1 = f_0 + \sum_{i=1}^{\ell}(f_i + f_{i-1}/2) &= \sum_{i=0}^{\ell} f_i + \frac{1}{2}\sum_{i=0}^{\ell-1} f_i \\ = F + \frac{1}{2}(F - f_\ell) &= \frac{3}{2}F - \frac{1}{2}f_\ell.\end{aligned}$$

Thus, we can conclude that $F \geq \frac{2}{3}(\ell + 1)$. This implies that the right-hand side of $\sum_{i=0}^{\ell} f_i IN_i$ is $F(k-1)/2 \geq \frac{2}{3}(\ell+1)(k-1)/2$. Combining with the calculation of the left-hand side, we obtain:

$$util(\sigma) \geq \frac{2}{3}(k-1)(\ell+1).$$

Since *FirstFit*$(\sigma) = k(\ell + 2)$, we conclude:

$$\frac{util(\sigma)}{\textit{FirstFit}(\sigma)} > \frac{2}{3}\frac{(k-1)(\ell+1)}{k(\ell+2)} = \frac{2}{3}\left(1 - \frac{1}{k}\right)\left(1 - \frac{1}{\ell+2}\right) > \frac{2}{3} - \frac{2}{3k} - \frac{2}{3(\ell+2)}.$$

□

Corollary 4.1.3. *On the inputs described in Lemma 4.1.2 with $k, \ell \to \infty$ FirstFit achieves competitive ratio $3/2$.*

Proof. The upper bound on the competitive ratio follows from Lemma 4.1.2, since we have $OPT(\sigma) \geq util(\sigma)$ and $util(\sigma)/\textit{FirstFit}(\sigma) = 2/3 + o(1)$. □

4.1.2 Lower Bound for Long-Running Uniform Servers

The bound on utilization in Lemma 4.1.2 is tight asymptotically as $k, \ell \to \infty$. It is witnessed by the following example. Fix $k, \ell \in \mathbb{N}$ and $\epsilon = 1/(k\ell)$.

- at time 0 we present k items of size $2/3$ each;
- at odd times $1, 3, 5, \ldots$ we present k items of size $1/3$ each;
- at even times $2, 4, 6, \ldots$ we present k items of size $1/3 + \epsilon$ each.

Arrival time of the last job is ℓ in the above. Assume ℓ is even for simplicity of the presentation. As can be seen in Figure 4.1, *FirstFit* opens k servers of duration $\ell + 2$ each and utilization is $2(2/3)k + 2(1/3)k(\ell/2) + 2(1/3 + \epsilon)k(\ell/2) = (2/3)k(\ell + 2) + \epsilon k\ell = (2/3)k(\ell + 2) + 1$. Thus, for this example we have:

$$\frac{util(\sigma)}{\textit{FirstFit}(\sigma)} = \frac{(2/3)k(\ell+2)+1}{k(\ell+2)} = \frac{2}{3} + \frac{1}{k(\ell+2)} = \frac{2}{3} + o(1).$$

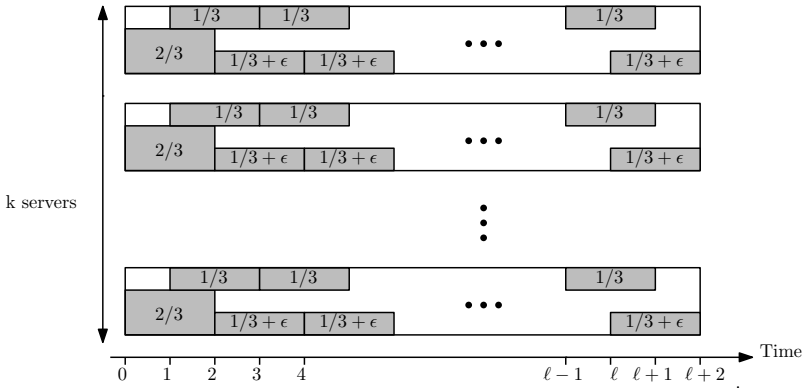

Figure 4.1: The bound in Corollary 4.1.3 is shown in a tight example.

4.2 Inputs for Dual-Core Servers

In the section[1], we focus on the *RSiC* problem in a dual-core server setup, where each server is equipped with exactly two cores, and each job requires one core to run. Consequently, a server can handle up to two jobs simultaneously. Therefore, we can say that each job has uniform duration, and the arrival times are spaced at exactly half the duration of items. For simplicity, we assume the durations of items are 2, and job arrival times are natural numbers. Let us first discuss the upper bound for this variant of the problem.

4.2.1 Upper Bound

We start with a simple proposition.

Proposition 4.2.1. *A solution to RSiC for input I in which at any time t, at most one server is half-empty is optimal.*

Proof. Suppose at time t, there is no server that is half-empty, that is, all servers are full. Then the number of items alive is even. If there is exactly one such server, then the number of alive items is odd. This implies that at any time t, the number of servers used is $\lceil s(\sigma, t) \rceil$ where $s(\sigma, t)$ is the sum of sizes of jobs that are alive at time t, which is a lower bound on the cost of a solution. Thus the solution must be opitmal. □

For the special case of input we consider, a server can contain at most two items. A *domino server* is defined as a server that has exactly two items placed in it at some t, and is closed immediately afterwards; that is, it accepts no more items. It remains active until time $t + 2$. A *chain server* is a server that has one job placed in it at every time step from some time t to time $t' \geq t$; since no items are placed in it at time $t' + 1$, it is released at time $t' + 2$. We say its start time is t and it remains active until time $t' + 2$. The *length* of a chain server is the number of items placed in it; for a chain server started at time t and closed at $t' + 1$, its length is $t' - t + 1$. It is clear that the cost of a domino server is 2 while the cost of chain server is $t' + 2 - t$.

[1]The results presented in this section are based on joint work with Dr. Shahin Kamali.

In Figure 4.4 (a), ALG has used a chain server of length 2 opened a time 0, and a chain server of length 1 opened at time 1, while in Figure 4.4 (b), OPT has used one chain server and one domino server.

We say an algorithm for *RSiC* forms a *domino* if at some time-step t it places two items into the same server to create a domino server. Alternatively, at any time t when a new job arrives, an algorithm may either *start a chain server* by placing a single job in a new server, or *extend a chain server* by placing a single job in an already existing chain server. We claim that forming a maximal set of dominos gives an optimal solution.

More precisely, for input sequence $(\sigma_1, \cdots, \sigma_n)$, with arrival times lying between 0 and k, assume $n_1, n_2, \cdots, n_k$ is the *cardinality sequence* where n_i is the number of items arriving at time i. We process items in increasing order of arrival times. At any arrival time i with $n_i > 0$, if n_i is even, then pack the items arriving at this time in pairs into domino servers. If n_i is odd, then use one job either to extend an existing chain server if one exists, or to start a new chain server if none exists. The remaining even number of items are packed into domino servers. Observe that this algorithm can be implemented as an online algorithm with a lookahead of one: it is enough to know when receiving an job whether or not there will be another job arriving at the same time.

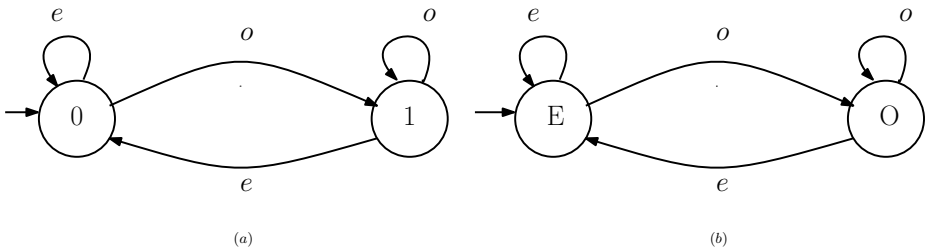

Figure 4.2: The change in number of active chains for (a) OPT. (b) ALG. The labels e and o on edges refer to even and odd n_i respectively. For OPT, the states correspond to the number of active chains, and for ALG, they correspond to the parity of the number of active chains c_i.

Observe that a new chain server is started at time i if n_{i-1} is even and n_i is odd; it is extended until we keep receiving an odd number of items, and the chain server is closed when we get an even number of items. See Figure 4.2 (a) for an illustration. Since there is at most one active chain server at any time, there is at most one server that has a single job packed in it. Optimality now follows from Proposition 4.2.1.

We call the above algorithm OPT. Denote an *AnyFit* algorithm by ALG. We proceed to show

that ALG has competitive ratio $\leq 5/4$.

Consider the cardinality sequence for an input I. Partition it into maximal sequences of non-zero integers separated by one or more zeroes. The items corresponding to these maximal subsequences are called *connected components* of the input. It is easy to see that for both ALG and OPT, the servers used for two connected components are disjoint and cannot overlap, that is, be active at the same time. Therefore, the cost of ALG (and OPT) is the sum of the cost paid by ALG (resp. OPT) for each connected component. Therefore, these connected components can be analyzed independently, and in the rest of this section, we consider only *connected* inputs in which all items of the cardinality sequence are non-zero, that is, there is at least one arrival in every step.

First we define a set of input sequences on which ALG is optimal. An input sequence is called an *ASC* if its cardinality sequence is an element of $(12+112)^*(1+11)$. For example, the cardinality sequence 112121121 corresponds to an *ASC* input. We say an input I is *NASC* if it is not *ASC*. A *NASC* input has a cardinality sequence that has at least one of the following properties: (a) has three or more consecutive 1s, (b) two or more consecutive 2s, (c) contains a number greater than 2 or (d) does not end with 1. For example, cardinality sequences 112, 12321, 122121, and 11121 are all cardinality sequences of *NASC* inputs.

Lemma 4.2.2. *ALG is optimal on connected ASC inputs.*

Proof. Let I be an *ASC* input with first arrival at time 1 and last arrival at time k. It is easy to see that ALG has one chain server starting at time 1, and active until time $k+2$. Note that except for time interval $[1, 2)$ $[k+1, k_2)$, this chain server always has 2 items in it. Also for each time when there are 2 arrivals, ALG opens a chain server with a single item, open for 2 time units. Since these times are within the time interval $[2, k+1)$, at all times, there is at most one server which is half-full. See Figure 4.3 for an example of the behavior of ALG on an *ASC* input. The lemma now follows from Proposition 4.2.1. □

Since ALG is optimal on *ASC* inputs, we turn our attention to *NASC* inputs. Suppose we have a chain formed by k items; then its cost is exactly $k+1$. Given that each domino has two items and cost 2, the cost of an algorithm with c chains for packing n items is exactly $n+c$. let C_{ALG} and C_{OPT} be the number of chains created by ALG and OPT respectively. Then on any input I with

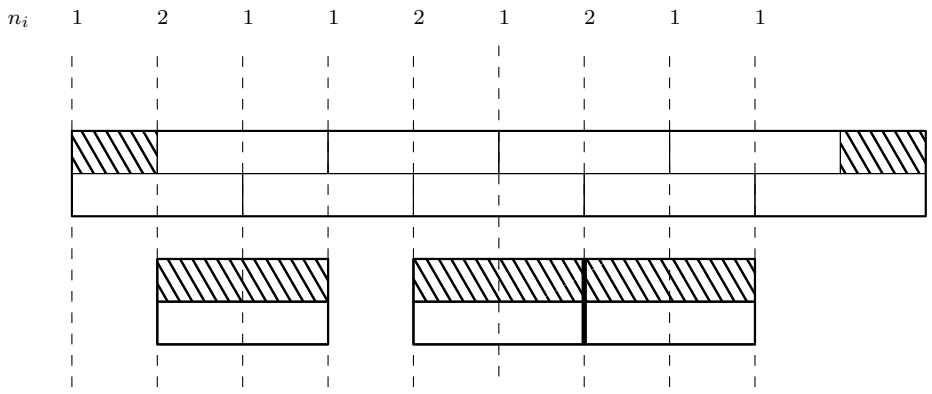

Figure 4.3: The assignment of ALG on the *ASC* input with cardinality sequence 121121211

n items,

$$\frac{ALG(I)}{OPT(I)} = \frac{n + C_{\text{ALG}}}{n + C_{\text{OPT}}} \tag{27}$$

In the lemma below we establish a relationship between the number of chains created by ALG and OPT.

Lemma 4.2.3. *$C_{FirstFit}(I) \leq 2C_{OPT}(I)$ for any connected NASC input I.*

Proof. As already described, at any time t, OPT has at most one active chain server. As shown in Figure 4.2(a), OPT creates a new chain server whenever the cardinality sequence switches from an even number to an odd number, and closes the chain server after the cardinality sequence switches from odd to even. On the other hand, we claim that ALG may create a new chain if the cardinality sequence switches from an even number to an odd number or vice versa, but never creates a new chain when there is no such switch.

Suppose ALG has c_i chains active at time i when n_i items arrive. We will show that (a) a new chain is not started if the parity of n_{i-1} is the same as the parity of n_i. We will also show inductively that (b) the parity of c_i is the same as the parity of n_{i-1}. This is clearly true at time 1, since there are 0 active chains, and $n_0 = 0$.

First we consider the case when the parity of n_i is the same as the parity of n_{i-1}, which by

inductive assumption equals the parity of c_i. If $c_i \geq n_i$, then n_i chains will continue and $c_i - n_i$ chains will be closed before time step $i+1$. Thus $c_{i+1} = n_i$, and they obviously have the same parity. Also note that no new chains were formed. So both (a) and (b) are true. If instead $c_i < n_i$, then all existing chains are extended, and the number of remaining items is $n_i - c_i$ which is even. ALG creates dominos with the remaining $n_i - c_i$ items and no new chains are created. Since $c_i + 1 = c_i$ which has the same parity as n_i, once again both (a) and (b) are true.

Next we consider the case that the parity of n_{i-1} and n_i are different, to finish the inductive proof of (b). As before, if $c_i \geq n_i$, no new chains are created, but n_i chains will continue, assuring that c_{i+1} has the same parity as n_i. Otherwise, all c_i chains are extended, and $n_i - c_i$ is odd, which means a new chain will be created, and so the parity of c_{i+1} is opposite that of c_i. By the inductive hypothesis, c_i and n_{i-1} have the same parity, which is opposite the parity of n_i in this case, thus assuring that c_{i+1} has the same parity as n_i. See Figure 4.2(b) for an illustration.

To summarize, OPT creates a new chain at time i if n_{i-1} is even and n_i is odd, while ALG may create a new chain if the parity of n_{i-1} and n_i are different. This shows that ALG starts a new chain at most twice as often as OPT, completing the proof of the lemma. □

Next we show that the number of chains created by OPT on a *NASC* input with n items is at most $n/3$.

Lemma 4.2.4. *$C_{OPT}(I) \leq n/3$ for any connected NASC input with n items.*

Proof. Let the cardinality sequence of the input I be $n_1, n_2, \ldots, n_k$, and assume that all n_i are positive. As mentioned earlier, OPT starts a new chain server when the cardinality sequence switches from an even number to an odd number. Let $j_1, j_2, \ldots, j_\ell$ be the set of times with $j_1 < j_2 < \cdots < j_\ell$ such that n_{j_i} is odd and $n_{j_i - 1}$ is even. Let S_i be the chain server opened by OPT at time j_i. For chain server S_i, we call items arriving between times j_i and $j_{i+1} - 1$ the *supporting items* of the server S_i, and let $W(S_i)$ be the number of its supporting items. That is,

$$W(S_i) = \begin{cases} \sum_{p=j_i}^{j_{i+1}-1} n_p & \text{for } 1 \leq i < \ell \\ \\ \sum_{p=\ell}^{k} n_p & \text{for } i = \ell \end{cases}$$

For example, for the cardinality sequence 12234111121, we see that OPT opens 4 servers at times 1, 4, 6, and 11. We have $W(S_1) = 5, W(S_2) = 7, W(S_3) = 6, W(S_4) = 1$. Observe that every job in the input is a supporting job of at most one server. If the cardinality sequence ends with an even number, for a chain server arriving at time j_i, there are an even number of items arriving in the time step $j_{i+1} - 1$ that are not part of the chain server, but are in its support set. Thus $W(S) \geq length(S) + 2$. If the cardinality sequence does not end with an even number, then this is still true for every chain server except the last server.

We say that the server S is *self-supporting* if $W(S) \geq 3$ and we call it *heavy* if $W(S) \geq 5$. If all chains created by OPT on I are self-supporting, then each chain has at least three distinct supporting items, and the lemma follows.

Suppose instead that there is a chain created by OPT that is not self-supporting. Then the chain is either of length 1 or 2, and corresponds to an arrival sequence 1 or 11, with no even number of items arriving afterwards. Such a chain can only be the last chain S created by OPT. But then we claim there must be at least one heavy chain. To see this, observe that if a chain is self-supporting but not heavy, it has weight either 3 or 4, in which case it corresponds to a cardinality subsequence 12 or 112. That is, if there are no heavy chains, the input must be of the form $(12 + 112)^*(1 + 11)$, a contradiction to the assumption that I is a *NASC* input. Thus there must be a heavy chain, which has a surplus of at least two supporting items that can instead support S. This completes the proof of the lemma. □

This brings us to the main result of this section.

Theorem 4.2.5. $\rho(ALG) = 5/4$.

Proof. For any input I, the cost of any algorithm is the sum of its cost on its connected components. Since by Lemma 4.2.2 ALG is optimal for *ASC* components, the worst-case ratio of the cost of ALG and OPT is achieved on an input with only *NASC* components. Using Equation 27, on any

such input I of n items, where ALG creates C_{ALG} chains and OPT creates C_{OPT} chains we have

$$\begin{aligned} \frac{\text{ALG}(I)}{\text{OPT}(I)} &= \frac{n + C_{\text{ALG}}}{n + C_{\text{OPT}}} && (28) \\ &\leq \frac{n + 2C_{\text{OPT}}}{n + C_{\text{OPT}}} && (29) \\ &\leq \frac{n + 2n/3}{n + n/3} = 5/4 && (30) \end{aligned}$$

where the last two inequalities follow from Lemmas 4.2.3 and 4.2.4 respectively. □

Corollary 4.2.6. *For any online algorithm ALG for RSiC on inputs where every job σ_i has size $1/3 < s(\sigma_i) \leq 1/2$, duration 2, and arrival time $i \in N$. Then $\rho(ALG) = 5/4$.*

Proof. The proof follows directly from Theorems 4.2.7 and 4.2.5. □

4.2.2 Lower Bound

Theorem 4.2.7. *Let ALG be an online algorithm for RSiC where all servers are dual-core servers, and jobs have duration 2, and arrival time $i \in N$. Then $\rho(ALG) \geq 5/4$.*

Proof. The adversary gives the first job σ_1 at time 0 and a second one at time 1. At the arrival time of job σ_2, if ALG packs σ_2 into the same server as σ_1, then ADV reveals the third job σ_3 also arriving at time 1, requiring ALG to open another server, see Figure 4.4. In this case, the cost incurred by ALG is 5, while OPT can pack σ_1 in one server and σ_2 and σ_3 together in another server, resulting in a total cost of 4. Thus, the ratio of the cost of ALG to the cost of OPT is $5/4$. If instead ALG packs the second job σ_2 into a new server, ADV ends the input, resulting in a total cost of 4 for ALG. Meanwhile, OPT can pack σ_1 and σ_2 into one server, resulting in a total cost of 3. Therefore, the ratio in this case is $4/3$. We conclude that $\rho(ALG) \geq 5/4$.

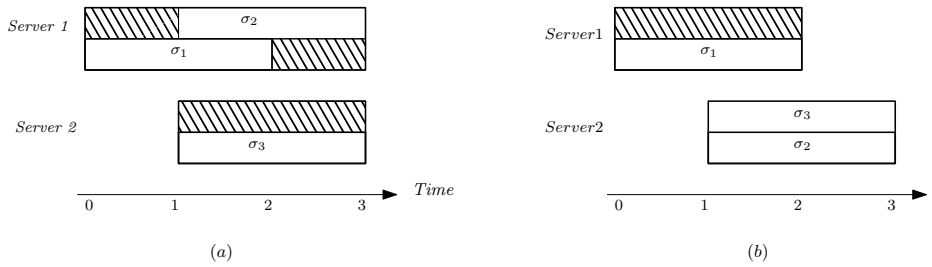

Figure 4.4: (a) ALG. (b) OPT. In this figures, the shaded areas indicate the wasted space within each server.

□

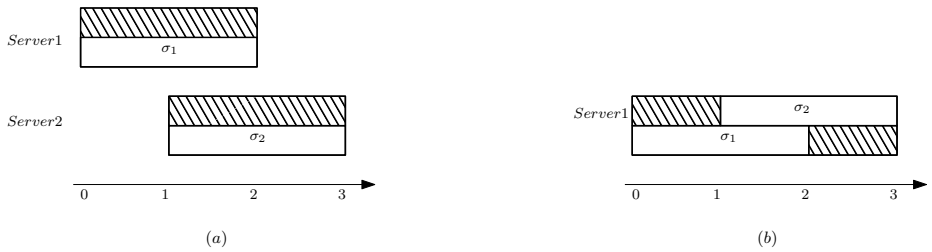

Figure 4.5: (a) ALG. (b) OPT. In the figures, the shaded areas indicate the wasted space within each server.

4.2.3 Lower Bound on Advice Complexity

In this thesis, so far we assume that no information about future items is available. However, this assumption is often unrealistic in real-world scenarios. In many cases, online algorithms have access to some information about future inputs. Models that try to capture such side-information are referred to as advice models. In this section, we consider the *RSiC* problem within a particular framework of the advice model. We begin with a short discussion of various advice models.

The advice model extends traditional online algorithms by providing additional information about future inputs, thereby relaxing the assumption of complete uncertainty. This extra information is provided by a trusted offline *oracle* with unbounded computational power, which has full knowledge of the input sequence. The oracle provides advice in the form of bits, creating a trade-off between the amount of advice received and the efficiency of the online algorithm. The main question in advice complexity is determining how much advice is necessary to improve the worst-case performance of an online algorithm. Specifically, we investigate the amount of advice required for an online algorithm to achieve optimal results or obtain a certain performance ratio. Depending

on the quantity of advice received, the algorithm may perform significantly better than an online algorithm without advice, potentially even optimally.

Several models of advice complexity exist for online problems. The first two, introduced by Dobrev et al. [24], are known as the *helper mode* and the *answer mode*. In the helper mode, the online algorithm receives a predefined number of advice bits—potentially zero—before processing each request. The advice complexity in this model is defined as the total number of bits required to achieve optimal performance. The answer mode operates similarly but allows the online algorithm to request advice bits as needed rather than receiving them in advance.

The model that became more standard (and the one we use in this thesis) is due to Böckenhauer et al. [10] and is called the *tape model*. In this model, the online algorithm has access to an infinite advice tape written by the oracle. The *oracle* has unbounded computational power and sees the entire input in advance. The oracle populates the infinite advice tape prior to algorithm receiving the first input item. At any point in time, the algorithm can consult the tape and read any number of advice bits (sequentially from the start of the tape). The advice complexity, as a function of input length n, in this model is measured by the total number of bits accessed by the algorithm on the worst-case input of length n. The oracle and the algorithm cooperate and act according to a pre-agreed protocol. The tape-advice model and the helper model are closely related. In fact, the advice complexity of an algorithm can differ by at most $O(\log n)$ between the two models. The tape-advice model became more standard, because it has a "cleaner" definition of advice length. Note that in the helper model, the oracle can send some additional information through the length of the advice string. For more details on advice complexity models, we refer interested readers to the survey by Boyar et al. [12].

In this section, we explore the *RSiC* problem within the tape-advice model. We aim to demonstrate that, even in the dual-core input setting— a highly relaxed version of the problem— it is impossible to achieve a competitive ratio better than $9/8$ with sub-linear advice. Our proof is based on a reduction from the *binary string guessing with known history (2-SGKH)* problem to this variant of the *RSiC* problem.

Let us first define the 2-SGKH problem:

Definition 4.2.1. Input: A binary sequence $x = (x_1, x_2, \cdots, x_n)$, where $x_i \in \{0, 1\}$ for $i =$

$1, 2, \ldots, n$.

Output: A sequence $y = (y_1, y_2, \ldots, y_n)$, where $y_i \in \{0, 1\}$, representing a guess of the input sequence.

Objective: Minimize the number of indices i such that $y_i \neq x_{i+1}$.

The following lower bound on the advice complexity of 2-SGKH problem is known [15]:

Theorem 4.2.8. *Fix $\alpha \in (1/2, 1]$. Any online algorithm for 2-SGKH that makes at most $(1-\alpha)n$ mistakes on inputs of length n needs at least $(1 - H(\alpha))n$ bits of advice, where $H(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \frac{1}{1-\alpha}$ is the binary entropy function.*

We are now ready to establish the main result of this section:

Theorem 4.2.9. *Fix $\alpha \in (1/2, 1]$. Any online algorithm for the variant of RSiC, where all servers are dual-core servers and jobs have duration 2 and integer arrival times, requires at least $\frac{1 - H(\alpha)}{3} N$ bits of advice to achieve competitive ratio $1 + \frac{1-\alpha}{4}$ on inputs of length N.*

Proof. We give an online reduction from the 2-SGKH problem to *RSiC*. Our reduction is gadget-based. At a high level, we define an algorithm for 2-SGKH based on an algorithm for *RSiC*. For each input x_i for the 2-SGKH, the reduction constructs a gadget – set of jobs to be scheduled on servers by the algorithm for *RSiC*. The gadget shall have the following properties:

(1) first part of the i^{th} gadget is given upon arrival of x_i; the behavior of the algorithm for *RSiC* is used to make a guess y_i;

(2) once x_{i+1} is revealed, the rest of the i^{th} gadget is revealed so as to achieve the following: if the guess was correct (i.e., $y_i = x_{i+1}$) then the algorithm for *RSiC* can schedule the entire i^{th} gadget optimally, and if the guess was incorrect (i.e., $y_i \neq x_{i+1}$) then the *RSiC* algorithm's decision on the first part of the gadget cannot be completed to an optimal solution once the second part of the gadget is revealed.

Thus, for all i, except $i = 1$, in step i the reduction algorithm gives a second part of the $i - 1^{\text{st}}$ gadget followed by the first part of the i^{th} gadget. Then we carefully relate parameters of the two

problems (2-SGKH and *RSiC*) to derive the desired result. The gadgets for different values of i do not overlap.

Next, we give the details of the above high-level plan. Suppose there is an online algorithm ALG with advice for the *RSiC* problem. Moreover, suppose that ALG achieves competitive ratio at most $1 + \frac{1-\alpha}{4}$ on inputs of length N, while using K bits of advice.

Consider input $(x_1, \ldots, x_n)$ to the 2-SGKH problem. We design an online algorithm $\widetilde{ALG}$ for 2-SGKH problem as follows. The algorithm shall maintain a variable t indicating current time in an instance for *RSiC*. Initially, t is set to 0. Upon arrival of input x_i for the 2-SGKH and assuming that $\widetilde{ALG}$ finished presenting the jobs for the $i-1^{\text{st}}$ gadget, $\widetilde{ALG}$ updates the time variable $t \leftarrow t+4$ and presents the first part of the i^{th} gadget which consists of two items to ALG: one with arrival time t, and another with arrival time $t+1$. Then, $\widetilde{ALG}$ observes how ALG packs these two items, and sets the guess y_i according to the following rules:

- If ALG places both items in the same server, then $\widetilde{ALG}$ sets $y_i = 0$.
- If ALG places the two items in separate servers, then $\widetilde{ALG}$ sets $y_i = 1$.

Once the value of x_{i+1} is revealed, $\widetilde{ALG}$ finishes the i^{th} gadget as follows:

- If $y_i = x_{i+1}$ then:
  - If $y_i = 0$ (i.e., ALG placed the two items on the same server), then the adversary does not reveal a new item, resulting in a cost of 3 for both ALG and OPT on the i^{th} gadget.
  - If $y_i = 1$ (i.e., ALG placed the two items in separate servers), then the adversary extends the previous gadget by revealing a new job at time $t+1$, resulting in a cost of 4 for both ALG and OPT on the i^{th} gadget (here, we assume without loss of generality that ALG schedules both items at time $t+1$ onto the same server).
- If $y_i \neq x_{i+1}$:
  - If $y_i = 0$, the adversary reveals an additional job at time $t+1$, resulting in a cost of 5 for ALG and 4 for OPT on the i^{th} gadget.

- If $y_i = 1$, the adversary does not reveal a new item, resulting in a cost of 4 for ALG and 3 for OPT on the i^{th} gadget.

Now for 2-SGKH, let us define the number of mistakes on 0s as m_0 (i.e., $y_i \neq x_{i+1} = 0$), the number of mistakes on 1s as m_1 (i.e., $y_i \neq x_{i+1} = 1$), and the number of correct guesses for 0s as c_0 (i.e., $y_i = x_{i+1} = 0$) and for 1s as c_1 (i.e., $y_i = x_{i+1} = 1$). Then the length of the instance for the 2-SGKH is $n = c_0 + c_1 + m_0 + m_1 + 1$. Also, observe that the length of the instance for *RSiC* is then $N = 2c_0 + 3c_1 + 3m_1 + 2m_0 = 2(c_0 + m_0) + 3(c_1 + m_1) \leq 3n$. Thus, we have:

$$n \geq N/3.$$

We can also calculate the cost of ALG and OPT on the *RSiC* instance (using the bullet-point discussion above):

$$ALG = 3c_0 + 4c_1 + 5m_1 + 4m_0$$

and

$$OPT = 3c_0 + 4c_1 + 4m_1 + 3m_0.$$

Since the competitive ratio of ALG is at most $1 + \frac{1-\alpha}{4}$, we obtain the following inequality:

$$\frac{3c_0 + 4c_1 + 5m_1 + 4m_0}{3c_0 + 4c_1 + 4m_1 + 3m_0} \leq 1 + \frac{1-\alpha}{4}.$$

Rearranging and simplifying the above expression we obtain:

$$\begin{aligned}
3c_0 + 4c_1 + 5m_1 + 4m_0 &< (1 + \frac{1-\alpha}{4})(3c_0 + 4c_1 + 4m_1 + 3m_0), \\
m_0 + m_1 &< (\frac{1-\alpha}{4})(3c_0 + 4c_1 + 4m_1 + 3m_0), \\
m_0 + m_1 &< (\frac{1-\alpha}{4})(4n), \\
m_0 + m_1 &< (1-\alpha)n.
\end{aligned}$$

In particular, it means that $\widetilde{ALG}$ makes at most $(1-\alpha)n$ mistakes. By Theorem 4.2.8, $\widetilde{ALG}$

must use at least $(1 - H(\alpha))n$ bits of advice. Note that the advice used by $\widetilde{ALG}$ is exactly the advice used by ALG in the reduction. Therefore, we conclude that

$$K \geq (1 - H(\alpha))n \geq \frac{1 - H(\alpha)}{3} N,$$

as desired. □

Corollary 4.2.10. *By taking $\alpha = 1/2$, the competitive ratio of RSiC cannot be better than $9/8$.*

4.3 Summary and Discussion

In this chapter, we addressed the *RSiC* problem under the specific scenario of equal job duration 2 and integer arrival times. For this setting, we established an upper bound for the *FirstFit* algorithm when all servers have the same start and finish times.

Subsequently, we further restricted the setting to the case where all servers are dual-core servers. For this variant, we established an upper bound of $5/4$ for the problem. Further, we demonstrated a matching lower bound of $5/4$ on the competitive ratio achievable by any online algorithm. Additionally, through a reduction from the binary guessing problem, we showed that with sub-linear advice bits, it is impossible to achieve a competitive ratio better than $9/8$.

There are many interesting research directions that the work in this chapter suggests. For example, one could try to extend the results to the k-core setting for $k \geq 3$. One could also consider arbitrary arrival times in the limited core setting and/or allow varying job durations. Studying different combinations of these extensions may lead to insights for the completely general setting. Moreover, there do not seem to be significant obstacles to analysis of each of these extensions individually, making them promising directions for future research.

Chapter 5

Restricted Number of Servers

In the previous chapters, the assumption is that there is an infinite number of identical servers available. In this chapter, we study the more realistic setting where there is a limited number of servers available. We consider a parameterized analysis, and consider families of inputs on which *FirstFit* needs at most k servers at any time, and provide tight bounds on the competitive ratio of *FirstFit* on such inputs. This type of parameterized analysis has the potential to provide insight into the original *RSiC* problem with no restriction on the number of servers. Seen in more practical terms, for the general input case, we analyze the version of *FirstFit* in which if none of the k available servers can accommodate the next job that arrives, then *FirstFit* simply discards the job. However, the *analysis* is restricted to the jobs accepted by *FirstFit*: we compare the rental cost paid by *FirstFit* with the optimal rental cost of serving only those jobs. We refer to this version of *RSiC* when there are only k servers available as k-*RSiC*. In effect, this restricts the space of input sequences to consist of only those inputs σ that satisfy *FirstFit*$(\sigma, t) \leq k$ for all t. More precisely, we consider the case where jobs have uniform durations but arbitrary arrival times, as shown in Table 5.1.

Algorithm	**Job Duration**	**Arrival Pattern**	**# of Arrivals**	**# of Dimensions**	**# of Servers**	**Extra**
FirstFit	*Equal Duration*	*Arbitrary*	*Arbitrary*	*1*	*Limited*	k *available servers*

Table 5.1: Overview of the settings for *RSiC* with equal duration and midterm arrivals.

The notions of competitiveness and competitive ratio extend naturally to restricted sets of inputs by requiring that the Inequality (1) holds for all input sequences σ from the restricted set only. We

denote the competitive ratio of *FirstFit* under the restriction of using at most k servers at a time by $\rho_k(\textit{FirstFit})$.

5.1 Theoretical Analysis of *FirstFit* for $k = 2, 3, 4$

The following simple upper bound will be used to establish tight competitive ratios for small values of k.

Lemma 5.1.1. $\rho_k(\textit{FirstFit}) \leq k$.

Proof. Consider an arbitrary input σ to k-*RSiC*. Due to the nature of the restriction, i.e., *FirstFit* using at most k servers at a time, we have: $\textit{FirstFit}(\sigma) \leq k \cdot span(\sigma)$. From Lemma 1.3.1 we have $OPT(\sigma) \geq span(\sigma)$. Combining these two observations proves the lemma. □

In the rest of this section we establish tight bounds on the competitive ratio of *FirstFit* for $k = 2, 3, 4$.

Theorem 5.1.2. $\rho_2(\textit{FirstFit}) = 2$.

Proof. The upper bound is given by Lemma 5.1.1. As for the lower bound, fix an arbitrary $n \in \mathbb{N}$ and sufficiently small $\epsilon > 0$ and $\delta > 0$, and define the input as follows. The first item is $(0, 1-\epsilon)$, and after that items arrive in n groups of 4. The i^{th} group (for $i \in \{1, \ldots, n\}$) consists of the following items:

$$((2i-1)-(2i-1)\delta, \epsilon), ((2i-1)-(2i-1)\delta, 2\epsilon), (2i-2i\delta, 1-\epsilon), (2i-2i\delta, \epsilon).$$

Figures 5.1 and 5.2 demonstrate this construction. Observe that *FirstFit* maintains two active servers: Server 1 has duration $(2-2\delta)n+1$ while Server 2 has duration $(2-2\delta)n+\delta$. Adding the cost of both servers, we obtain $\textit{FirstFit}(\sigma) = 4n-4\delta n+1+\delta$. Meanwhile, OPT always keeps one server active at a time, with small overlaps between the times they are open. When opened, each server remains open for duration exactly 1. Each of the groups costs 2, and the initial group costs 1. Thus, the overall cost of OPT is $OPT(\sigma) = 2n+1$. Combining these observations, we obtain:

$$\rho_2(\textit{FirstFit}) \geq \frac{\textit{FirstFit}(\sigma)}{OPT(\sigma)} = \frac{4n - 4\delta n + 1 + \delta}{2n+1} \geq 2 - \frac{1 - \delta + 4\delta n}{2n+1} \to 2 \text{ as } n \to \infty,$$

where the last inequality holds for $\delta = n^{-\alpha}$ with $0 < \alpha < 1$. □

We can also prove the above lower bound by using the following simpler construction: choose an arbitrary natural number n and set δ to be small enough, then define the input as follows: the first item is $(0, \frac{1}{2})$, followed by n groups of 4 items each. The i^{th} group, where i ranges from 1 to n, contains the following items: $((2i-1)-(2i-1)\delta, \frac{1}{2}), ((2i-1)-(2i-1)\delta, \frac{1}{2}), (2i-2i\delta, \frac{1}{2}), (2i-2i\delta, \frac{1}{2})$. It can be verified that the cost of *FirstFit* and OPT are the same as in the previous example. However, the ratio between *FirstFit* and volume is approximately 2, whereas in the example used for the proof of Theorem 5.1.2, this ratio is nearly 4. Thus the earlier example highlights the fact that utilizing total job volume as a lower bound for the cost of OPT is not a viable approach.

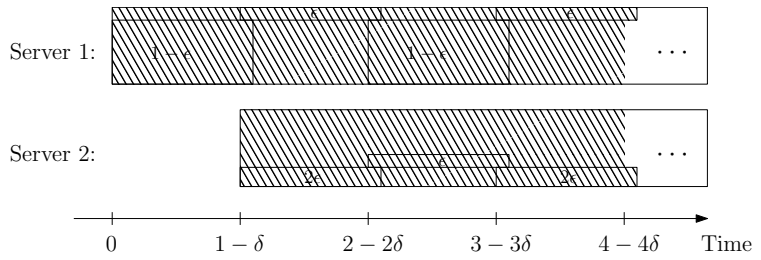

Figure 5.1: Schedule of *FirstFit* on the adversarial input demonstrating $\rho_2(FirstFit) \geq 2$.

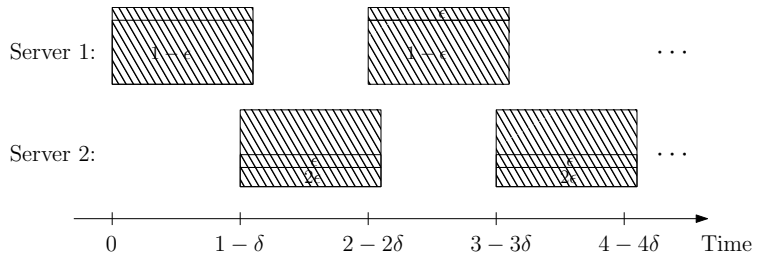

Figure 5.2: Schedule of OPT on the adversarial input demonstrating $\rho_2(FirstFit) \geq 2$.

Theorem 5.1.3. $\rho_3(\textit{FirstFit}) = 3$.

Proof. The upper bound is given by Lemma 5.1.1. For the lower bound, fix an arbitrary $n \in \mathbb{N}$

and sufficiently small $\epsilon > 0$ and $\delta > 0$, and define the input as follows. The input begins with four items: $(0, 1/3 - \epsilon), (0, 1/3 - 2\epsilon), (0, 2/3 - \epsilon), (0, 2/3 + 2\epsilon)$. *FirstFit* will place the first two items into the first server which will have occupancy $2/3 - 2\epsilon$, and the remaining two items will be placed into two additional servers. Then n groups of 3 items each follow. The i^{th} group items are the following:

$$(i - i\delta, 1/3 + \epsilon/2^{i-1}), (i - i\delta, 1/3 + \epsilon/2^{i-1}), (i - i\delta, 1/3 - \epsilon/2^{i-2}).$$

Consider how *FirstFit* behaves on the items for the first group. The first item has size $1/3 + \epsilon$ and arrives at time $1 - \delta$. The first server has enough remaining capacity to accommodate this item at time $1 - \delta$. The second item cannot be placed into the first server, since it does not have enough space at time $1 - \delta$, so it is placed into the second server, and similarly, the third item is placed into the third server.

Next, consider how *FirstFit* behaves on items for the second group. At the time of arrival of the second group items, i.e., $2 - 2\delta$, the occupancy in *FirstFit* servers is $1/3 + \epsilon, 1/3 + \epsilon$, and $1/3 - 2\epsilon$, respectively. Thus, the first item of size $1/3 + \epsilon/2$ is placed into the first server, the second item of size $1/3 + \epsilon/2$ is placed into the second server. Observe that this raises occupancy to $1/3 + \epsilon + 1/3 + \epsilon/2 = 2/3 + (3/2)\epsilon > 2/3 + \epsilon$ for the first two servers at time $2 - 2\delta$. Thus, when the third item arrives at time $2 - 2\delta$, there is not enough space in the first two servers to accommodate it (it has size $1/3 - \epsilon$), thus it is placed into the third server. It is easy to see by induction that this process continues for all the remaining groups. Thus, *FirstFit* maintains 3 servers open over the entire span, so *FirstFit*$(\sigma) = 3(n - n\delta + 1)$. As for OPT, it can begin by placing $\{(0, 1/3 - \epsilon), (0, 2/3 - \epsilon)\}$ items in one server, and $\{(0, 1/3 - 2\epsilon), (0, 2/3 + 2\epsilon)\}$ items into the second server at time 0. Observe that the items in each group add up to exactly one, so it can fit all items of a group into one server, which is open for a duration of 1. Thus, we have $OPT(\sigma) = 2 + n$. See Figures 5.3 and 5.4.

Combining the above, we obtain:

$$\rho_3(FirstFit) \geq \frac{\textit{FirstFit}(\sigma)}{OPT(\sigma)} = \frac{3n - 3n\delta + 3}{n+2} = 3 - \frac{3 + 3n\delta}{2+n} \to 3 \text{ as } n \to \infty,$$

assuming we take δ sufficiently small, e.g., $\delta = 1/n$.

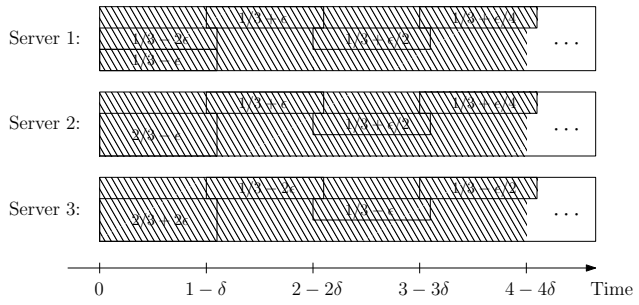

Figure 5.3: Schedule of *FirstFit* on the adversarial input demonstrating $\rho_3(FirstFit) \geq 3$.

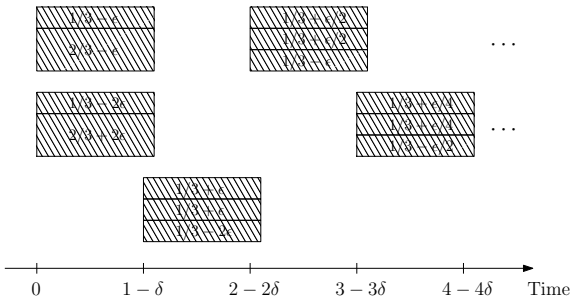

Figure 5.4: Schedule of OPT on the adversarial input demonstrating $\rho_3(FirstFit) \geq 3$.

□

We remark that the lower bound shown in the above theorem implies a lower bound for the general case of unrestricted inputs and number of arrivals.

Corollary 5.1.4. $\rho(\textit{FirstFit}) \geq 3$

The above bound is an improvement on the previously best known lower bound of ≈ 2.519 shown in Corollary 3.3.2 on the competitive ratio of *FirstFit*.

Finally we prove a tight bound on the performance of *FirstFit* for the case of 4 servers. Clearly the lower bound shown for 3 servers also applies to this case. To prove the upper bound, we use

a novel technique of partitioning the span into intervals based on the number of servers used by *FirstFit* and OPT. We then show that the total duration of intervals in which *FirstFit* uses 4 servers while OPT uses only one server cannot be that large.

Theorem 5.1.5. $\rho_4(\textit{FirstFit}) = 3$.

Proof. The lower bound follows from Theorem 5.1.3. For the upper bound, consider an input sequence σ of n items. We refer to an arrival or a departure of a job as an event. Without loss of generality, we can assume that all event times are distinct (otherwise, we can infinitesimally perturb arrival and departure times without any significant effect on costs of *FirstFit* and OPT). Let $t_1 \leq t_2 \leq \ldots \leq t_{2n}$ be the ordered sequence of times of all events. Define the duration between i^{th} and $(i+1)^{\text{st}}$ events by $\Delta t_i = t_{i+1} - t_i$.

Let $n_i \in \{0, 1, 2, 3, 4\}$ denote the number of servers active at time t_i in the *FirstFit* schedule , and let $m_i \geq 0$ denote the number of servers active at time t_i in OPT. Note that since there are no events between t_i and t_{i+1}, the *FirstFit* schedule has n_i active servers during the time interval $[t_i, t_{i+1})$, and OPT has m_i active servers in the same interval. Since servers are released as soon as all jobs assigned to them are finished, it follows that $n_i > 0$ if and only if $m_i > 0$ (if $n_i > 0$ or $m_i > 0$ then some job must be active during time interval $[t_i, t_{i+1})$. In addition, time intervals with $n_i = 0$ and $m_i = 0$ can be ignored since they do not affect the costs of *FirstFit* and OPT.

Let $D(j_1, j_2)$ denote the *total* duration of intervals i with $n_i = j_1$ and $m_i = j_2$. Using the introduced notation, we can express the costs of *FirstFit* and OPT as follows:

$$\textit{FirstFit}(\sigma) = \sum_{i=1}^{2n-1} n_i \Delta t_i = \sum_{j_1=1}^{4} \sum_{j_2 \geq 1} j_1 D(j_1, j_2)$$

and

$$OPT(\sigma) = \sum_{i=1}^{2n-1} m_i \Delta t_i = \sum_{j_1=1}^{4} \sum_{j_2 \geq 1} j_2 D(j_1, j_2).$$

To prove the theorem we need to establish that

$$\textit{FirstFit}(\sigma) \leq 3 \cdot OPT(\sigma)$$

which is equivalent to the following:

$$\sum_{j_1=1}^{4} \sum_{j_2 \geq 1} j_1 D(j_1, j_2) \leq \sum_{j_1=1}^{4} \sum_{j_2 \geq 1} 3j_2 D(j_1, j_2).$$

This inequality is, in turn, equivalent to:

$$\sum_{j_1=1}^{4} \sum_{j_2 \geq 1} (3j_2 - j_1) D(j_1, j_2) \geq 0.$$

Observe that

$$\begin{aligned}
\sum_{j_1=1}^{4} \sum_{j_2 \geq 1} (3j_2 - j_1) D(j_1, j_2) &= \sum_{j_1=1}^{4} (3 - j_1) D(j_1, 1) \\
&\quad + \sum_{j_1=1}^{4} \sum_{j_2 \geq 2} (3j_2 - j_1) D(j_1, j_2) \\
&\geq -D(4, 1) + \sum_{j_1=1}^{4} \sum_{j_2 \geq 2} D(j_1, j_2),
\end{aligned}$$

where the inequality follows by dropping non-negative terms from the first term (corresponding to $j_1 \in \{1, 2, 3\}$) and observing that $3j_2 - j_1 \geq 2$ for the values of j_1 and j_2 in the second term.

Thus, to prove the theorem it suffices to show the following inequality:

$$D(4, 1) \leq \sum_{j_1=1}^{4} \sum_{j_2 \geq 2} D(j_1, j_2) \tag{31}$$

The above inequality says that the total duration of intervals where *FirstFit* uses 4 servers while OPT uses 1 server is not much larger than the total duration of intervals where OPT uses at least 2 servers (regardless of how many servers are used by *FirstFit*).

Let $[i_1, i_2]$ be a maximal contiguous sequence of indices such that for all i with $i_1 \leq i \leq i_2$ we have $n_i = 4$ and $m_i = 1$. We claim that $m_{i_1 - 1} \geq 2$. If not, we would have $m_{i_1 - 1} = 1$ and since $[i_1, i_2]$ is maximal, we must have $n_{i_1 - 1} \leq 3$. As $n_{i_1} = 4$, this implies that i_1 is an arrival event which forced *FirstFit* to open a new server, which means the total size of items at time t_{i_1} exceeds 1, contradicting $m_{i_1} = 1$. Let i_0 be the smallest index below i_1 such that for all i with

Indices			i_0		i_1-1	i_1			i_2	
n_i	3	3	4	3		4	4	4	4	4
m_i	1	2	3	2		1	1	1	1	2

Table 5.2: A sequence of indices and the corresponding n_i, m_i pairs that illustrating i_0, i_1, i_2.

$i_0 \leq i < i_1$ we have $m_i \geq 2$. See Table 5.2 for an example illustrating i_0, i_1, i_2. We will show that $t_{i_2} - t_{i_1} \leq t_{i_1} - t_{i_0}$.

First, we show that $t_{i_2} - t_{i_1} \leq 1$. Suppose, for contradiction, that $t_{i_2} - t_{i_1} > 1$. Since all jobs have duration 1, all jobs that were active at time t_{i_1} must have finished by time t_{i_2}. However, since $n_{i_2} = 4$, all 4 servers are active at time t_{i_2}. This means that some job was scheduled by *FirstFit* onto server 4 between t_{i_1} and t_{i_2}. This implies that this job did not fit into server 1, which means that the total size of items at the time of arrival of this job exceeded 1. This contradicts the fact that $m_i = 1$ for all i with $i_1 \leq i \leq i_2$.

Second, we show that $t_{i_1} - t_{i_0} \geq 1$. Prior to time t_{i_0} OPT has one server, during time interval $[t_{i_0}, t_{i_1})$ OPT has at least two servers, and at t_{i_1} OPT has one server. If the only server in OPT that was active prior to time t_{i_0} is never released during $[t_{i_0}, t_{i_1})$ then we can immediately conclude that $t_{i_0} - t_{i_1} \geq 1$. This is because in this case some other server must have been opened and released between t_{i_0} and t_{i_1}, but such a server would have duration at least 1 – (the duration of every job in the input). Now, consider the case in which the one server active in OPT immediately prior to t_{i_0} is released some time during $[t_{i_0}, t_{i_1})$ while another server that was opened in OPT at time t_{i_0} remained active during the entire $[t_{i_0}, t_{i_1}]$ interval. Suppose for contradiction that $t_{i_1} - t_{i_0} < 1$. Let X_j denote the total size of jobs that were on server j in *FirstFit* immediately prior to t_{i_0} (note that some X_j may be 0 indicating that *FirstFit* was using less than 4 servers). Let Y_j denote the total size of jobs that were added to server j by *FirstFit* during time interval $[t_{i_0}, t_{i_1})$. Since OPT had only one server open immediately prior to t_{i_0} we have $\sum_j X_j \leq 1$. Since we assumed that $t_{i_1} - t_{i_0} < 1$ all jobs counted in the Y_j's are still active at time t_{i_1}. Since OPT has only one server open at t_{i_1} it implies that $\sum_j Y_j \leq 1$. Also, by the rules of *FirstFit* scheduling we have $Y_j + X_{j-1} + Y_{j-1} > 1$ (jobs in Y_j were not placed in server $j-1$). Note that we have 3 such inequalities (for $j \in \{2, 3, 4\}$) since $n_{i_1} = 4$. Adding these inequalities we obtain $2\sum_j Y_j + \sum_j X_j > 3$; however, this contradicts $\sum_j X_j \leq 1$ and $\sum_j Y_j \leq 1$ that we observed earlier.

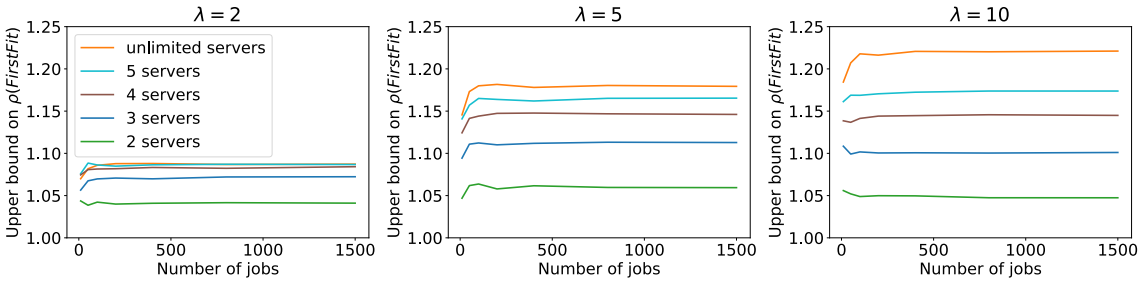

Figure 5.5: The effect of number of jobs on the performance of *FirstFit*. The performance of *FirstFit* is seen to be stable with at least 500 jobs for different choices of λ. The results are averaged over 100 trials.

This implies the inequality in Equation (31), since we established that any run of indices with $n_i = 4$ and $m_i = 1$ must be preceded by a run of indices (of at least the same total duration) with $m_i \geq 2$. This finishes the proof of the theorem. □

5.2 Experimental Results

In the previous section, we showed that in the worst case, *FirstFit* can have a cost that is twice the cost of OPT when using 2 servers, or even three times the cost of OPT when using 3 or 4 servers. However, these worst case inputs were carefully constructed, and are unlikely to occur in practice. In this section, we analyze the empirical performance of the *FirstFit* algorithm for *RSiC* and k-*RSiC* on *random* inputs. More specifically, we investigate *FirstFit* with respect to inputs generated by a random Poisson process with , *rejection sampling*, as described below. We consider the scenarios where *FirstFit* is limited to using $2, 3, 4, 5$ servers at a time, as well as where *FirstFit* has access to unlimited number of servers.

All our experiments were executed on a personal desktop with a quad-core 3.3 GHz Intel Core i3-3220 CPU. The desktop had 8 GB of RAM. The desktop was running Ubuntu OS version 22.04.01. The code was written in Python version 3.10.6.

Input Generation: We generate random inputs for our experiments based on the following parameters:

- $k \in \mathbb{N} \cup \{\infty\}$ is the number of servers available for *FirstFit*;
- $\lambda > 0$ is the average arrival rate in the first step of the random process (described below);

- $N \in \mathbb{N}$ is the target number of accepted jobs;
- $\ell, u \in [0, 1]$ with $\ell < u$ are the lower and upper bounds on sizes of jobs.

The random input is generated in two steps. In the first step we randomly sample a sequence of arrival times $a_1, a_2, \ldots$ by initializing $a_1 = 0$ and sampling inter-arrival times from an exponential distribution with parameter λ. Recall that the PDF of the exponential distribution is $f(x; \lambda) = \lambda e^{-\lambda x}$ for $x \geq 0$ and $f(x; \lambda) = 0$ for $x < 0$. Thus, higher values of λ result in shorter inter-arrival times. The resulting sequence of arrival times is a Poisson process, and this gives another interpretation for λ, namely, as an average number of arrival times in any time window of size 1. Since we consider the case of jobs of equal duration, we set the finishing times of jobs as $f_i = a_i + 1$. We assign each job size s_i to be an independently sampled uniformly random number from the range $[\ell, u]$.

In the second step of the process, we execute *FirstFit* on the input created in the first step. If a job does not require *FirstFit* to have more than k active servers at the time of job's arrival then this job is accepted; otherwise this job is rejected and removed from the input. We refer to this step as *rejection sampling*. The process continues until N jobs are accepted, which form a single random instance. Experimentally we found that generating $10N$ jobs in the first step was sufficient to guarantee that the second step succeeds (i.e., produces N accepted jobs) for all our experiments.

Performance Metric: Given parameters (k, λ, N, ℓ, u) of the experiment we repeat several trials (typically, 100 unless stated otherwise). In each trial, we generate a random input as described above and compute the cost of *FirstFit*. Since we do not have an efficient algorithm to compute OPT, we do not compute OPT exactly; instead, we use the lower bound from Proposition 1.3.4. Recall that the lower bound is ceiling of the total size of jobs active at time t. We average the cost of *FirstFit* over all trials, average the lower bound values over all trials, and report the ratio of the averages as an upper bound on the competitive ratio of *FirstFit*.

5.2.1 The Effect of the Number of Jobs

Since we are mainly interested in the asymptotic performance of *FirstFit*, first we wanted to establish a reasonable value of parameter N (the number of accepted jobs) to be used in our experiments. Having N too large would be computationally prohibitive, but having N too small could result in wrong interpretations of the asymptotic behaviour. We ran our experiments with values of N in $\{10, 50, 100, 200, 400, 800, 1500\}$ for three choices of λ, namely, $\lambda = 2$, $\lambda = 5$, and $\lambda = 10$. In this experiment we fix $\ell = 0$ and $u = 1$, therefore job sizes are uniformly distributed between 0 and 1. Figure 5.5 shows the results of this experiment.

Observe that the performance metric for *FirstFit* stabilizes at around $N = 500$ for all considered choices of λ. Thus, we decided to use $N = 500$ in all our remaining experiments in this section. The plots in Figure 5.5 exhibit several interesting patterns, which we discuss next.

First of all, observe that the upper bounds on competitive ratios in all the experiments and for all considered choices of k are quite close to 1 (and all of them are less than 1.25), indicating an excellent performance of *FirstFit* on these inputs. This is far better than the worst-case bounds of 2 (for $k = 2$) and 3 (for $k \geq 3$) that we obtained theoretically. This is not too surprising, since we had to carefully arrange arrival times and sizes of items to construct worst-case instances, and there is negligible probability of generating such patterns randomly. Perhaps, there are other worst-case inputs, but the empirical results demonstrate that random inputs are much nicer. It is worth noting that actual competitive ratio of *FirstFit* on these inputs can be even better than what is suggested by Figure 5.5, since we use a lower bound on OPT instead of the actual value of OPT. Understanding the gap between OPT and the lower bound on OPT is an important open problem, which needs further investigation.

Secondly, observe that for all the choices of λ considered in this experiment the performance of *FirstFit* is monotone with respect to k: having more servers leads to worse performance. This may seem counter-intuitive, but it is important to remember that inputs are also parameterized by k. Perhaps, it is more accurate to say that inputs which can be processed with fewer *FirstFit* servers tend to result in tighter schedules. Lastly, note that the gaps between the performance of *FirstFit* for different choices of k seem to increase with λ. The results of the experiments in the following

sections will show that as λ increases even more, the competitive ratio of *FirstFit* converges to 1 for k servers, for all $k \in \{2, 3, 4, 5\}$.

5.2.2 The Effect of the Job Arrival Rate

In the previous subsection we observed that the competitive ratio seems to increase with increasing values of λ. The natural question is how the performance of *FirstFit* with limited number of servers depends on λ and why. We ran the experiment with $N = 500$, $\ell = 0$, $u = 1$, and for all integral choices of λ between 1 and 100. Figure 5.6 shows the results of this experiment.

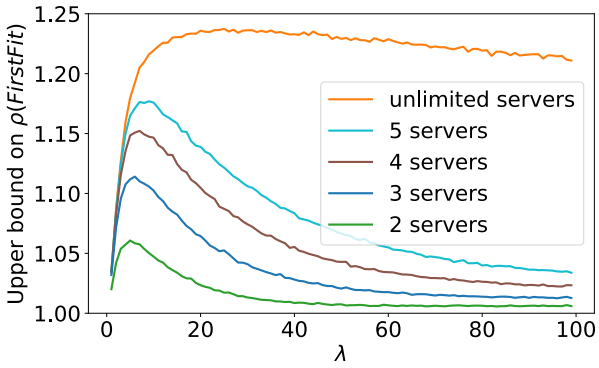

Figure 5.6: The effect of the job arrival rate on the performance of *FirstFit*. The total number of accepted jobs is 500. The results are averaged over 100 trials.

First, observe that all performance curves for various choices of k are uni-modular, i.e., the performance of *FirstFit* tends to get worse with increasing value of λ, reaches a peak, and then starts to improve. This suggests that for different choices of k there is a value of λ that results in the most difficult instances. Table 5.3 shows the experimentally derived worst values of λ for each value of k.

k	λ	$\rho(FirstFit) \leq$
2	5	1.06
3	6	1.11
4	7	1.15
5	9	1.18
∞	25	1.24

Table 5.3: Values of λ that result in worst performance for various values of k.

Figure 5.6 suggests that for every fixed value of k the competitive ratio of *FirstFit* tends to

1 as λ tends to infinity. Indeed, we conjecture this to be so. To understand the reason behind such a behavior of *FirstFit*, it is important to understand the effect of rejection sampling on the performance of *FirstFit*. Consider some fixed value of k and a large value of λ. Suppose that a large number of jobs have been generated according to the Poisson process of arrival times that are now being filtered through the rejection sampling phase. Let us denote the occupancy of i^{th} server of *FirstFit* at time t by $S_i(t)$. Observe that a job of size s arriving at time t is accepted if and only if $s \leq 1 - \min_{i \in [k]} S_i(t)$. Since job sizes are generated uniformly at random in the interval $(0, 1]$, the probability of this happening for a single job is $1 - \min_{i \in [k]} S_i(t)$.

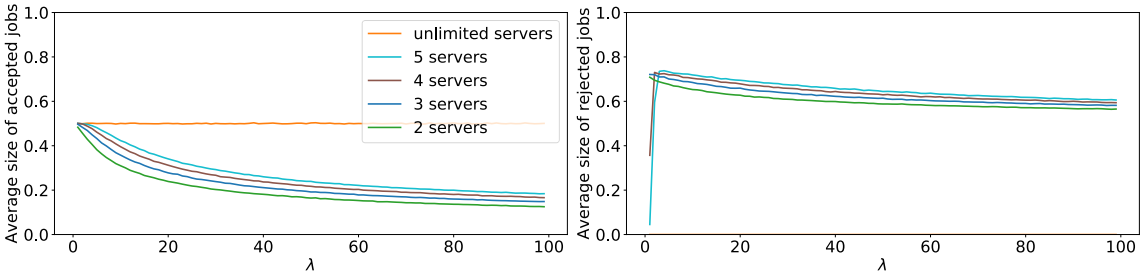

Figure 5.7: The effect of the job arrival rate on average size of accepted and rejected jobs. The total number of accepted jobs is 500. The results are averaged over 100 trials.

This means that as servers of *FirstFit* get filled in with jobs, there is a tendency to accept smaller jobs and reject larger jobs. As λ gets larger, the probability that some job is accepted during time interval $[t, t + dt]$ increases, which results in higher occupancy during this time interval, which in turn results in higher preference towards even smaller jobs. In the limit, taking $\lambda = \infty$ can be interpreted as having an infinite supply of jobs all arriving at time 0 and with job sizes uniformly distributed between 0 and 1. It is clear that when *FirstFit* is limited to k servers then all k servers would be completely full once the input passes through the rejection sampling stage. There is, of course, the question of whether the performance of *FirstFit* is continuous at $1/\lambda = 0$, and Figure 5.6 seems to suggest so. Another interesting question that is suggested by this plot is whether the case of *FirstFit* with unlimited number of servers tends to competitive ratio 1 or not. This question is left for future research.

In order to test our hypothesis that the average size of *accepted* jobs tends to decrease with increasing λ and for a fixed k, we plotted λ versus the average size of accepted jobs (as well as the average size of rejected jobs) in Figure 5.7.

Using *FirstFit* with a bounded number of servers also affects the *effective* arrival rate of the jobs. Recall that λ describes the arrival rate of jobs in the first step of the random input generation process. Since many of the jobs may be rejected, it is interesting to investigate what would be the actual arrival rate of jobs after the rejection sampling. We refer to this rate as the effective arrival rate, and we compute it empirically in the above experiment. The results are shown in Figure 5.8. The interpretation of this figure is as follows. Take for example $\lambda = 80$ and the case of $k = 5$ *FirstFit* servers. Then after the rejection sampling, on average there will be about 20 jobs with arrival time in any given time window of duration 1, there will be about 60 jobs rejected with arrival time in any given time window of duration 1. It would be interesting to be able to predict the effective arrival rate, given k and λ.

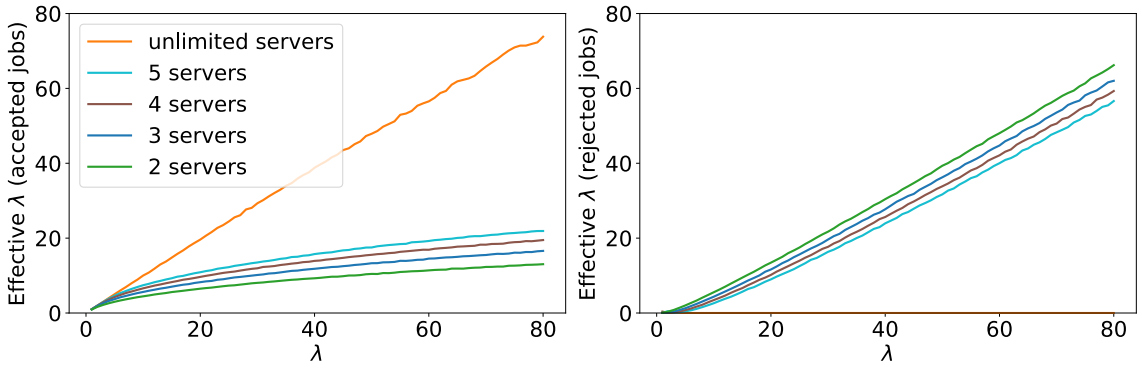

Figure 5.8: The effect of the job arrival rate on effective rate of accepted jobs and effective rate of rejected jobs. The total number of accepted jobs is 500. The results are averaged over 100 trials.

5.2.3 The Effect of Job Sizes

In this subsection we investigate how the interval $(\ell, u]$ from which job size is sampled uniformly at random affects the performance of *FirstFit*. We ran our experiments with $N = 500$ for three choices of λ, namely, $\lambda = 2$, $\lambda = 5$, and $\lambda = 10$, and for the following job size intervals: $(0, 0.1]$, $(0.1, 0.2]$, $(0.2, 0.3]$, $(0.3, 0.4]$, and $(0.4, 0.5]$. Note that if job sizes are restricted to $(0.5, 1]$ then *FirstFit* is optimal since both *FirstFit* and OPT must rent a new server for each incoming job, as no jobs of sizes greater than 0.5 that overlap in time can be scheduled on a single server. The results of our experiments are reported in Figure 5.9.

First, observe that in almost all plots the ordering of performance of *FirstFit* in terms of k is

consistent with what we found for the job size interval $(0, 1]$ in Section 5.2.1.

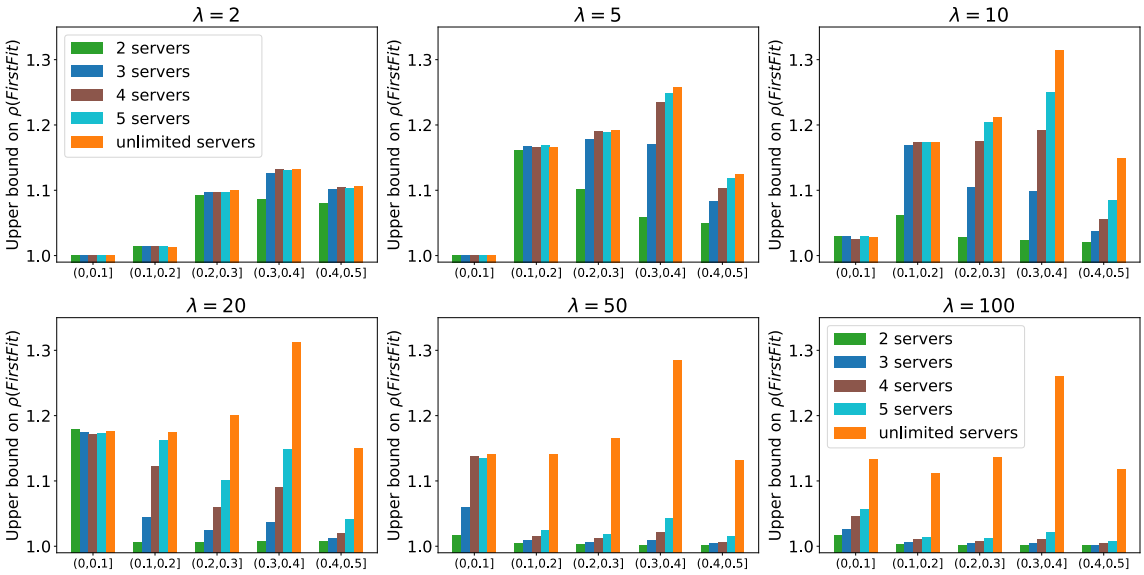

Figure 5.9: The effect of the interval $(\ell, u]$ from which job size is sampled on the performance metric of *FirstFit*. The total number of accepted jobs is 500. The results are averaged over 100 trials.

Secondly, for all ranges of item size, and for k servers for all values of $k \in \{2, 3, 4, 5\}$, the competitive ratio of *FirstFit* appears to converge to 1 as λ increases to 50 and 100. However, we argue below that the reasons for this are different for small and large item sizes.

For small job sizes, in the interval $(0, 0.1]$, when λ is very small (2 or 5) , all items can be accommodated on a single server, and when λ is large, the resulting schedule of *FirstFit* results in tightly packed servers nearly matching the volume of all jobs. The competitive ratio is seen to be very close to 1 in both situations, as seen in Figure 5.9.

For large job sizes, in the interval $(0.4, 0.5)$, both *FirstFit* and OPT can fit at most two items in a server at any time. How competitive *FirstFit* is depends on how well the two items in a server are aligned in terms of time. As λ increases, items arrive closer to each other, resulting in a more aligned packing of *FirstFit*. We observe from Figure 5.9 that for every value of $k \in \{2, 3, 4, 5\}$, the competitive ratio of *FirstFit* goes down with increasing λ.

Looking at each value of k individually, it is clear that there is a particular range of job sizes which results in the worst performance, but this range changes with λ. For example, when $k = 3$ and $\lambda = 2$, the upper bound on competitive ratio of *FirstFit* is largest for the job size interval $(0.3, 0.4]$, whereas for the same $k = 3$ and $\lambda = 5$, the upper bound on competitive ratio of *FirstFit* is largest

for the job size interval $(0.2, 0.3]$. As λ increases from 2 to 10, the gaps between performance of *FirstFit* for various values of k increase, and as λ increases even further, the gaps decrease, which is consistent with what we observed in previous sections.

It is interesting to note that the worst performance overall (among these experiments) is obtained by *FirstFit* with unlimited number of servers with job sizes in the interval $(0.3, 0.4]$, where the upper bound on competitive ratio is a bit over 1.3. This is worse than the performance of 1.24 achieved by *FirstFit* with unlimited number of servers when job sizes are in the interval $(0, 1]$. We hypothesize that this is because the schedule of *FirstFit* is quite sensitive to items of sizes close to $1/3$, which are essential to many worst-case theoretical constructions.

5.3 Summary and Discussion

In this chapter, we investigated the *RSiC* problem, focusing on minimizing the total cost of rented servers when assigning jobs of equal duration in a k-server setting. We established tight bounds on the worst-case competitive ratio of the *FirstFit* algorithm, proving a ratio of 2 for $k = 2$ and 3 for both $k = 3$ and $k = 4$, improving on the previous lower bound of approximately 2.518. Our experimental results revealed that, after 500 jobs, the competitive ratio stabilizes below 1.32 across all values of k and arrival rates λ. Initially, the ratio increases with λ before decreasing as λ grows, trending toward 1. Moreover, job size plays a significant role, with sizes in the range $[0.3, 0.4]$ yielding the highest competitive ratio.

A natural next step in the theoretical part of this chapter would be to extend the analysis to larger values of k. In the proof of $k = 4$, we had to show that the total duration of intervals where the *FirstFit* algorithm uses 4 servers while the OPT uses only 1 server is small. However, trying to extend this result to larger values of k, such as $k \geq 5$, is really challenging. For instance, when $k = 5$, we would need to show that the total duration of intervals where *FirstFit* uses 4 servers and OPT uses 1 server, plus twice the duration when *FirstFit* uses 5 servers and OPT uses 1 server, is not much larger than the total duration where OPT uses at least two servers. This is really difficult to show since we need to deal with many different cases, each corresponding to various job and server configurations. As a result, proving these results for larger values of k remains an open

problem for future research.

Chapter 6

Instance Incomparability of Some *RSiC* Algorithms

In the previous chapters, we analyzed the performance of algorithms using worst-case analysis. While some algorithms have the same worst-case performance, their behavior on specific inputs can vary significantly. In fact, even though their worst-case performances align, their results on particular instances may differ. In this chapter[1], we investigate this phenomenon for *RSiC*. The setting discussed in this chapter is outlined in Table 6.1.

Job Duration	Arrival Pattern	# of Arrivals	# of Dimensions	# of Servers
Arbitrary	*Arbitrary*	*Arbitrary*	*1*	*Unlimited*

Table 6.1: A summary of the *RSiC* setting for the instance incomparability of algorithms.

6.1 Instance Dominance and Instance Incomparability

Johnson, in his Ph.D. thesis [41], introduced an alternative approach by directly comparing the costs of solutions produced by two algorithms on the same input, without referencing the offline optimal solution. He evaluated algorithms such as *FirstFit*, *BestFit*, *Almost Worst Fit*, and *Next-K-Fit*. The *Almost Worst Fit* algorithm places a new item in the second emptiest open bin if possible. If the item does not fit there, it attempts to place it in the emptiest bin, and as a last resort, it opens

[1]The results presented in this section are based on joint work with Dr. Yaqiao Li.

a new bin. On the other hand, the *Next-K-Fit* algorithm modifies *NextFit* by keeping the last $K \geq 2$ bins open simultaneously, packing the item into the first bin where it fits.

Johnson demonstrated that while some of these algorithms have the same competitive ratio, their costs can vary significantly on certain inputs. To address this, he proposed evaluating the $\limsup$ of the cost ratios between two such algorithms, considering families of inputs $\{\sigma_n\}$ where the total item size grows without bound. Based on this analysis, he provided specific examples illustrating the following relationships:

$$\limsup_{n\to\infty} \frac{\textit{FirstFit}(\sigma_n)}{\textit{BestFit}(\sigma_n)} = \limsup_{n\to\infty} \frac{\textit{Almost Worst Fit}(\sigma_n)}{\textit{BestFit}(\sigma_n)} = \limsup_{n\to\infty} \frac{\textit{Next-K-Fit}(\sigma_n)}{\textit{BestFit}(\sigma_n)} = \frac{3}{2}, \quad \text{for } K \geq 2$$

and

$$\limsup_{n\to\infty} \frac{\textit{BestFit}(\sigma_n)}{\textit{FirstFit}(\sigma_n)} = \limsup_{n\to\infty} \frac{\textit{BestFit}(\sigma_n)}{\textit{Next-K-Fit}(\sigma_n)} = \limsup_{n\to\infty} \frac{\textit{BestFit}(\sigma_n)}{\textit{Almost Worst Fit}(\sigma_n)} = \frac{4}{3}, \quad \text{for } K \geq 2$$

Johnson conjectured that these values represent the maximum possible ratios between the respective pairs of algorithms. Recently, Levin [49] proved that these ratios correspond to the worst-case scenarios, and established this by proving matching upper bounds on the cost ratios of the respective algorithms, validating Johnson's conjecture. In this chapter, we aim to conduct a similar comparison of algorithms for the 1-dimensional *RSiC* problem. The algorithms discussed include *NextFit*, *FirstFit*, *BestFit*, *LastFit*, *WorstFit*, *MoveToFront*, as well as a new algorithm we propose, named *Greedy*. The *Greedy* algorithm orders servers in decreasing order of their *finishing times*, that is, the maximum of the finishing times of items currently in the server. It assigns the newly arrived item to the first server in the order that has sufficient capacity. If no such server exists, *Greedy* opens a new server and assign the item to it. A detailed analysis of this algorithm is presented in Section 7.4. Before we begin the analysis, we introduce the following key definition:

Definition 6.1.1. For a minimization problem, algorithm A is said to be *instance-dominate* algorithm B if:

$$\forall \text{ instance } \sigma : \ \text{cost}(A, \sigma) \leq \text{cost}(B, \sigma)$$

If neither A instance dominates B nor B instance dominates A, we say that A and B are

instance-incomparable.

6.2 Catalog of Instances

In this section, we present some examples and evaluate the performance of the algorithms on these cases. These examples serve as essential components for proving the general Theorem 6.2.1, which we will explore at the end of this chapter.

Instance A: Consider the input sequence $\sigma = \{\sigma_1, \cdots, \sigma_{2n}\}$. Assume that all items arrive at time 0 in their indexed order. For all σ_i where i is odd, the duration is 1, and for all σ_i where i is even, the duration is μ. The sizes are defined as follows: for every $1 \leq i \leq 2n$:

$$s(\sigma_i) = \frac{1}{2}, \quad \text{if } i \text{ is odd}$$

$$s(\sigma_i) = \epsilon, \quad \text{if } i \text{ is even}$$

This instance is also mentioned in [57].

Choose any ϵ satisfying $0 < \epsilon \leq 1/(4n)$ and let n be an odd number. A simple analysis shows that: For *FirstFit*, a new server is opened for σ_1. When σ_2 arrives, *FirstFit* assigns it to the first server. However, upon the arrival of σ_3, *FirstFit* needs to open a new server since there is not enough space in the first server. When σ_4 arrives, *FirstFit* starts checking the servers in the order they were opened and assigns σ_4 to the first available server that has enough capacity, which is the first server. Following this pattern, *FirstFit* assigns all σ_i where i is even, to the first server and places every two σ_i where i is odd into separate servers, as illustrated in Figure 6.1 part (a). The total cost of *FirstFit* on this input will be $(n-1)/2 + \mu$, since all servers except the first one have a duration of 1, and the first server has a duration of μ.

Similarly, *BestFit* will place all the even items into server 1, as it has the smallest available space. *Greedy* will follow the same strategy because, after item 2 is packed, the additional cost of placing the long-duration items becomes zero.

Now, let us examine the performance of *NextFit* on the same input. Recall that *NextFit* always keeps only one server open at a time. It opens the first server to serve item σ_1. When σ_2 arrives,

there is still enough space in the first server, so *NextFit* assigns it there. Upon the arrival of σ_3, the first server no longer has enough capacity, so *NextFit* closes it, opens a second server, and assigns σ_3 to it. Item σ_4 can also be assigned to this new server. This pattern continues for the rest of the input, with *NextFit* assigning each pair of items to a new server. As a result, each server will incur a cost of μ, and *NextFit* will open n servers in total—one for each pair. See Figure 6.1, part (b). Therefore, the total cost of *NextFit* for this input will be $n\mu$.

Now, let us consider how the other algorithms perform. Recall that *MoveToFront*, when assigning an item to a server, moves that server to the front of the list of servers. Therefore, *MoveToFront* assigns the first and second items to the first server. Upon the arrival of the third item, *MoveToFront* opens a new server to accommodate it and brings it to the front of the list. Consequently, item σ_4 will be assigned to this new server. This pattern repeats for the rest of the input, with *MoveToFront* assigning each pair of items to a new server. *LastFit* behaves similarly to *MoveToFront*, but with one key difference: *LastFit* does not move servers to the front. Instead, it assigns items to the most recently opened server. *WorstFit* assigns each item to the server with the most available space. So, when σ_4 arrives, *WorstFit* will assign it to the second server, as it has more available space than the first server. This pattern continues until the end of the input. Therefore, *MoveToFront*, *LastFit*, and *WorstFit* all have the same total cost as *NextFit*.

In summary, for instance A, the costs for different algorithms are as follows:

$$\textit{FirstFit}(A) = \textit{BestFit}(A) = \textit{Greedy}(A) = (n-1)/2 + \mu$$

while

$$\textit{MoveToFront}(A) = \textit{NextFit}(A) = \textit{LastFit}(A) = \textit{WorstFit}(A) = n\mu$$

Instance B: This instance is constructed in three stages. Assume the input sequence $\sigma = \{\sigma_1, \cdots, \sigma_{2n+1}\}$ is given. All items arrive at time 0, according to the order specified below.

In stage one, n items arrive, all have duration 1, and all have size $1-\epsilon$. In the second stage, a single item arrives, it has duration μ, and it has size 2ϵ. In the last stage, n items arrive, all have duration μ, and all have size ϵ. Choose any ϵ satisfying $0 < \epsilon < 1/(n+2)$.

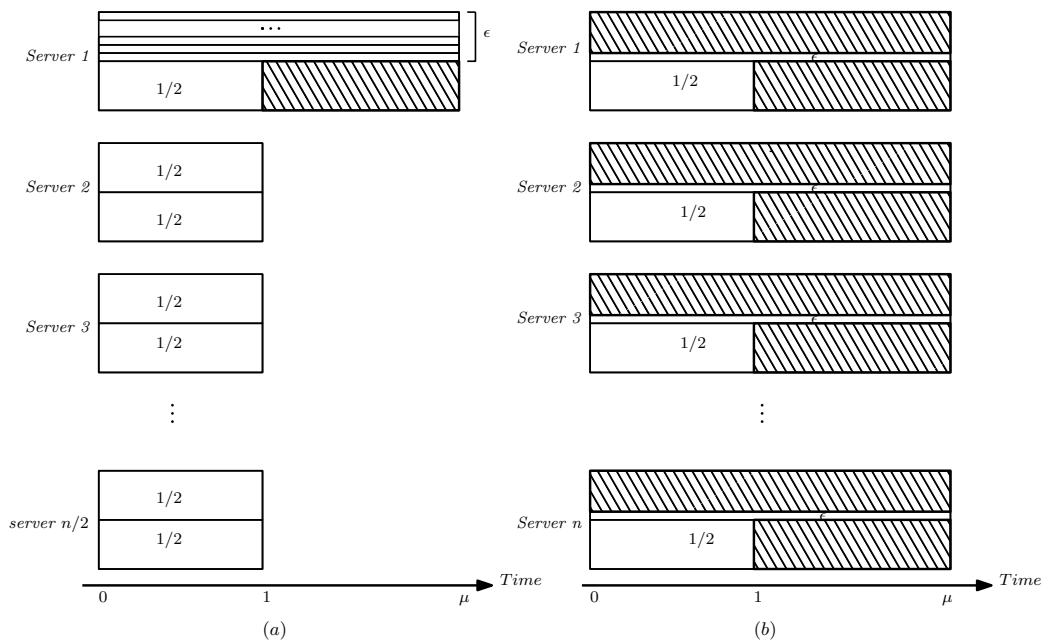

Figure 6.1: Example A and the assignment of (a) *FirstFit*, *BestFit* and Greedy. (b) *NextFit*, *MoveToFront*, *LastFit*, and *WorstFit*.

Let us analyze how *FirstFit* behaves on this input. In the first stage, *FirstFit* opens n servers, each with a duration of 1, to assign each item, as each item has a size of $1 - \epsilon$. In the second stage, *FirstFit* opens a new server because an item with a size of 2ϵ cannot fit in any of the previous n servers. In the third stage, *FirstFit* assigns each incoming item to the first available server that has enough space, extending the duration of the first n servers to μ by assigning one job to each, See Figure 6.2 part (a). Therefore, the total cost of *FirstFit* on this input is $(n + 1)\mu$. *BestFit* behaves similarly to *FirstFit* on this instance, assigning items in the third stage to the first n servers, as they have less available space compare to the $(n + 1)$th server.

Now, let us analyze the performance of the other algorithms on this example. First, consider the *Greedy* algorithm. In the first stage, *Greedy* assigns each item of size $1 - \epsilon$ to a new server, resulting in n servers, each with a duration of 1. In the second stage, *Greedy* opens a new server for the $(n + 1)$-th item, which has a size of 2ϵ and a duration of μ. Since this server has the longest finishing time among all the servers and still has enough remaining space, *Greedy* assigns all items from the third stage to this server. See Figure 6.2, part (b). Consequently, the total cost of *Greedy* for this input is $n + \mu$.

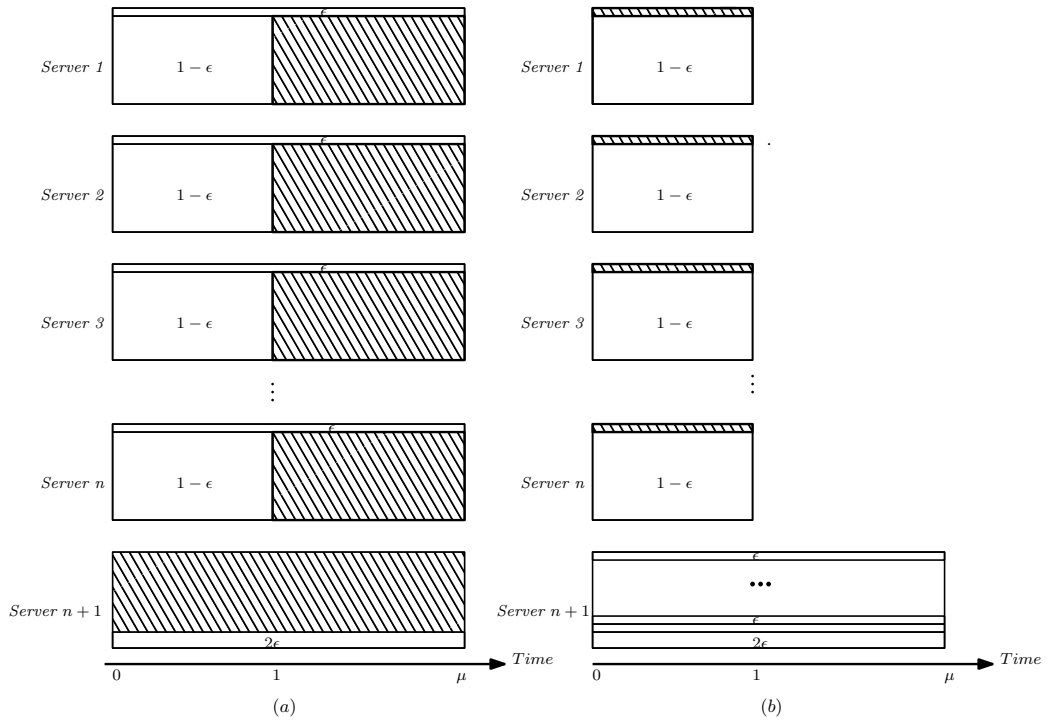

Figure 6.2: Instance B and the assignment of (a) *FirstFit* and *BestFit*. (b) *Greedy*, *MoveToFront*, *LastFit*, *WorstFit*, and *NextFit*.

The algorithms *MoveToFront*, *NextFit*, *LastFit*, and *WorstFit* exhibit the same assignment behavior. Since *LastFit* assigns the items from stage 3 to the $(n+1)$-st server, which is the least used server, it follows the same pattern. *MoveToFront* does similarly, as the $(n+1)$-st server is the one at the front of the list. *WorstFit* also places the items in the $(n+1)$-st server, since it has the most available space among all the servers, with only an item of size 2ϵ assigned to it. Finally, for *NextFit*, the $(n+1)$-st server is the only open server, as the rest of the servers have been closed. Therefore, for instance B:

$$\textit{FirstFit}(B) = \textit{BestFit}(B) = (n+1)\mu$$

while

$$\textit{MoveToFront}(B) = \textit{Greedy}(B) = \textit{NextFit}(B) = \textit{LastFit}(B) = \textit{WorstFit}(B) = n + \mu$$

Instance C: In this example, we aim to compare the performance of algorithms *FirstFit* and *BestFit*

in a specific scenario, that unfolds in two stages. An input sequence $\sigma = \{\sigma_1, \cdots, \sigma_{2n+1}\}$ is provided, with all items arriving at time 0 in a specified order.

During the first stage, $n+1$ items $\sigma_1, \ldots, \sigma_n, \sigma_{n+1}$ arrive, each with a duration of 1. The sizes are defined as follows: for $1 \leq i \leq n$, $s(\sigma_i) = 1 - 2^{-(n+1-i)}$, and $s(\sigma_{n+1}) = 1/2 + \epsilon$.

In the second stage, n items $\sigma_{n+2}, \ldots, \sigma_{2n+1}$ arrive, each with a duration of μ. The sizes are given by $s(\sigma_i) = 2^{-(i+1)} + \epsilon$; for $1 \leq i \leq n$. Here, ϵ is chosen such that $0 < \epsilon \leq (n+1)^{-1} \cdot 2^{-(n+1)}$. Thus the total input is:

$$\sigma = \{1 - \frac{1}{2^n}, 1 - \frac{1}{2^{n-1}}, \ldots, 1 - \frac{1}{2}, \frac{1}{2} + \epsilon, \frac{1}{2^2} + \epsilon, \frac{1}{2^3} + \epsilon, \ldots, \frac{1}{2^{n+1}} + \epsilon\}$$

For this instance, both *FirstFit* and *BestFit* assign the first $n+1$ items to $n+1$ new servers during the first stage. Upon the arrival of items in the second stage, *FirstFit* extends the first n servers to duration μ by allocating each server an item with size $\frac{1}{2^{i+1}} + \epsilon$, starting form the server n to server 1. Thus, the total cost for *FirstFit* is $n\mu + 1$. The same assignment applies to *Greedy* as well.

In contrast, *BestFit* attempts to assign second stage items to servers with the highest current load. Consequently, all second stage items can be accommodated in just one server which is the last server, resulting in a cost of $n + \mu$ for *BestFit* on this input instance. This same assignment strategy also applies to *LastFit*, *NextFit*, and *MTF*.

WorstFit follows a similar approach to these algorithms, with one key difference: it assigns the first item of the second stage to server $n-1$ and then places the remaining items in the last server. This results in a total cost of $(n-1) + 2\mu$.

In summary, for this input instance C, we have:

$$\textit{FirstFit}(C) = \textit{Greedy}(C) = n\mu + 1$$

while

$$\textit{BestFit}(C) = \textit{LastFit}(c) = \textit{NextFit}(c) = \textit{MoveToFront}(C) = n + \mu$$

and

$$\textit{WorstFit}(C) = (n-1) + 2\mu$$

To better understand this instance, let us consider a simple example.

Example 6.2.1. For $n = 5$, the input sequence σ is constructed as follows. At the first stage, $n + 1$ items arrive at time 0, and in the second stage, n items arrive, also at time 0. Thus, the sequence is given by:

$$\sigma = \{\frac{31}{32}, \frac{15}{16}, \frac{7}{8}, \frac{3}{4}, \frac{1}{2}, \frac{1}{2} + \epsilon, \frac{1}{4} + \epsilon, \frac{1}{8} + \epsilon, \frac{1}{16} + \epsilon, \frac{1}{32} + \epsilon, \frac{1}{64} + \epsilon\}$$

for some $0 < \epsilon \leq (n + 1)^{-1} \cdot 2^{-(n+1)}$. The first 6 items belong to the first stage with duration 1, and the remaining items are part of the second stage with duration μ.

For this small example, let us examine how *FirstFit* handles the packing of items. Initially, *FirstFit* opens 6 servers to accommodate the first 6 items in the first stage. When the first item in the second stage arrives, which has a size of $\frac{1}{4} + \epsilon$, the first server with enough capacity is server 5. Thus, *FirstFit* packs this item into server 5. The item with size $\frac{1}{8} + \epsilon$ is packed into server 4, as it is the first available server with sufficient capacity for this item. For the items $\frac{1}{16} + \epsilon$, $\frac{1}{32} + \epsilon$, and $\frac{1}{64} + \epsilon$, *FirstFit* assigns them to servers 3, 2, and 1, respectively. With these assignments, *FirstFit* extends the duration of all servers except the last one to μ, resulting in a total cost of $5\mu + 1$.

Now let us see how *BestFit* does the assignment for this example. *BestFit* opens 6 servers for the items in the first stage, similar to *FirstFit*. However, at the arrival of the first item in the second stage, an item with size $\frac{1}{4} + \epsilon$, *BestFit* tries to pack it into the server with the highest load that has enough capacity, which is the last server with a total load of $\frac{1}{2} + \epsilon$. Thus, *BestFit* assigns the item to server 6, and the load becomes $\frac{3}{4} + 2\epsilon$.

For the next item with size $\frac{1}{8} + \epsilon$, there are three potential servers to accommodate it: servers 4, 5, and 6. Among these, server 6 has the highest load; therefore, *BestFit* packs this item into the last server. This pattern repeats, and *BestFit* assigns all the items in the second stage into the last server, resulting in a total cost of $5 + \mu$.

Instance D: Consider the input sequence $\sigma = \{\sigma_1, \cdots, \sigma_{2n}\}$. Assume that all items arrive at time 0 in their indexed order. All σ_i where i is odd have a duration of μ, and all σ_i where i is even have a duration of 1. The sizes are defined as follows: for every $1 \leq i \leq 2n$,

$$s(\sigma_i) = 2\epsilon \quad \text{if } i \text{ is odd}$$

$$s(\sigma_i) = 1 - \epsilon \quad \text{if } i \text{ is even}$$

where ϵ satisfies $0 < \epsilon \leq \frac{1}{2n}$.

A simple analysis shows that *NextFit* opens a new server for σ_1. When σ_2 arrives, *NextFit* has to open a new server for it since there is not enough space in the first server, and then closes the first server. This pattern repeats for the entire input, causing *NextFit* to open a new server for each item, as *NextFit* keeps only one server open at a time, see Figure 6.3 part (a). The total cost of *NextFit* for this input will be $n\mu + n$.

Now, let us evaluate how other *AnyFit* algorithms handle this input, focusing initially on the behavior of *MoveToFront* for input D. *MoveToFront* opens a new server for σ_1 and another for σ_2, as the first server lacks sufficient space. Upon the arrival of σ_3, *MoveToFront* places it into the first server, where it fits, before opening a new server for σ_4. This pattern repeats: *MoveToFront* opens a new server for each even-indexed item σ_i while assigning all odd-indexed items σ_i to the first server, see Figure 6.3 part (b). This results in a total cost of $n + \mu$.

The same behavior applies to all other considered *AnyFit* algorithms, including *FirstFit*, *BestFit*, *LastFit*, *WorstFit*, and *Greedy*. All odd-indexed items are packed into the first server, while a new server must be opened for each even-indexed item, leading to a total cost of $n + \mu$.

In summary, for this input instance D, we have:

$$\textit{NextFit}(D) = n\mu + n$$

while

$$\textit{FirstFit}(D) = \textit{BestFit}(D) = \textit{MoveToFront}(D) = \textit{Greedy}(D) = \textit{WorstFit}(D) = \textit{LastFit}(D) = n+\mu$$

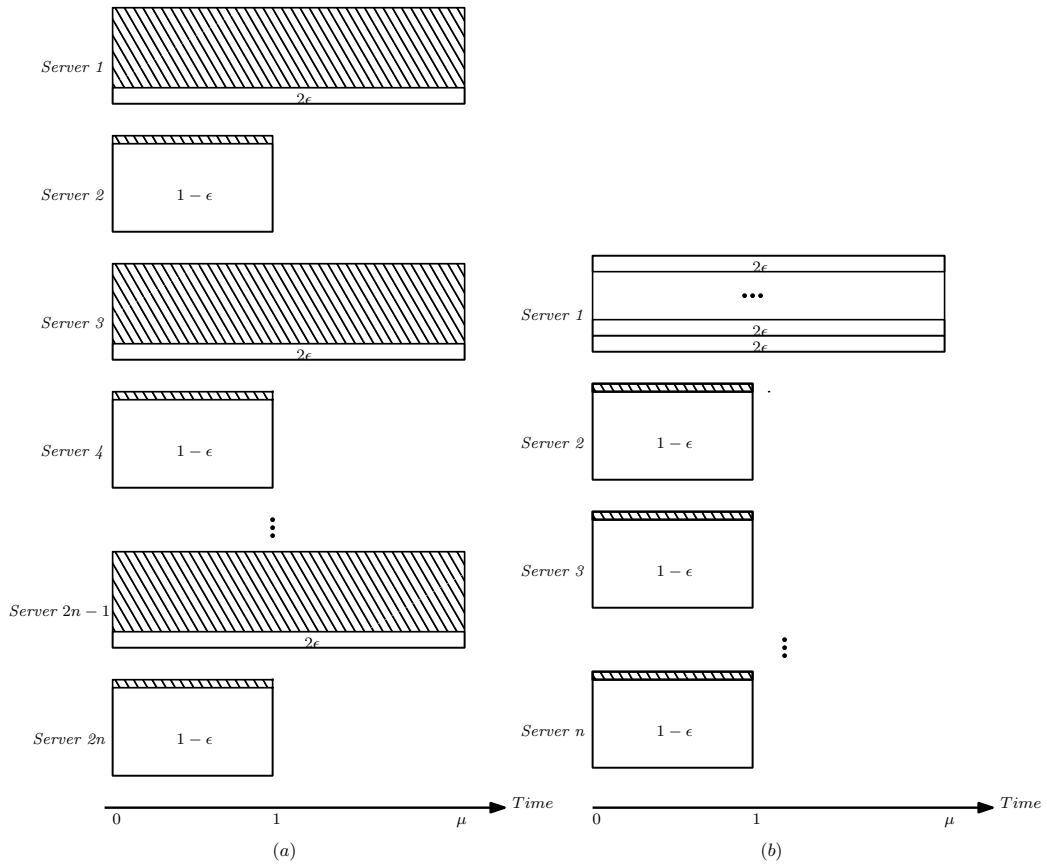

Figure 6.3: Instance D and the assignment of (a) *NextFit*. (b) All *AnyFit* algorithms.

Instance E: The scenario unfolds in two stages. The input sequence $\sigma = \{\sigma_1, \ldots, \sigma_{2n}\}$ with all items arriving at time 0 in a specified order. In the first stage, n items arrive. The first item σ_1 has a size of $1/2$ and a duration of 1. The next $n-1$ items each have a size of $1-\epsilon$ and also a duration of 1. In the second stage, another n items arrive. The first item in this stage has a size of 2ϵ and a duration of 1, followed by $n-1$ items, each with a size of ϵ and a duration of μ. Suppose ϵ satisfies: $0 < \epsilon \leq \dfrac{1}{2(n+1)}$.

Let us see how *LastFit* performs on this input instance. During the first stage, *LastFit* opens n servers for the first n items: the first server contains the item with size $1/2$, while the remaining servers each contain an item of size $1-\epsilon$. In the second stage, for the first item (of size 2ϵ), *LastFit* starts checking the servers from the most recently opened to the first to find a server with enough space. The only server with enough space to accommodate 2ϵ is the first server, so *LastFit* assigns this item there. For the remaining $n-1$ items (each of size ϵ), *LastFit* places each one in one

of the servers that already contain items of size $1 - \epsilon$. This extends the duration of each of these servers to μ, see Figure 6.4 part (a). Thus, the total cost of *LastFit* on this input will be $n\mu$. On this input, *BestFit* performs the same assignment. For the first-stage items, there is no alternative for any algorithm, as *LastFit*'s assignment is the only viable option. In the second stage, *BestFit* assigns the first item (size 2ϵ) to the first server, as it is the only server capable of accommodating it. For the remaining items in the second stage, *BestFit* assigns each to one of the servers containing an item of size $1 - \epsilon$, since these are the most loaded servers with exactly ϵ of available space. Thus, *BestFit* produces the same assignment as *LastFit*, resulting in a total cost of $n\mu$.

On the other hand, all other *AnyFit* algorithms among considered ones including *MoveToFront*, *WorstFit*, and *Greedy* perform on this input as following. Focus initially on the *MoveToFront* algorithm. In the first stage, *MoveToFront* behaves exactly like *LastFit*, opening n servers to assign the first n items. In the second stage, *MoveToFront* assigns the item of size 2ϵ to the first server because it is the only server that can accommodate this item. For the remaining items, since the first server is at the front of the list according to the *MoveToFront* algorithm, all the items will be assigned to this server by *MoveToFront*, see Figure 6.4 part (b). As a result, the total cost for *MoveToFront* is $(n - 1) + \mu$. The *FirstFit* and *WorstFit* algorithms also produce the same assignment. In the first stage, they do not have any option other than assigning items as *LastFit* does, opening n servers. In the second stage, *FirstFit* assigns the first item (size 2ϵ) to the first server, as it has sufficient space and is the first server in the list. The remaining items are also assigned to this server for the same reason. Similarly, *WorstFit* assigns the first item of the second stage to the first server because it has the most available space among all servers. The rest of the items in the second stage follow the same assignment pattern.

The *Greedy* algorithm behaves in a similar manner. When the first item of the second stage arrives, the first server still has enough space to accommodate it, so *Greedy* assigns it there. For the remaining items in this stage, the first server becomes the one with the longest duration, prompting *Greedy* to assign all these items to it as well. As a result, all these algorithms (*MoveToFront*, *FirstFit*, *WorstFit*, and *Greedy*) yield a total cost of $(n - 1) + \mu$.

The *NextFit* algorithm exhibits a slightly different behavior on this instance. After assigning the first item of size $1/2$, *NextFit* closes the first server and opens a new one to place the second item,

as the first server lacks sufficient capacity. Consequently, for the first n items in the initial stage, *NextFit* opens n servers. In the second stage, *NextFit* places all items into a single new server with a duration of μ, resulting in a total cost of $n + \mu$.

Therefore, for instance E:

$$\textit{LastFit}(E) = \textit{BestFit}(E) = n\mu$$

while

$$\textit{FirstFit}(E) = \textit{MoveToFront}(E) = \textit{WorstFit}(E) = \textit{Greedy}(E) = (n-1) + \mu$$

and

$$\textit{NextFit}(E) = n + \mu$$

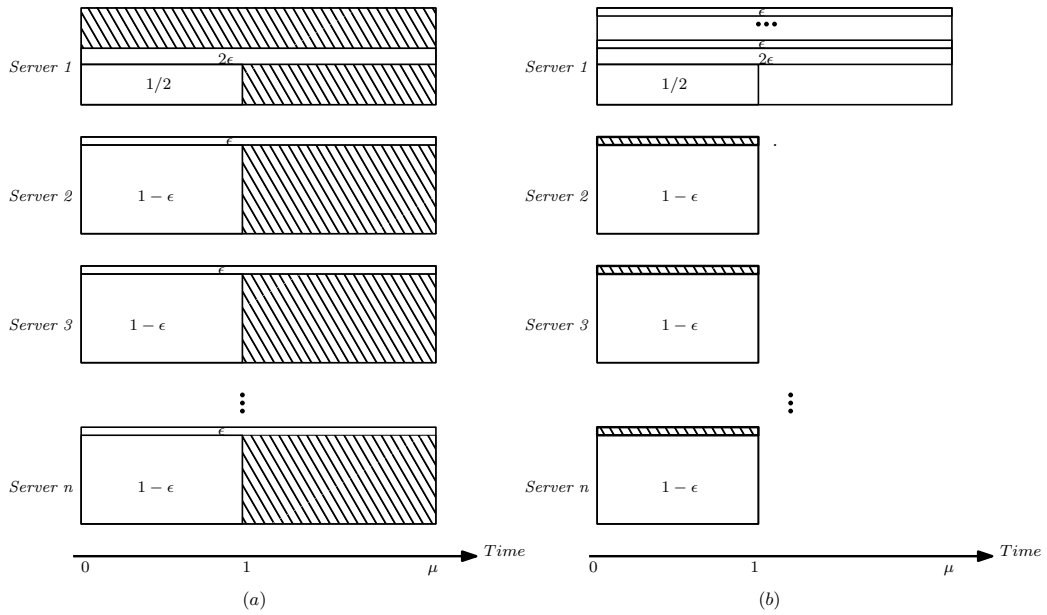

Figure 6.4: Instance E and the assignment of (a) *LastFit* and *BestFit*. (b) *FirstFit*, *MoveToFront*, *WorstFit*, *Greedy*. The assignment of *NextFit* is similar to part(b) with one extra server for all the items in the second stage.

Instance F: Consider the input sequence $\sigma = \{\sigma_1, \cdots, \sigma_{3n-1}\}$, which is constructed in two stages, with all items arriving at time 0. In the first stage, n items arrive, each with a duration of 1. Most items have a size of $1/2 + \epsilon$, except for the last item, which has a size of $2/3$. In the second stage,

$\sigma_{n+1}, \sigma_{n+2}, \cdots, \sigma_{3n-1}$ items arrive. Among these, the first $n-1$ items have a duration of 1 and a size of $1/2 - 2\epsilon$, while the last n items each have a duration of μ and a size of ϵ. Suppose ϵ satisfies $0 < \epsilon \leq \dfrac{1}{3n}$.

In this scenario, *LastFit* opens a new server for each item in the first stage, resulting in a total of n servers. When items arrive in the second stage, *LastFit* checks the servers from the most recently opened to the first, looking for available space. For item σ_{n+1}, *LastFit* finds that the second-to-last server (server $n-1$) has enough space and assigns the item there. The next item, σ_{n+2}, is assigned to server $n-2$. This pattern continues, till all the first $n-1$ items in the second stage, packed with items with size $1/2 + \epsilon$ in the first $n-1$ servers. Then *LastFit* assigns all the last n items each with size ϵ into the last server, see Figure 6.5 part (a). This results in a total cost of $n - 1 + \mu$ for *LastFit*. The same assignment is also applies to *WorstFit*, since at the arrival of the first n items in the second stage, the first $n-1$ servers are with the most available space compare to the last server. All the other considered *AnyFit* algorithms, including *MTF*, *FirstFit*, *BestFit*, and *Greedy*, behave differently on this instance. For example, consider *MTF*. The *MoveToFront* algorithm behaves the same as *LastFit* during the first stage and also for the first n items in the second stage. However, since *MoveToFront* brings the server where an item is placed to the front of its list of servers, the arrival of the last n items in the second stage (each with a size of ϵ) causes them to be packed into the first $n-1$ servers. As a result, the duration of these servers is extended to μ, while the last server remains unaffected. This leads to a total cost of $(n-1)\mu + 1$, as shown in Figure 6.5, part (b). The same assignment also works for *FirstFit*, *BestFit*, and *Greedy*.

The *NextFit* algorithm demonstrates slightly different behavior in this instance. During the first stage, *NextFit* also opens n servers to accommodate the items from this stage. However, since *NextFit* keeps only one server open at a time, it must close server n and open a new one upon the arrival of the first item in the second stage, which has a size of $1/2 - 2\epsilon$. Consequently, *NextFit* needs to open $n/2$ servers for the first n items in this stage. Additionally, for the final n items, each with a size of ϵ and duration μ, it must open one more server. This results in a total cost of $n + n/2 + \mu$.

Therefore, for instance F:

$$\textit{LastFit}(F) = \textit{WorstFit}(F) = n - 1 + \mu$$

while

$$\textit{MoveToFront}(F) = \textit{FirstFit}(F) = \textit{BestFit}(F) = \textit{Greedy}(F) = n\mu$$

and

$$\textit{NextFit}(F) = 3n/2 + \mu$$

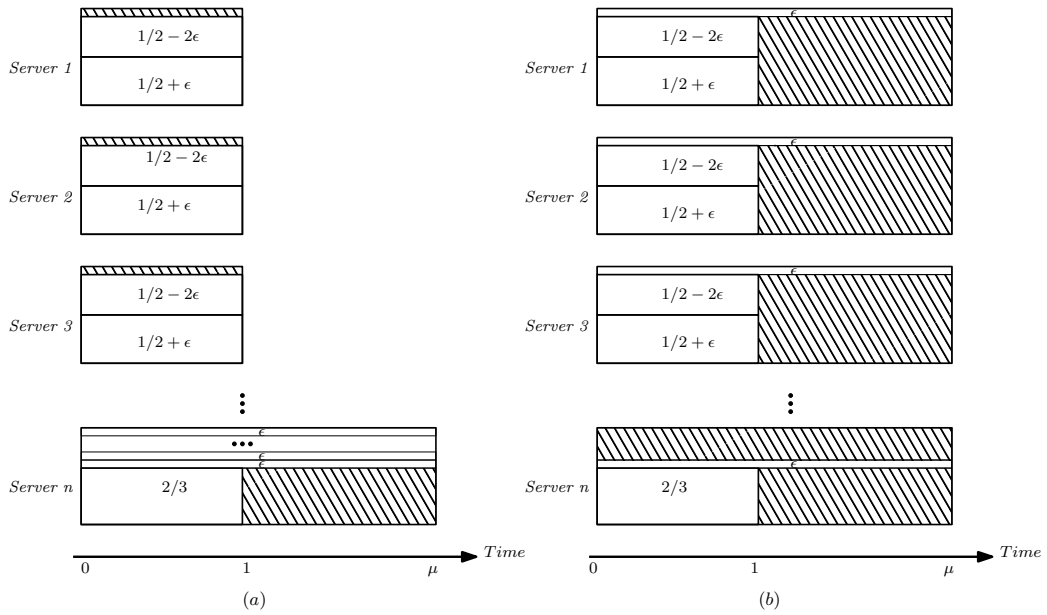

Figure 6.5: Instance F and the assignment of (a) *LastFit* and *WorstFit*. (b) *MoveToFront*, *FirstFit*, *BestFit*, and *Greedy*.

The *NextFit* algorithm exhibits a slightly different behavior on this instance. In the first stage *NextFit* also opens m servers to accommodate the items form the first stage. However, since *NextFit* only keeps one server open at each time, at the arrival of the first item in the second stage which is the item with size $1/2 - 2\epsilon$, *NextFit* has to close the server n and opens a new one. Thus, *NextFit* needs to open $n/2$ servers for the first n items in this stage, and for the last n items with size ϵ and duration μ it needs to open another server, which leads to a cost of $n + n/2 + \mu$ in total.

of size $1/2$, *NextFit* closes the first server and opens a new one to place the second item, as the

first server lacks sufficient capacity. Consequently, for the first n items in the initial stage, *NextFit* opens n servers. In the second stage, *NextFit* places all items into a single new server with a duration of μ, resulting in a total cost of $n + \mu$.

Instance G: This instance is constructed in three stages. Consider the input sequence:

$$\sigma = \{\sigma_1, \ldots, \sigma_{2n+1}\}$$

All items arrive at time 0, according to the following order. In stage 1, $n - 1$ items of size $1 - \epsilon$ all with duration 1 arrive. In stage two, three items σ_n, σ_{n+1} and σ_{n+2} arrive all have duration 1, and sizes are the following: $s(\sigma_n) = 1/2$, $s(\sigma_{n+1}) = 1 - \epsilon$, $s(\sigma_{n+2}) = 1/2$. In stage three, $n - 1$ items of size ϵ all have duration μ arrive where ϵ satisfies $0 < \epsilon \leq (2n)^{-1}$.

NextFit on this input instance performs as follows: In the first stage, it opens $n - 1$ servers, each with a duration of 1, for each item, and close each server before the next server is opened. For each item in the second stage, *NextFit* has to open a new server. All the items from the third stage could be fit in the last server, extending its duration to μ, see Figure 6.6 part (a). Thus, the cost of *NextFit* on this instance is $n + 1 + \mu$.

All the *AnyFit* algorithms among considered ones do the same assignment for this instance. Let us explain this assignment by seeing how *MoveToFront* does the assignment. *MoveToFront* acts similarly to *NextFit* in the first stage. However, in the second stage, *MoveToFront* places $s(\sigma_n)$ and $s(\sigma_{n+2})$ together in one server and item $s(\sigma_{n+1})$ in a separate server. Therefore, upon the arrival of items in the third stage, *MoveToFront* has to extend the duration of all the previous servers, except for one, to μ by assigning one item of size ϵ to each of them, see Figure 6.6 part (b). This results in a cost of $(n - 1)\mu + 2$. The same assignment applies to all *AnyFit* algorithms. Therefore, for instance G:

$$\textit{NextFit}(G) = n + 1 + \mu$$

while

$$\textit{FirstFit}(G) = \textit{BestFit}(G) = \textit{MoveToFront}(G) = \textit{LastFit}(G) = \textit{WorstFit}(G) = \textit{Greedy}(G) = (n-1)\mu+2$$

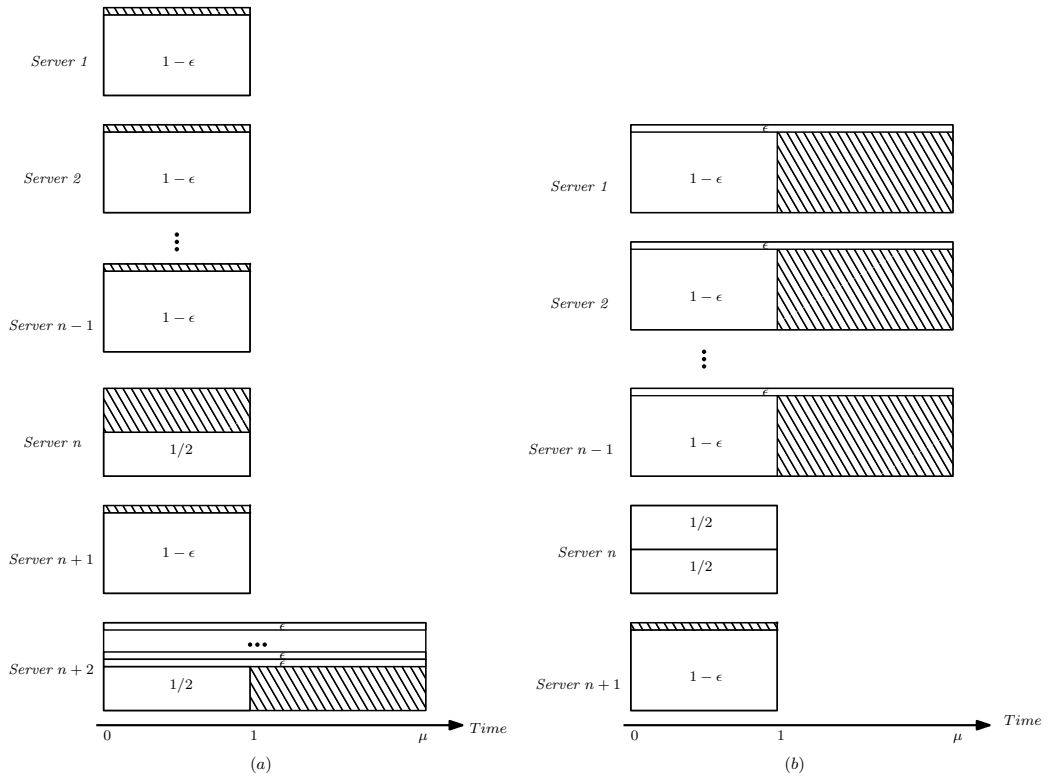

Figure 6.6: Instance G and the assignment of (a) *NextFit*. (b) All *AnyFit* algorithms.

Instance H: Consider the following instance, which demonstrates that while all other algorithms achieve a bounded competitive ratio, *WorstFit* exhibits an unbounded competitive ratio. This instance is constructed with k distinct phases. In the first phase, there are two distinct durations, while in subsequent phases, there is only one duration. Each phase has a single arrival time.

In the first phase, a sequence of items $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{2n}\}$ arrives at time 0 in the specified order. Each odd-indexed item, σ_{2i-1}, has a duration of 1 and a size of $1/2$. Each even-indexed item, σ_{2i}, has a duration of μ and a size ϵ_{2i}, where ϵ_{2i} (for $i \in \mathbb{N}$) are in increasing order and satisfy $\epsilon_{2i} < 1/2n$. At time 1, all items of size $1/2$ depart.

In the second phase, at time $\mu-\delta$ (for a small δ), a new sequence of items $\sigma = \{\sigma_{2n+1}, \sigma_{2n+2}, \cdots, \sigma_{3n}\}$ arrives. These items all have the same duration, μ, and sizes ϵ_i satisfying two conditions:

(1) ϵ_{2n+i} (for $i = 1, 2, \ldots, n$) are in increasing order.

(2) $\epsilon_{2n} < \epsilon_{2n+1}$, ensuring that the smallest items in this phase are larger than the largest items from the previous phase.

At time μ, the σ_{2i} items from the first phase depart.

In the third phase, at time $2\mu - \delta$, another sequence of items $\sigma = \sigma_{3n+1}, \sigma_{3n+2}, \ldots, \sigma_{4n}$ arrives. These items have the same duration, μ, and sizes ϵ_i satisfying the similar conditions as the previous phase:

(1) ϵ_{3n+i} (for $i = 1, 2, \ldots, n$) are in increasing order.

(2) $\epsilon_{3n} < \epsilon_{4n+1}$.

At time 2μ, the σ_{2n+i} items from the second phase depart. This process repeats for k phases.

Analyzing this setup reveals the inefficiency of *WorstFit*. Now, let us examine the details of how *WorstFit* assigns items in each phase.

In phase 1, *WorstFit* places the first two items into the first server. When the third item arrives, *WorstFit* must open a new server, as the third item cannot fit into the first server. Therefore, when the fourth item arrives, the second server has more available space than the first, so *WorstFit* assigns the new item to the second server. This pattern continues, with *WorstFit* using n servers to accommodate the items in phase 1, each with a duration of μ. At time 1, all the odd-indexed items from phase 1 depart.

Items from phase 2 arrive one by one at time $\mu - \delta$, each with a size ϵ_{2i} in increasing order. The first item is placed in the first server, as it has the most available space. After the first item is assigned, the first server no longer has the most available space, so *WorstFit* assigns the second item to the second server. This pattern repeats, and all items in phase 2 are assigned to servers, extending their duration by $\mu - \delta$. As a result, the duration of each server becomes $2\mu - \delta$. At time μ, all even-indexed items from phase 1 depart the system, leaving each server occupied by one item from phase 2.

At time $2\mu - 2\delta$, items from phase 3 arrive. Similar to phase 2, these items are assigned to servers, extending the duration of each server by $\mu - \delta$. This pattern continues, and *WorstFit* extends the duration of each server by approximately μ in each phase. Consequently, *WorstFit* uses n servers per phase, resulting in a total server time of $kn\mu$ across k phases, see Figure 6.7.

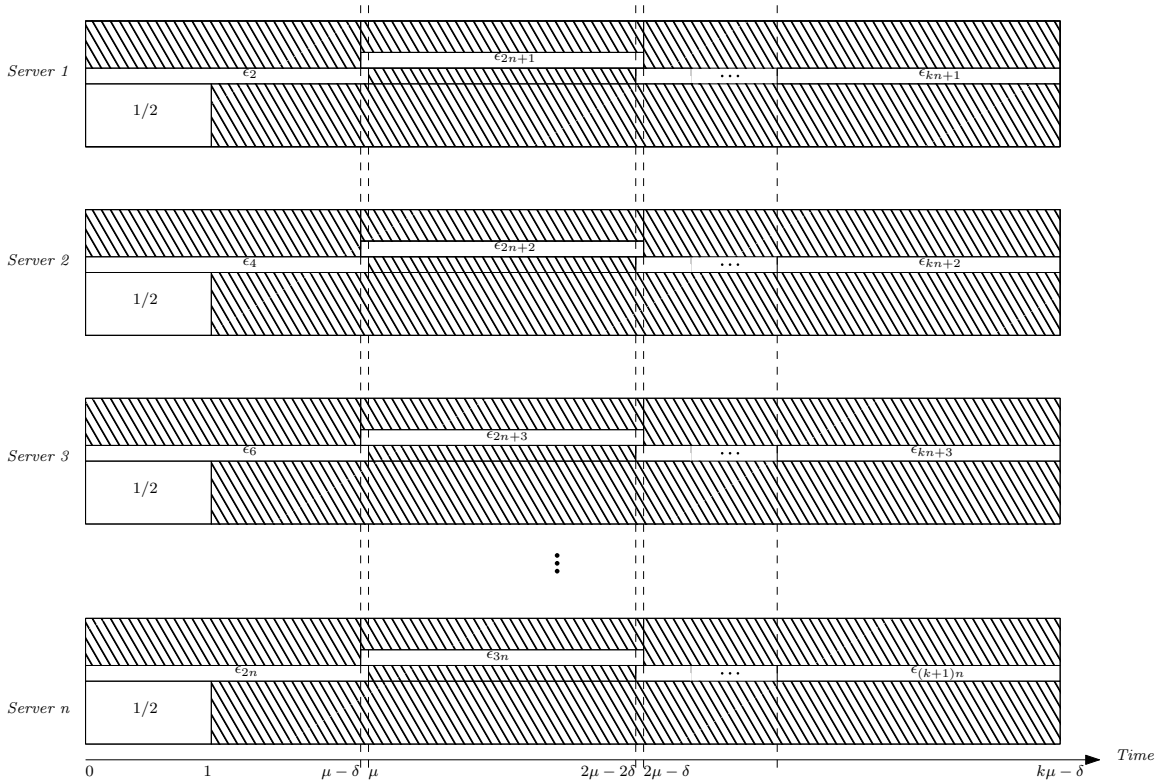

Figure 6.7: Instance H and the assignment of *WorstFit*.

In contrast, the other considered *AnyFit* algorithms, as well as *NextFit*, effectively pack items into fewer servers. For example, consider *BestFit*. During the first phase, all σ_{2i} items fit into the first server because their cumulative size does not exceed 1, given that each $\epsilon_{2i} < 1/2n$. Similarly, in each subsequent phase, all newly added items also fit into the same server. As a result, the total server time for *BestFit* is approximately $(n-1)/2 + k\mu$, as illustrated in Figure 6.8. This same assignment strategy applies to both *FirstFit* and *Greedy*.

On the other hand, *LastFit* behaves like *WorstFit* in the first stage. However, in the second stage, it places the remaining items into the last server, n, leading to a total cost of $(n-1) + k\mu$. This same assignment pattern is also observed with *MTF* and *NextFit*.

Therefore, for instance H:

$$\textit{WorstFit}(H) = kn\mu$$

while

$$\textit{FirstFit}(H) = \textit{BestFit}(H) = \textit{Greedy}(H) = (n-1)/2 + k\mu$$

and

$$\textit{MoveToFront}(H) = \textit{LastFit}(H) = \textit{NextFit}(H) = (n-1) + k\mu$$

As n grows arbitrarily large, the competitive ratio of *WorstFit* compared to other algorithms approaches infinity, demonstrating the unbounded nature of *WorstFit*'s performance on this instance.

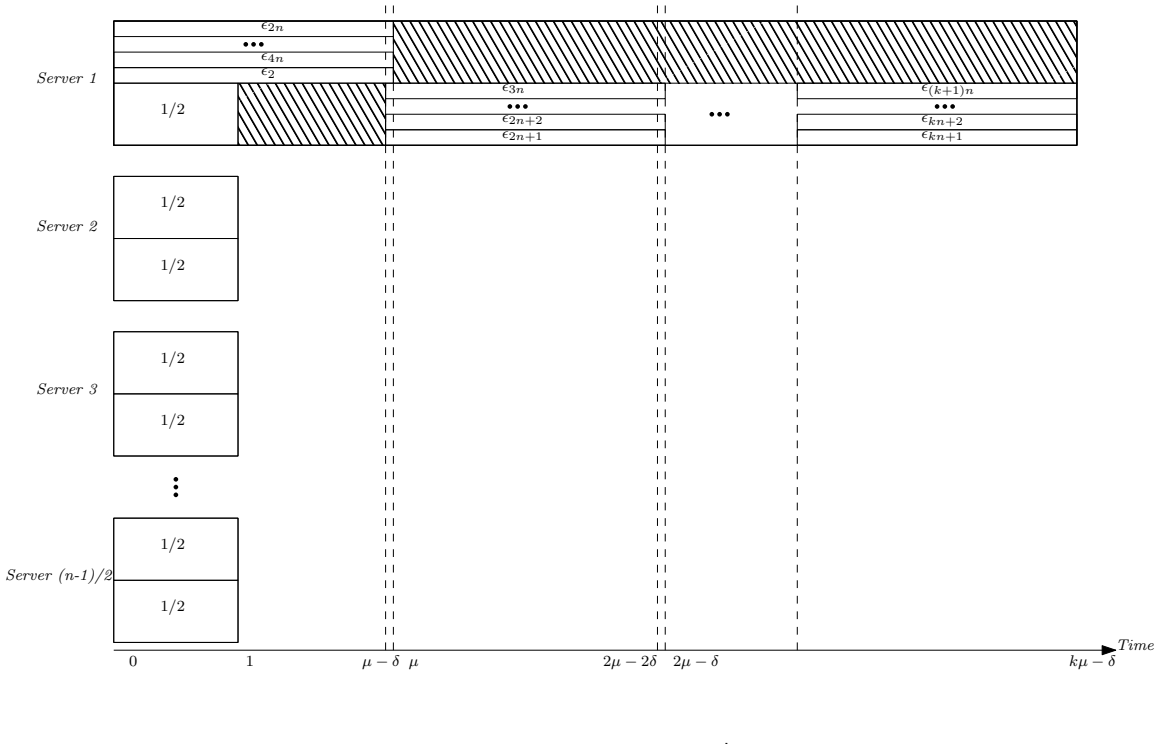

Figure 6.8: Instance H and the assignment of *BestFit*, *FirstFit*, and *Greedy*. Note that *LastFit*, *MTF*, and *NextFit* follow a similar structure; the key difference is that they keep the last server open to accommodate jobs from stage 2 and beyond.

Instance I: This instance demonstrates that *BestFit* has an unbounded competitive ratio. The instance unfolds over multiple stages as follows:

At time 0, a sequence of $2n$ items $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{2n}\}$ arrives in a specific order, odd-indexed items σ_{2i-1} each have a duration of 1 and a size of $1 - \epsilon_i$. And all even-indexed items σ_{2i} each have:

- Duration between $\mu - \delta$ and μ, for small δ.

- Sizes ϵ_i satisfying $\epsilon_{2i} > \epsilon_{2(i+1)} > \ldots > \epsilon_{2n}$ (i.e., sizes decrease with increasing index).

At time 1, all odd-indexed items depart since they have duration 1. In the second stage, within the time interval $[\mu - \delta, \mu]$, a new sequence of items $\sigma = \{\sigma_{2n+1}, \sigma_{2n+2}, \ldots, \sigma_{3n}\}$ arrives one after another. These items have the following properties:

- Duration between $\mu - \delta$ and μ.
- Sizes $\epsilon_i < 1/n^2$ that satisfies:

  (1) $\epsilon_{2n} > \epsilon_{2n+1} > \epsilon_{2n+2} > \ldots > \epsilon_{3n}$.

  (2) $\epsilon_{2n-2} > \epsilon_{2n} + \sum_{i=2n+1}^{3n} \epsilon_i$.

During this stage, each new item ϵ_{2n+i} is placed in bin B_i. The previous item in B_i (which arrived before time $\mu - \delta$) departs before the next item in the sequence arrives.

In the time interval $[2\mu - \delta, 2\mu]$, a third sequence $\sigma = \{\sigma_{3n+1}, \sigma_{3n+2}, \ldots, \sigma_{4n}\}$ arrives. These items have durations between $\mu - \delta$ and μ, and their sizes ϵ_i adhere to the following condition: $\epsilon_{3n} > \epsilon_{3n+1} > \epsilon_{3n+2} > \ldots > \epsilon_{4n}$. Similar to the previous stage, after each item ϵ_{3i} is placed, the old item in bin B_i (the item that arrived before time $2\mu - \delta$) departs before the next item arrives. This pattern repeats for k stages.

This example highlights the inefficiency of the *BestFit* algorithm. In the first phase, *BestFit* opens n servers and assigns pairs of items to each server. As a result, at time 1, when the odd-indexed items depart, the servers only contain smaller items. In the next phase, when the first item arrives, *BestFit* assigns it to the server with the highest load (which is the first server). The oldest item in that server then departs, causing this server to no longer have the highest load. When the second item arrives, *BestFit* places it in the server with the next highest load, which is now the second server. The oldest item in this server then departs, and this server no longer holds the highest load. This process continues for each incoming item, with each being assigned to the server with the current highest load, and the oldest item in that server departing. As a result, all n servers remain open throughout all k phases, see Figure 6.9. Therefore, the total cost of *BestFit* is $kn\mu$.

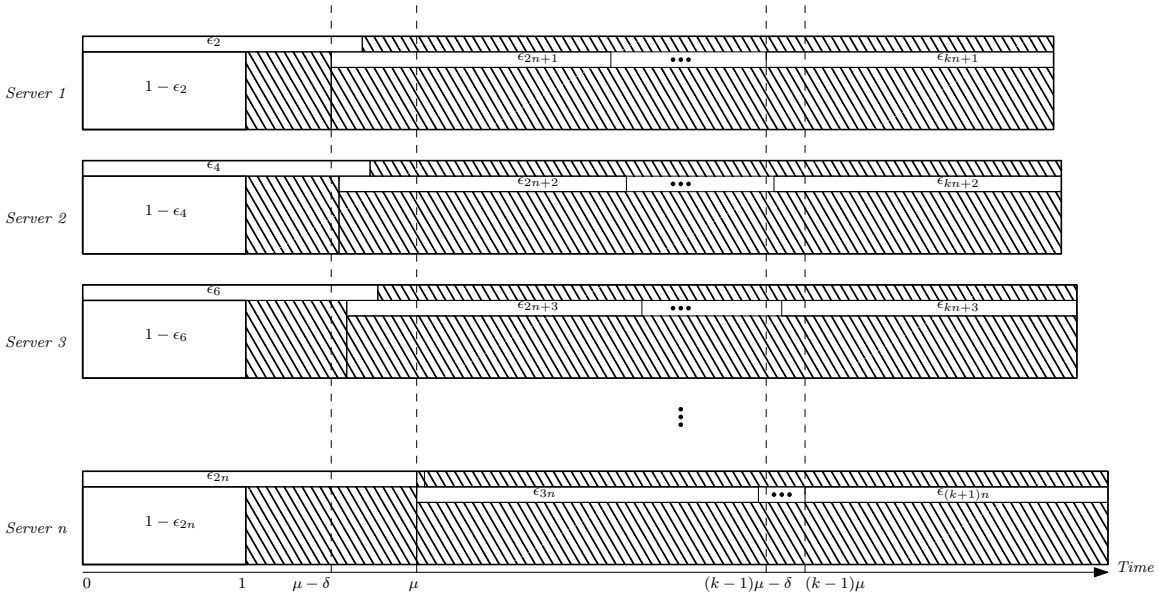

Figure 6.9: Instance I and the assignment of *BestFit*.

In contrast, all other considered algorithms place all items into a single server during the second and subsequent phases, resulting in a total cost of $(n-1)\mu+k\mu$. Notably, except for *FirstFit*, which extends the first server, all other algorithms extend the last server, see Figure 6.10. However, the total cost remains the same.

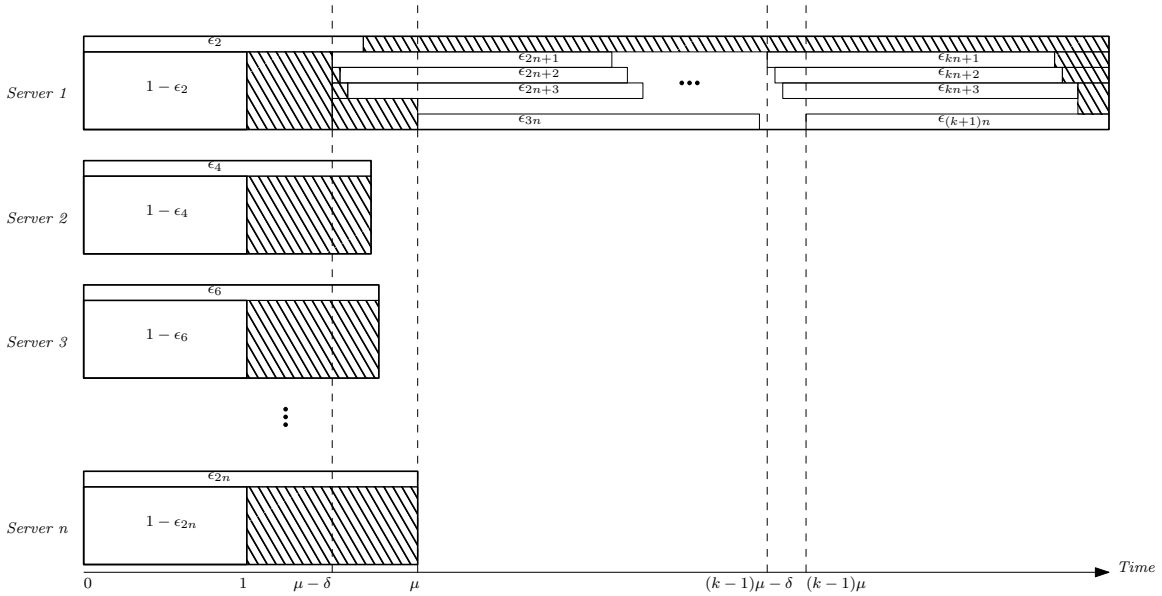

Figure 6.10: Instance I and the assignment of *FirstFit*. The *Greedy*, *LastFit*, *WorstFit*, *MoveToFront*, and *NextFit* algorithms also produce the same assignment. However, these algorithms differ by extending the last server rather than the first server during the assignment process.

Therefore, for instance I:

$$\textit{BestFit}(I) = kn\mu$$

while

$$\textit{FirstFit}(I) = \textit{BestFit}(I) = \textit{Greedy}(I) = \textit{MoveToFront}(I) = \textit{LastFit}(I) = \textit{NextFit}(I) = (n-1)\mu + k\mu$$

As a result, as n grows larger, the ratio of *BestFit* compares to other algorithms approaches infinity, showing the unbounded nature of *BestFit*'s assignment on this instance.

Instance J: This example demonstrates that, in this specific instance, *Greedy* is less efficient than *BestFit*. The instance is structured in three stages. In the first stage, which begins at time 0, n items of size $1-\epsilon$, each with a duration of 1, arrive sequentially, followed by a single item of size 2ϵ with a duration of μ. In the second stage, also at 0, another n items of size ϵ, each with a duration of 1, arrive. Finally, in the third stage, occurring at time $1-\delta$, for a small δ, n items of size ϵ with a duration of $\mu-1$ arrive. Assume that $\epsilon = 1/(n+2)$.

In this scenario, *Greedy* opens $n+1$ servers to accommodate the items from the first stage. Each of the first n servers holds one item of size $1-\epsilon$, while the last server contains the single item of size 2ϵ. When the second stage items arrive, *Greedy* assigns all n items to the last server, which already has a duration of μ, ensuring no additional cost is incurred. However, when the third stage items arrive, the last server lacks sufficient capacity. As a result, *Greedy* extends the duration of the first n servers to μ by assigning one item of size ϵ to each, leading to a total cost of $(n+1)\mu$, see Figure 6.11, part (a).

The same outcome applies to *LastFit*, *WorstFit*, and *MoveToFront* because, like *Greedy*, these algorithms assign the second-stage items to the last server and extend the duration of the first n servers to accommodate the third-stage items. On the other hand, *NextFit* behaves similarly for the first and second stages but differs in its handling of the third stage. Since *NextFit* keeps only one server open at a time, it closes server $n+1$ and opens a new server to accommodate the third-stage items. This results in a total cost of $n+1+\mu$.

BestFit behaves similarly to *Greedy* in the first stage, opening $n+1$ servers to accommodate the items. However, in the second stage, *BestFit* assigns one item of size ϵ to each of the first n servers, as they all have a maximum load of $1-\epsilon$. In the third stage, *BestFit* assigns all items arriving at time $1-\delta$ to the last server, resulting in a total cost of $n+\mu$. This outcome is illustrated in Figure 6.11, part (b).

The same assignment applies to *FirstFit* because *FirstFit* always prioritizes placing new items in the first server with available space in the order of their opening. Consequently, all second-stage items are assigned to the first n servers, and all third-stage items are packed into the last server by *FirstFit*.

Therefore, for instance J:

$$\textit{Greedy}(J) = \textit{LastFit}(J) = \textit{WorstFit}(J) = \textit{MoveToFront}(J) = (n+1)\mu$$

and

$$\textit{NextFit}(J) = n+1+\mu$$

while

$$\textit{BestFit}(J) = \textit{FirstFit}(J) = n + \mu$$

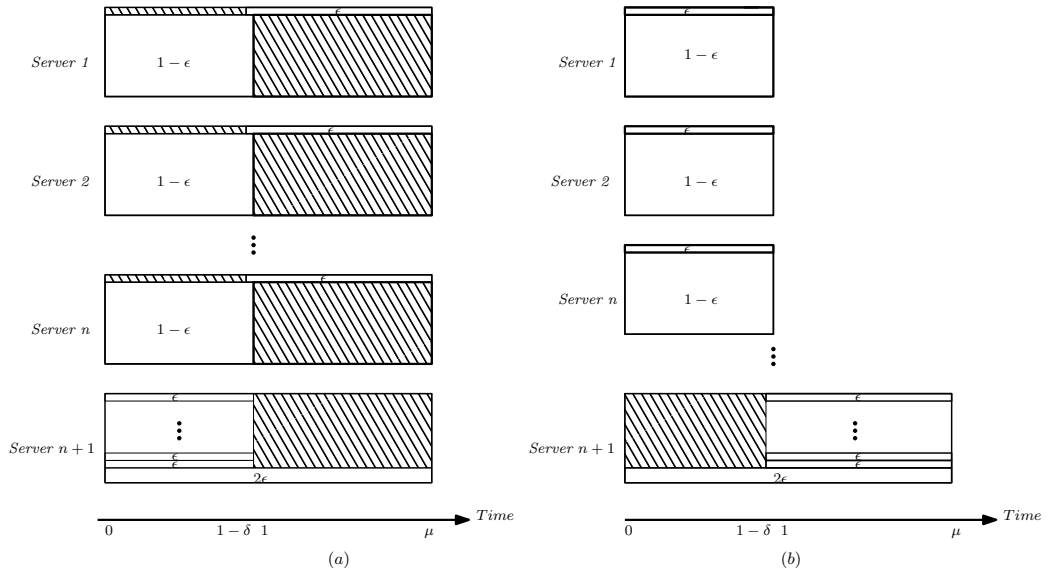

Figure 6.11: Instance J and the assignment of (a) *Greedy*, *LastFit*, *WorstFit*, *MoveToFront*. (b) *BestFit* and *FirstFit*.

Instance K: This instance is constructed in two stages. In the first stage, all items arrive at time 0 in a specified order. The first $n-1$ items have a size of $1-\epsilon$ and a duration of $1+\delta$, where $\delta > 0$ is a small positive value. These are followed by a single item with a size of 2ϵ and a duration of 1. In the second stage, occurring at time 1, n items of size ϵ, each with a duration of $\mu - 1$, arrive. Assume that $\epsilon = 1/(n+2)$.

Greedy handles this instance as follows. In the first stage, it opens n servers to accommodate the n items. Each of the first $n-1$ servers contains one item of size $1-\epsilon$ with a duration of $1+\delta$, while the last server holds the item of size 2ϵ with a duration of 1. In the second stage, *Greedy* assigns each item of size ϵ to one of the already-opened servers, placing one item per server. This assignment extends the duration of all n servers to μ, as illustrated in Figure 6.12, part (a). Consequently, the total cost of *Greedy* is $n\mu$.

This same assignment applies to *FirstFit* and *BestFit*. *FirstFit* always places new items in the first server with available space, following the order in which the servers were opened, while *BestFit* assigns items to the server with the highest current load. Both strategies result in the same final

configuration as *Greedy*.

MoveToFront behaves identically to *Greedy* during the first stage, opening n servers and assigning $n-1$ items of size $1-\epsilon$ with a duration of $1+\delta$ to the first $n-1$ servers, and the item of size 2ϵ with a duration of 1 to the last server. However, in the second stage, *MoveToFront* assigns all n items of size ϵ to the last server, as it is the first server in the list due to the "move-to-front" strategy. This results in extending the duration of the last server to μ while leaving the other $n-1$ servers unchanged. Consequently, the total cost for *MoveToFront* is $(n-1)+\mu$, as shown in Figure 6.12, part (b).

The same assignment applies to *NextFit*, *LastFit*, and *WorstFit*, as these algorithms also prioritize assigning the second-stage items to the last server, leading to the same total cost. Therefore, for instance K:

$$\textit{Greedy}(K) = \textit{FirstFit}(K) = \textit{BestFit}(K) = n\mu$$

while

$$\textit{MoveToFront}(K) = \textit{LastFit}(K) = \textit{NextFit}(K) = \textit{WorstFit}(K) = (n-1)+\mu$$

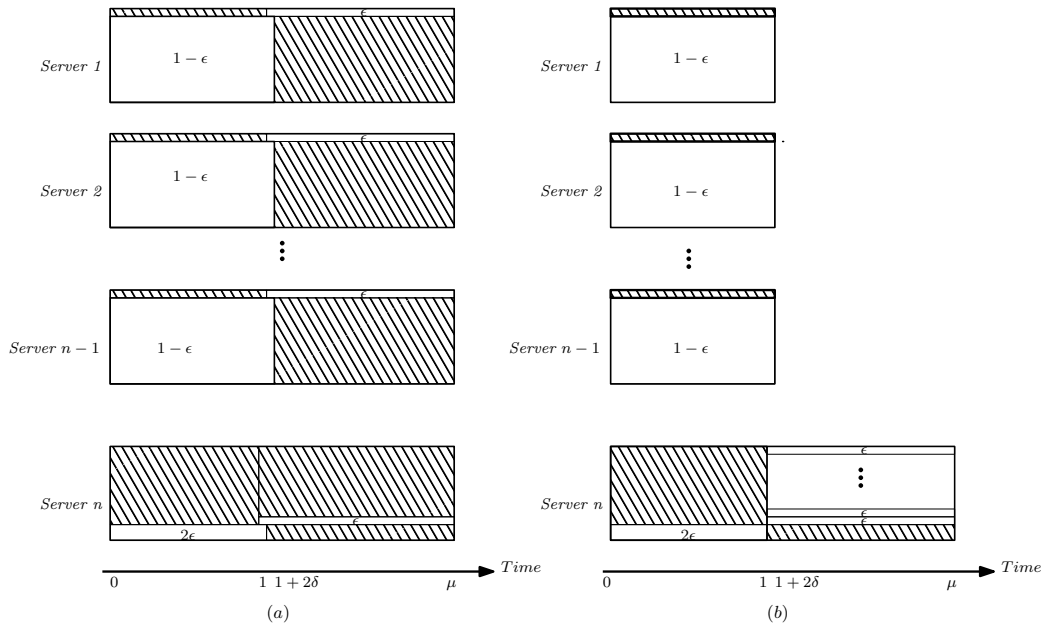

Figure 6.12: Instance K and the assignment of (a) *Greedy*, *FirstFit*, and *BestFit*. (b) *MoveToFront*, *NextFit*, *LastFit*, and *WorstFit*.

Instance L: This instance is constructed in two stages. In the first stage, all items arrive at time 0 in a specified order. The first item has a size of 2ϵ and a duration of 1, followed by $n-1$ items, each of size $1-\epsilon$ and duration $1+\delta$, where $\delta > 0$ is a small positive value. In the second stage, occurring at 1, n items of size ϵ, each with a duration of $\mu - 1$, arrive.

On this input instance, *Greedy* handles the first stage by opening n servers for the n items. Each server accommodates one item, resulting in $n-1$ servers holding items of size $1-\epsilon$ with a duration of $1+\delta$, and the first server holding the item of size 2ϵ with a duration of 1.

In the second stage, *Greedy* assigns the items of size ϵ to the servers with the latest finishing times. It begins by assigning the items to servers 2 through n, which are already at a duration of $1+\delta$, and then assigns the final item to the first server. This process extends the duration of all servers to approximately μ, as illustrated in Figure 6.13, part (a). Consequently, the total cost of *Greedy* is $n\mu$.

This same assignment applies to *LastFit*, *MoveToFront*, and *BestFit*. Since *LastFit* assigns items to the most recently opened server. Therefore, it distributes the second-stage items across all n servers, extending the duration of each server to μ. *BestFit* assigns items to the server with the highest current load. For the second-stage items, *BestFit* first fills the last $n-1$ servers with items of size $1-\epsilon$, and then assigns the final item to the first server, which still has available space. And *MoveToFront* behaves similarly by reordering servers whenever it packs one items into it, leading to the same outcome as *Greedy*.

FirstFit handles this instance in a manner similar to *Greedy* during the first stage, opening n servers to accommodate the items. Each server holds one item, with the first $n-1$ servers containing items of size $1-\epsilon$ and duration $1+\delta$, and the last server holding the item of size 2ϵ with a duration of 1.

In the second stage, however, *FirstFit* assigns all items of size ϵ to the first server, as it has sufficient space to accommodate all these items. This assignment extends the duration of the first server to μ while leaving the remaining $n-1$ servers unchanged. As a result, the total cost of *FirstFit* is $(n-1)+\mu$, as shown in Figure 6.13, part (b). *WorstFit* performs similarly because it prioritizes assigning items to the server with the most available free space. In this case, the first server has the most available capacity, leading to the same outcome as *FirstFit*.

NextFit also behaves similarly to *FirstFit* and *WorstFit*, with one key difference: since *NextFit* keeps only one server open at any time, it assigns the first item of size ϵ from the second stage to the last server, extending its duration to μ. For the remaining items, *NextFit* opens a new server with a duration of $\mu - 1$. This results in a total cost of approximately $n + \mu$.

Therefore, for instance L:

$$\textit{Greedy}(L) = \textit{LastFit}(L) = \textit{MoveToFront}(L) = \textit{BestFit}(L) = n\mu$$

while

$$\textit{FirstFit}(L) = \textit{WorstFit}(L) = n - 1 + \mu$$

and

$$\textit{NextFit}(L) = n + \mu$$

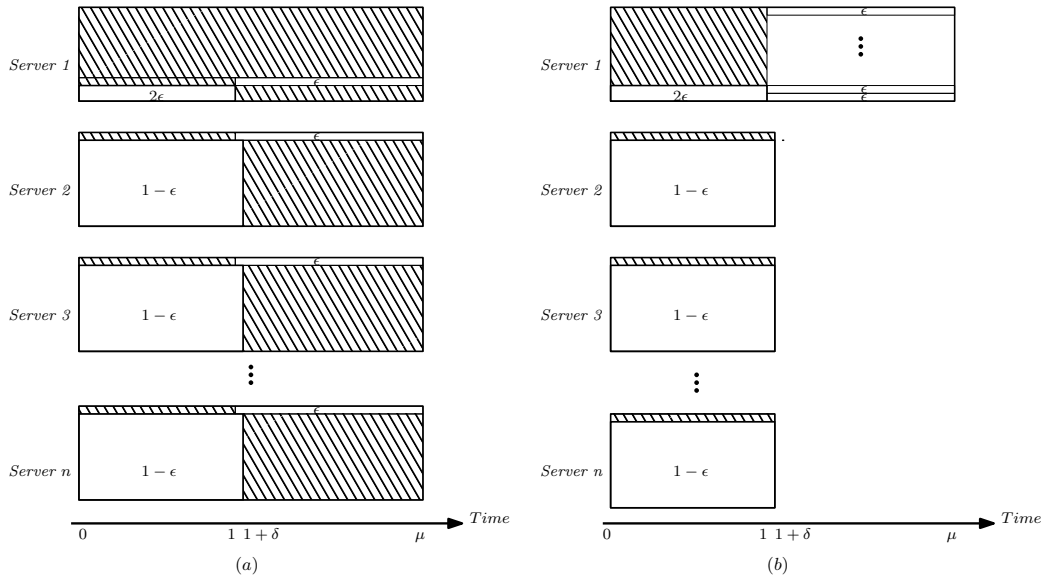

Figure 6.13: Instance L and the assignment of (a) *Greedy*, *LastFit*, *MoveToFront*, and *BestFit*. (b) *FirstFit*, and *WorstFit*.

Table 6.2 provides a breakdown of the distinct sizes, arrival times, and durations for each instance, highlighting the complexity of each example for comparison.

Instance	# of Distinct Sizes	# of Distinct Arrival Times	# of Distinct Duration
A	2	1	2
B	3	1	2
C	$\Theta(n)$	1	2
D	2	1	2
E	4	1	2
F	4	1	2
G	5	1	2
H	$\Theta(n)$	$\Theta(n)$	2
I	$\Theta(n)$	$\Theta(n)$	2
J	3	2	3
K	3	2	3
L	3	2	3

Table 6.2: Summary of instance characteristics including number of sizes, distinct arrival times, and duration counts.

In Table 6.3, we summarize the results. Let $\mathcal{A}$ and $\mathcal{B}$ denote two algorithms. Let σ denote one instance. Observe that, if we construct an instance σ for which $\mathcal{A}(\sigma)/\mathcal{B}(\sigma) \geq \rho$, this automatically implies that ρ is also a lower bound for algorithm $\mathcal{A}$ because $\mathcal{A}(\sigma) \geq \rho \cdot \mathcal{B}(\sigma) \geq \rho \cdot OPT(\sigma)$. Hence, every number in each row is a lower bound of the corresponding row algorithm. Of course, all these lower bounds have been established in existing works, except for *LastFit*. On the other hand, if algorithm ALG has an upper bound ρ, then $\mathcal{A}(\sigma)/\mathcal{B}(\sigma) \leq \rho$ holds for every instance σ because $\mathcal{A}(\sigma) \leq \rho \cdot \mathrm{OPT}(\sigma) \leq \rho \cdot \mathcal{B}(\sigma)$. Hence, all ratios of separation we showed in Table 6.3 are tight up to the constant in front of μ. To clarify this further, consider two algorithms, *FirstFit* and *LastFit*. As discussed, for a given example B, we have *FirstFit*$(B) \geq \mu \cdot$ *LastFit*(B). Additionally, we know that for all σ we have *FirstFit*$(\sigma) \leq (\mu + 3) \cdot \mathrm{OPT}(\sigma) \leq (\mu + 3) \cdot$ *LastFit*(σ). Thus, we can conclude that *FirstFit*$(B) = \Theta(\mu) \cdot$ *LastFit*(B).

	FirstFit	*MoveToFront*	*NextFit*	*BestFit*	*LastFit*	*WorstFit*	*Greedy*
FirstFit	1	B, μ	B, μ	C, μ	B, μ	B, μ	B, μ
MoveToFront	A, 2μ	1	G, μ	A, 2μ	F, μ	F, μ	A, 2μ
NextFit	A, 2μ	D, μ	1	A, 2μ	D, μ	D, μ	A, 2μ
BestFit	H, ∞	H, ∞	H, ∞	1	H, ∞	I, ∞	H, ∞
LastFit	A, 2μ	E, μ	E, μ	A, 2μ	1	E, μ	A, 2μ
WorstFit	I, ∞	H, ∞	H, ∞	H, ∞	H, ∞	1	H, ∞
Greedy	L, μ	K, μ	L, μ	J, μ	K, μ	L, μ	1

Table 6.3: Theoretical comparison of algorithms. (A, k) in the table entry (i, j) indicate that algorithm j is better than algorithm i on instance A by a factor k.

Theorem 6.2.1. *Consider set* $C = \{$*NextFit*, *FirstFit*, *BestFit*, *LastFit*, *WorstFit*, *MoveToFront*, *Greedy*$\}$ *of algorithms for RSiC. For every algorithm* $A \in C$*,* A *is instance-incomparable with any algorithm* $B \in C - \{A\}$.

Proof. The proof is provided through instances A to L above, which demonstrate that within set C, there is no algorithm that consistently outperforms all other algorithms. □

6.3 Summary and Discussion

In this chapter, we investigated the instance dominance concept for a set of algorithms for the *RSiC* problem. Our analysis showed that none of the algorithms among the considered set achieve true instance dominance across all possible input instances. Through a series of examples and detailed comparisons, we highlighted the restriction in each algorithm. Additionally, our set of instances can serve as a guide for the development of future algorithms.

The findings raise an important question about the feasibility of achieving instance dominance for the *RSiC* problem. While it is clear that no algorithm we discussed meets this ideal, we must consider whether it is possible to construct an algorithm that is truly instance-dominant, or whether such a level of performance is unattainable given the inherent complexity of the problem.

Chapter 7

The Case of Multi-Parameter Jobs

In the previous chapters, we examined the *RSiC* problem in a one-dimensional setting, where each job requires only a single resource, such as a specific amount of CPU. However, in real-world scenarios, jobs typically demand multiple resources, including CPU, memory, and I/O bandwidth. This setting is referred to as the d-dimensional setting. In this chapter[1], we extend our analysis to the *RSiC* problem within this d-dimensional framework. The scenario we focus on is outlined in Table 7.1. We first establish an upper bound for this version of the problem and then prove a lower bound for the case of $d = 1$ in the randomized setting. Additionally, we introduce a new clairvoyant algorithm for the d-dimensional *RSiC* problem and demonstrate its upper bound. Finally, we present the results of an experimental evaluation in an average-case scenario, where nearly all existing algorithms for *RSiC* are assessed using randomly generated synthetic data.

Job Duration	**Arrival Pattern**	**# of Arrivals**	**# of Dimensions**	**# of Servers**
Arbitrary	*Arbitrary*	*Arbitrary*	d	*Unlimited*

Table 7.1: A summary of the *RSiC* setting for multi-parameter jobs.

7.1 Preliminaries

Before discussing the results for the problem in this setting, we need to review the following simple properties of the L_∞ norm of a vector $v \in \mathbb{R}^d_{\geq 0}$ denoted by $\|v\|_\infty$ and that equals $\max_{j\in[d]} v_j$.

[1]The results presented in this section are based on joint work with Dr. Yaqiao Li.

We shall make frequent use of the following classical inequalities:

Proposition 7.1.1. *For any set of vectors $v_1, v_2, \cdots, v_n \in \mathbb{R}^d_{\geq 0}$, we have the following:*

$$\left\| \sum_{i=1}^{n} v_i \right\|_\infty \leq \sum_{i=1}^{n} \|v_i\|_\infty \leq d \cdot \left\| \sum_{i=1}^{n} v_i \right\|_\infty$$

7.2 A $\Theta(d\sqrt{\log \mu})$ Algorithm via a Direct-Sum Property of *RSiC*

In this section we give a $O(d\sqrt{\log \mu})$ upper bound for d-dimensional *RSiC* by showing a direct sum property of the problem. This is provably better in the worst case than the previously proposed algorithms for d-dimensional *RSiC* [57], which were shown to have a lower bound of $\Omega(d\mu)$ on their competitive ratio.

Given an arbitrary algorithm ALG for 1-dimensional *RSiC*, we define an algorithm, call it $\text{ALG}^{\oplus d}$, that works for d-dimensional *RSiC*, as follows. Let σ be an input instance for the d-dimensional problem. We partition σ as follows: $\sigma = \sigma^{(1)} \cup \cdots \cup \sigma^{(d)}$, where $\sigma^{(j)}$ is the subset of jobs r for which $\|s(r)\|_\infty$ is achieved at the j-th dimension. When $\|s(r)\|_\infty$ is achieved in more than one dimension, we break the tie arbitrarily. It is easy to see that this partitioning can be done online.

The algorithm $\text{ALG}^{\oplus d}$ is defined as follows: on the arrival of a job r, decide online a unique dimension j in which its size is maximum, and assign $r \in \sigma^{(j)}$, then, apply ALG (for 1-dimensional *RSiC*) to process $\sigma^{(j)}$, pretending that the instance is 1-dimensional by only looking at the size at the j-th coordinate, and ignoring the sizes of other dimensions, and assigning to servers that only contain jobs in $\sigma^{(j)}$.

Theorem 7.2.1. *Let* ALG *be an arbitrary deterministic algorithm for* 1*-dimensional RSiC. Then,* $\text{ALG}^{\oplus d}$ *works correctly for any* d*-dimensional RSiC, and* $\rho(\text{ALG}^{\oplus d}) = d \cdot \rho(\text{ALG})$*. Moreover, the guarantee on the competitive ratio holds for both strict and asymptotic competitive ratios.*

Proof. Firstly, we show that the algorithm $\text{ALG}^{\oplus d}$ does not violate the size constraint, i.e., the total size of all jobs in every server does not exceed 1^d. To see this, consider an arbitrary job $r \in \sigma$ and suppose it is put into a bin B by $\text{ALG}^{\oplus d}$. By the definition of $\text{ALG}^{\oplus d}$, we know before r is put

into B, either server B is empty (i.e., has not been created yet) in which case after r is assigned to server B the size constraint is trivially respected, or server B is nonempty. In the latter case, it only contains jobs in $\sigma^{(j)}$. In this case, we have

$$\|s(r) + s(B,t)\|_\infty = s(r)_j + s(B,t)_j \leq 1,$$

where the equality follows from the fact that every job in server B is in $\sigma^{(j)}$ and also $r \in \sigma^{(j)}$. The inequality follows by the fact that we apply algorithm ALG on $r \in \sigma^{(j)}$.

Next, we show $\rho(\mathrm{ALG}^{\oplus d}) \leq d \cdot \rho(\mathrm{ALG})$. By an abuse of notation, let $\mathrm{ALG}(\sigma^{(j)})$ denote the cost of $\mathrm{ALG}^{\oplus d}(\sigma)$ on the subset of inputs $\sigma^{(j)}$. Since $\sigma^{(j)} \subseteq \sigma$, one has $\mathrm{OPT}(\sigma^{(j)}) \leq \mathrm{OPT}(\sigma)$ for every j. Let $\mathrm{OPT}'(\sigma^{(j)})$ denote the cost of the optimal solution that processes $\sigma^{(j)}$ by only focusing on the size of the j-th dimension. Then, for every $\rho > \rho(\mathrm{ALG})$ there exists a $c > 0$ such that $\mathrm{ALG}(\sigma^{(j)}) \leq \rho \cdot \mathrm{OPT}'(\sigma^{(j)}) + c$ for every j. Observe that we have $\mathrm{OPT}'(\sigma^{(j)}) = \mathrm{OPT}(\sigma^{(j)})$. This is because every job in $\sigma^{(j)}$ satisfies that the size at the j-th dimension is the largest. With these, and by the definition of $\mathrm{ALG}^{\oplus d}$, we have

$$\begin{aligned}
\mathrm{ALG}^{\oplus d}(\sigma) &= \sum_{j \in [d]} \mathrm{ALG}(\sigma^{(j)}) \\
&\leq \sum_{j \in [d]} \Big(\rho \cdot \mathrm{OPT}'(\sigma^{(j)}) + c \Big) \\
&= \sum_{j \in [d]} \Big(\rho \cdot \mathrm{OPT}(\sigma^{(j)}) + c \Big) \\
&\leq \sum_{j \in [d]} (\rho \cdot \mathrm{OPT}(\sigma) + c) = d \cdot \rho \cdot \mathrm{OPT}(\sigma) + cd.
\end{aligned}$$

Since cd is a constant independent of input, and this inequality holds for all $\rho > \rho(\mathrm{ALG})$, it follows that $\rho(\mathrm{ALG}^{\oplus d}) \leq d\rho(\mathrm{ALG})$. Moreover, if $c = 0$ then $cd = 0$, so the competitive ratio guarantee preserves strictness.

Lastly, we show $\rho(\mathrm{ALG}^{\oplus d}) \geq d \cdot \rho(\mathrm{ALG})$. Let H be an arbitrary 1-dimensional instance, from which we construct a d-dimensional instance σ as follows. For every job $h = (a(h), f(h), s(h)) \in H$, create d jobs in σ that have the same arrival and finishing time as h, and the size vectors are the

d column vectors of the matrix $s(h) \cdot I_d$ where I_d is the $d \times d$ identity matrix. Clearly, every $\sigma^{(j)}$ is simply a copy of H in dimension j, while having 0's in all other dimensions. Hence, $\mathrm{ALG}^{\oplus d}(\sigma) = d \cdot \mathrm{ALG}(H)$. Furthermore, observe that $\mathrm{OPT}(H) = \mathrm{OPT}(\sigma)$, where here by an abuse of notation we use $\mathrm{OPT}(H)$ to denote the cost of the optimal algorithm for the 1-dimensional instance H, and $\mathrm{OPT}(\sigma)$ to denote the cost of the optimal algorithm for the d-dimensional instance σ. Hence,

$$\rho(\mathrm{ALG}^{\oplus d}) \geq \frac{\mathrm{ALG}^{\oplus d}(\sigma)}{\mathrm{OPT}(\sigma)} = \frac{d \cdot \mathrm{ALG}(H)}{\mathrm{OPT}(\sigma)} = d \cdot \frac{\mathrm{ALG}(H)}{\mathrm{OPT}(H)}.$$

Because H is arbitrary, the desired lower bound follows. □

We note that Theorem 7.2.1 is rather general – it can apply to strict, as well as asymptotic competitive ratio guarantees, and it can be applied to clairvoyant and non-clairvoyant algorithms for 1-d. For example, applying Theorem 7.2.1 to *Departure Strategy* and *Duration Strategy* algorithms[2] yields $\Theta(d\sqrt{\mu})$ and $\Theta(d \log \mu / \log\log \mu)$ competitive algorithms in dimension d, respectively. The theoretical state-of-the-art clairvoyant algorithm in dimension 1 is the Hybrid Algorithm, *Hybrid Algorithm*, introduced in [1] and shown to have a competitive ratio $\Theta(\sqrt{\log \mu})$. Applying Theorem 7.2.1 to *Hybrid Algorithm* gives us the following.

Corollary 7.2.2. *The algorithm Hybrid Algorithm$^{\oplus d}$ for d-dimensional clairvoyant RSiC has a competitive ratio $\Theta(d\sqrt{\log \mu})$.*

7.3 Lower Bound for Randomized Algorithms for $d = 1$

Since d-dimensional *RSiC* generalizes vector bin packing (by having all items arrive at time 0 and have the same duration 1), the lower bound of $\Omega(d/\log^2 d)$ (due to Azar et al. [2] and Balogh et al. [6]) on the competitive ratio of randomized algorithms applies to *RSiC*. This lower bound result does not give anything non-trivial for small values of d. In this section, we complement this lower bound by giving a stronger lower bound for $d = 1$.

Theorem 7.3.1. *In 1-dimensional non-clairvoyant RSiC, any randomized algorithm has a competitive ratio at least $\frac{1-e^{-1}}{2} \cdot \mu$.*

[2]These algorithms are introduced in [60]. We shall revisit them in Section 7.5.

Proof. We use Yao's principle. Consider the following distributional input: k^2 jobs each of size $1/k$, uniformly at random pick k among k^2 jobs to be of duration μ, and let the rest $k^2 - k$ jobs to be of duration 1. Let ALG be an arbitrary deterministic algorithm. We show that in expectation ALG has cost $\Omega(k\mu)$. Since OPT has cost $\leq k - 1 + \mu$, this gives the competitive ratio $\Omega(\mu)$ as desired.

Let $A_1, \ldots, A_m$ be the m servers that ALG uses for the above instance. Let $|A_i|$ denote the number of jobs in A_i, then $1 \leq |A_i| \leq k$. We partition these m servers into p groups $B_1, \ldots, B_{p-1}, B_p$ such that the number of jobs in each group B_i contains $\geq k$ jobs and $< 2k$ jobs, except perhaps the last group B_p which may contain less than k jobs. Note that such a partition exists by simply partitioning greedily. Hence, $p \geq k^2/2k = k/2$. Let $|B_i|$ denote the number of jobs in group B_i. Then, for every $1 \leq i \leq p-1$,

$$\begin{aligned}
&\Pr[B_i \text{ contains at least one job of duration } \mu] = 1 - \frac{\binom{k^2-|B_i|}{k}}{\binom{k^2}{k}} \\
&\geq 1 - \frac{\binom{k^2-k}{k}}{\binom{k^2}{k}} \geq 1 - (1 - \frac{1}{k})^k \geq 1 - e^{-1}.
\end{aligned}$$

Let $X_i \in \{0, 1\}$ be a random variable denoting whether group B_i contains some job of duration μ or not. Then, by the linearity of expectation, the expected cost of ALG is at least

$$\begin{aligned}
\mu \cdot \mathbb{E}\left[\sum_{i=1}^{p-1} X_i\right] &= \mu \cdot \sum_{i=1}^{p-1} \mathbb{E}[X_i] \geq \mu \cdot (p-1)(1 - e^{-1}) \\
&\geq \mu \cdot (k/2 - 1)(1 - e^{-1}).
\end{aligned}$$

Hence, the competitive ratio is at least $\frac{1-e^{-1}}{2} \cdot \frac{k-2}{k-1+\mu} \cdot \mu$. For every μ, since we can pick k to be arbitrarily large, we get the ratio is at least $\frac{1-e^{-1}}{2} \cdot \mu$ as claimed. □

7.4 *Greedy* and the Class of Monotone *AnyFit* Algorithms

In this section, we want to discuss the *Greedy* algorithm in details; a new clairvoyant algorithm that surprisingly has not been studied earlier. Recall that *Greedy* orders servers in decreasing order of their *finishing times*, that is, the maximum of the finishing times of jobs currently in the server.

It assigns the newly arrived job to the first server in the order that has sufficient capacity. If no such server exists, *Greedy* opens a new server and assign the job to it. *Greedy* is a natural and easy-to-implement algorithm that uses the greedy heuristic of assigning the incoming job to the server that will incur the *least additional cost* to the algorithm. *Greedy* belongs to the class of *AnyFit* algorithms; recall that an algorithm for *RSiC* is said to be an *AnyFit* algorithm if it opens a new server only in case that a new incoming job cannot be accommodated on any of the currently active servers. The main result of this section is to show that the competitive ratio of *Greedy* is at most $3\mu d + 1$. In fact, we show that this bound holds for a large subclass of *AnyFit* algorithms that we introduce below.

Most *AnyFit* algorithms use an ordering of active servers, and assign the next job to the first server in the ordering with enough available space. In this case, we say that the algorithm *employs an ordering*. For example, *FirstFit* orders servers based on their opening times, *BestFit* orders servers based on their remaining capacity, and *MoveToFront* moves the server to which a job is assigned to the first position in the ordering. However, *RandomFit* does not employ any specific ordering; it assigns jobs to servers randomly. Observe that the ordering of servers could be fixed as in *FirstFit* and *LastFit*, or it could change when a job arrives, as in *BestFit*, *WorstFit*, and *MoveToFront*, as well as when a job leaves, as in *BestFit* and *WorstFit*.

Consider an algorithm ALG that employs an ordering. We say that a server S is higher in the ordering than S' at time t if S appears closer to the beginning of the ordering than S'. Consider $t < t'$ and define $A(t, t')$ to be the set of servers that are alive at t and t'.

Definition 7.4.1. An *AnyFit* algorithm ALG is called *monotone* if

- it employs an ordering, and
- for every $t < t'$ and every server $S \in A(t, t')$: if S did not receive any new jobs during the interval (t, t') then every server in $A(t, t')$ that is higher than S in the ordering at time t is still higher than S in the ordering at time t'.

Note that for a monotone *AnyFit* algorithm a server S can move up in the ordering between t and t' only if either some server that was higher than S at time t was released before t', or S received a job during the interval (t, t').

It is clear that *AnyFit* algorithms employing a static ordering such as *FirstFit* and *LastFit* have the monotone property. Observe that in *MoveToFront*, a server that receives a job moves to the first position in the ordering, and the relative position of other servers stays the same. Since the only way for a server to move ahead of other servers in the ordering is for it to receive a job, *MoveToFront* obeys the monotone property. However, in *BestFit*, a server may move down in the ordering when a job departs, causing the server to have more available space. Thus *BestFit* does not obey the monotone property.

Observe that *Greedy* is, indeed, a monotone *AnyFit* algorithm, since a server S moves ahead of another server S' in the ordering of *Greedy* if and only if S receives a job that causes its finishing time to be higher than that of S'.

We need the following lemma before we prove the main result of this section. We denote the sum of L_∞ norms of sizes of jobs with arrival time in the interval (t, t') for $t < t'$ by $s_\infty(\sigma, t, t')$, i.e., $s_\infty(\sigma, t, t') = \sum_{i:t<a_i<t'} \|s_i\|_\infty$.

Lemma 7.4.1. *For $0 < \alpha \leq T$, we have: $\int_0^T s_\infty(\sigma, t-\alpha, t)dt = \alpha \sum_{i=1}^n \|s_i\|_\infty$.*

Proof.

$$\begin{aligned}
\int_0^T s_\infty(\sigma, t-\alpha, t)dt &= \int_0^T \sum_{i=1}^n \mathbf{1}(t-\alpha < a_i < t) \|s_i\|_\infty dt \\
&= \sum_{i=1}^n \int_0^T \mathbf{1}(t-\alpha < a_i < t) \|s_i\|_\infty dt \\
&= \sum_{i=1}^n \|s_i\|_\infty \int_0^T \mathbf{1}(t-\alpha < a_i < t)dt \\
&= \sum_{i=1}^n \alpha \|s_i\|_\infty .
\end{aligned}$$

□

Now, we are ready to prove the main result of this section.

Theorem 7.4.2. *Let* ALG *be a monotone AnyFit algorithm. Then, $\rho(\mathrm{ALG}) \leq 3\mu d + 1$.*

Proof. We claim that for an arbitrary t it holds that

$$\mathrm{ALG}(\sigma, t) \leq s_\infty(\sigma, t-2\mu, t) + s_\infty(\sigma, t-\mu, t) + 1. \tag{32}$$

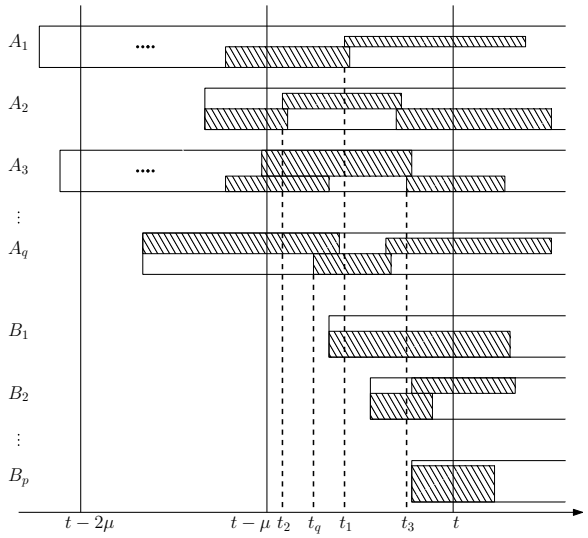

Figure 7.1: Configuration of servers in interval $[t - 2\mu, t]$. Note that the A_i servers are ordered according to the ordering of ALG at time $t - \mu$ while the B_i servers are ordered by opening time.

Observe that each server in $A(t - \mu, t)$ must have received a job during the time interval $(t - \mu, t)$, otherwise a server alive at time $t - \mu$ would have been released by time t, since the duration of each job is at most μ. Suppose there are q servers in $A(t - \mu, t)$ named $A_1, A_2, \ldots, A_q$, ordered according to the ordering of ALG at time $t - \mu$. Let t_i be the earliest time in $(t - \mu, t)$ when a job with the size vector s_i arrived in server A_i. Let $B(t - \mu, t)$ denote the set of new servers that were opened during time $(t - \mu, t)$ that are still alive at time t. Suppose there are p such servers called $B_1, B_2, \ldots, B_p$ ordered by their opening times $t'_1, t'_2, \ldots, t'_p$. Let s'_i be the size vector of the first job placed into B_i. See Figure 7.1 for an illustration. Note that we have $\mathrm{ALG}(\sigma, t) = p + q$.

Consider some $i \in \{2, \ldots, q\}$. Observe that A_{i-1} preceded A_i in the ordering of ALG at time $t - \mu$, no job arrived in A_i during time interval $(t - \mu, t_i)$, and ALG is monotone. Thus, A_{i-1} precedes A_i in the ordering of ALG immediately prior to arrival of job s_i. Thus, ALG must have tried placing s_i into server A_{i-1} at time t_i, but could not fit it in (since s_i was ultimately placed into A_i). This happened because in some coordinate the total size of jobs in server A_{i-1} plus the size of the job s_i in that coordinate exceeded the capacity. Thus, we can conclude that

$$\|s_i + s(A_{i-1}, t_i)\|_\infty > 1. \tag{33}$$

For the B_i servers, since ALG is an *AnyFit* algorithm (it opens a new server only if it has to), we have:

$$\left\|s'_1 + s(A_q, t'_1)\right\|_\infty > 1 \text{ and } \left\|s'_i + s(B_{i-1}, t'_i)\right\|_\infty > 1 \text{ for } 2 \leq i \leq p \tag{34}$$

Also, note that

$$\sum_{i=2}^{q} \|s(A_{i-1}, t_i)\|_\infty + \left\|s(A_q, t'_1)\right\|_\infty + \sum_{i=2}^{p} \left\|s(B_{i-1}, t'_i)\right\|_\infty \leq s_\infty(\sigma, t - 2\mu, t), \tag{35}$$

since all jobs that are alive in server A_{i-1} at time t_i, as well as in A_q at time t'_1, must have arrived between $t - 2\mu$ and t, and the A_i and the B_j servers partition the set of relevant jobs. In addition, we have

$$\sum_{i=2}^{q} \|s_i\|_\infty + \left\|s'_1\right\|_\infty + \sum_{i=2}^{p} \left\|s'_i\right\|_\infty \leq s_\infty(\sigma, t - \mu, t), \tag{36}$$

since the s_i and the s'_i jobs have arrival times between $t - \mu$ and t, and the jobs are distinct.

Combining the observations in (33) and (34), we obtain

$$\begin{aligned} q + p - 1 &< \sum_{i=2}^{q} \|s_i + s(A_{i-1}, t_i)\|_\infty + \left\|s'_1 + s(A_q, t'_1)\right\|_\infty \\ &\quad + \sum_{i=2}^{p} \left\|s'_i + s(B_{i-1}, t'_i)\right\|_\infty \\ &\leq s_\infty(\sigma, t - 2\mu, t) + s_\infty(\sigma, t - \mu, t), \end{aligned}$$

where the second inequality follows from Proposition 7.1.1 and application of (35) and (36). This establishes Inequality (32).

To finish the proof of the theorem, we integrate this inequality over possible values of t, i.e.:

$$\begin{aligned}
\mathrm{ALG}(\sigma) &= \int_0^T \mathrm{ALG}(\sigma, t)dt \\
&\leq \int_0^T (s_\infty(\sigma, t-2\mu, t) + s_\infty(\sigma, t-\mu, t) + 1)dt \\
&= 2\mu \sum_{i=1}^{n} \|s_i\|_\infty + \mu \sum_{i=1}^{n} \|s_i\|_\infty + \mathrm{span}(\sigma) \\
&\leq 3\mu\, \mathrm{util}(\sigma) + \mathrm{span}(\sigma) \\
&\leq 3\mu d\, \mathrm{OPT}(\sigma) + \mathrm{OPT}(\sigma),
\end{aligned}$$

where the second equality follows from two applications of Lemma 7.4.1, and the last inequality is an application of Proposition 1.3.1. □

We remark that the analysis in the theorem above holds for both clairvoyant and non-clairvoyant monotone *AnyFit* algorithms. In Murhekar et al. [57], it shows $(\mu+1)d$ is a lower bound for *AnyFit*. Thus, we obtain the following corollary:

Corollary 7.4.3. $(\mu + 1)d \leq \rho(\textit{Greedy}), \rho(\textit{LastFit}) \leq 3d\mu + 1$.

As mentioned earlier, for the case of $d = 1$, a weaker upper bound $6\mu+8$ can be obtained via the proof in [45]. We conjecture that the correct competitive ratio for *Greedy* is $\mu d + O(d)$. Although *Greedy* has a worse competitive ratio than *Hybrid Algorithm*$^{\oplus d}$, our experimental results described in Section 7.5 show that the competitive ratio of *Greedy* is much better in the average case, and *Greedy* has the best performance in practice of all known algorithms for d-dimensional *RSiC*.

7.5 Experiments

In this section, we provide a thorough evaluation of the average-case performance of almost all existing non-clairvoyant as well as clairvoyant algorithms for the *RSiC* problem.

7.5.1 Experimental Setup

We evaluate the performance of different algorithms using randomly generated input sequences for d-dimensional *RSiC*, for $d \in \{1, 2, 4, 5\}$, closely adhering to the experimental setup detailed in [45] for the 1-dimensional case. In the experiments, we assume that each server has size E^d where $E = 1000$, and each job is assumed to have a size in $\{1, 2, \cdots, E\}^d$. For a given integral span value T; for $T \in \{1000, 5000, 10000\}$, we assume that each job arrives at an integral time step within the interval $[0, T - \mu]$ and has an integral duration in $[1, \mu]$, for $\mu \in \{1, 2, 5, 10, 100\}$. Each experimental instance comprises a sequence of $N = 10000$ jobs, with the size and duration of each job selected randomly from their respective ranges, assuming a uniform distribution. The reported upper bound on competitive ratio of each studied algorithm is computed as the ratio of the average cost of the algorithm over 100 input sequences, and the average of the lower bound on OPT given by Lemma 1.3.3 for these instances.

All our experiments were executed on a personal laptop with a Dual-core 2.3 GHz Intel Core i5 CPU. The laptop had 8 GB of RAM. The laptop was running Mac OS version 12.6.4. The code was written in C++ using VS code version 1.38.1.

7.5.2 Implemented Algorithms

We implemented both clairvoyant and non-clairvoyant algorithms. In Table 7.2, you can find a detailed overview of each one.

p

7.5.3 Experimental Results

Our experimental results for $d \in \{1, 2, 4, 5\}$ are shown in Tables 7.3, 7.4, 7.5, and 7.6. We validated our results against those in [45] for $d = 1$ and for the algorithms implemented there. Our results are slightly different as we use a better lower bound to compute the competitive ratio; when using the same lower bound as [45], our results match exactly.

First we note that for all algorithms, the experimentally derived competitive ratio on random inputs is much better than the worst-case bounds derived theoretically. This is not surprising as the

Algorithm	Description
NextFit	Keeps only one open server at each time.
Modified NextFit	Assigns jobs with sizes greater than a specific threshold separately from the other jobs using the *NextFit* algorithm.
FirstFit	Monotone *AnyFit* that orders servers in increasing order of opening time.
Modified FirstFit	Assigns jobs with sizes greater than a specific threshold separately from the other jobs using the *FirstFit* algorithm.
LastFit	Monotone *AnyFit* that orders servers in decreasing order of opening time.
BestFit	*AnyFit* that orders servers in increasing order of remaining capacity.
WorstFit	*AnyFit* that orders servers in decreasing order of remaining capacity.
RandomFit	*AnyFit* algorithm that orders servers randomly.
MoveToFront	Monotone *AnyFit* that orders servers in decreasing order of the last time a job was assigned to it.
Departure Strategy [60]	The span is split into intervals of length τ each, where $\tau > 0$ is a constant. Classifies jobs into categories according to their departure times. Each category contains all jobs that depart in a time interval of length τ.
Duration Strategy [60]	Classifies the jobs into categories such that the max/min job duration ratio for each category is a given constant α. Given a base job duration b, each category includes all the jobs with durations between $b\alpha^{i-1}$ and $b\alpha^{i}$ for an integer i.
Hybrid Algorithm (*HA*) [1]	Classifies jobs according to their length and their arrivals. Suppose the maximum duration of jobs in the input sequence is μ. Then all the jobs whose lengths are in the range $[2^{i-1}, 2^{i}]$ for integer $1 \leq i \leq \lceil \log \mu \rceil + 1$ and whose arrival times are in the time interval $[(c-1)2^{i}, c2^{i})$ for an integer c are put into the same category.
New Hybrid	*Hybrid Algorithm*$^{\oplus d}$ as defined in Section 7.2.
Greedy	Monotone *AnyFit* as defined in Section 7.4.

Table 7.2: Implemented Algorithms. Similar to [45], we adopt the parameters for *ModifiedNextFit* and *ModifiedFirstFit* as $E^d/(\mu+1)$ and $E^d/(\mu+7)$, respectively. This choice of values is designed to optimize the competitive ratio of these algorithms, as indicated in [45, 52].

worst-case inputs are carefully constructed to beat the given algorithm and are unlikely to occur in practice. Comparing the three tables, we see that the competitive ratio for every algorithm and every value of T and μ increases with increasing d. In general, the competitive ratio also increases with μ, keeping other parameters constant.

We note that many algorithms have versions that separate servers into separate categories, and assign jobs to servers in a particular category based on their sizes. In general, such modifications of algorithms do not perform better than the original versions on random inputs, even though they have better worst-case competitive ratios. For example, *MFF* performs worse than *FirstFit*, *MNF* performs worse than *NextFit*, and *Hybrid Algorithm*$^{\oplus d}$ performs worse than *Hybrid Algorithm* in our experiments. We can also see that all clairvoyant algorithms except *Greedy* classify jobs and

	T=1000					T=5000					T=10000				
	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$
	Non-clairvoyant														
NextFit	1.27	1.37	1.45	1.49	1.52	1.12	1.20	1.32	1.40	1.51	1.06	1.10	1.20	1.31	1.50
MNF	1.31	1.39	1.43	1.48	1.52	1.19	1.29	1.41	1.47	1.52	1.11	1.19	1.31	1.39	1.51
WorstFit	1.41	1.39	1.36	1.33	1.29	1.16	1.20	1.26	1.28	1.29	1.06	1.09	1.16	1.22	1.29
FirstFit	1.42	1.36	1.30	1.27	1.22	1.17	1.20	1.24	1.25	1.23	1.07	1.10	1.16	1.21	1.24
MFF	1.51	1.44	1.35	1.30	1.23	1.25	1.29	1.33	1.32	1.25	1.13	1.17	1.24	1.28	1.25
BestFit	1.51	1.41	1.31	1.24	**1.11**	1.17	1.21	1.25	1.26	**1.16**	1.07	1.10	1.17	1.22	1.19
LastFit	1.35	1.34	1.29	1.25	1.17	1.14	1.18	1.23	1.24	1.19	1.05	1.08	1.15	1.20	1.21
Random Fit	1.49	1.41	1.34	1.28	1.18	1.17	1.21	1.26	1.27	1.21	1.07	1.10	1.17	1.22	1.23
MoveToFront	1.32	1.32	1.28	1.24	1.16	1.13	1.17	1.22	1.24	1.19	1.05	1.08	1.15	1.20	1.20
	Clairvoyant														
Departure Strategy	1.42	1.36	1.30	1.27	1.17	1.17	1.20	1.24	1.25	1.21	1.07	1.10	1.16	1.21	1.23
Duration Strategy	1.42	1.40	1.35	1.31	1.23	1.17	1.20	1.24	1.25	1.24	1.07	1.16	1.26	1.33	1.29
Hybrid Algorithm	**1.12**	**1.25**	1.32	1.33	1.25	**1.03**	1.22	1.36	1.39	1.31	**1.01**	1.15	1.30	1.39	1.34
New Hybrid	**1.12**	**1.25**	1.32	1.33	1.25	**1.03**	1.22	1.36	1.40	1.31	**1.01**	1.15	1.30	1.39	1.34
Greedy	1.28	1.27	**1.22**	**1.19**	1.13	1.12	**1.15**	**1.19**	**1.20**	**1.16**	1.05	**1.07**	**1.13**	**1.17**	**1.17**

Table 7.3: Average competitive ratio results for the *RSiC* problem when $d = 1$.

	T=1000					T=5000					T=10000				
	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$
	Non-clairvoyant														
NextFit	1.40	1.49	1.59	1.65	1.73	1.12	1.20	1.36	1.48	1.69	1.05	1.09	1.21	1.35	1.65
MNF	1.44	1.52	1.61	1.65	1.73	1.17	1.25	1.39	1.49	1.69	1.09	1.13	1.23	1.36	1.65
WorstFit	1.46	1.45	1.44	1.42	1.38	1.14	1.19	1.29	1.35	1.39	1.05	1.08	1.17	1.26	1.38
FirstFit	1.49	1.45	1.42	1.40	1.35	1.15	1.20	1.29	1.34	1.37	1.06	1.09	1.17	1.26	1.37
MFF	1.50	1.47	1.43	1.41	1.35	1.16	1.21	1.30	1.35	1.37	1.06	1.09	1.18	1.26	1.37
BestFit	1.48	1.44	1.40	1.37	1.26	1.14	1.19	1.28	1.33	1.31	1.05	1.08	1.17	1.25	1.33
LastFit	1.39	1.40	1.39	1.36	1.29	1.12	1.17	1.27	1.32	1.32	1.05	1.08	1.16	1.24	1.33
Random Fit	1.48	1.45	1.42	1.39	1.30	1.14	1.19	1.28	1.34	1.34	1.05	1.08	1.17	1.26	1.35
MoveToFront	1.38	1.39	1.38	1.36	1.28	1.12	1.17	1.27	1.32	1.32	1.05	1.07	1.16	1.24	1.33
	Clairvoyant														
Departure Strategy	1.48	1.45	1.42	1.40	1.30	1.15	1.20	1.29	1.34	1.35	1.06	1.09	1.17	1.26	1.37
Duration Strategy	1.48	1.49	1.48	1.47	1.38	1.15	1.20	1.29	1.34	1.37	1.06	1.12	1.23	1.34	1.44
Hybrid Algorithm	**1.23**	1.37	1.45	1.47	1.38	**1.07**	1.21	1.36	1.45	1.45	**1.02**	1.11	1.25	1.37	1.48
New Hybrid	1.42	1.54	1.62	1.65	1.64	1.17	1.29	1.46	1.57	1.65	1.09	1.16	1.31	1.46	1.65
Greedy	1.36	**1.36**	**1.34**	**1.32**	**1.24**	1.12	**1.16**	**1.25**	**1.30**	**1.29**	1.04	**1.07**	**1.15**	**1.23**	**1.30**

Table 7.4: Average competitive ratio results for the *RSiC* problem when $d = 2$.

servers into different categories, and do not have good performance.

An interesting finding is that *Greedy* has the best performance in almost all cases, among all clairvoyant and non-clairvoyant algorithms. Clearly, *Greedy* being a clairvoyant algorithm uses the information on finishing time of jobs to its advantage. However, the other clairvoyant algorithms generally do not exhibit good performance, with the exception of *Hybrid Algorithm* for the case $\mu = 1$. It is important to note that *Greedy* is the only monotone *AnyFit* algorithm among the clairvoyant algorithms we have implemented. The other clairvoyant algorithms do not belong to the monotone *AnyFit* algorithms category.

Among non-clairvoyant algorithms, the best algorithms are generally *MoveToFront* and *LastFit*, which are also both monotone *AnyFit* algorithms. Surprisingly, as in [45], in our experiments *BestFit*

	T=1000					T=5000					T=10000				
	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$
	Non-clairvoyant														
NextFit	1.48	1.57	1.69	1.75	1.88	1.12	1.20	1.38	1.52	1.80	**1.04**	1.08	1.20	1.36	1.75
MNF	1.50	1.58	1.69	1.75	1.88	1.13	1.20	1.38	1.52	1.81	1.05	1.09	1.20	1.36	1.75
WorstFit	1.47	1.51	1.55	1.56	1.53	1.11	**1.18**	**1.32**	1.43	1.55	**1.04**	**1.07**	1.18	1.30	1.54
FirstFit	1.49	1.52	1.56	1.57	1.54	1.12	1.19	1.33	1.43	1.55	**1.04**	1.08	1.18	1.30	1.54
MFF	1.49	1.52	1.55	1.57	1.54	1.12	1.19	1.33	1.43	1.55	**1.04**	**1.07**	1.18	1.31	1.54
BestFit	1.47	1.50	1.54	1.54	1.48	1.12	**1.18**	**1.32**	1.43	1.52	**1.04**	**1.07**	1.18	1.30	1.52
LastFit	1.43	1.48	1.52	1.53	1.48	1.11	**1.18**	**1.32**	1.42	1.52	**1.04**	**1.07**	**1.17**	1.30	1.52
Random Fit	1.47	1.50	1.54	1.55	1.50	1.12	**1.18**	**1.32**	1.43	1.53	**1.04**	**1.07**	1.18	1.30	1.53
MoveToFront	1.43	1.48	1.52	1.53	1.48	1.11	**1.18**	**1.32**	1.42	1.52	**1.04**	**1.07**	**1.17**	1.30	1.52
	Clairvoyant														
Departure Strategy	1.49	1.52	1.55	1.57	1.52	1.12	1.19	1.33	1.43	1.56	**1.04**	1.08	1.18	1.31	1.55
Duration Strategy	1.49	1.52	1.55	1.57	1.54	1.12	1.19	1.33	1.43	1.56	**1.04**	1.08	1.18	1.31	1.55
Hybrid Algorithm	**1.38**	1.50	1.59	1.64	1.60	**1.10**	1.20	1.37	1.50	1.64	**1.04**	1.09	1.21	1.36	1.63
New Hybrid	1.53	1.62	1.73	1.79	1.88	1.15	1.24	1.42	1.57	1.83	1.06	1.11	1.24	1.40	1.79
Greedy	1.43	**1.47**	**1.51**	**1.52**	**1.46**	1.11	**1.18**	**1.32**	**1.41**	**1.50**	**1.04**	**1.07**	**1.17**	**1.29**	**1.51**

Table 7.5: Average competitive ratio results for the *RSiC* problem when $d = 4$.

	T=1000					T=5000					T=10000				
	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$	$\mu=1$	$\mu=2$	$\mu=5$	$\mu=10$	$\mu=100$
	Non-clairvoyant														
NextFit	1.49	1.58	1.70	1.77	1.90	1.11	1.19	1.37	1.52	1.82	**1.04**	1.08	1.19	1.35	1.76
MNF	1.50	1.58	1.70	1.77	1.90	1.12	1.19	1.37	1.52	1.82	**1.04**	1.08	1.19	1.35	1.76
WorstFit	1.47	1.52	1.59	1.61	1.61	1.11	**1.18**	**1.33**	1.45	1.61	**1.04**	**1.07**	1.18	**1.31**	1.60
FirstFit	1.48	1.53	1.59	1.62	1.62	1.11	**1.18**	1.34	1.46	1.62	**1.04**	**1.07**	1.18	**1.31**	1.60
MFF	1.48	1.53	1.59	1.62	1.62	1.11	**1.18**	1.34	1.46	1.62	**1.04**	**1.07**	1.18	**1.31**	1.60
BestFit	1.47	1.52	1.58	1.60	1.57	1.11	**1.18**	**1.33**	1.45	1.60	**1.04**	**1.07**	1.18	**1.31**	1.59
LastFit	1.45	1.51	1.57	1.60	1.57	1.11	**1.18**	**1.33**	1.45	1.60	**1.04**	**1.07**	1.18	**1.31**	**1.58**
Random Fit	1.47	1.52	1.58	1.61	1.59	1.11	**1.18**	**1.33**	1.45	1.61	**1.04**	**1.07**	1.18	**1.31**	1.59
MoveToFront	1.45	1.51	1.57	1.60	1.57	1.11	**1.18**	**1.33**	1.45	1.60	**1.04**	**1.07**	1.18	**1.31**	1.59
	Clairvoyant														
Departure Strategy	1.48	1.53	1.59	1.62	1.61	1.11	**1.18**	1.34	1.46	1.63	**1.04**	**1.07**	1.18	**1.31**	1.61
Duration Strategy	1.48	1.55	1.63	1.67	1.66	1.11	**1.18**	1.34	1.46	1.63	**1.04**	1.08	1.19	1.34	1.65
Hybrid Algorithm	**1.42**	1.53	1.63	1.68	1.68	**1.10**	1.19	1.37	1.51	1.70	**1.04**	1.08	1.20	1.35	1.68
New Hybrid	1.52	1.61	1.72	1.79	1.91	1.13	1.21	1.39	1.55	1.85	1.05	1.09	1.21	1.37	1.79
Greedy	1.45	**1.50**	**1.56**	**1.59**	**1.55**	1.11	**1.18**	**1.33**	**1.44**	**1.59**	**1.04**	**1.07**	**1.17**	**1.31**	**1.58**

Table 7.6: Average competitive ratio results for the *RSiC* problem when $d = 5$.

is one of the better algorithms, especially for higher values of μ, where its performance ratio equals or betters that of *MoveToFront* and *LastFit*[3]. Recall that the worst-case competitive ratio of *BestFit* is unbounded as shown in [50].

In [45] and [57], two key factors, namely *alignment* and *packing* are identified as contributing to the performance of an algorithm for *RSiC*. The first factor is about how effectively jobs are aligned into servers in terms of their durations, while the second factor evaluates how tightly the jobs are packed together in servers. *AnyFit* algorithms (except *WorstFit*) do well in terms of packing, but do not consider alignment. Conversely, *NextFit* tries to align jobs but does not do as well with packing.

[3]While *BestFit* also has excellent performance in the experiments of [57], it does not beat *MoveToFront* for any value of d or μ. However, their experimental setup is somewhat different to ours and that in [45].

The authors of [45] stipulate that by assigning the next job to the server that has the *most recently arrived job*, *MoveToFront* succeeds in terms of aligning jobs well in terms of time, while it also succeeds in packing since it is an *AnyFit* algorithm.

The logic behind the first factor – alignment – is that jobs arriving around the same time are likely to depart around the same time as well. Therefore, an algorithm that groups jobs with similar departure times is expected to yield better performance. *Greedy* follows this principle by choosing the server in such a way that minimally extends the finishing time when placing a job into it. Similarly, *MoveToFront* contributes to addressing this factor by placing a job in the recently used server. Thereby both algorithms addressing this factor and enhancing overall performance. On the other hand, all *AnyFit* algorithms, including *Greedy* and *MoveToFront*, possess the second factor – packing. This is because these algorithms aim to avoid opening new servers and prefer packing jobs into already opened servers. By doing so, they achieve a more efficient use of resources, which results a better performance.

Considering the clairvoyant algorithms, all algorithms except *Greedy* do not fall under the *AnyFit* category, meaning they do not tightly pack jobs into bins. *Greedy* is the only algorithm in this category that focuses on packing jobs tightly. The other four algorithms open new bins even when there are already opened bins for other categories of jobs. The *Hybrid Algorithm* and *New Hybrid* attempt to group jobs with similar arrival and duration times together, reserving bins for each category and then packing them together. While achieving excellent alignment, the packing seems to suffer a lot, resulting in bad performance overall on the input instances considered in this paper. It is important to highlight that, although Table 2.1 shows that *Hybrid Algorithm*$^{\oplus d}$ has better worst-case performance than *Greedy* across all dimensions, these ratios are asymptotic. Our experiments are limited to $\mu \leq 100$, so for significantly larger values of μ, it is possible that *Hybrid Algorithm*$^{\oplus d}$ could further surpass *Greedy*.

A final interesting finding is that the difference in performance between the algorithms appears to narrow for $d = 5$. Further research is needed to understand this phenomenon, but one reason could be that because the sizes of jobs in different dimensions are chosen independently, it is harder for any algorithm to achieve a good packing, which diminishes the difference between the algorithms.

7.6 Summary and Discussion

In this chapter, we addressed the d-dimensional *RSiC* problem and proposed two new algorithms: *Greedy*, which excels in practical performance, and *Hybrid Algorithm*$^{\oplus d}$, which offers the best theoretical worst-case performance among all existing algorithms. We introduced a new class of *AnyFit* algorithms known as monotone algorithms and proved that they, including *Greedy*, achieve a competitive ratio of $\Theta(d\mu)$. Our study also includes a general direct-sum theorem that extends 1-dimensional *RSiC* algorithms to d-dimensional, with *Hybrid Algorithm*$^{\oplus d}$ deriving as a result. Additionally, our experiments demonstrated that the *Greedy* algorithm is among the top performers in average-case scenarios, surpassing both clairvoyant and non-clairvoyant algorithms.

In this chapter, we established that *Greedy* has a performance bound of at most $3\mu d + O(d)$. However, we conjecture that its actual theoretical performance is at most $d\mu + O(d)$. This conjecture arises from the observation that our proof for the monotone *AnyFit* algorithm is somewhat loose, as it relies on upper bounds for server costs in the intervals $[t - 2\mu, t]$ and $[t - \mu, t]$. Improving the upper bound for the monotone *AnyFit* algorithms would be an interesting direction for future research.

Chapter 8

Experiments on Real-World Data

In this chapter, we evaluate the performance of nearly all existing clairvoyant and non-clairvoyant algorithms for the *RSiC* problem. Additionally, we introduce new algorithms, derived from combinations of existing algorithms, which outperforms all previously known algorithms in our experimental evaluations. Unlike prior studies that exclusively utilized synthetic data, we rely on real-world Azure data from the "Protean: VM Allocation Service at Scale" study [36]. This dataset captures large-scale virtual machine (VM) allocation across Azure's availability zones, offering a more practical perspective on the operational effectiveness of these algorithms.

8.1 Description of the Dataset

We begin by describing the dataset, which captures a segment of Microsoft's Azure Compute workload consisting of a set of VM requests. Each VM request corresponds to a specific VM type, detailing requirements for CPU (core), memory (RAM), SSD (solid-state drive), NIC (network bandwidth), and HDD (hard drive), with resource values presented in fractional units. Collected over a 14-day period, the dataset includes all VMs that overlapped with this timeframe. Jobs continued to be observed for 76 days after the data collection period. Jobs with a start time prior to the observation period are assigned a negative start time. Jobs that did not conclude within 76 days of the observation period's end are given a null end time. VMs are classified by priority—high (0) and

low (1)—where low-priority VMs may be evicted in favor of high-priority VMs. This comprehensive dataset provides a realistic basis for analyzing VM allocation strategies in cloud environments.

8.1.1 Analysis of the Dataset

Hadary et al. [36] provided valuable insights into the nature of VM workloads, highlighting their significant variability and non-uniform distribution. The analysis revealed that certain VM types dominate the workload, with some comprising nearly half of the total, while others occur much less frequently. A majority of VMs require relatively few cores, yet a subset of VM types exhibits significantly higher demands for computational resources, requiring a greater number of cores.

The authors also examined temporal variations in resource demand, highlighting sharp surges during job arrivals and gradual declines as jobs are completed. Additionally, the authors observed substantial variation in VM durations, with many VMs running for only a few minutes, while others last for weeks or even months.

Another noteworthy finding was the diurnal patterns in VM requests. Peak demand typically occurs on Tuesdays, Wednesdays, and Thursdays, with request volumes dropping overnight but rising during the day, particularly with sharp spikes at 12:00 PM and 4:00 PM. This variability highlights the varying nature of workload demand, as observed in the study.

Figure 8.1 illustrates the frequency of VMs based on their core sizes, memory sizes, SSD sizes, and NIC sizes that were assigned to one specific machine, called machine 0 on the dataset. The figure reveals that for all these resource types, the majority of VMs require only a small fraction of the total available capacity. Specifically, for core sizes, approximately 80% of VMs utilize less than 10% of the total core capacity, demonstrating the lightweight nature of most workloads. A similar trend is evident for memory, SSD, and NIC sizes, where the demand is predominantly concentrated within the lower ranges of the available resource capacity.

This consistent pattern across resource types highlights that while the large majority of VMs impose minimal demands, a small subset of resource-intensive VMs stand out as outliers, consuming a disproportionately large share of resources. These findings emphasize the importance of designing

resource allocation strategies that are optimized to efficiently handle both the lightweight majority and the resource-heavy minority, ensuring balanced and effective utilization of computational infrastructure.

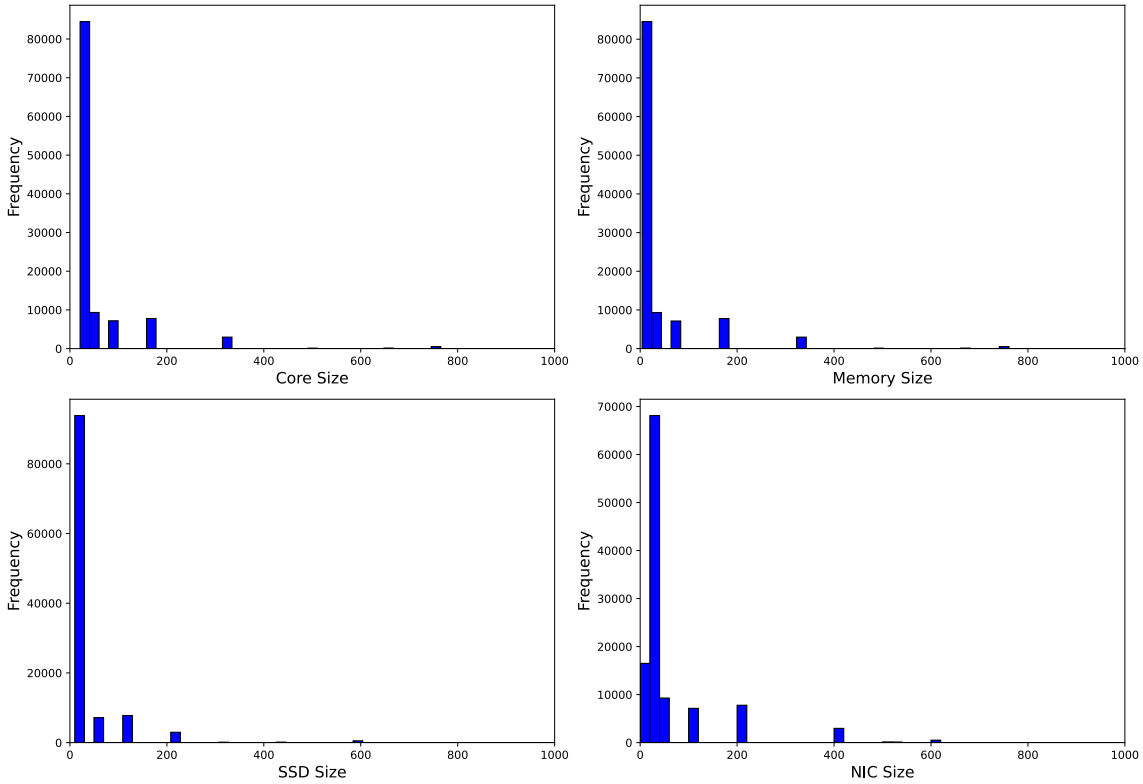

Figure 8.1: Computational resources demand for all dimensions for machine 0. Note that since all type 0 machines are identical, their computing capacities are the same such as memory, cores, and other resources. All values are represented as integers and we scale all resource capacities to 1000 in order to standardize the analysis.

In Figure 8.2, we present the total resource demand for each dimension over time. At each time step, the figure shows the cumulative demand for all resources. The resource demand surges sharply during job arrivals, reflecting the significant initial load as incoming jobs request resources simultaneously. In contrast, the tail of the plot corresponds to the period when jobs are completing and leaving the system, leading to a gradual decline in total resource demand. The monitoring period spans 14 days, during which the peak reflects the maximum demand from active jobs, while the tail illustrates the steady reduction as jobs finish and release resources.

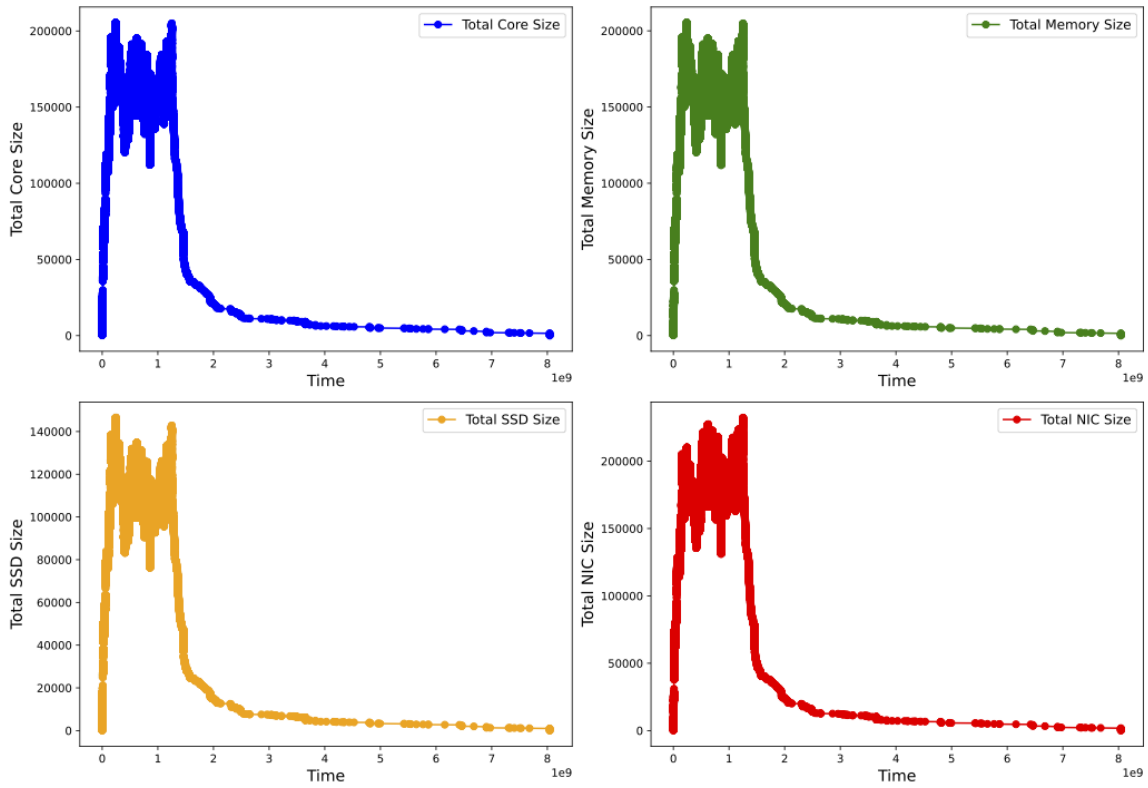

Figure 8.2: Sum of the computational resources demand for each dimensions at each specific time for machine 0. Note that all values are represented as integers.

For the same machine shown in Figure 8.1, we plot the duration of VMs assigned to it. As illustrated in Figure 8.3, the majority of VMs have very short duration as discussed in [36].

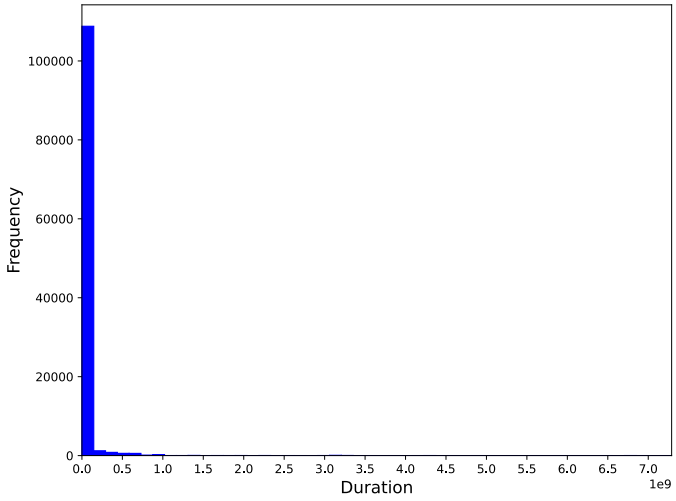

Figure 8.3: The frequency of each job duration is shown for all jobs. Note that all values are represented as integers.

8.2 Experimental Setup

In our experiments, each virtual machine (VM) is treated as a job request, representing the demand for computational resources within the Azure environment. We evaluate the performance of various algorithms using the Azure dataset, assigning these jobs to servers. The dataset includes nearly 34 different types of machines, each capable of accommodating a significant volume of jobs. Due to resource and time constraints, our analysis is restricted to machines 0 and 1. Each job comprises five resource dimensions: CPU, memory, SSD, NIC, and HDD. However, due to the frequent presence of null values in the HDD dimension, we discarded it from our analysis and focused on the remaining four dimensions. As we mentioned earlier, in this dataset, negative start times indicate jobs that were already active at the onset of data collection, while null end times represent jobs that continued running beyond the 90-day observation period. To ensure accuracy, we exclude jobs with negative start times or null end times from our analysis.

As previously noted, the resource values are expressed in fractional units. For consistency with earlier experiments involving synthetic data, we convert all resource values to integers without loss of generality. In the experiments, we assume that each server has a size of E^4 where $E = 1000$, and each job is presumed to have a size in $\{1, 2, \cdots, E\}^4$.

We consider a configuration where jobs with a priority of 1 are excluded from both machines 0 and 1, as the eviction policy makes their duration data unreliable. This exclusion is reasonable because, as shown in Table 8.1, the majority of jobs—89.51%—in machine 0 have a priority of 0, and a similar distribution is expected for machine 1. Additionally, machine 1 receives nearly 4,000,000 job requests, with 3,775,455 of them classified as priority 0. However, due to computational constraints, we analyze only a subset of these jobs. To enable a fair comparison, the sampled dataset for machine 1 is chosen to be approximately the same size as that of machine 0. Specifically, each priority 0 job is independently selected with a probability of $112,552/3,775,455$. Our experiments are conducted under the following configurations:

(1) Machine 0 handling both priority 0 and priority 1 jobs.

(2) Machine 0 restricted to handling only priority 0 jobs.

(3) Machine 1 processing a randomly sampled sequence of jobs, uniformly selected from its priority 0 job requests. To ensure consistency, we generate two independent random samples for machine 1.

Table 8.1 provides a detailed summary of the datasets, including the number of jobs and the corresponding μ ratio for each configuration. It highlights that the μ values vary significantly across datasets.

	Machine 0, Priority 0	Machine 0, Priority 0 and 1	Machine 1, Priority 0 - Sample 1	Machine 1, Priority 0 - Sample 2
Number of Jobs	$112,552$	$125,784$	$112,299$	$112,546$
μ	$11,794,735$	$11,794,735$	$104,761,079$	$52,083,077$

Table 8.1: Number of jobs and the ratio μ for each dataset.

The reported upper bound on the competitive ratio for each algorithm is the ratio of the algorithm's cost over input sequences to the cost of the lower bound on OPT as specified in Lemma 1.3.3 for these instances.

All experiments were conducted on a personal laptop equipped with an Apple M1 chip, featuring an 8-core CPU and 16 GB of RAM. The laptop operated on macOS Sonoma 14.5, and the code was developed in C++ using Visual Studio Code version 1.93.1.

The algorithms we implemented are identical to those described in Section 7.5.2. For detailed descriptions, refer to Table 7.2.

8.3 Experiments on the Full Dataset

As detailed in Section 8.2, our experiments are conducted on machines 0 and 1. A comprehensive summary of the experimental results across all datasets is presented in Table 8.2.

Algorithms	Machine 0, Priority 0, $\mu = 11,794,735$	Machine 0, Priority 0 and 1, $\mu = 11,794,735$	Machine 1, Priority 0-Sample 1, $\mu = 104,761,079$	Machine 1, Priority 0-Sample 2, $\mu = 52,083,077$
	Non-clairvoyant			
NextFit	4.48	3.70	8.88	8.85
MNF	4.48	3.70	8.88	8.85
WorstFit	2.81	2.76	3.47	3.51
RandomFit	2.55	2.64	3.14	3.13
MoveToFront	2.48	2.14	3.15	3.16
LastFit	2.24	1.93	3.11	3.14
BestFit	2.21	1.99	2.99	2.85
FirstFit	2.00	1.76	2.71	2.69
MFF	2.00	1.76	2.71	2.69
	Clairvoyant			
Departure Strategy	6.17	4.88	13.53	13.20
Duration Strategy	1.83	1.69	2.23	2.22
Greedy	1.64	1.54	1.85	1.78
New Hybrid	1.30	1.24	1.67	1.67
Hybrid Algorithm	1.29	1.21	1.36	1.35

Table 8.2: Competitive ratio results for the *RSiC* problem on real data. Algorithms are listed in decreasing order of competitive ratio for Machine 0, Priority 0.

Now, let us focus on analyzing the experimental results. To begin, we note that the competitive ratios of all implemented algorithms, as shown in Table 8.2, have almost the same ratio order for all datasets, despite some differences in their ratios. *NextFit* and *MNF* are the worst and *FirstFit* and *MFF* are the best among all non-clairvoyant algorithms. On the other hand, for clairvoyant algorithms, *Hybrid Algorithm* is the best and *Departure Strategy* is the worst. This variation is due to the differing number of jobs and the distinct values of μ for each machine. In the case of machine 0, the competitive ratio follows the same pattern whether jobs have mixed priorities (0 and 1) or only priority 0. Therefore, given this consistent pattern, we focus the remainder of our analysis on machine 0 with priority 0.

Next, we observe that for all algorithms, the results derived from the real dataset are worse than those obtained from the randomly generated inputs. This outcome is not surprising, as the random inputs were chosen within $\mu = \{1, 2, 5, 10, 100\}$. In Table 8.1, for all datasets that we run the algorithms, μ is much larger. As discussed in Section 7.5.3, the competitive ratio of all algorithms generally increases with μ, assuming other parameters remain constant.

We also observe that the differences in algorithm performance are more pronounced when compared to the random input data. For instance, the competitive ratio of *NextFit* on synthetic data with $d = 4$, $T = 1000$, and $\mu = 100$ is 1.87611, while the ratio for *FirstFit* is 1.53345. In contrast, on the real dataset, the ratio for *NextFit* is 4.483 on machine 0 with priority 0, whereas *FirstFit* has a ratio of 2.00454, nearly half that of *NextFit*. This highlights the critical importance of selecting the appropriate algorithm for real datasets, as choosing an ineffective algorithm can lead to significant costs for server providers.

We find that the algorithms *NextFit* and *FirstFit* share the same competitive ratio as their modified counterparts, *MFF* and *MNF*. As noted in Section 7.5.3, while the modified versions exhibit improved worst-case performance, they did not yield favorable results in experiments involving random data. In contrast, our analysis of real datasets indicates that these modified versions maintain the same competitive ratio as the original algorithms. This performance similarity suggests that the majority of jobs in the real dataset have sizes less than half of the server capacity (< 500), as the modified algorithms differentiate jobs based on whether their sizes exceed or fall below this threshold.

In investigating the hypothesis, for machine 0, we found that out of a total of $112,552$ jobs of priority 0, only 656 jobs (approximately 0.58%) had a core size greater than 500. Similarly, the number of jobs with memory exceeding 500 was also 656, constituting about 0.58% of the total. For SSD, only 514 jobs (around 0.46%) exceeded a size of 500, while for NIC, 637 jobs (approximately 0.57%) surpassed this threshold. These low percentages support our hypothesis about why *FirstFit* and *MFF*, as well as *NextFit* and *MNF*, perform similarly.

Interestingly, among the non-clairvoyant algorithms, *FirstFit* exhibits the strongest performance. Although *MoveToFront* excelled in experiments with random input sequences, it shows significantly poorer results compared to *FirstFit* in experiments with real data. We suspect that the restrictions on

the selection of μ in the random input may account for this discrepancy. Specifically, in the random input sequence, μ is chosen from the set $\{1, 2, 5, 10, 100\}$. To further investigate this hypothesis, we conduct additional tests in the following section.

The results also reveal that nearly all clairvoyant algorithms, with the exception of the *Departure Strategy*, perform exceptionally well. We suspect that the poor performance of the *Departure Strategy* is due to the large value of μ and the wide span, as it categorizes jobs based on their duration, which may not be effective under these conditions.

Although *Greedy* outperforms *Hybrid Algorithm* in experiments with random data, *Hybrid Algorithm* demonstrates an advantage when applied to real-world datasets. For example, in the random data experiments with $d = 5$ and $m = 1$, *Hybrid Algorithm* achieves a better average ratio of 1.42 compared to *Greedy*'s performance which is 1.45. Notably, this is the only case in the random data experiments where *Hybrid Algorithm* outperforms *Greedy*. For other values of μ, $\{2, 5, 10, 100\}$, *Greedy* consistently performs better.

However, on real data, *Hybrid Algorithm* consistently outperforms all other algorithms, including *Greedy*, the competitive ratio of *Hybrid Algorithm* is 1.29 while *Greedy* has ratio 1.64 . We hypothesize that this disparity arises because real datasets typically feature much higher μ values. This advantage can be attributed to *Hybrid Algorithm*'s strategy of grouping jobs with similar arrival and duration into the same servers. As discussed in Section 7.5.3 and noted in [45], jobs arriving at the same time are more likely to have similar durations. *Hybrid Algorithm* capitalizes on this property by efficiently assigning jobs with identical arrival times and durations to the same server.

In the following section, we explore this hypothesis further by restricting the jobs to smaller μ values to evaluate the impact on algorithm performance.

8.4 Experiments on μ-Filtered Data

We hypothesized that the difference in relative ratio between algorithms on real data and synthetic data are caused by much higher values of μ in real data. To evaluate our hypothesis, we filtered the dataset to include only jobs within a specified range determined by μ. The filtering process began by sorting all jobs in ascending order based on their duration. For each job in the sorted

list, we treated its duration as the lower bound of an interval. Specifically, for a job with duration d_{lower}, we considered the interval $[d_{\text{lower}}, \mu \cdot d_{\text{lower}}]$ and counted the number of jobs that fell within this range. By iterating through all jobs, we identified the interval that contained the maximum number of jobs.

Once the interval with the highest density of jobs was determined, we selected the jobs within this interval for experimentation. The analysis focused exclusively on jobs associated with machine 0 and priority 0, while excluding any jobs where μ exceeded the values in the set $\{1, 2, 5, 20, 100\}$. For machine 0 with priority 0, the initial number of jobs before applying the filter is $112,552$. The results of this experiment are presented in Table 8.3.

Algorithms	Machine 0, Priority 0, $\mu = 1$	Machine 0, Priority 0, $\mu = 2$	Machine 0, Priority 0, $\mu = 5$	Machine 0, Priority 0, $\mu = 10$	Machine 0, Priority 0, $\mu = 100$
Number of jobs	13	28,051	49133	55,382	73,215
	Non-clairvoyant				
NextFit	1	1.29	1.50	1.62	3.05
MNF	1	1.31	1.48	1.58	3.05
WorstFit	1	1.74	1.87	1.84	1.98
FirstFit	1	1.16	1.17	1.18	1.23
MFF	1	1.25	1.27	1.24	1.23
BestFit	1	1.21	1.22	1.23	1.31
LastFit	1	1.20	1.21	1.22	1.32
Random Fit	1	1.90	2.10	2.05	1.75
MoveToFront	1	1.16	1.20	1.22	1.40
	Clairvoyant				
Departure Strategy	1	11.26	12.18	10.76	8.55
Duration Strategy	1	1.16	1.20	1.24	1.24
Hybrid Algorithm	1	1.59	1.90	1.82	1.39
New Hybrid	1	1.60	1.93	1.84	1.42
Greedy	1	1.13	1.15	1.14	1.12

Table 8.3: Competitive ratio results for the *RSiC* problem on machine 0 for jobs with priority 0 after filtering the jobs based on μ.

As shown in Table 8.3, the ratios for various algorithms look similar to the results derived from the synthetic data, particularly for smaller values of μ. This trend suggests that imposing restrictions on μ can improve algorithm performance. However, certain algorithms, such as *NextFit*, *MNF*, and the *departure Strategy*, still exhibit significant deviations from the synthetic data results. This divergence may be attributed to the number of jobs and the large span value in the real data input, which is considerably larger than that in the synthetic data.

8.5 New Combined Algorithms

From the experiments described in Section 8.3, we can conclude that for datasets with larger values of μ and a large span, *Hybrid Algorithm* is the optimal choice. For smaller values of μ, *Greedy* performs better in the clairvoyant case. In the non-clairvoyant case, when μ is large, *FirstFit* is the preferred algorithm, while *MTF* generally performs well in the other case. Based on these observation, an interesting idea arises: what if we design an algorithm that dynamically selects between two existing algorithms based on a threshold? For example, such an algorithm could combine two algorithms A and B, where A is used for short duration of jobs while B is used for the jobs with long duration. By carefully selecting an appropriate threshold, incoming jobs could be assigned to the most appropriate algorithm. This raises a compelling question: could this combined approach surpass the performance of current algorithms?

To answer this question, we propose several new algorithms designed to enhance the performance of the packing process, surpassing all existing approaches in both clairvoyant and non-clairvoyant scenarios. For each algorithm, we categorize servers into two distinct types:

(1) Type A servers, which are used by algorithm A to pack jobs with durations less than a predefined threshold τ,

(2) Type B servers, which are used by algorithm B to pack jobs with durations greater than or equal to the threshold τ.

8.5.1 Combined Clairvoyant Algorithms

As shown in Table 8.2, all the existing clairvoyant algorithms except the *Departure Strategy* perform well on the real database. However, in this section, we want to achieve a better performance by proposing new algorithms that combine the existing algorithms based on the parameter τ.

The clairvoyant algorithms that we propose are as the following:

- ***Greedy-Hybrid***: In this algorithm, the *Greedy* algorithm is applied to type A and *Hybrid Algorithm* is applied to type B servers.

- ***Greedy-Greedy***: In this algorithm, the *Greedy* algorithm is applied to both type A and type B servers.

- ***Greedy-Duration***: In this algorithm, the *Greedy* algorithm is applied to type A and *Duration Strategy* is applied to type B servers.

We also introduce a new clairvoyant algorithm that differs slightly from the previous approach but still utilizes the parameter τ to allocate each job. The new algorithm is defined as follows:

- ***New Greedy***: If assigning a new job to the best server chosen by the *Greedy* algorithm causes its finishing time to exceed τ, the *New Greedy* algorithm opens a new server for the job. Otherwise, the job is packed into the server in the same way as the standard *Greedy* algorithm.

The main challenge lies in identifying the optimal value of τ to maximize the performance of these new algorithms. To tackle this, we employ a grid search approach, testing values of τ ranging from the shortest to the longest job duration. Initially, we set the range for τ to be one of $[10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^{10}]$ to determine whether a threshold exists that improves performance compared to the existing algorithms. Figure 8.4 compares the performance of these algorithms across various threshold settings with that of the *Hybrid Algorithm* and *Greedy* algorithms.

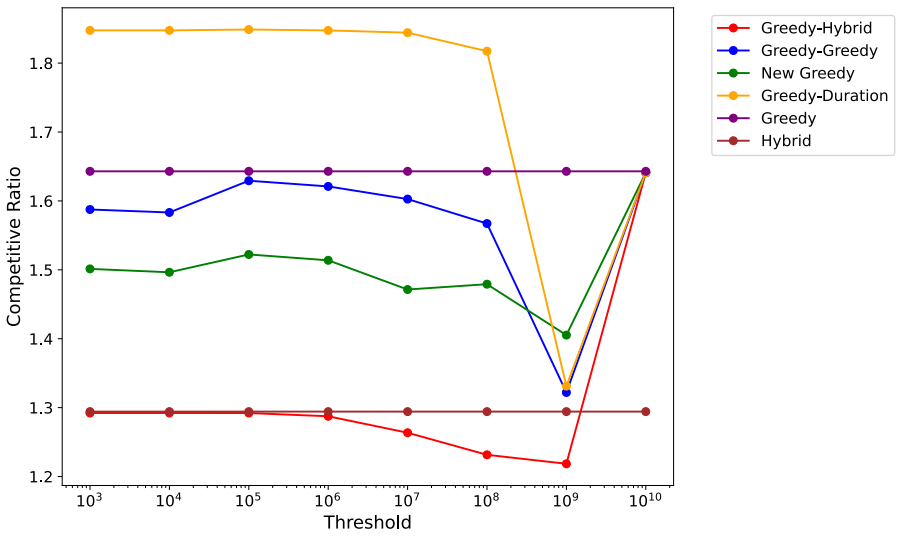

Figure 8.4: Performance of the new combined clairvoyant algorithms for $\tau \in [10^3, 10^{10}]$.

As shown in Figure 8.4, after $\tau = 10^6$, the *Greedy-Hybrid* algorithm achieves competitive ratios better than *Hybrid Algorithm*, with the best ratio occurring at $\tau = 10^9$, where it reaches 1.21. At this threshold, the competitive ratios of *Greedy-Greedy*, *New Hybrid*, and *Greedy-Duration* are also very close to that of *Hybrid Algorithm*. This is noteworthy, as these algorithms are significantly simpler compared to *Hybrid Algorithm* while maintaining strong performance at this threshold.

We hypothesize that this threshold serves as the dividing point between two categories of jobs. Jobs with durations exceeding this threshold tend to increase the algorithm's cost, as they are packed together with shorter-duration jobs. As shown in Figure 8.5, the majority of jobs have short durations, while only a few have significantly longer durations. In this figure, you can see how this threshold compares to the overall distribution of job durations. Specifically, for this threshold, almost all the jobs, 99.76% ($112,277$ out of $112,552$)—fall below it, while only 0.24% (275 out of $112,552$)—exceed it. The distribution for the other thresholds can be found in Table 8.4.

Threshold	*Jobs Below Threshold*	*Jobs Exceeding Threshold*
10^3	0.01%	99.99%
10^4	2.02%	97.98%
10^5	19.39%	80.61%
10^6	31.22%	68.78%
10^7	79.21%	20.79%
10^8	95.72%	4.28%
10^9	99.76%	0.24%
10^{10}	100%	0%

Table 8.4: Job distribution for different thresholds on job durations.

Consequently, we focus on τ values around this interval to explore the potential for achieving even better ratios. As a result, we select τ to range from 10^8 to 10^{10}, with increments of 10^8. Therefore, we aim to evaluate a total of 100 points to observe any improvements, See Figure 8.6.

Based on these experiments, we observe that the *Greedy-Hybrid* algorithm achieves a competitive ratio of 1.18 at a threshold of $\tau = 400,000,000$, outperforming *Hybrid Algorithm*. Similarly, both the *Greedy-Greedy* and *Greedy-Duration* algorithms achieve a desirable competitive ratio of 1.28 at $\tau = 2,500,000,000$. Additionally, the *New Greedy* algorithm achieves a competitive ratio of 1.41 at $\tau = 1,100,000,000$. These results are particularly impressive, especially for *Greedy-Greedy*, as it is a very simple algorithm to implement and achieves a competitive ratio better than

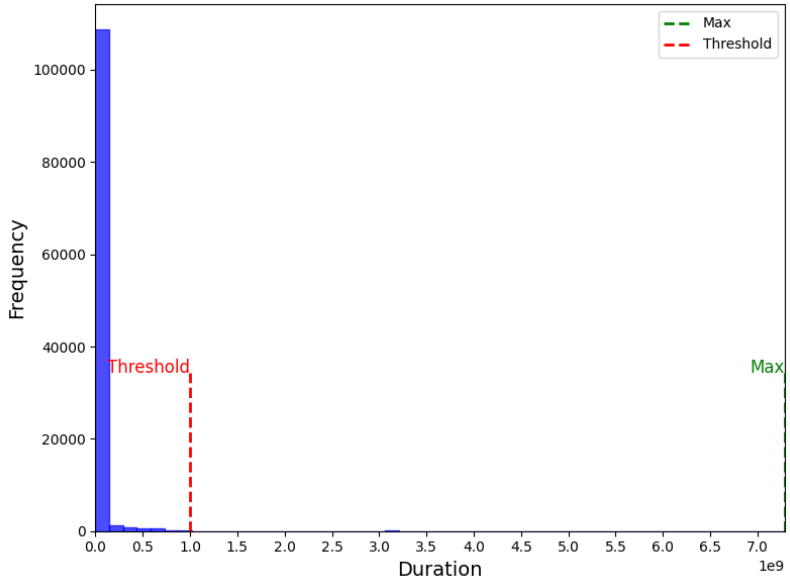

Figure 8.5: Distribution of jobs compared to the threshold. Note that 99.76% of jobs fall below threshold 10^9, while only 0.24% exceed it.

Hybrid Algorithm, which aligns with our objectives.

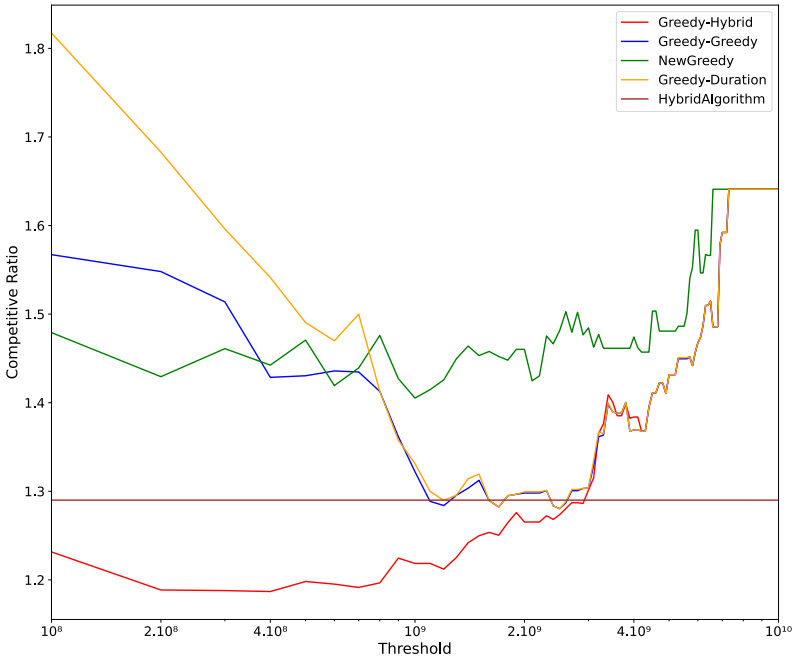

Figure 8.6: Comparison between the competitive ratio of the new clairvoyant combined algorithms.

8.5.2 Combined Weakly Clairvoyant Algorithms

As shown in Table 8.2, both *FirstFit* and *BestFit* consistently achieve the best competitive ratios in nearly all scenarios within the non-clairvoyant setting. Based on this observation, we propose new algorithms that combine *FirstFit* and *BestFit* using the parameter τ. In this setting, jobs are classified as long or short by the user, and the scheduler does not require knowledge of their actual durations at the time of arrival. Consequently, this setting is neither fully clairvoyant nor fully non-clairvoyant, which we refer to as weakly clairvoyant. Therefore, the algorithms we propose are weakly clairvoyant. The algorithms we introduce for this setting are as follows:

- ***FirstFit-FirstFit***: In this algorithm, the *FirstFit* algorithm is applied to both type A and type B servers, where jobs are assigned according to whether their durations are smaller or larger than the threshold τ.
- ***FirstFit-BestFit***: In this algorithm, the *FirstFit* algorithm is applied to type A and *BestFit* is applied to type B servers.
- ***BestFit-BestFit***: In this algorithm, the *BestFit* algorithm is applied to both type A and type B servers.
- ***BestFit-FirstFit***: In this algorithm, the *BestFit* algorithm is applied to type A and *FirstFit* is applied to type B servers.

For these algorithms, we adopted the range of τ used in the experiments with clairvoyant algorithms, setting τ to vary between 10^8 and 10^{10}. Figure 8.7 presents the competitive ratios of the combined algorithms for various thresholds within this range. As shown in the figure, the performances of *FirstFit-BestFit* and *FirstFit-FirstFit* are nearly identical, as are those of *BestFit-FirstFit* and *BestFit-BestFit*. The results indicate an improvement over all existing non-clairvoyant algorithms. Notably, setting $\tau = 1,300,000,000$ allows *FirstFit-FirstFit* to achieve a competitive ratio of 1.41 and *FirstFit-BestFit* to reach 1.42, representing a significant improvement over the algorithms listed in Table 8.2. Similarly, *BestFit-BestFit* and *BestFit-FirstFit* achieve competitive ratios of 1.45 when τ is set to $1,300,000,000$ and $1,000,000,000$, respectively.

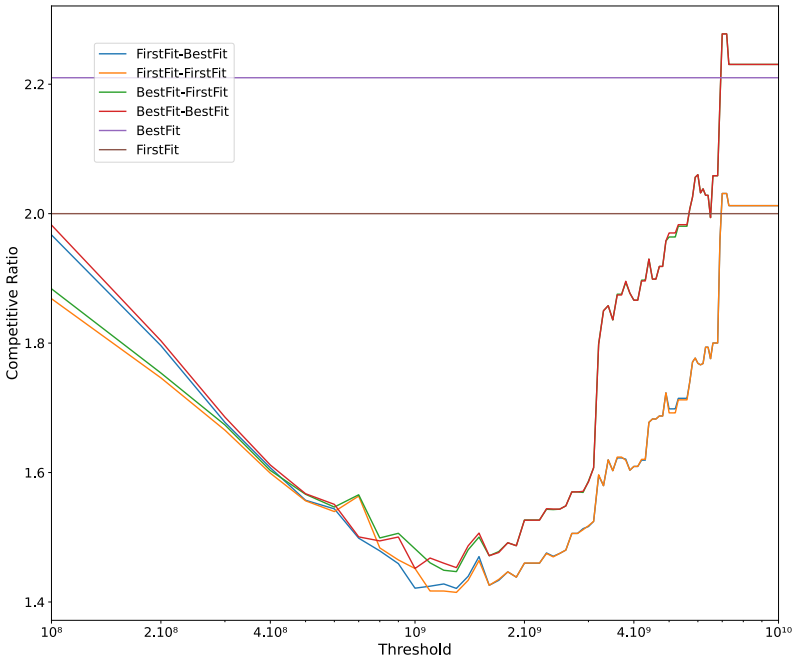

Figure 8.7: The performance of the weakly clairvoyant combined algorithms over different values of τ.

8.6 Wasted-Space and Duration of Servers

In this section, we aim to understand why different algorithms exhibit varying levels of performance, why some perform well while others do not. Our analysis will consider both clairvoyant and non-clairvoyant scenarios. As outlined in Chapter 1, the total cost of an algorithm for the *RSiC* problem is determined by the total cost of all servers utilized by the algorithm, where the cost of each server is proportional to its duration. Consequently, the performance of an algorithm is generally a linear function of the number of servers it uses and their respective durations. It is evident that the number of servers utilized by an algorithm depends on how efficiently jobs are packed into the servers. In the rest of this section, we analyze how these factors influence algorithm performance.

8.6.1 Clairvoyant Algorithms

As shown in Table 8.2, among the clairvoyant algorithms, *Hybrid Algorithm* and *New Hybrid* demonstrate superior performance, while the *Departure Strategy* performs poorly, and *Greedy*

shows moderate performance. To provide a clearer understanding and comparison of these algorithms, Table 8.5 and Figure 8.8 present the number of servers utilized by each algorithm and the corresponding server durations. In Figure 8.8, the servers are first sorted in ascending order by duration to create the plot. The x-axis represents the cumulative number of servers, while the y-axis displays the corresponding sorted duration values. This visualization helps us understand how frequently servers share the same duration, allowing us to assess how effectively the algorithm optimizes server allocation.

Algorithms	*Greedy*	*Hybrid Algorithm*	*New Hybrid*	*Duration Strategy*	*Departure Strategy*
Number of Servers	$1,078$	$24,322$	$16,796$	$4,953$	$76,893$

Table 8.5: Number of servers that each clairvoyant algorithm used on the full dataset.

From Table 8.5 and Figure 8.8, we observe that *Departure Strategy* employs the highest number of servers, nearly $76,000$ in total. Most of these servers have short durations, but there is a subset with significantly long durations, contributing to its poor performance. In contrast, *Greedy* utilizes far fewer servers—approximately $1,000$—which is significantly less than the number used by the *Departure Strategy*. On the other hand, the *Hybrid Algorithm* algorithm, which achieves the best performance among clairvoyant algorithms, uses around $24,000$ servers. Notably, only a small portion of these servers have long durations. This efficient balance of server usage and duration likely explains why *Hybrid Algorithm* outperforms the other algorithms.

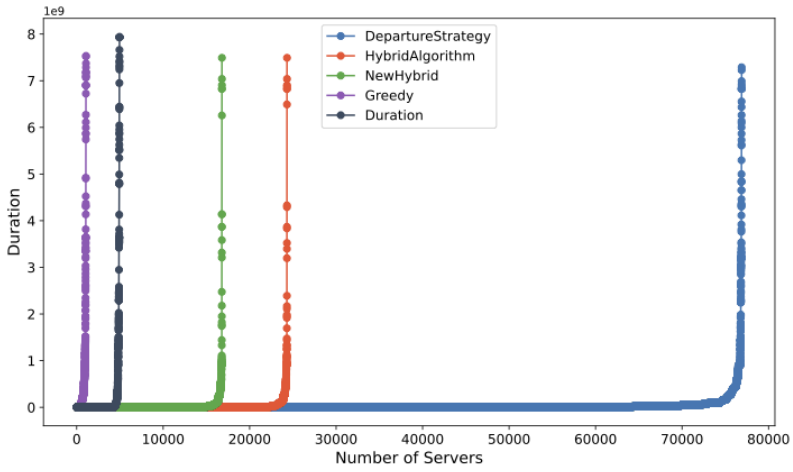

Figure 8.8: Duration of servers for clairvoyant algorithms.

We observe that while *Greedy* uses fewer servers than *Hybrid Algorithm*, its performance is significantly worse. Why is this the case? To answer this question, we conducted additional experiments to compare how tightly these algorithms pack jobs into each server.

In our database, each job is defined in four dimensions: core, memory, NIC, and SSD. In Figure 8.9, we plot the cumulative distribution wasted space for each dimension across the servers used by each algorithm. The x-axis represents the cumulative number of servers, while the y-axis displays the corresponding sorted wasted space values for each dimension. From these plots, we can see that the servers utilized by *Hybrid Algorithm* have significantly less wasted space across all dimensions. This indicates that *Hybrid Algorithm* achieves tighter packing of jobs, which contributes to its superior performance.

On the other hand, while *Greedy* 's performance is worse than *Hybrid Algorithm*, it performs better than *Departure Strategy*. This is because *Greedy* exhibits less wasted space in all dimensions—core, memory, NIC, and SSD—compared to *Departure Strategy*. The tighter packing of jobs in *Greedy* leads to improved performance relative to *Departure Strategy*, even though it still falls short of the efficiency achieved by *Hybrid Algorithm*. Finally, algorithms like *Departure Strategy* show significantly higher wasted space across all dimensions, resulting in poor utilization of server resources and the lowest overall performance.

8.6.2 Non-clairvoyant Algorithms

Now, let us analyze the performance of the non-clairvoyant algorithms. From Table 8.2, we observe that among all the non-clairvoyant algorithms, *FirstFit* demonstrates the best performance, while *NextFit* performs the worst. Why is this the case?

To investigate, we conducted the same experiments as we did for the clairvoyant algorithms, comparing the number of servers used and their durations across all non-clairvoyant algorithms. The results, summarized in Table 8.6 and illustrated in Figure 8.10, highlight the differences in server utilization and duration across these algorithms.

Algorithms	*NextFit*	*MNF*	*WorstFit*	*FirstFit*	*MFF*	*BestFit*	*LastFit*	*Random Fit*	*MoveToFront*
Number of Servers	6,549	6,549	971	709	709	446	401	362	442

Table 8.6: Number of servers that each non-clairvoyant algorithm used on the full dataset.

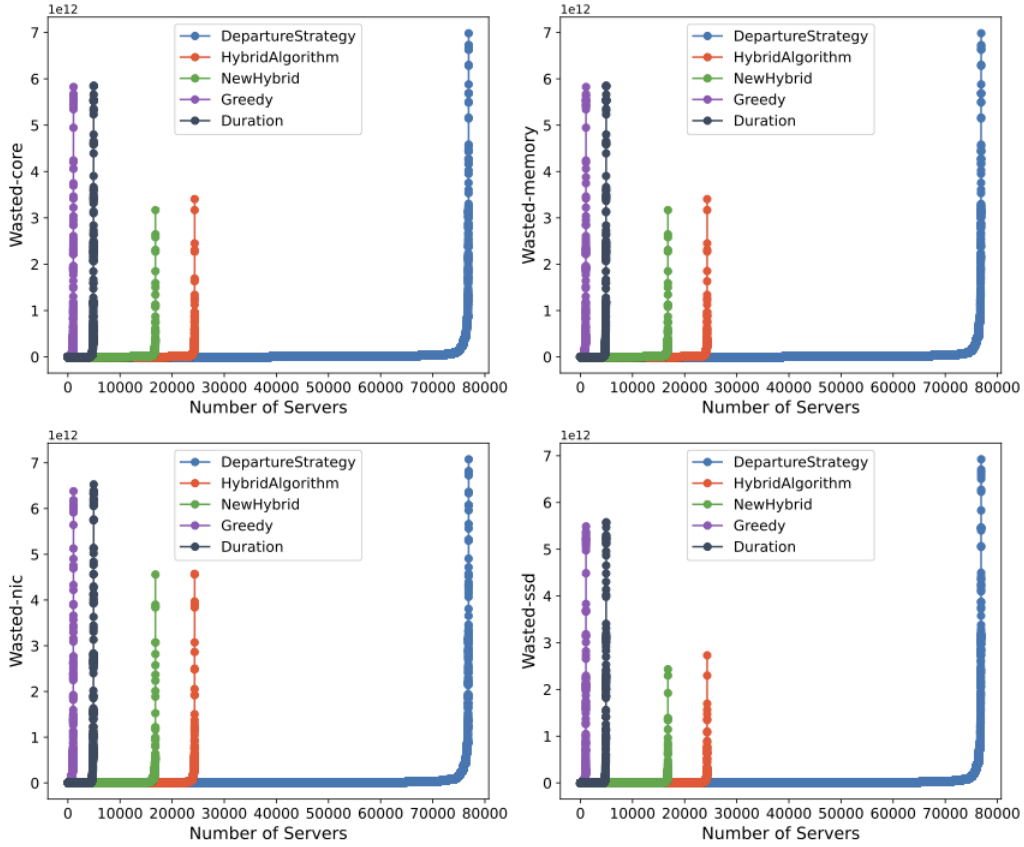

Figure 8.9: Wasted-space of servers for clairvoyant algorithms.

From this figure, we observe that *NextFit* utilizes nearly 6500 servers in total. While most of these servers have a short duration, there is a subset with very long durations, which negatively impacts its performance. In contrast, algorithms like *FirstFit* and *MFF* use significantly fewer servers—around 700 in total—which is considerably less compared to *NextFit*.

As shown in Table 8.2, the performance of almost all non-clairvoyant algorithms, except for *NextFit* and *MNF*, is relatively close, with *FirstFit* outperforming the rest. To understand why *FirstFit* performs better, we compare these algorithms from the perspective of wasted space across each dimension. The corresponding wasted resources are depicted in Figure 8.11.

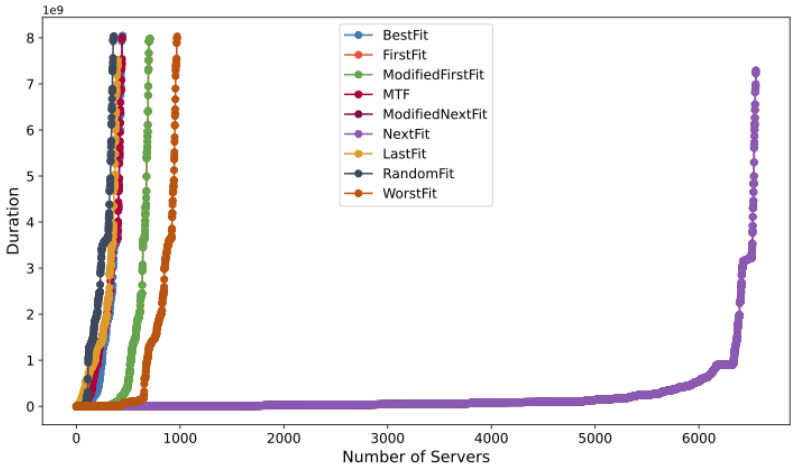

Figure 8.10: Duration of servers for non-clairvoyant algorithms. Note that due to overlapping values, *MFF* hides *FirstFit*, and *NextFit* similarly hides *MNF*.

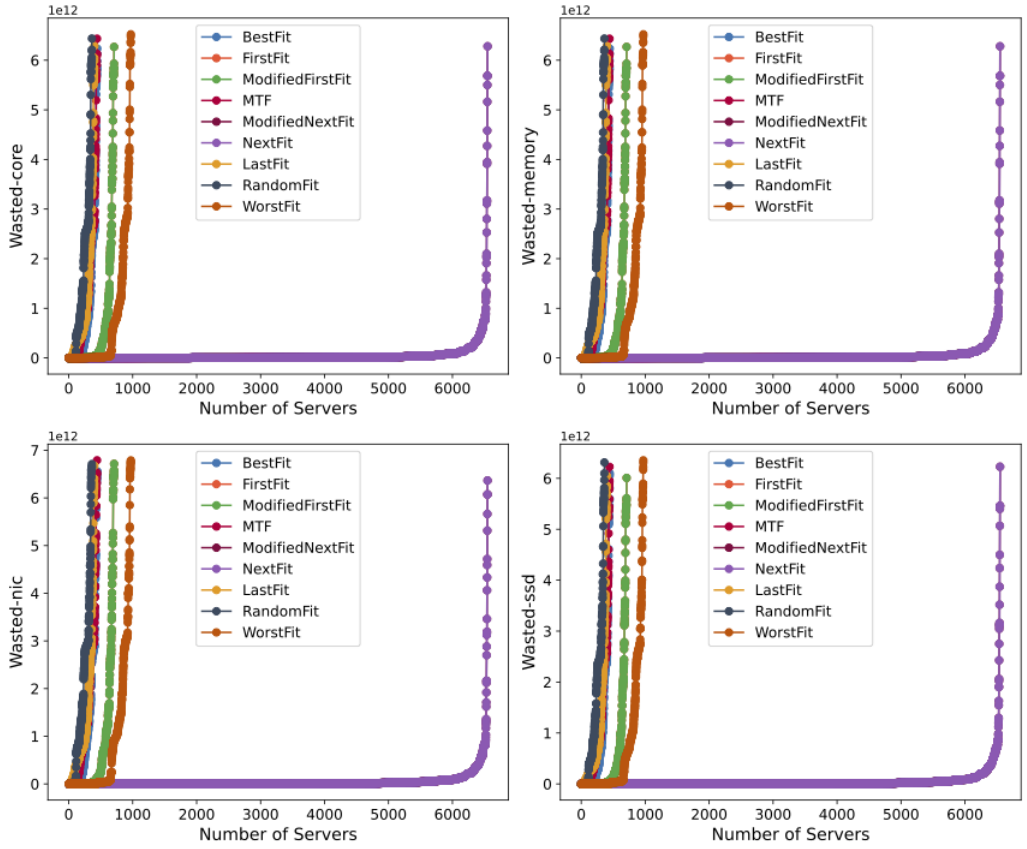

Figure 8.11: Wasted-space of servers for non-clairvoyant algorithms. Note that due to overlapping values, *MFF* hides *FirstFit*, and *NextFit* similarly hides *MNF*.

8.7 Summary and Discussion

In this chapter, we evaluated the performance of existing clairvoyant and non-clairvoyant algorithms for the *RSiC* problem using real-world datasets. To the best of our knowledge, this is the first study to analyze the *RSiC* problem with real-world data. We also proposed new algorithms that demonstrate superior performance in experimental evaluations. Our analysis provides valuable insights into the effectiveness of these algorithms and establishes a benchmark to guide future research in this area.

Recall that in the clairvoyant setting job durations are known at the time of job arrivals; in the weakly clairvoyant setting, jobs are categorized as long or short duration before arrival at the scheduler; and in non-clairvoyant setting nothing is known about the job durations at the time of arrivals. Our experiments with real world data confirm that the clairvoyant algorithms tend to perform better than the weakly clairvoyant algorithms which in turn tend to perform better than non-clairvoyant algorithms.

In the clairvoyant setting, *Hybrid Algorithm* emerged as the most effective algorithm. However, *Greedy* stands out as a simpler alternative, avoiding the complexity of *Hybrid Algorithm*, which involves server categorization and extensive server usage. By combining the algorithms we were able to improve on the competitive ratio of *Hybrid Algorithm*. In particular, *Greedy-Hybrid* exhibits the best performance of all known algorithms including *Hybrid Algorithm* in terms of empirically derived competitive ratio but is as difficult to implement. Consequently, *Greedy-Greedy* which also beats *Hybrid Algorithm* offers a more practical solution.

In the weakly clairvoyant setting, *FirstFit-FirstFit* and *FirstFit-BestFit* have similar performance and improve upon the empirically derived competitive ratio of both *FirstFit* and *BestFit*. Since *BestFit* has infinite competitive ratio in the worst case, we recommend using *FirstFit-FirstFit* as a safer alternative.

In the non-clairvoyant setting, *FirstFit* exhibited the strongest performance overall. While *MoveToFront* excelled in experiments with random input sequences, its performance with real-world data was significantly weaker than that of *FirstFit*. Based on these findings, we recommend using *MoveToFront* for datasets with small μ values and *FirstFit* for datasets with larger μ values.

Chapter 9

Conclusion

In this thesis, we investigated the *RSiC* problem, motivated by job allocation to servers in cloud computing applications. This problem generalizes the bin packing problem by introducing additional constraints. Bin packing is one of the fundamental problems in combinatorial optimization with many practical applications, such as cutting stock problem and other problems alike. Despite 50 years of research, bin packing remains a challenging problem. Yet, it represents only a restricted version of the *RSiC* problem. Given the difficulty of *RSiC*, this thesis tackled the problem by first exploring simpler variations.

First, we dealt with the case of *equal job durations* and *two arrival times*. We analyzed two well-known algorithms, *NextFit* and *FirstFit*, and established bounds on their performance. We proved a tight bound for competitive ratio of *NextFit* in this regime and in fact for the general case of equal duration of jobs. For *FirstFit* we derived a new upper bound by applying the weight function technique to the *RSiC* problem. This was the first time that this technique was applied to the *RSiC* problem. Building on this, we extended our analysis to the long-running servers version of the problem and established upper bounds for this case.

Next, we explored another variation where *servers are dual-core*, *jobs arrive at integer times*, and *jobs have equal duration of* 2. For such inputs, the states of ALG and OPT can be described and analyzed using a finite state machine. This enables us to show a tight bound on the competitive ratio of *AnyFit* algorithms. By a reduction from the binary guessing problem to this problem, we demonstrated that sub-linear advice is not enough to solve the problem near optimally.

We also addressed the *RSiC* problem with *limited number of servers*, introducing a novel interval partitioning technique to establish tight competitive ratio bounds. This technique shows potential for application in broader, unrestricted cases. Lastly, we formalized the notion of *instance-dominance* and demonstrated that no single known algorithm is instance-dominant across all possible inputs.

The following questions remain open for these simpler variations of *RSiC*:

- For the case of equal duration of jobs, we improved the lower bound to 2.519 while the upper bound for *FirstFit* is $\mu + 1 = 4$. Is it possible to close this gap?
- Is it possible to generalize the results for the dual-core server setting to the k-core server setting?
- Is it possible to extend the analysis of the upper bound for the case with at most 4 servers to improve the upper bound on the competitive ratio of *FirstFit* in the $k \geq 5$ server case?

Another approach we took to address the *RSiC* problem was to consider it in the d-dimensional setting for both clairvoyant and non-clairvoyant algorithms. In this setting, we conducted a comprehensive study, establishing a direct sum property that transforms 1-dimensional algorithms into d-dimensional algorithms with competitive ratios scaled by d. We proposed and analyzed the *Greedy* algorithm, *Greedy* assigns the current job into the server that incurs the least additional cost. We introduced a new analytical technique that extended to a broader class of monotone algorithms, including *FirstFit*, *LastFit*, and *MoveToFront*.

In the d-dimensional setting, the following is a list of open problems that we found interesting for the possible future works:

- Does there exist a randomized algorithm for 1-dimensional clairvoyant *RSiC* with a competitive ratio of $O(1)$?
- For the d-dimensional *RSiC* problem in the non-clairvoyant setting, the lower bound for any *AnyFit* algorithm is $(\mu + 1)d$, and the upper bound is $2\mu d + 1$. Is it possible to close the gap between the upper and lower bounds?
- For d-dimensional clairvoyant *RSiC*, is $\Omega(d\sqrt{\log \mu})$ a lower bound for *any* algorithm?

To evaluate the performance of different algorithms under average-case scenarios, we conducted a comprehensive experimental study of this problem. The experiments utilized both synthetic random data and real-world datasets. Notably, this study is the first to assess algorithmic performance on real-world datasets for the *RSiC* problem, offering valuable insights into its practical applications. We have identified the best performing algorithms for each of the following three settings: clairvoyant, weakly clairvoyant, and non-clairvoyant.

In conclusion, our research introduced novel algorithms, established new performance bounds, and advanced analytical techniques to deepen the theoretical understanding of the *RSiC* problem. Additionally, we conducted comprehensive experimental studies to evaluate algorithm performance across diverse scenarios, offering both theoretical insights and practical guidance for real-world applications.

Bibliography

[1] Y. Azar and D. Vainstein. Tight bounds for clairvoyant dynamic bin packing. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–21, 2019.

[2] Y. Azar, I. R. Cohen, S. Kamara, and B. Shepherd. Tight bounds for online vector bin packing. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, pages 961–970, 2013.

[3] Y. Azar, I. R. Cohen, A. Fiat, and A. Roytman. Packing small vectors. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1511–1525. SIAM, 2016.

[4] Y. Azar, I. R. Cohen, and A. Roytman. Online lower bounds via duality. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1038–1050. SIAM, 2017.

[5] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new and improved algorithm for online bin packing. *arXiv preprint arXiv:1707.01728*, 2017.

[6] J. Balogh, I. R. Cohen, L. Epstein, and A. Levin. Truly asymptotic lower bounds for online vector bin packing. In M. Wootters and L. Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, volume 207 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[7] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions:

inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006.

[8] N. Bansal, A. Caprara, and M. Sviridenko. A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM Journal on Computing*, 39(4):1256–1278, 2010.

[9] N. Bansal, M. Eliáš, and A. Khan. Improved approximation for vector bin packing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1561–1579. SIAM, 2016.

[10] H. J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In *International Symposium on Algorithms and Computation*, pages 331–340. Springer, 2009.

[11] J. Boyar, G. Dósa, and L. Epstein. On the absolute approximation ratio for first fit and related results. *Discrete Applied Mathematics*, 160(13-14):1914–1923, 2012.

[12] J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen. Online algorithms with advice: A survey. *ACM Computing Surveys (CSUR)*, 50(2):1–34, 2017.

[13] D. J. Brown. A lower bound for on-line one-dimensional bin packing algorithms. Technical report, University of Illinois at Urbana-Champaign, 1979.

[14] N. Buchbinder, Y. Fairstein, K. Mellou, I. Menache, and J. Naor. Online virtual machine allocation with lifetime and load predictions. *ACM SIGMETRICS Performance Evaluation Review*, 49(1):9–10, 2021.

[15] H. J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, and A. Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theoretical Computer Science*, 554:95–108, 2014.

[16] J. W. T. Chan, T. W. Lam, and P. W. Wong. Dynamic bin packing of unit fractions items. *Theoretical Computer Science*, 409(3):521–529, 2008.

[17] B. Chandra. Does randomization help in on-line bin packing? *Information Processing Letters*, 43(1):15–19, 1992.

[18] C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM Journal on Computing*, 33(4):837–851, 2004.

[19] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[20] E. G. Coffman, Jr, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 12(2):227–258, 1983.

[21] E. G. Coffman Jr, J. Csirik, L. Rónyai, and A. Zsbán. Random-order bin packing. *Discrete Applied Mathematics*, 156(14):2810–2816, 2008.

[22] D. Coppersmith and P. Raghavan. Multidimensional on-line bin packing: algorithms and worst-case analysis. *Operations Research Letters*, 8(1):17–20, 1989.

[23] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[24] S. Dobrev, R. Královič, and D. Pardubská. Measuring the problem-relevant information in input. *RAIRO-Theoretical Informatics and Applications*, 43(3):585–613, 2009.

[25] G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is $ffd(i) \leq \frac{11}{9}opt(i) + \frac{6}{9}$. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007.

[26] G. Dósa and J. Sgall. First fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2013.

[27] G. Dósa and J. Sgall. Optimal analysis of best fit bin packing. In *International Colloquium on Automata, Languages, and Programming*, pages 429–441. Springer, 2014.

[28] L. Epstein. On variable sized vector packing. *Acta Cybernetica*, 16(1):47–56, 2003.

[29] U. Faigle, W. Kern, and G. Turán. On the performance of on-line algorithms for partition problems. *Acta cybernetica*, 9(2):107–119, 1989.

[30] W. Fernandez de La Vega and G. S. Lueker. Bin packing can be solved within 1+ ε in linear time. *Combinatorica*, 1(4):349–355, 1981.

[31] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.

[32] G. Galambos, H. Kellerer, and G. J. Woeginger. A lower bound for on-line vector-packing algorithms. *Acta Cybernetica*, 11(1-2):23–34, 1993.

[33] M. R. Garey, R. L. Graham, and J. D. Ullman. Worst-case analysis of memory allocation algorithms. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, pages 143–150, 1972.

[34] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C.-C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[35] M. R. Gary and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[36] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, et al. Protean:{VM} allocation service at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 845–861, 2020.

[37] X. Han, C. Peng, D. Ye, D. Zhang, and Y. Lan. Dynamic bin packing with unit fraction items revisited. *Information Processing Letters*, 110(23):1049–1054, 2010.

[38] A. Hebbar, A. Khan, and K. Sreenivas. Bin packing under random-order: Breaking the barrier of 3/2. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4177–4219. SIAM, 2024.

[39] B. Hiller and T. Vredeveld. Probabilistic alternatives for competitive analysis. *Computer Science-Research and Development*, 27(3):189–196, 2012.

[40] R. Hoberg and T. Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2625. SIAM, 2017.

[41] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[42] D. S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, 1974.

[43] D. S. Johnson and M. R. Garey. A 71/60 theorem for bin packing. *Journal of complexity*, 1 (1):65–106, 1985.

[44] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[45] S. Kamali and A. López-Ortiz. Efficient online strategies for renting servers in the cloud. In *Proceedings of SOFSEM 2015: Theory and Practice of Computer Science: 41st International Conference on Current Trends in Theory and Practice of Computer Science*, pages 277–288. Springer, 2015.

[46] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 312–320. IEEE, 1982.

[47] H. Kellerer and V. Kotov. An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing. *Operations Research Letters*, 31(1):35–41, 2003.

[48] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985.

[49] A. Levin. Comparing the costs of Any Fit algorithms for bin packing. *Operations Research Letters*, 50(6):646–649, 2022.

[50] Y. Li, X. Tang, and W. Cai. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 2–11, 2014.

[51] Y. Li, X. Tang, and W. Cai. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 2–11, 2014.

[52] Y. Li, X. Tang, and W. Cai. Dynamic bin packing for on-demand cloud resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):157–170, 2015.

[53] F. M. Liang. A lower bound for on-line bin packing. *Information processing letters*, 10(2): 76–79, 1980.

[54] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[55] G. S. Lueker. An average-case analysis of bin packing with uniformly distributed item sizes. Technical report, UC Irvine: Donald Bren School of Information and Computer Sciences, 1982.

[56] A. Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8 (4):265–368, 2013. URL http://dx.doi.org/10.1561/0400000057.

[57] A. Murhekar, D. Arbour, T. Mai, and A. B. Rao. Brief announcement: Dynamic vector bin packing for online resource allocation in the cloud. *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 307–310, 2023.

[58] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, 1989.

[59] R. Ren. *Combinatorial algorithms for scheduling jobs to minimize server usage time*. PhD thesis, 2018.

[60] R. Ren and X. Tang. Clairvoyant dynamic bin packing for job scheduling with minimum server usage time. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 227–237, 2016.

[61] R. Ren, X. Tang, Y. Li, and W. Cai. Competitiveness of dynamic bin packing for online cloud server allocation. *IEEE/ACM Transactions on Networking*, 25(3):1324–1331, 2016.

[62] M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1-3):203–227, 1991.

[63] T. Rothvoß. Approximating bin packing within $\mathcal{O}(\log \mathrm{OPT} \cdot \log \log \mathrm{OPT})$ bins. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 20–29. IEEE, 2013.

[64] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics (NRL)*, 41(4):579–585, 1994.

[65] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[66] X. Tang, Y. Li, R. Ren, and W. Cai. On first fit bin packing for online cloud server allocation. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 323–332, 2016.

[67] J. Ullman. The performance of a memory allocation algorithm. Technical report, Princeton University. Department of Electrical Engineering. Computer Science Laboratory, 1971.

[68] R. Van Bevern, A. Melnikov, P. V. Smirnov, and O. Y. Tsidulko. On data reduction for dynamic vector bin packing. *Operations Research Letters*, 51(4):446–452, 2023.

[69] A. Van Vliet. An improved lower bound for on-line bin packing algorithms. *Inf. Process. Lett.*, 43(5):277–284, 1992.

[70] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[71] G. J. Woeginger. There is no asymptotic ptas for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.

[72] P. W. Wong, F. C. Yung, and M. Burcea. An 8/3 lower bound for online dynamic bin packing. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 44–53. Springer, 2012.

[73] B. Xia and Z. Tan. Tighter bounds of the first fit algorithm for the bin-packing problem. *Discrete Applied Mathematics*, 158(15):1668–1675, 2010.

[74] A. C. C. Yao. New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2):207–227, 1980.