Adaptive Resource Allocation in High-Performance Distributed Multimedia Computing

Ran, Yang, 1978 –
Adaptive Resource Allocation in High-Performance Distributed Multimedia Computing
ISBN: 978-90-8570-739-4

Printed by CPI Wöhrmann Print Service, The Netherlands.

VRIJE UNIVERSITEIT

Adaptive Resource Allocation in High-Performance Distributed Multimedia Computing

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op maandag 23 mei 2011 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Ran Yang

geboren te Zhejiang, China

promotor:	prof.dr. R.D. van der Mei
copromotoren:	dr. S. Bhulai
	dr. F.J. Seinstra

Acknowledgements

This thesis is the result of a four-year Ph.D. studies at the Vrije Universiteit (VU) in Amsterdam. Standing on the threshold of a new stage in life, I can look back at a fantastic and also very instructive period at the VU.

These years of research and also the writing of this thesis could never have been successful without the help and support of several people, some to the scientific content, and others through friendship and support. Therefore, I would like to take this opportunity to express my gratitude to them for their support and valuable advice.

First of all I would like to express my deep acknowledgement and gratitude to my supervisors Rob van der Mei and Ger Koole for giving me the opportunity to become a Ph.D. student for this research project, which was funded by the Netherlands Organisation for Scientific Research (NWO). In addition, I am deeply grateful to Rob van der Mei for all the insight, encouragement, guidance and leadership he has given to me during my studies and while writing this thesis.

Besides, I would like to thank Sandjai Bhulai for discussions on my research work and giving me valuable suggestions on numerous occasions. Moreover, his infectious enthusiasm has certainly had its positive effect on my work. In addition, I also thank him for the great and pleasant cooperation.

Furthermore, I am grateful to Dennis Roubos for his continuous advice, useful input and helping me to solve the problems in writing simulation programs. His useful comments have certainly contributed to an improved quality of this work. Besides, it was very nice to work together with him.

During my research, I have also cooperated with many great people at the Department of Computer Science. Henri Bal, Frank Seinstra and Jason Maassen, thank you very much for the fruitful cooperation and the realisation of this thesis.

I am also indebted to all colleagues, without mentioning names, for the nice period we have spent together.

This thesis is also dedicated to two of my middle school teachers, Dongling Pan and Jianwei Yang who taught me a good study habit.

Last but not least, I would like to express my sincere appreciation to my family and relatives for supporting and encouraging me during all those years. In particular, I would like to thank my mother, Shulian Fan, and my father, Renlin Yang, for teaching me to sense the joy of pursuance of truth and enrich our understanding ever since my childhood. I would thank my brothers Jianguang, Jianjun and my sister Nianhua for providing a loving company. I would thank my mother-in-law, Guixiang Liang for taking care of my sons during my Ph.D. studies. Most of all, I want to thank my husband Dongsheng Zhao and my two sons Chengji and Xingqi for providing me a nice family.

Ran Yang
December 2010

Contents

Chapter 1

Introduction

In recent years, the increasing importance of multimedia data, in particular in the form of still pictures and video, has boosted demands for automatic extraction, comparison, and processing of features from such data - and has led to the new research domain of Multimedia Content Analysis (MMCA). In the very near future, computerized access to the content of multimedia data will be a problem of phenomenal proportions, as digital video may produce data at extremely high rates, and multimedia archives steadily run into petabytes of storage [4]. At the same time, applications in MMCA must often run under strict time constraints, even under variable workloads, data-dependent computation, and dynamic resource utilization. As individual computers cannot satisfy the high computational demands, distributed supercomputing on large collections of compute clusters (Grids) is rapidly becoming indispensable.

1.1 Multimedia Content Analysis and Grid Computing

The application and research area of MMCA considers all aspects of the automated extraction of new knowledge from large multimedia data streams and archives. In part, the MMCA domain is driven by the requirements of emerging applications, ranging from the automatic comparison of forensic video evidence, to searching publicly available digital television archives, and real-time analysis of video data obtained from surveillance cameras in public locations [89].

Due to the increasing storage of multimedia data, the challenge is to discover and interpret tiny fractions of useful information in a whirlwind of meaningless noise. The following two statistics can provide us with an intuition

about its scale: there were about 121.4 million digital still cameras sold in 2010 [1], and video already accounts for more than half of the Internet traffic, with YouTube along taking 10% [108]. As another example, the Aqua instruments (the earth observing system) produce more than 750 gigabytes of data per day for analysis and processing [5].

A complicating factor is that applications in MMCA often must run under strict time constraints. For example, to avoid delays in queues of people waiting, a biometric authentication system must identify a person's identity within several seconds. Large autonomous applications, such as the automatic detection of suspect behavior in video data obtained from surveillance cameras, may even need to work under real-time restrictions.

One way to deal with this complication is to apply approximate algorithms, albeit at the expense of accuracy and the loss of useful access results. A better solution is to exploit distribution of data and computation over compute networks. Also, in many emerging MMCA problems, the generation, processing, storage, indexing, querying, retrieval, and visualization of multimedia data are integrated aspects, all taking place at the same time and - potentially - at multiple administrative domains. This increasing need for computing and storage resources can be satisfied only by adopting techniques from Grid computing.

Grid computing [33] enables the sharing, selection, and aggregation of a wide variety of geographically distributed computational resources (such as supercomputers, compute clusters, and storage systems) for solving large-scale resource-intensive problems in science, engineering, and commerce. This approach to computing can be most simply thought of as a massively large power "utility" grid, such as the electric power network where power generators are distributed. The users are able to access the compute power without bothering about the source of resources and its location.

Grid computing is not only expected to satisfy the high computational demands and the strict time constraints of MMCA applications, but also to popularize a model of computational economy [18]. It creates a market for service providers to provide computational resource rental services to consumers who have this requirement. Whenever an MMCA application needs to be processed by extensive computational resources, the consumers can hire additional resources from service providers. The consumers are interested in minimizing the cost for what they use, whilst the service providers aim to maximize the profit for what they supply. The balance point of the re-

source price may vary from time to time and from one pair of consumers and providers to another. This raises the need for generating optimal resource-allocation policies for low-cost or high-profit access for computational service consumers or providers.

1.2 Problem description

In a services-based execution scenario, a client program (typically a local desktop computer) connects to one or more remote multimedia servers, each running on a (different) compute cluster. At application run-time, the client application sends video/audio frames (e.g., captured by a camera) to any number of available servers, each performing the analysis in a data parallel manner [83]. A highly complicating factor is the strong variability in the availability of hardware and software resources. Therefore, a fundamental problem of parallel computing in a Grid environment is to achieve high execution efficiency in the presence of the dynamically changing hardware and software availability.

In MMCA applications there are several sources of dynamic behavior. First, the sources of computer power are often shared by numerous applications that may make the available capacity scarce and varying over time. Although technological improvements will increase the bandwidth of single wide-area links beyond the Gbits/second range, the demands of emerging MMCA problems will increase - rather than reduce - competition for resources. Second, in MMCA applications the amount of data that needs to be processed often changes wildly over time. Two realistic examples of real-time multimedia analysis are:

- Application type 1 (A1): the comparison of objects and individuals in video streams obtained from multiple surveillance cameras.
- Application type 2 (A2): iris-scan based identification and automatic fingerprint checks (e.g., to be performed at international airports).

In the former application type, the job-arrival process is fairly constant and predictable, whereas in the latter one the arrival process has a random nature, and the job-arrival epochs may be hard to predict at small time scales. For both application types it is essential to optimize the use of the available resources, which results in two classes of problems: (1) *when* to transfer

a job to the resources, so as to obtain the highest service utilization possible while minimizing the buffering time for that job, and (2) *how many* resources to assign to the job such that the utilization costs are minimized whilst the QoS-constraint is met. In practice, these two problems are intertwined. Because of their dynamic behavior, MMCA applications must be made variability-tolerant by means of controlled adaptive resource utilization. This raises the need for new stochastic runtime performance-control methods that properly react to changing circumstances.

1.3 Objective of the thesis

In this thesis, we focus on the development, analysis and optimization of performance models and stochastic control schemes that can be used to make time-constrained multimedia applications tolerant to the dynamics of the environment. More specifically, we consider two classes of resource-assignment problems: (1) the proper *timing* of job processing, and (2) the cost-optimal *assignment of processor capacity* under QoS-constraints.

A key requirement to our models and methods is that they should be simple and easily implementable, yet effective, and adaptive to system variations. The solutions presented in this thesis form a step towards the realization of the ultimate goal of the research on Grid computing: to provide inexpensive and easy-to-use "wall socket" computing over a distributed set of resources [83].

1.4 Overview of the thesis

In Chapter 2 we brief discuss the basic concepts of prediction methods and Markov Decision Processes (MDPs) that are important and needed for the other chapters, and we provide a literature overview of these methods in the context of Grid computing.

In Chapter 3 we discuss the so-called "resource utilization" (RU) problem and the "just-in-time" (JIT) communication problem in MMCA applications in which the amount of data that needs to be processed is enormous. The RU problem focuses on determining the optimal number of compute nodes used by each multimedia server, properly balancing the complex tradeoff between computation and communication. The JIT problem aims to tune the transmission of newly generated data sent to remote servers, so as to obtain

the highest service utilization, while minimizing the need for buffering. For the RU problem, we develop a simple and easy-to-implement method to determine the optimal number of nodes to be employed, which is based on the classical binary-search method for non-linear optimization and is independent of the specifics of the system. The JIT problem is addressed by a smart adaptive control method that properly reacts to the continuously changing circumstances in large-scale Grid environment. Extensive experimental validation shows that our optimization approaches are highly effective.

In Chapter 4 we study optimal resource allocation in time-reservation systems. In such systems, jobs arrive at a service facility and receive service in two steps; in the first step information is gathered from the customer, which is then sent to a pool of computing resources, and in the second step the information is processed, after which the customer leaves the system. Here, two decisions should be made: (1) *when* to reserve computing power from the pool of resources, such that the job does not have to wait for the start of the second service step and that the processing capacity is not wasted due to the job still being serviced at the first step, and (2) *how many* processors to allocate for the second processing step such that reservation and holding costs are minimized. We decompose the problem into two parts. First, we show that the near-optimal reservation moment is given by the difference of the mean service time in the first step and the mean reservation time. Then, we apply dynamic programming to show that the near-optimal resource-allocation policy follows a step function with as extreme policy the bang-bang control for given structures of the cost function and the service rate function.

In Chapter 5, we extend the second part of the model in Chapter 4 to multi-queue systems. In such systems, each service facility poses a constraint on the maximum expected sojourn time of a job. The decision should be made to dynamically allocate the servers over the different facilities such that the sojourn-time constraints are met at minimal costs. We model this problem as a Markov decision problem and derive the structural properties of the relative value function via standard induction-based arguments. These properties, which are hard to derive for multi-dimensional systems (together with the properties described in Chapter 6), give a full characterization of the optimal policy.

In Chapter 6 we study a special case of the model described in Chapter 5, in which there is only a single queue. We show via dynamic programming that (1) the optimal allocation policy is work-conserving, and (2) the optimal

number of servers follows a step function with as extreme policy the bang-bang control policy. Moreover, (3) we provide conditions under which the bang-bang control policy is optimal. The derivation of these results is based on a combination of direct arguments and induction, which are not generalizable to multiple queues. Therefore, these characterizations of the optimal policy are not directly applicable in the multi-queue setting. However, it provides a good foundation of exploring the additional optimal allocation properties in multi-queue systems.

In Chapter 7, we study the optimal allocation policy for multi-queue systems with time-varying arrivals. We consider the problem under two different cases. In the first case, the time-varying parameters are known beforehand, and we show how the optimal policy can be obtained numerically. On the contrary, the second case considers the optimal allocation problem *without* full knowledge of the job arrival rates. For this case, we use both a prediction method and a stochastic approximation method to track the time-varying parameters to obtain near-optimal policies. Numerical results show that our techniques are highly effective.

Finally, in Chapter 8 we validate our resource-allocation policies in an experimental setting. The results show that our methods are extremely effective in practical scenarios.

1.5 Publications

This scientific contents of this thesis have been or will be published in the open literature. Chapter 3 is based on [116, 117, 113], Chapter 4 is based on [112], Chapter 5 is based on [110], Chapter 6 is based on [111], Chapter 7 is based on [115], and finally, the contents of Chapter 8 are based on [114].

Chapter 2

Preliminaries: Methods and Techniques

To run Multimedia Content Analysis (MMCA) applications under strict time constraints, information regarding processing time of the available resources must be incorporated. Since fluctuations in the available resources (e.g., computing power, bandwidth, workload) may have a great impact on the processing time, it is crucial to have effective and efficient prediction methods to achieve the right information and the important performance metrics on the resources. Apart from satisfying the time constraint of MMCAs, Grid computing is also expected to popularize a model of computational economy. It gives rise to a class of stochastic models that aims to minimize the average utilization costs by proper on-the-fly actions, while at the same time meeting a time constraint. A framework to model such decision problems is provided by Markov Decision Processes (MDPs). In this chapter we summarize the basic concepts of forecasting methods and MDPs.

This chapter is organized as follows. In Section 2.1 we discuss the basic concepts of prediction methods that are important and needed for the other chapters, and provide an overview of these methods used in the context of Grid computing. In Section 2.2 we discuss the basic concepts of MDPs and provide an overview of MDPs in the context of Grid computing.

2.1 Prediction methods

2.1.1 Basic concepts

Among the existing predictive methods there is a huge difference in the way previously obtained data is handled. In some cases one wants to adapt very

quickly to observed changes in the data, while there are also cases in which this behavior is not desired. The forecasting methods that we use in the sequel can be classified into four categories:

- mean-based methods
- median-based methods
- exponential smoothing methods
- Robbins-Monro approximation method.

The simplest *mean-based methods* take the average of all observed data as the forecast. Let Y_t be the data available at time t. Then the forecasted value at $t+1$, F_{t+1}, is given by

$$F_{t+1} = \frac{1}{t} \sum_{i=1}^{t} Y_i.$$

The *adapted mean-based methods* [105] use arithmetic averages over some portion of the measurement history to predict the next measurement. In particular, the extent of the history taken into account depends on the sliding-window parameter K, specifying the number of previous measurements for the arithmetic average, i.e.,

$$F_{t+1} = \frac{1}{K} \sum_{i=t-K+1}^{t} Y_i.$$

The parameter K is changed by -1, 0, or $+1$ over time based on the prediction error.

The *median-based methods* use a median as an estimator. Define $Sort_K$ as the sorted sequence of the K most recently observed values and $Sort_K(j)$ as the j-th value of the sorted sequence. Then the forecasted value at $t+1$ with the choice of K is given by

$$F_{t+1} = \begin{cases} Sort_K((K+1)/2) & \text{if } K \text{ is odd,} \\ \big[Sort_K(K/2) + Sort_K(K/2+1)\big]/2 & \text{if } K \text{ is even.} \end{cases}$$

Adapted median-based methods [105] adapt the parameter K in the same way as in the mean-based methods above. Note that the prediction of these

methods are not influenced much by asymmetric outliers, since this does not affect the median greatly.

The methods described above use an equal set of weights to past data. However, these weights can decay in an exponential manner from the most recent to the most distant data point. In this case, exponential smoothing [16, 17, 47, 104] is suitable for the prediction. Denote α $(0 < \alpha < 1)$ as the weight of the most recent observation Y_t and $1 - \alpha$ as the weight of the most recent forecast F_t. Then the forecast for the next period is given by

$$F_{t+1} = \alpha Y_t + (1 - \alpha) F_t.$$

Denote by $w(i)$ the weight for the i-th previous measurement. Then, function $w(.)$ is defined by

$$w(i) = \alpha(1 - \alpha)^i.$$

The *Robbins-Monro approximation method* [57] is a stochastic approximation method. The estimation is updated according to the following relation

$$F_{t+1} = F_t + \varepsilon_t(Y_t - F_t),$$

where ε_t is a parameter (possibly) depending on t. The intuition behind the update rule is the following. In case the observed processing time is higher than estimated, the prediction for the next processing time is increased by a small amount of the difference, and vice versa. When $\varepsilon_t = 1$ for all t, then the prediction for the next processing time is equal to the last observation.

2.1.2 Applications in Grid computing

In the literature, prediction methods have been proven successful to forecast the properties of a Grid. Prediction techniques can be classified into analytical, artificial intelligence (AI), and statistical methods [56]. Analytical techniques construct models by hand or use automatic code instrumentation. AI methods, such as neural network-based method, predict the future performance of resources or applications by learning from historical data and classifying the information. Statistical approaches analyze the successive historical data using the statistical methods (e.g., time series analysis [87]) in an effort to predict the data in future. Experience has taught that even some seemingly random or very noisy series can be modeled and predicted to

a usable error margin using statistical methods [31]. Therefore, we restrict ourselves in this thesis only on the statistical prediction methods to forecast the properties of a Grid.

To predict job runtimes in a Grid environment accurately, it is essential to have a method that effectively reacts to the peaks and level switches in job runtimes. For this purpose, Dobber et al. [28] develop the Dynamic Exponential Smoothing (DES) methods based on the traditional exponential smoothing (ES) methods. Sonmez et al. [92] use the mean-based, median-based and ES for predicting job runtimes and job queue waiting times, whilst Berman et al. [13] choose the Network Weather Service (NWS) prediction algorithm for the same purpose. To predict different properties of a Grid, the NWS algorithm selects between the following three prediction methods: mean-based methods, median-based methods and Autoregressive (AR) methods. For instance, Wolski et al. [106] take the NWS algorithm to predict resource availability. Furthermore, Smith et al. [88] and Guim et al. [42] aim to predict the total running times of parallel applications. The former one uses the mean-based and the Linear Regression (LR) method, while the latter one uses mean-based and median-based methods. Moreover, AR is applied by Zhang et al. [120] and Wu et al. [107] to predict CPU load and by Qiao [75] to predict the network traffic. To improve the accuracy of the prediction, the basic forecasting methods can be applied adaptively (e.g., adapted mean-based methods, adapted median-based methods and adaptive ES-based methods). These adapted prediction methods have shown to be very accurate. Apart from the basic forecasting methods, some research areas are interested in predictors that estimate the possibility of an event from its likelihood and prior probability as its probability conditional to its characteristics, such as Bayesian inference used in [66] to predict the resource availability in a Grid.

Recall that in the context of MMCA, prediction methods should be simple and easily implementable, yet effective because of the strict time requirement of the multimedia application. Therefore, in this thesis we only use prediction methods that are simple and fast, yet fairly accurate.

2.2 Markov Decision Processes

2.2.1 Basic concepts

In this thesis, we split Markov decision problems up into two categories: unconstrained and constrained Markov decision problems. The basic concepts are outlined below. We refer to [73] and [10] for a more detailed description.

2.2.1.1 Unconstrained Markov decision problem

A Markov Decision Process (MDP) is defined by the quadruple $(\mathcal{X}, \{\mathcal{A}_x | x \in \mathcal{X}\}, p, c)$, where the set $\mathcal{X}$ denotes the state space, $\mathcal{A}_x$ is the set of actions from which an action can be chosen at state $x \in \mathcal{X}$, $p(x, a, y)$ is the transition probability function describing the probability of going from state x to y when action $a \in \mathcal{A}_x$ is chosen, and $c(x, a)$ is the cost function describing the real-valued costs incurred by the system in state x and when action a is chosen.

Let X_t and A_t be the random variables for the state at time t and the corresponding action at that time epoch, respectively. The idea is that depending on the state X_t an action A_t is selected, according to some policy $\pi : \mathcal{X} \to \mathcal{A}_x$. Thus $A_t = \pi(X_t)$. Let $V_t^\pi(x)$ be the total expected cost in $0, \ldots, t-1$, when starting at 0 in x, under policy π. Then, the objective is to find a policy π^* such that $\lim_{t\to\infty} V_t^{\pi^*}(x)/t$, the long-run expected average cost (per time unit), is minimized. For the optimal policy π^* it holds that

$$V^{\pi^*}(x) + g^{\pi^*} = \min_{a \in \mathcal{A}_x} \Big[c(x, a) + \sum_{y \in \mathcal{X}} p(x, a, y) V^{\pi^*}(y) \Big], \qquad x \in \mathcal{X}. \quad (2.1)$$

This equation is called the optimality equation or Bellman equation. Often the superscript is left out: g and V are simply the average cost and value function of the optimal policy. By solving the optimality equation, π^* can be obtained. However, the optimality equation is hard to solve analytically in practice. Alternatively, the optimal actions can also be obtained by recursively defining $V_{l+1} = TV_l$ for arbitrary V_0, where T is the dynamic programming operator acting on V. For $l \to \infty$, the maximizing actions converge to the optimal ones (for existence and convergence of solutions and optimal policies we refer to Puterman [73]). The backward recursion

```
Let n = 0 and V_n(x) = 0 for all x ∈ 𝒳, and let ε be some
small number.
repeat:
     n := n + 1
     for each x ∈ 𝒳 do:
          V_n(x) := min_{a∈𝒜_x} {c(x,a) + Σ_{y∈𝒳} p(x,a,y)V_{n-1}(y)}
     end do.
     max := −10^10
     min := 10^10
     for each x ∈ 𝒳 do:
          min := MIN(min, V_n(x) − V_{n-1}(x))
          max := MAX(max, V_n(x) − V_{n-1}(x))
     end do.
until (max − min < ε).
V(x) := V_n(x) − V_n(0) for all x ∈ 𝒳.
g := (min + max)/2.
```

Algorithm 2.1. Pseudo code for the value-iteration algorithm.

equation is given by

$$V_{t+1}(x) = \min_{a \in \mathcal{A}_x} \Big[c(x,a) + \sum_{y \in \mathcal{X}} p(x,a,y) V_t(y) \Big]. \tag{2.2}$$

The pseudo code for value iteration is shown in Algorithm 2.1. The value iteration algorithm stops when for all states $x \in \mathcal{X}$, the difference $V_n(x) - V_{n-1}(x)$ is less than ε away from the average costs g. Therefore, the average costs are obtained by $g = (\min + \max)/2$ in the method that is also the optimal average costs g^*.

2.2.1.2 Constrained Markov decision problem

In this section we describe the so-called constrained Markov decision problem, i.e., where we aim to determine the optimal policies that minimize the average costs with subject to meet a constraint on the mean sojourn time of a customer or a job in the whole system. Denote the time constraint by α and let W be the sojourn time of an arbitrary customer in the system.

Then, the constrained Markov decision problem is

$$\min_{\pi} g(\pi) \quad \text{subject to} \quad \mathbb{E}W \leq \alpha.$$

Note that due to Little's Law the number of jobs L in the system can be related to the sojourn time W by $\mathbb{E}L = \lambda \mathbb{E}W$. Using this formula, the constrained Markov decision problem can be rewritten as an unconstrained Markov decision problem using Lagrange multipliers (see Altman [10]). Denote Lagrange multipliers by $\mathcal{L}$. Now, the optimality equation and the backward recursion equation for constrained MDP can be given as follows: for $x \in \mathcal{X}$

$$V^{\pi^*}(x) + g^{\pi^*} = \min_{a \in \mathcal{A}_x} \Big[\mathcal{L}\frac{x}{\lambda} + c(x,a) + \sum_{y \in \mathcal{X}} p(x,a,y) V^{\pi^*}(y) \Big], \tag{2.3}$$

$$V_{t+1}(x) = \min_{a \in \mathcal{A}_x} \Big[\mathcal{L}\frac{x}{\lambda} + c(x,a) + \sum_{y \in \mathcal{X}} p(x,a,y) V_t(y) \Big]. \tag{2.4}$$

For each value of $\mathcal{L}$, the optimal policy is a non-decreasing curve, and as $\mathcal{L}$ increases, the value of $\mathbb{E}W$ decreases (see Section 4.3.2 for a proof). There is a difference in the optimal policy between the one in the unconstrained case and in the constrained case. Note that there is a $\mathcal{L}^*$ for which $\mathbb{E}W_{\mathcal{L}^*} \geq \alpha$ and $\mathbb{E}W_{\mathcal{L}^*+\varepsilon} < \alpha$ for a small $\varepsilon > 0$. In the constrained case, the optimal policy is to randomize between the associated policies $\pi(\mathcal{L}^*)$ and $\pi(\mathcal{L}^* + \varepsilon)$ so that exactly $\mathbb{E}W = \alpha$ is achieved. The optimal policy thus randomizes between two threshold policies.

2.2.2 Applications in Grid computing

Markov Decision Processes (MDPs) [98] have been proven to be effective for modeling decision problems. The idea of MDPs is to make decisions not only based on the immediate rewards or costs, but to take into account the future dynamics of the system as well. As Grid computing has a strongly stochastic nature [28], MDPs have received a lot of attention in recent papers. Yu et al. [119] use an MDP approach to schedule sequential workflow task execution. The scheduling algorithm aims to minimizes the execution cost while meeting the overall deadline. Lee et al. [60] propose a grid policy administrator with an MDP-based resource management scheme that considers the management cost and QoS guarantee. In sensor-grid computing, there are many cases in which some response is needed from the sensor-grid system,

but the best action to take in different situations is not known in advance. In this context, Tham [99] advocates to use MDPs or a reinforcement learning approach [98] to realize optimal distributed autonomous decision making.

To make optimal decisions, one needs to solve the optimality equations. However, the optimality equation is hard to solve analytically in practice. In 1957, Bellman showed that iterative algorithms to solve the optimality equations converge to the correct value function, which is the solution to the optimality equations. In 1978, Puterman and Shin [74] came up with a modified value iteration and policy iteration algorithm to speed up the process.

The prediction methods and MDPs discussed above will be used in the remaining chapters.

Chapter 3

Resource Optimization in Distributed Real-Time Multimedia Applications

Multimedia applications running in real-time environments often must run under strict time constraints, e.g., to analyze video frames at the same rate as a camera produces them. Commonly, such applications run under a services-based scenario, in which video content analysis is being performed by a set of remote multimedia servers — each running independently on a different compute cluster.

For optimized use of the available resources it is first essential to determine the optimal number of compute nodes used by each multimedia server, properly balancing the complex tradeoff between computation and communication. In this chapter we refer to this issue as the "resource utilization" (RU) problem. Next, it is important to tune the transmission of newly generated data (e.g., a video frame) to the occupation of the remote servers, so as to obtain the highest *service utilization* possible, while minimizing the buffering time for individual video frames at the server side. We refer to this latter issue as the problem of "just-in-time" (JIT) communication.

Motivated by these observations, we first develop a simple and easy-to-implement method to determine the optimal number of nodes to be employed by each multimedia server. Our method is based on the classical binary search method for non-linear optimization and is independent of the (usually unknown) specifics of the system. Second, we address the JIT problem by introducing a smart adaptive control method that properly reacts to the continuously changing circumstances in distributed systems. Extensive experimental validation of the two approaches on a real distributed system shows that our optimization approaches are indeed highly effective.

3.1 Introduction

Multimedia content analysis (MMCA) operated in a realtime environment pose very strict requirements on the obtained processing times, while off-line applications have to perform within "tolerable" time frames. To adhere to strict time constraints, large-scale multimedia applications typically are being executed on distributed systems consisting of large collections of compute clusters.

In an execution scenario in which a number of multimedia servers are being executed on a distributed set of compute clusters, the *resource optimization* problem can be separated into two main parts. First, it is essential to determine the optimal number of compute nodes used by each individual multimedia server. This part of the optimization problem generally depends on a priori system information, including the multimedia server application itself, and the specifics of the computing environment (e.g., network characteristics, CPU power, memory, etcetera). In this context, it is essential to properly balance the following trade-off: if the number of compute nodes employed by a multimedia server is too low, the processing power is insufficient to meet the strict time constraints of real-time applications; if the number of compute nodes is too high, the parallelization overhead will cause a degradation of the computational performance. This problem is referred to as the *resource utilization* (RU) problem throughout this chapter. Clearly, as researchers in the MMCA domain generally are not usually experts in parallel computing, there is an urgent need for *simple* and *easily implementable*, yet *effective* methods (in terms of the number of evaluation steps) for determining the optimal level of parallelism. Also, the method should be easily *adaptable* to inherently dynamic changes in the distributed environment.

Second, based on the result achieved from the RU problem, it is essential to employ the allocated resources efficiently by sending data (e.g., video frames) to each multimedia server at carefully determined moments in time, in order to obtain the highest *service utilization* possible, and to minimize the service response time. Clearly, if an available multimedia server is currently unoccupied, analysis results for a video frame can be obtained in the fastest possible way. Unfortunately, keeping a multimedia server mostly idle is a waste of compute resources. Alternatively, sending video frames to a multimedia server as soon as possible may cause a need for queuing of video frames at the server side. Having to wait for the processing of previously queued data may result in an unacceptably long delay between the moment

of data generation and result calculation. Hence, to optimally balance server utilization and response time, it is essential to tune the transmission of video frames to the occupation of the remote multimedia servers. Due to variations in transmission latencies and other variabilities in the computing environment, however, it is difficult to accurately tune the sending of video frames to the variable response time of a multimedia server. In this chapter we refer to this issue as the *just-in-time* (JIT) communication problem.

To solve the JIT problem, we need effective prediction methods that react to the continuously changing circumstances in distributed systems. An immediate consequence of a JIT approach is that a multimedia server always analyzes the most recently generated (or, "up-to-date") video frames; no server response delays are introduced due to frame buffering at either the client side or at the server. Clearly, this is an important, even critical requirement in real-time applications.

The main contributions of this chapter are as follows: (1) we provide an innovative solution to the RU problem, which - in contrast to existing methods - is a fully dynamic, runtime approach, and (2) we provide a solution to the JIT problem which is entirely new, as - to our knowledge - it has not been addressed in the literature before. Our solution requires only limited (runtime) benchmarking, which is performed in a transparent and portable manner. Also, our solution is independent of the specific implementation of the applications at hand, making our solution highly sustainable (as it is immediately applicable, even after the application is altered).

The chapter is organized as follows. In Section 3.2 we present related work, and address the pros and cons of existing methods. Section 3.3 presents the experimental setup, and describes example applications. Section 3.4 presents our proposed approaches, which are further formulated in Section 3.5. Section 3.6 discusses our experimental results. Finally, in Section 3.7 we present our conclusions and address topics for further research.

3.2 Related work

Previous work in this field can be categorized into two groups. The first group, relevant to our RU problem, incorporates the general performance optimization problem of computer systems. The second group, relevant to our JIT problem, relies on statistical predictions of system behavior.

Roughly speaking, techniques to general performance estimation can be clas-

sified into one of three main categories: (1) measurement, (2) modeling and (3) hybrid methods. Estimation techniques that belong to the second category can be further divided into the subcategories of (2a) mathematical analysis and (2b) simulation [50].

Performance estimation by measurement is generally performed on a real system under conditions that reflect typical workload and behavior. Execution times of real problems are then inferred from measured results [101, 59]. Application of this approach has several drawbacks. First, in many cases the complete system to be evaluated has yet to be developed, and may change over time. Second, even if a complete system is available it is often not clear what workload is realistic or typical. Finally, if the measurement process is biased towards certain aspects of the underlying hardware, the measurement technique may not be applicable to other platforms.

Benchmarking is an alternative technique within the category of measurement, which is often used for comparison of multiple computer systems (e.g., see [24, 46, 103, 30, 20]). Rather than reflecting typical behavior, benchmarks often represent non-typical, artificial workloads. In comparison with direct measurement, benchmarking has the advantage that the system to be evaluated does not have to be available. The use of non-typical workloads, however, often has a negative effect on the accuracy of the performance estimations. A solution — albeit complex — is to capture results for small instruction mixes and a variety of workloads, and to interpret the measurement results with utmost care [29, 94].

Performance modeling can be applied in cases where direct measurement is too costly, or where the computer system to be evaluated is not available. In the category of mathematical analysis, models range from simple (linear) algebraic expressions to complex formalisms such as queueing networks [50, 81]. In general, such models have a high response time due to their ease of evaluation. An additional advantage is that parameter values may be varied to observe their relative impact on performance. However, to obtain high estimation accuracy, the large number of model parameters may violate the simplicity and applicability constraints.

In simulation models behavior and workloads are described (imitated) in a special computer program — usually an annotated or otherwise adapted version of a "real" program [50, 72]. Performance predictions are obtained by monitoring the execution of the adapted program. The main advantage of simulation models is that dynamic system behavior is easily captured. Also,

simulation makes it easy to "zoom in" on interesting or expensive parts of a system. A disadvantage is that the system to be evaluated must be available, at least in some rudimentary form. Another drawback is that it is a costly method for obtaining even moderately accurate performance estimates.

In hybrid estimation techniques a combination of measurement and modeling is applied [64, 109]. Such techniques have the advantage that the complexity of using either measurement or modeling in isolation can be avoided, while a high level of estimation accuracy can still be obtained. As an example of approaches in this direction, Saavedra-Barrera et al. [79] have measured system performance for *sequential* Fortran programs in terms of an Abstract Fortran Machine (AFM), an approach referred to as narrow spectrum benchmarking. The AFM-based approach provides a solution to the problem of the high complexity of complete analytical study of computer systems. The drawback of the approach, however, is that system variance is almost completely ignored. For applications working on extensive dense data fields (e.g., image data structures) this is a too crude restriction as variations in the hit ratio of caches and system interrupts often have a significant impact on performance [41, 82].

Other performance estimation techniques that incorporate more detailed behavioral abstractions relating to the major components of a computer system [50, 62] need tens - if not hundreds - of platform-specific machine abstractions to obtain truly accurate estimations. Consequently, the requirements of simplicity and applicability to the MMCA domain are not satisfied. To overcome this problem, Seinstra et al. [84] have designed a new model for performance estimation of *parallel* image and video processing applications running on clusters, based on the Abstract Parallel Image Processing Machine (APIPM). The APIPM model has been used in a large set of realistic image and video processing applications to find the optimal number of compute nodes. The main advantage of this model is that predictions are based on the analysis of a small number of rather high level system abstractions (i.e., represented by the APIPM instruction set). The main limitation of this model, however, is that the instruction set and its related performance values are parameterized with a very large number of instruction behavior and workload indicators. As such, the model does not meet our requirements, as obtaining accurate performance values for all possible parameter combinations is both costly and complex.

For our JIT problem, we argue that existing prediction methods (i.e., the adapted mean-based methods [105], the adapted median-based methods [105],

exponential smoothing methods [16, 17, 47, 104], and the Robbins-Monro Stochastic Approximation method [57] discussed in Chapter 2) are not capable of adhering to the specific requirements of JIT communication. One important problem with existing methods is that *random peaks* can be observed in the processing time of each multimedia server. These delays cause accumulative errors in predicting the exact moments of video frame transmission, resulting in significant deviations from the optimal strategy. Similarly, existing methods cannot deal with *periodic peaks* very well either. These observations have raised a need for additional policies to amend these particular problems.

3.3 Grid testbeds

In a Grid environment, resources have different capacities and many fluctuations exist in load and performance of geographically distributed nodes [27]. As the availability of resources and their load continuously varies over time, the repeatability of the experimental results is hard to guarantee under different scenarios in a real Grid environment. Also, the experimental results are very hard to collect and to observe. Hence, it is wise to perform experiments on a testbed that contains the key characteristics of a Grid environment on the one hand, and that can be managed easily on the other hand. To meet these requirements, we perform all of our experiments on the DAS-3 Grid test bed [2], described in Section 3.3.1 below.

3.3.1 DAS-3

DAS-3 (The Distributed ASCI Supercomputer 3), see Table 3.1 and Figure 3.1, is a five-cluster wide-area distributed system, with individual clusters located at four different universities in The Netherlands: VU University Amsterdam (VU), Leiden University (LU), University of Amsterdam (UvA), and Delft University of Technology (TUD). The MultimediaN Consortium (UvA-MN) [6] also participates with one cluster, located at UvA. As one of its distinguishing features, DAS-3 employs a novel internal wide-area interconnect based on optical 10G links (StarPlane [7]).

DAS-3 is heterogeneous in design. The research specifically planned for DAS-3 includes topics like Grid computing, performance analysis, Virtual Laboratory design, and distributed image and video content analysis and

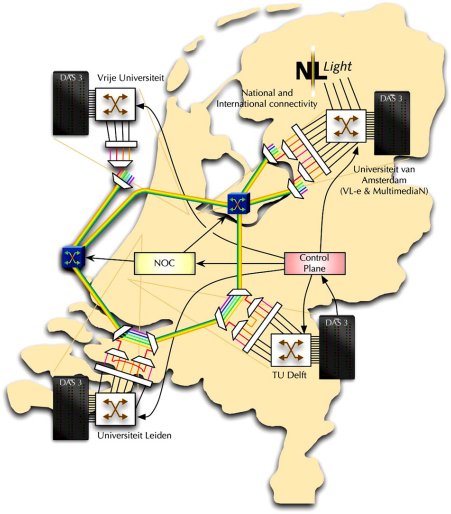


Figure 3.1. The Distributed ASCI Supercomputer 3.

visualization.

Cluster	Nodes	Type	Speed	Memory	Storage	Node HDDs	Network
VU	85 dual	dual-core	2.4 GHz	4 GB	10 TB	85 × 250 GB	Myri-10G and GbE
LU	32 dual	single-core	2.6 GHz	4 GB	10 TB	32 × 400 GB	Myri-10G and GbE
UvA	41 dual	dual-core	2.2 GHz	4 GB	5 TB	41 × 250 GB	Myri-10G and GbE
TUD	68 dual	single-core	2.4 GHz	4 GB	5 TB	68 × 250 GB	GbE (no Myri-10G)
UvA-MN	46 dual	single-core	2.4 GHz	4 GB	3 TB	46 × 1.5 TB	Myri-10G and GbE

Table 3.1. Overview of the DAS-3 cluster sites.

3.3.2 Example applications

In our experiments, we use DAS-3 to run a real-time multimedia application (referred to as "Aibo"), as well as an off-line application (referred to as "TRECVID").

The Aibo application demonstrates real-time object recognition performed by a Sony Aibo robot dog [83] (see Figure 3.2). Irrespective of the application of a robot, the general problem of object recognition is to determine which, if any, of a given repository of objects, appears in an image or video stream. It is a computationally demanding problem that involves a non-trivial trade-off between specificity of recognition (e.g., discrimination between different

faces) and invariance (e.g., to shadows, or to differently colored light sources). Due to the rapid increase in the size of multimedia repositories of "known" objects [37], state-of-the-art sequential computers no longer can live up to the computational demands, making high-performance computing (potentially at a world-wide scale, see also Figure 3.2) indispensable. The TRECVID

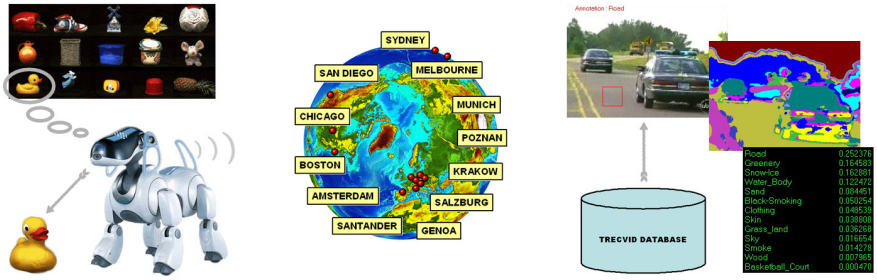


Figure 3.2. Our example real-time (left) and off-line (right) distributed multimedia applications, which are capable of being executed on a world-wide scale. The real-time application constitutes a visual object recognition task performed by a robot dog (Aibo). The off-line application constitutes our TRECVID system.

application represents a multimedia computing system that has been applied successfully in recent editions of the international NIST TRECVID benchmark evaluation for content-based video retrieval [45, 90]. The aim of the TRECVID application is to find semantic concepts (e.g., vegetation, cars, people, etc.) in hundreds of hours of news broadcasts, amongst others from ABC and CNN. The TRECVID concept detection task is, in general terms, defined as follows: Given the standardized TRECVID video data set, a common shot boundary reference for this data set, and a list of feature definitions, participants must return for each concept a list of at most 2000 shots from the data set, ranked according to the highest possibility of detecting the presence of that semantic concept. TRECVID is computationally intensive; for thorough analysis it easily requires about 16 seconds of processing per video frame on the fastest sequential machine at our disposal [85]. Consequently, the required time for participating in the TRECVID evaluation using a single computer easily can take over one year of processing.

Both applications have been implemented using the so-called Parallel-Horus software architecture, that allows programmers to write parallel and distributed multimedia applications in a fully sequential manner [83]. The automatic parallelization and distribution of both applications results in

services-based execution: a client program (typically a local desktop machine) connects to one or more *multimedia servers*, each running on a (different) compute cluster. Each multimedia server is executing in a fully data parallel manner, thus resulting in transparent *task parallel execution of data parallel services*.

More specifically, in both applications, before any processing takes place, a connection is established between the client application and a multimedia server. As long as the connection is available, the client can send video frames to this server. Each received video frame is scattered by this server into many pieces over the available compute nodes. Normally, each compute node receives one partial video frame for processing. The computations at all compute nodes take place in parallel. When the computations are completed, the partial results are gathered by the communication again and the final result is returned to the client.

3.4 Proposed approaches

In practice, running CPU-intensive applications in large-scale distributed computing environments typically consists of two phases: (1) an *initialization phase* to determine the optimal number of compute nodes L^*, and (2) the *main phase* to actually run the application on the L^* parallel nodes. In this chapter, each of our proposed approaches is used in one of these phases.

3.4.1 Resource utilization problem

First, we propose a simple method for on-the-fly determination of the "optimal" level of parallelism. Unlike analytical methods, our parallel multimedia server together with the underlying execution platform is treated as a black box from the resource allocator's point of view. This is due to our need of obtaining a *general* and *robust* approach to solve the optimization problem.

With our software and hardware assumed as black boxes, we are faced with the problem of having to deal with a search space that is unlimited in theory (and in practice limited only by the total number of available nodes in a given cluster system). As a result, it is essential to apply heuristics that can reduce our search space significantly. In this context, extensive experimental observations for realistic, large-scale problems in MMCA have revealed

the following three important properties of optimal resource allocations (see Sections 3.5.1.1 and 3.5.1.2 for details):

First, in many cases the optimal number of compute nodes is found to be a power of 2, i.e., of the form 2^m for some $m = 0, 1, \ldots$. This observation is important because it leads to a drastic reduction of the set of possible solutions. For example, if the number of available compute nodes is L_{max}, the size of the solution space is reduced from L_{max} (i.e., the number of elements in the index set $\{1, \ldots, L_{max}\}$) to $\lfloor \log_2(L_{max}) \rfloor$ (i.e., the number of elements of the set $\{2^0, 2^1, \ldots, 2^K\}$ where $K = \lfloor \log_2(L_{max}) \rfloor$). Here the symbol $\lfloor x \rfloor$ represents the largest integer $\leq x$.

Second, on compute nodes consisting of multiple CPUs (and potentially multiple cores), for a fixed number of compute elements, using more compute nodes and less CPUs per node yields better performance.

Third, if the compute cluster processing time is denoted by $S(L)$, with L the number of compute nodes, then there exists a threshold value L^* such that $S(L)$ decreases fast as a function of L for $L < L^*$, whereas $S(L)$ flattens out, and may even increase, for $L > L^*$. L^* is commonly referred to as the *engineering knee*. Moreover, in practice using too many compute nodes may be very costly. L^* should be the smallest number that matches the conditions specified above.

It should be noted here that our first two observations above may not be (and probably are not) true for all potential target systems. For such systems, however, other heuristics will apply, which can then be used for our search space reduction. Such other heuristics do not affect the manner in which our search is applied.

Based on the above observations, our proposed method is aimed at determining L^* as the optimal point of operation. The method takes the idea of the well-known classical binary search method for non-linear optimization, and converges if the relative improvement of $S(L)$ with respect to L (on a log scale) is close enough to 0 (say $5 - 10\%$). In Section 3.5 we will give a complete formulation of our method.

3.4.2 JIT communication problem

A simple execution approach to solve the JIT communication problem, which we refer to as the back-to-back method (BBM), is to perform the sending of a newly generated video frame exactly after a result has been received from

the same server (see Figure 3.3). Using the BBM method, any video frame processed by a multimedia server is guaranteed to be most up-to-date. A drawback of BBM, however, is that the server is idle when it has processed a frame and is waiting for the next one. In a bottleneck situation, the video frame transmission time from the client to the server (Tc_1) and the time to send a result back (Tc_2) may be long. In practice, Tc_1 is normally very close to Tc_2, thus we denote them by Tc. Then, the service utilization (SU) using BBM is given by

$$SU = \frac{Ts}{Ts + 2 \cdot Tc},$$

where Ts is denoted as the service processing time of a video frame. Obviously, if the communication time increases, service utilization decreases.

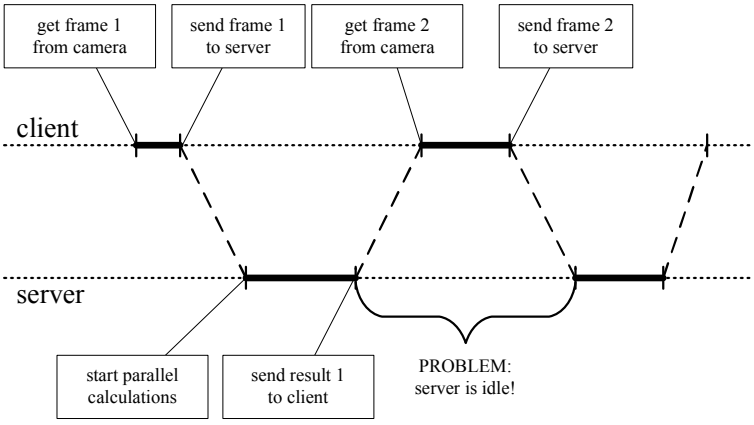


Figure 3.3. BBM approach for video frame transmission.

An alternative approach, referred to as the buffer storage method (BSM), is to establish a buffer at the server side. As long as the buffer is not full, the client is allowed to keep sending frames to the server. When the server is busy, the frames will be stored in the buffer before being processed (see Figure 3.4). Using BSM, service utilization can reach 100%. However, the drawback is that the data in the buffer may have become outdated *before* the actual video content analysis even takes place, due to the long waiting time. A solution would be to simply remove outdated frames at the server side. This, however, leads to (a lot of) unnecessary traffic between client and server, which should be avoided as resources are scarce.

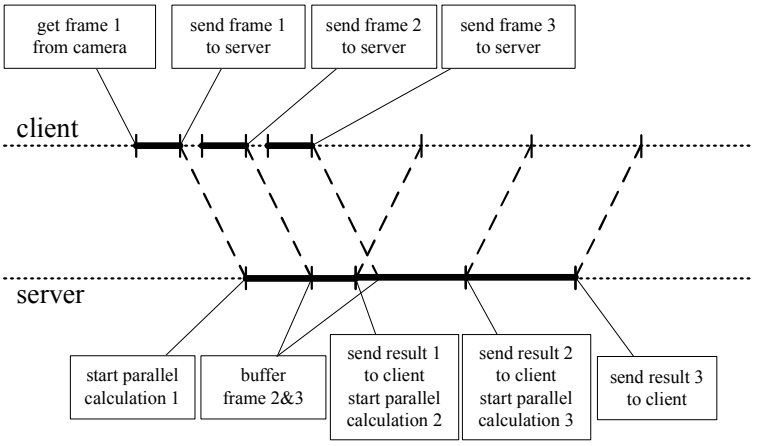


Figure 3.4. BSM approach for video frame transmission.

Given the previous two methods, the optimal strategy would be to send each $(i+1)$-st frame with a delay after sending the i-th frame. The delay is exactly the processing time of the i-th frame. For instance, if the service processing time of the current frame equals Ts_i, sending the next frame after a period of Ts_i will give an optimal solution. With this strategy, the server gets the most up-to-date frame and the service utilization is unity (see Figure 3.5). Unfortunately, Ts_i is unknown before the result of the current frame is returned back to the client side. It is therefore essential to have an accurate prediction of the processing time of video frame data.

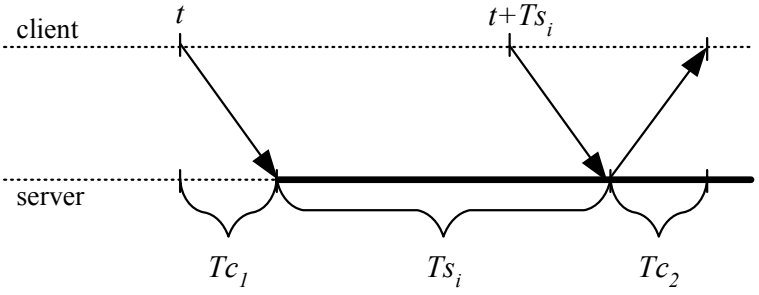


Figure 3.5. An optimal solution for video frame transmission.

We have observed that existing predictive methods are all capable of generating an accurate trend line based on the processing time of previous frames. However, for our JIT communication problem, these methods are not suffi-

ciently optimized for particular cases. The first problem appears when the processing time of certain frames suddenly become much longer (e.g., a peak) than the expected Ts obtained from a trend line. The sudden change breaks the rhythm of frame transmission and causes accumulative waiting times for all subsequent frames, even when the processing time returns back to the expected Ts (see Figure 3.6).

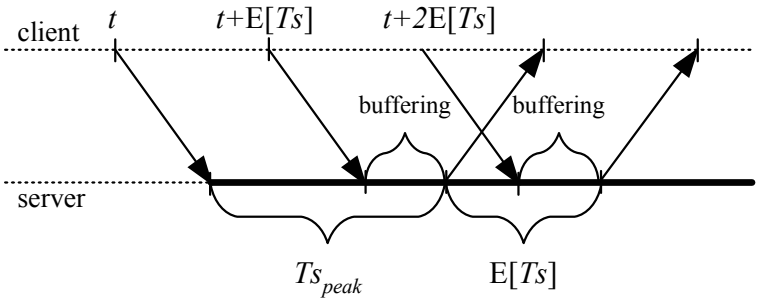


Figure 3.6. All frames are affected continuously by sudden long process times.

Apart from random peaks, a second complication is that one can observe processing times to have periodic peaks. If the service processing time of frame i is predicted as a peak, then the sending of frame $(i + 1)$ should be delayed to prevent a long buffering time. None of the prediction methods mentioned above can effectively deal with random peaks very well, nor do these pay attention to periodic characteristics.

We propose two policies to amend these problems. The first, referred to as the *one-before-last-measurement* (BLM) policy, is to restore the rhythm of transmission by removing the extra delay observed at an earlier moment. The second, referred to as the *peak-prediction* (PP) policy, is to find the periodic characteristics of the peaks in processing times and then to predict occurrence of subsequent peaks. Our proposed prediction methods, including the BLM and PP policies, provide good solutions for our JIT communication problem.

3.5 Method formulation

This section describes the two proposed modeling approaches in detail. The approaches are based on the results of extensive experimentation performed on the DAS-3 distributed cluster system (described in Section 3.3).

3.5.1 Resource utilization problem

In our example applications (described in Section 3.3.2 above), video frames are being processed on a per-cluster basis, using a varying number of compute nodes on each cluster, each consisting of multiple CPUs. The compute cluster (or *service*) processing time is defined as a function $S(L, n)$ of the number of compute nodes $L = 1, \ldots, m_{max}$ and the number of CPUs per node $n = 1, 2, \ldots$. Our goal is to minimize the cost function $S(L, n)$ over the set of possible values of (L, n); thus, we are searching for the point $(\hat{L}, \hat{n})$ where the function $S(L, n)$ attains its minimum. In this context, it is important to note that the set of possible combinations (L, n) may be very large, and that in practice, finding the optimum $(\hat{L}, \hat{n})$ may be very time consuming. Therefore, our goal is to develop a simple, but effective, heuristic method to obtain a nearly-optimal solution within a short time frame. To this end, we first discuss a number of observations that we collected during our extensive experiments, leading to a dramatic reduction of the set of possible value of (L, n). Subsequently, the method to approach the optimal (L, n) is described in detail.

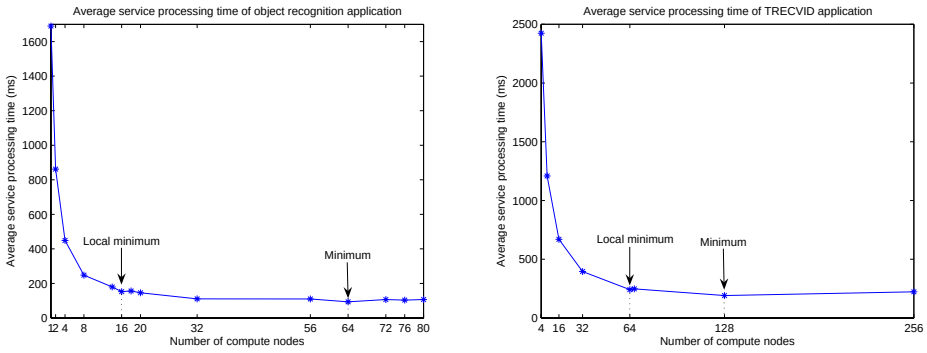
3.5.1.1 Reduction of the solution space

Many combinations (L, n) lead to the same total number $T = L \cdot n$ of CPUs. The following observations, made for our particular example applications, rule out many possibilities:

Observation 1: *The optimal number of CPUs often is a power of 2.*

In our experiments, we consistently observed that the optimal number of CPUs is found to be a power 2. For example, Figure 3.7 shows the average processing times for our two example applications: (a) Aibo, and (b) TRECVID. The results show that both *local* and *global* minima are consistently found when the total number of CPUs is a power of 2. This observation leads to a dramatic reduction of the set of possible solutions. Namely, if the number of available compute nodes is L_{max}, the number of available CPUs in each compute node is n_{max}, then the solution set is reduced to $\mathbb{X} := \{(2^p, 2^q), p = 0, \ldots, P, q = 0, \ldots, Q\}$, where $P := \lfloor \log_2(L_{max}) \rfloor$, and $Q := \lfloor \log_2(n_{max}) \rfloor$.

Observation 2: *Using more compute nodes, yet less CPUs per node, is generally better.*



(a) Aibo application

(b) TRECVID application

Figure 3.7. Average service processing time versus number of compute nodes.

Another important observation from our experimental results is that for the same total number of CPUs $T = L \cdot n$, using more compute nodes L and fewer CPUs per node n provides better performance. That is, for the same total number of CPUs $T = 2^m$, where the solution set should be $\mathbb{X} := \{(2^p, 2^q), p + q = m\}$, among them, q should be as small as possible. This observation is illustrated by Figure 3.8, where we consider the case $T = 64$ for three combinations $(L, n) \in \{(64, 1), (32, 2), (16, 4)\}$; the results show that the combination (64,1) has better performance than (32,2) and (16,4). This is explained by the fact that the compute nodes in DAS-3 are linked by a fast local Myrinet interconnect, whereas the CPUs within a single node communicate over a shared memory bus, which is less efficient. Note that the 'burstiness' of the perceived processing times is explained by random operating system interference, and by automatic garbage collection performed by the Java virtual machine.

Based on these observations, the solution set can be reduced drastically. For instance, for a system having 85 nodes and 4 CPUs per node, the reduced solution space is $\mathbb{X} = \{(2^p, 1), p = 0 \ldots 6\} \cup \{(64, 2), (64, 4)\}$.

In a general form, to determine the optimal number of compute nodes and CPUs per node, the solution space is reduced to the combinations $\mathbb{X} = \{(2^p, 1), p = 0 \ldots P\} \cup \{(2^P, 2^q), q = 1 \ldots Q\}$, where $P := \lfloor \log_2(L_{max}) \rfloor$, and $Q := \lfloor \log_2(n_{max}) \rfloor$. For simplicity, we use $(2^{(P+q)}, 1)$ instead of $(2^P, 2^q)$ for our notation, although $(2^{(P+q)}, 1)$ does not exist.

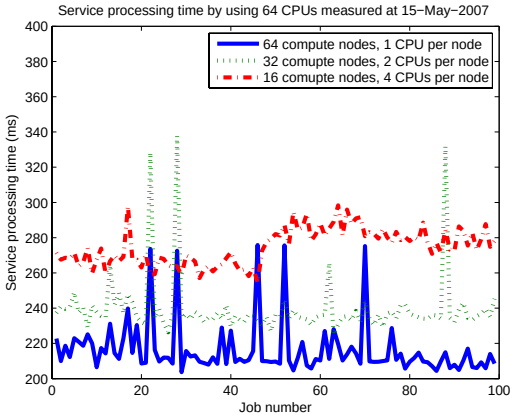


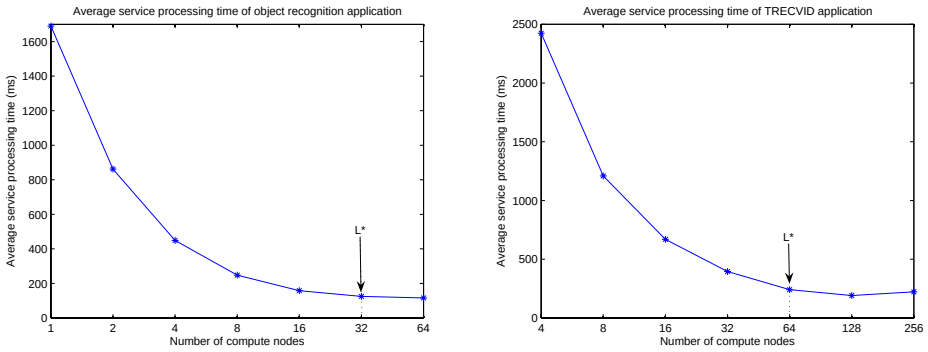
Figure 3.8. More compute nodes and less CPUs per node is better.

3.5.1.2 Steps to approach the optimal (L, n)

From the reduced solution space, we iteratively increase the total number of CPUs to find the optimal (L, n). When the number of applied compute nodes becomes larger, the parallelization overhead increases, and may even become dominant. Our experimental results show that there exists a threshold value m^* such that $S(2^m, 1)$ decreases fast for $m < m^*$, whereas $S(2^m, 1)$ flattens out, and may even increase, for $m > m^*$. As an illustration, Figure 3.9 shows the average service processing times for the Aibo- and TRECVID-applications for different values of $L = 2^m$. In both cases, we observe that there exists some *saturation point* $L^* = 2^{m^*}$ such that increasing the number of parallel nodes L beyond L^* does not lead to a significant reduction of the service processing times. Throughout, $L^* = 2^{m^*}$ will be referred to as the *engineering knee* and is regarded as the (near-)optimal point of operation.

3.5.1.3 LDS method

To find the engineering knee L^*, we have developed an *Logarithmic Dichotomy Search* (LDS) method. The LDS method follows the idea of a well-known conventional binary search (CBS) algorithm [53] which aims to find a particular value in a sorted list. Compared to the CBS strategy, the LDS method makes progressively better guesses, and proceeds closer to the optimal value. Let the elements in the solution set $\mathbb{X}$ be denoted by



(a) Aibo application

(b) TRECVID application

Figure 3.9. Engineering knee of Aibo and TRECVID applications.

$(e_0, \ldots, e_K)$, with $K = P + Q$, where P and Q are defined in Section 3.5.1.1. The LDS strategy selects the median element in the set $\mathbb{X}$, denoted by e_{Mid}. Define ε as the desired minimal improvement in the service processing time by increasing the number of compute nodes. If $\frac{S(e_{\text{Mid}})-S(e_{\text{Mid}+1})}{S(e_{\text{Mid}})} > \varepsilon$, then we repeat this procedure with a smaller list, and we keep only the elements $(e_{\text{Mid}+1}, \ldots, e_K)$. If $\frac{S(e_{\text{Mid}})-S(e_{\text{Mid}+1})}{S(e_{\text{Mid}})} \leq \varepsilon$, then the list in which we search becomes $(e_1, \ldots, e_{\text{Mid}})$. Pursuing this strategy iteratively, it narrows the search by a factor of two each time, and finds the minimum value that satisfies our requirement after $\log_2(K)$ iterations.

Note that the selection of ε is very important in finding the engineering knee. A large ε means that we are easily satisfied with the improvement. However, the result may not be close to the actual optimum. Setting ε to a very small value or even zero certainly will let us find the engineering knee (which is close to, or equal to, the optimal number of compute nodes), but this may take an undesirably long time. Hence, in practice ε is always a small positive number which is close to, but not equal to, zero. The pseudo code for our LDS method for the solution space $\mathbb{X}$ is given in Algorithm 3.1.

It is worth noting that there is still room for improvement. In our implementation, we obtain the runtime information using individual number of compute nodes by sequential measurements. Actually, if there are enough processors, we can do several measurements simultaneously by parallel tech-

```
Low := 0
High := K
While (Low < High) {
        Mid := ⌊(Low+High)/2⌋
        if S(e_Mid) ≤ S(e_Mid+1)/(1−ε) {High = Mid;}
        else {Low = Mid +1;}
        end if;
}
Optimal number of compute nodes := High.
```

Algorithm 3.1. Pseudo code of LDS strategy.

nique. It is named as parallel LDS strategy. In this thesis, we will not give further discussion on the possible improvement.

3.5.2 JIT communication problem

The following continues with a detailed formulation of the proposed solution for the JIT problem. The notations used here are defined as follows:

- Ts_i: the processing time of the i-th frame.
- Tc_i: the communication time of sending the i-th frame from the client to the server.
- t_i: the time point when the client sends the i-th frame to the server.
- r_i: the time point when the client receives i-th result from the server.

Trend line
As shown in Figure 3.5, if we can predict the service processing time of the current frame accurately, then sending the next frame after the predicted time unit should provide an optimal solution. Therefore, we investigated several conventional prediction methods (i.e., adapted mean-based methods, adapted median-based methods, exponential smoothing methods, and Robbins-Monro Stochastic Approximation method described in Chapter 2) for predicting the service processing time. In the experiments using

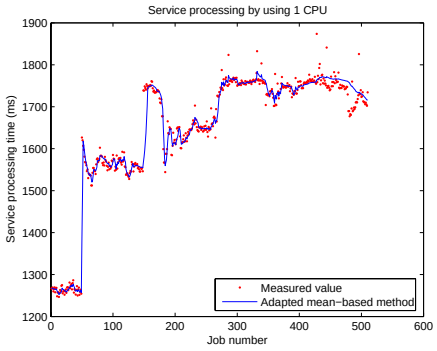
adapted mean-based methods, adapted median-based methods, and exponential smoothing methods the initial value of K is set to 20. Note that when using exponential smoothing methods, if K is larger than the amount of the available measurements data, then we rescale the weights $w(i)$ such that $\sum_{i=1}^{K} w(i) = 1$. In addition, the initial value of α is set to 0.5. In the experiments using Robbins-Monro approximation method we set $\varepsilon_t = 0.5$ for all t. We found that, based on the earlier service processing times, and by using any of these prediction methods, an accurate trend line can be generated. Figure 3.10 gives an illustration of the predicted service processing time versus the measured value of running an example application using one compute node and a single CPU only.

Periodicity of the peaks
Another important observation from our experimental results is the occurrence of periodic peaks when using large numbers of compute nodes. Because our multimedia applications are partially implemented in Java, the *Java garbage collector* [3] has an influence on the service processing time. In case of large service processing times, the effect of garbage collection generally is insignificant and can be ignored. This is the situation as depicted in Figure 3.10. In contrast, when the service processing time is small compared to the garbage collection time, the periodic peaks are significant. We ran an example application using 64 compute nodes (using one CPU per node) during three different periods in time. From these data sets, we notice that there is a deterministic period of the occurrences of certain specific peaks (see Figure 3.11).

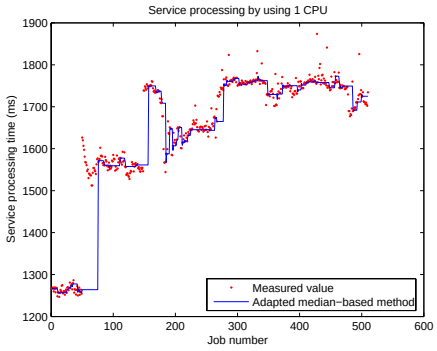
3.5.2.1 Method

Based on the experimental results, we conclude that an effective prediction method for our application must have the following characteristics: (1) it must be able to generate an accurate trend line of the service processing time, (2) it should be able to deal with outliers in the observed processing time as soon as possible, and (3) it must be able to predict when the next peak occurs. In this section, we discuss our BLM and PP policies in detail.

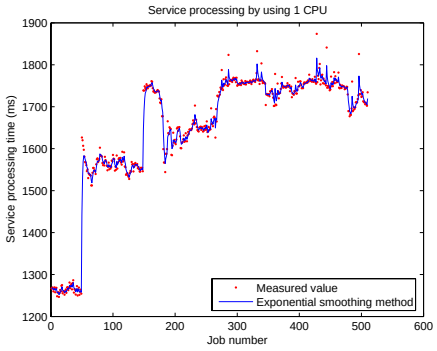
BLM Policy
Our first policy to deal with peaks is called "one-before-last-measurement" (BLM) policy. This policy determines the optimal sending time under the following three cases.



(a) adapted mean-based methods



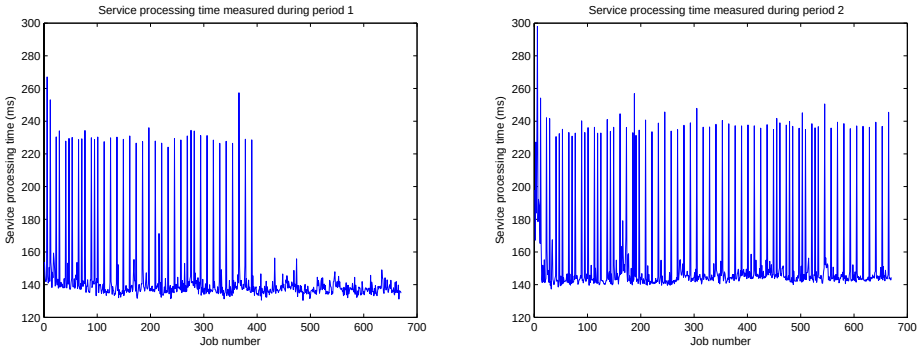
(b) adapted median-based methods



(c) exponential smoothing methods

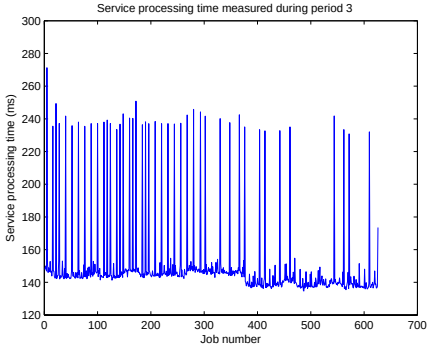
(d) Robbins-Monro method

Figure 3.10. Trend line generated by different prediction methods.



(a) measurement 1

(b) measurement 2



(c) measurement 3

Figure 3.11. Service processing time taken at different times.

Case 1: waiting for sending
The i-th job will not be sent until the result of the $(i-k)$-th job becomes available to the client. Because we must take care that the server has enough jobs to process, we cannot use the last measurement data as a predictor (also indicated by Harchol-Balter and Downey [44]). Therefore k must be larger or equal to 2. Throughout this chapter, we focus on the case that $E[Tc] \leq \frac{E[Ts]}{2}$. Here $E[Ts]$ and $E[Tc]$ represent the expected service processing time and the communication time respectively. In this case, we set $k = 2$. This implies that at most one job is waiting in the buffer at the server side. As a result, the occurrence of cumulative waiting times can be prevented. In the case that $E[Tc] > \frac{E[Ts]}{2}$, we only need to enlarge the value of k. Hence, for $k = 2$, we have the following equation,

$$t_i \geq r_{i-2}. \tag{3.1}$$

This equation implies that the i-th video frame is sent after the result of the $(i-2)$-th frame is received by the client. Figure 3.12 gives an illustration.

Case 2: sending immediately
Obviously, if the result of the $(i-1)$-th frame is received, the i-th frame must be sent immediately. Therefore, we have

$$t_i \leq r_{i-1}. \tag{3.2}$$

Case 3: adjusting sending time
The sending time of the i-th frame is also decided by the relationship between the expected service processing time and measured service processing time of the $(i-2)$-th frame Ts_{i-2}. If $Ts_{i-2} > E[Ts]$, then it is optimal to send the i-th frame at $r_{i-2} + E[Ts] - 2 \cdot E[Tc]$. Figure 3.12(a) gives an example. In case $Ts_{i-2} \leq E[Ts]$, the optimal sending moment is at $t_{i-1} + E[Ts]$. See Figure 3.12(b). Hence, we get the following equation,

$$t_i = \begin{cases} r_{i-2} + E[Ts] - 2 \cdot E[Tc] & \text{if } Ts_{i-2} > E[Ts], \\ t_{i-1} + E[Ts] & \text{otherwise.} \end{cases} \tag{3.3}$$

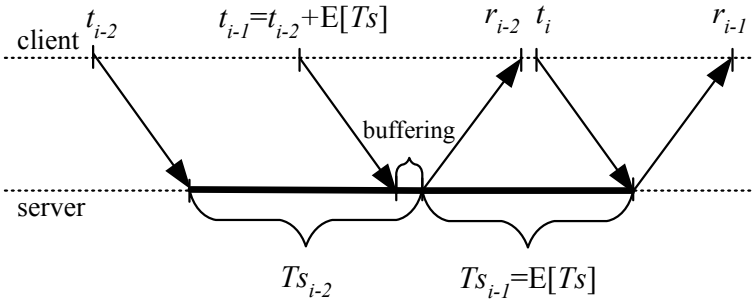
Note that using the receiving time of the $(i-2)$-th frame to determine the sending time of i-th frame indirectly takes into account the variation of the communication time between the client and the server. Therefore, the assumption $Tc_1 = Tc_2$ is not necessary any longer. Combining Equations (3.1), (3.2), and (3.3), the optimal sending time of i-th frame is given

by

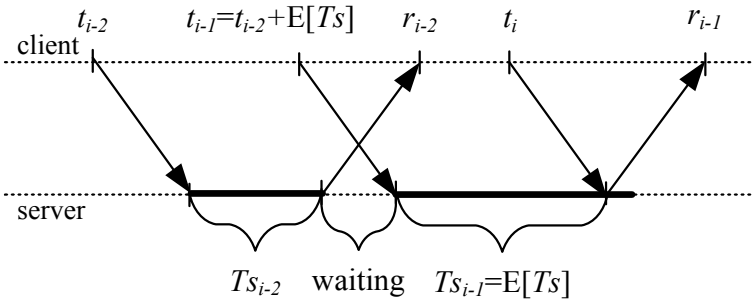
$$t_i = \min(r_{i-1}, \max(r_{i-2}, t_{i-1} + E[Ts], r_{i-2} + E[Ts] - 2E[Tc])). \quad (3.4)$$

Peak Policy

Our second method, called peak-policy (PP), tries to predict the next outlier based on historical observations. We define an outlier (i.e., a peak) as significantly different from the average processing time if the observation is much larger than the average (say 1.2 times larger). Based on the occurrences of peaks in the previous observations, we try to predict when the next peak will occur. Motivated by experiments, we observe that there is a deterministic period of the occurrences of peaks; see Figure 3.11 for the experimental results. Denote $\mathbb{P} = \{i | Ts_i \text{ is peak}\}$ as the set of peaks



(a) Optimal sending time in case of $TS_{i-2} > E[Ts]$.



(b) Optimal sending time in case of $TS_{i-2} \leq E[Ts]$.

Figure 3.12. Overview of the BLM Policy.

and denote by $\widetilde{p}_j$ the j-th element of $\mathbb{P}$. Let k be an integer number. If $\widetilde{p}_j - \widetilde{p}_{j-1} = \cdots = \widetilde{p}_{j-(k+1)} - \widetilde{p}_{j-k}$ then we say that there is a deterministic period of duration $d = \widetilde{p}_j - \widetilde{p}_{j-1}$, and we expect the next peak to occur at job number $j + d$. Note that k defines the number of previous peaks that should have occurred equidistantly with length d such that we consider the peaks as periodical events. The optimal value of k is not known beforehand. Therefore, we will start with an arbitrary value and adjust it as time evolves. Suppose that $k = 3$, and we observe three peaks each having distance d, then the method predicts that the next peak occurs after processing of d frames. If it turns out that the prediction is wrong, then we increase k by 1, since probably $k = 3$ was too low. In case the prediction is correct, then we decrease k by 1, such as to try a smaller number. To prevent meaningless values for k, we restrict k to be in $[3, \infty)$.

By combining the BLM and PP policies with one of the prediction methods to predict service processing times, we obtain our final model to deal with the JIT communication problem in real-time applications.

3.6 Experimental results

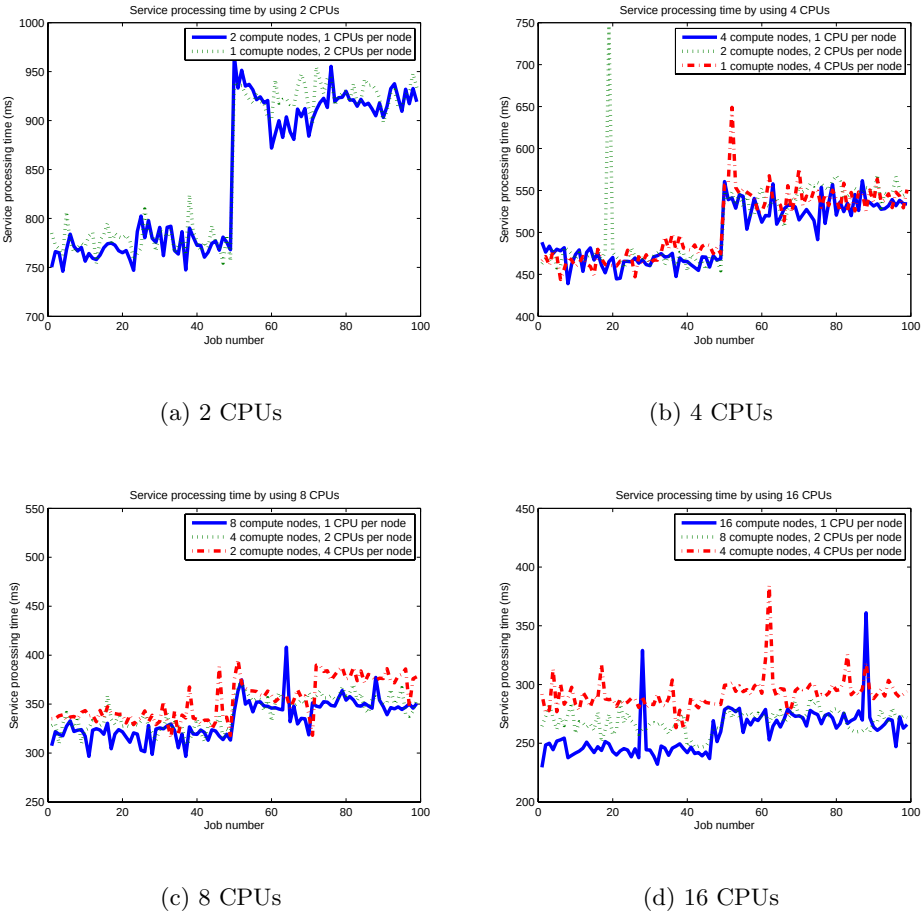
In this section we present the results of our experiments performed on the DAS-3 system. Even though our methods have been applied successfully on all DAS-3 clusters, results are shown here only for the largest cluster (VU University Amsterdam) consisting of 85 compute nodes with 4 CPUs per node. For application-specific performance results on DAS-3 as a whole, and even on a world-wide set of compute clusters, we refer to [83].

3.6.1 Resource utilization problem

We start our discussion with the numerical results of the average service processing times versus a varying total number of compute nodes. In addition, the simplicity of the LDS strategy to determine the optimal number of compute nodes is validated.

First, denote the possible solution space of the compute nodes and the number of CPUs per node as $\mathbb{O}$, where $\mathbb{O} = \{(L, n), L \in [1, \ldots, 85]$ and $n \in [1, \ldots, 4]\}$. To show that using more compute nodes and less CPUs per node provides better performance in general, we ran our real-time "Aibo" application on a varying numbers of CPUs (2, 4, 8, 16, 32, 64, and 128

CPUs). We compared the obtained service processing times for a fixed total number of CPUs, while varying the number of CPUs per nodes. The results are shown in Figure 3.13 and Figure 3.14. These figures demonstrate that for small numbers of CPUs (say, ≤ 16), the service processing time is largely independent of the ratio between the total number of employed CPUs and the number of employed CPUs per node. As the number of CPUs increases, it becomes obvious that a wider distribution of the CPUs, that is, using less CPUs per node and more compute nodes, provides better performance.



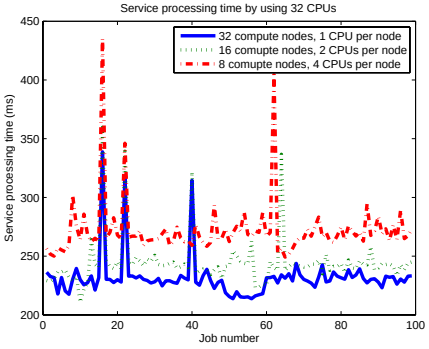
(a) 2 CPUs

(b) 4 CPUs

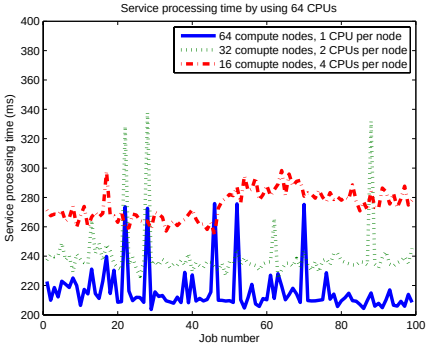
(c) 8 CPUs

(d) 16 CPUs

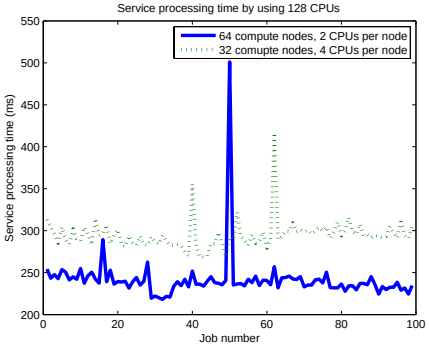
Figure 3.13. Service processing time of the Aibo application using 2, 4, 8 and 16 CPUs, respectively.



(a) 32 CPUs



(b) 64 CPUs



(c) 128 CPUs

Figure 3.14. Service processing time of the Aibo application using 32, 64 and 128 CPUs, respectively.

We also compared the service processing time for our off-line TRECVID application, on a varying total number of CPUs (16, 64 and 128 CPUs). The results are tabulated in Table 3.2. For this application we have a similar conclusion: more compute nodes and less CPUs per node provides the best performance results.

(L, n)	(16, 1)	(8, 2)	(4, 4)	(64, 1)	(32, 2)	(16, 4)	(64, 2)	(32, 4)
$S(L, n)$	669.28	682.44	736.56	241.62	244.90	263.01	190.70	218.27

Table 3.2. Average service processing time of the TRECVID application (in ms).

In Section 3.4, we mentioned that the optimal number of compute nodes is consistently found to be a power of 2. Combining this result and the observations above, we reduced the original space $\mathbb{O}$ with $85 \times 4 = 340$ possible solutions to the space $\mathbb{X}$ with 9 possible solutions, where $\mathbb{X} = \{(2^i, 1), i \in [0, \ldots, 6]\} \cup (64, 2) \cup (64, 4)$. Based on $\mathbb{X}$, we apply our LDS method to find the minimum value after $\lfloor \log_2 9 \rfloor = 3$ steps. We use Table 3.3 to explain the three steps taken in the Aibo application when $\varepsilon = 0.1$. We continue to approach the optimal number of compute nodes L^* by doubling the total number of compute nodes, until the relative improvement is less than 10%. Here the index of the elements of $\mathbb{X}$ is denoted as $[0, 1, \ldots, 8]$. Then the LDS method is applied. In the first step, we have Low = 0 and High = 8, and thus

$$\text{Mid} = \left\lfloor \frac{\text{Low} + \text{High}}{2} \right\rfloor = 4.$$

Step	Low	High	Mid	$S(e_{\text{Mid}})$	$S(e_{\text{Mid+1}})$	relative improvement	Action
1	0	8	4	(16, 1) 152.26	(32, 1) 110.64	0.27	keep high half
2	5	8	6	(64, 1) 93.58	(64, 2) 108.55	-0.15	keep low half
3	5	6	5	(32, 1) 110.64	(64, 1) 93.58	0.15	finish, return index 6

Table 3.3. Three steps to approach the optimal (L, n).

Therefore, we measure the service processing time using $2^4 = 16$ and $2^5 = 32$ compute nodes and 1 CPU per node. The measured average service

processing times and the calculated relative improvement are shown in the first row of Table 3.3. Because the relative improvement using 32 compute nodes compared to 16 compute nodes is 0.27 ($> \varepsilon$), we conclude that 16 compute nodes is not optimal. Therefore, we continue searching for the optimal. In the second step, the index value 5 (i.e., 32 compute nodes) is set as the value of Low. The value of High remains the same. Therefore Mid = 6. When calculating the relative improvement using 64 compute nodes and 2 CPUs per node compared to 2^6 compute nodes, we find that the improvement (-0.15) is less than ε. Therefore, in the third step, the value of High is reset to 6, and Low remains the same. In this case, Mid = 5. The improvement of using 2^6 compute nodes compared to 2^5 is more than ε. Thus, Low is reset to 6, such that Low is equal to High, and the whole procedure is finished. The LDS method returns index 6 as the optimal solution. This means, for $\varepsilon = 0.1$, the optimal number of CPUs is $2^6 = 64$ compute nodes. Table 3.4 shows the value of $S(L, n)$ for the Aibo application for different values of (L, n), and where ε is varied as 0.1, 0.2 and 0.3. Table 3.5 shows the results for the TRECVID application. The optimal L^* that we found for both applications for different values of ε are listed in Table 3.6. In this table, we notice that with larger ε, the L^* remains the same or decreases.

Note that our method is very simple to implement. Besides this, it is very effective because of the small number of steps required to find the optimal number of compute nodes. In addition, by varying ε, we are able to obtain the optimal result related to the desired improvement in the service processing time by increasing the number of compute nodes.

$\varepsilon = 0.1$	$(L,\ n)$	(16, 1)	(32, 1)	(64, 1)	(64, 2)
	$S(L, n)$	152.26	110.64	93.58	108.55
$\varepsilon = 0.2$	$(L,\ n)$	(16, 1)	(32, 1)	(64, 1)	(64, 2)
	$S(L, n)$	152.26	110.64	93.58	108.55
$\varepsilon = 0.3$	$(L,\ n)$	(4, 1)	(8, 1)	(16, 1)	(32, 1)
	$S(L, n)$	448.57	247.72	152.26	110.64

Table 3.4. Average service processing time of the Aibo application (in ms).

$\varepsilon = 0.1$	(L, n)	(16, 1)	(32, 1)	(64, 1)	(64, 2)	(64, 4)
	$S(L, N)$	669.28	395.79	241.62	190.70	222.61
$\varepsilon = 0.2$	(L, n)	(16, 1)	(32, 1)	(64, 1)	(64, 2)	(64, 4)
	$S(L, N)$	669.28	395.79	241.62	190.70	222.61
$\varepsilon = 0.3$	(L, n)	(16, 1)	(32, 1)	(64, 1)	(64, 2)	
	$S(L, N)$	669.28	395.79	241.62	190.70	

Table 3.5. Average service processing time of the TRECVID application (in ms).

ε	L^*
0.1	64 (64,1)
0.2	32 (32,1)
0.3	16 (16,1)

(a) Aibo

ε	L^*
0.1	128 (64,2)
0.2	128 (64,2)
0.3	64 (64,1)

(b) TRECVID

Table 3.6. Value of the engineering knee.

3.6.2 JIT communication problem

The following presents the results of our experiments relating to the JIT problem. The results are also used as input for a trace-driven simulation in order to validate our final model for determining the exact transmission moments of video frames. We limit our experiments to the Aibo application, as this is the one that needs to run under strict real-time requirements. The application is ran on 64 compute nodes using 1 CPU per node.

First, we apply the BBM method (see Figure 3.3). In our experiment, we found that the average service processing time (i.e., $E[Ts]$) and the average communication time (i.e., $E[Tc]$) between client and server amount to 143.629 ms and 11.694 ms, respectively. In this case, the server utilization is about 85%, and the average waiting time per frame is 0 ms. Consider that the service utilization using the BBM method is given by $E[Ts]/(E[Ts] + 2 \cdot E[Tc])$. This implies that when $E[Tc]$ is negligible, the BBM method approaches the optimal strategy. However, in a bottleneck situation where $E[Tc]$ is long relative to $E[Ts]$, the BBM method performs badly.

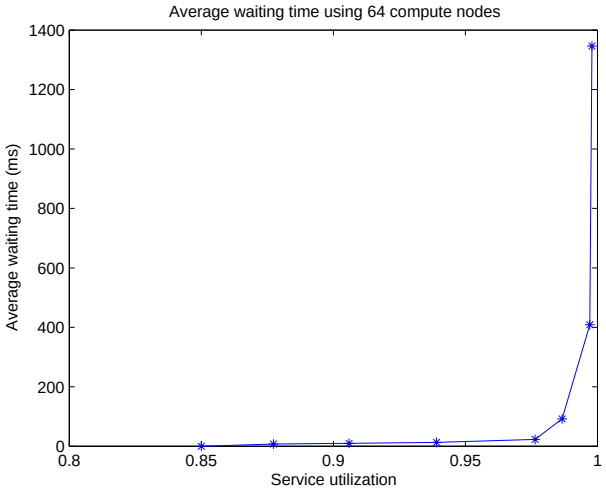


Figure 3.15. Average waiting time using 64 compute nodes.

The server utilization can be increased by sending frames with smaller intervals. However, if a sudden change (a peak) in service processing time takes place, all incoming frames are affected. A particularly difficult situation occurs when a series of long service times occurs, such that the waiting time of frames increases rapidly due to the accumulation of perceived gaps. In our experiments, we used simulation to evaluate the impact of changing the time interval between sending subsequent frames. The time interval is reduced in 5 steps according to Table 3.7. $E[Ts]$ and $E[Tc]$ in Table 3.7 are adjusted by one of the prediction methods. Recall that Figure 3.10 shows that all predic-

Simulation index	Time interval
1	Ts_{BBM}
2	$2E[Tc] + E[Ts]$
3	$1.5E[Tc] + E[Ts]$
4	$E[Tc] + E[Ts]$
5	$0.5E[Tc] + E[Ts]$
6	$0.375E[Tc] + E[Ts]$
7	$0.25E[Tc] + E[Ts]$
8	$E[Ts]$

Table 3.7. Time interval between sending two sequential frames.

tion methods are capable of generating accurate trend lines. Therefore, in this chapter, we only consider one of these (namely the exponential smoothing methods) as a representative prediction method. Figure 3.15 shows that the average waiting time increases significantly as the service utilization approaches 100%. Hence, the prediction methods are not sufficient for our just-in-time communication problem.

In our final model, in which one of the prediction methods is combined with the BLM and PP policies, we can achieve high service utilization while keeping the average waiting time low. By using the exponential smoothing methods with our policies, we obtain service utilization of about 98%, and an average waiting time per frame of around 7 ms. If we define the *waiting time percentage* (WP) as

$$\text{WP} = \frac{\text{total waiting time}}{\text{total waiting time} + \text{total service processing time}},$$

then we obtain a WP of around 3.5%. Because of the lower value of WP, we can compare the performance of our final model to the BBM method by looking at the service utilization. Define the *gain in service utilization* Gain(SU) as follows:

$$\text{Gain(SU)} = \frac{\text{service utilization with final model}}{\text{service utilization with BBM method}}. \tag{3.5}$$

Figure 3.16 shows the gain of our final model related to the BBM method for different values of $E[Tc]/E[Ts]$. In this figure, we notice that the gain in utilization is almost linear in $E[Tc]/E[Ts]$. This can be explained by the fact that the service utilization in the final model is very close to 1 and the service utilization belonging to the simple strategy can be approximated by $E[Ts]/(E[Ts] + 2 \cdot E[Tc])$. Hence, based on Equation 3.5, we have

$$\text{Gain(SU)} \approx \frac{1}{E[Ts]/(E[Ts] + 2 \cdot E[Tc])} = 1 + 2\,\frac{E[Tc]}{E[Ts]}.$$

For this reason, the gain in the service utilization is increasing nearly linearly with $E[Tc]/E[Ts]$.

The last comparison is done to evaluate the benefit brought by our policies. For the prediction method of exponential smoothing, we compare the performance of our final model to the prediction method by looking at the average waiting time. Define the *gain in the average waiting time* as follows:

$$\text{Gain(w)} = \frac{\text{average waiting time with prediction method}}{\text{average waiting time with final model}}.$$

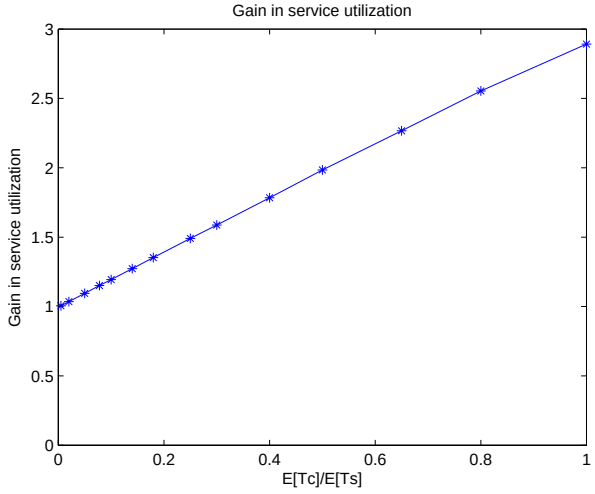


Figure 3.16. Gain in the service utilization.

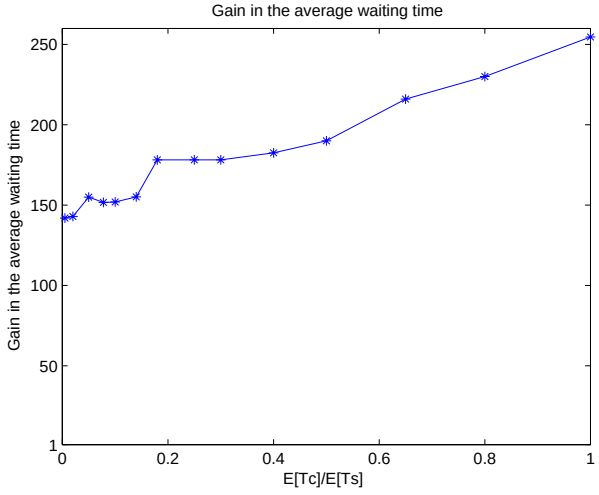


Figure 3.17. Gain in the average waiting time.

The results of this comparison are shown in Figure 3.17. The observation that the final model can gain so much in the average waiting time is explained by the following example. Assume that during processing, only one peak takes place and that, after that peak, there are still 100 frames to be processed. In this situation the use of prediction methods causes all following 100 frames to be delayed by the peak. But using our final model, there is only 1 following frame affected by the peak. Thereafter, the sending times of the next 99 frames are corrected. Thus no error accumulation occurs. Therefore, we conclude that our final model, incorporating BLM and PP, are indispensable and effective for just-in-time communication.

3.7 Conclusion and further research

In this chapter we first explored the relation between the service processing time of distributed multimedia applications and the number of compute nodes for a varying number of CPUs. We observed that there exists an engineering-knee threshold value L^* such that the service processing time decreases fast as a function of L for $L < L^*$, whereas the service processing time flattens out, and may even increase, for $L > L^*$. To find L^*, we first reduce the possible solution set, and then apply our LDS method to find L^*. Extensive validation has shown that our method is fast and effective.

Specifically, we have found that our method can find optimal resource utilization for an average-sized cluster system in no more than three evaluation steps. As a result, we conclude that our method adheres to all requirements as stated in the introduction: it is simple, easily implementable, and effective. In addition, our method takes into account system variation. Even though our focus was on the MMCA domain, our approach is general enough to be applicable in other domains as well.

Second, we have explored the JIT communication problem, that requires high service utilization on the one hand, and short service response time on the other. Using a BBM method, the waiting time is zero. However, service utilization decreases when the communication time between client and server increases. By applying existing prediction methods to this problem, service utilization can be increased. However, at the same time, the average waiting time of video frames increases even faster. This can be explained by the fact that existing prediction methods do not pay attention to peaks in the service processing time. For this reason, we have developed two innovative

policies, BLM and PP. Using the first policy, cumulative waiting times are avoided by postponing transmission of a new job when a peak is detected. The second policy is used to predict possible peaks. If we can predict the moment when a peak occurs, then we can send new jobs at the right time. Combining these two policies with any of the existing prediction methods described in this chapter, we achieve our final model to solve the just-in-time communication problem.

Our JIT model has been validated in our experiments. Moreover, we have extensively investigated the gain of our final model related to the BBM method, as well as the prediction methods without incorporating our newly developed policies. From our experimental results we conclude that our final model strongly outperforms the other methods. Specifically, we observed that, in comparison to other methods, our final model improves server utilization from 85% to 98%, and reduces the average waiting time per frame by a factor of up to 250.

The work described in this chapter is part of a larger strive to bring the benefits of high-performance computing to the multimedia community. One important aim, in this respect, is to make large-scale distributed multimedia applications variability tolerant by way of controlled adaptive resource utilization. This raises the need for new stochastic control methodologies that react to the continuously changing circumstances in large-scale Grid systems. Whereas the current chapter focuses on optimization of resource utilization under a rather static repetitive workload, whilst taking into account system variations, further sources of variability exist.

First, in MMCA applications the amount of data that needs to be processed often changes wildly over time. For one, this is because data compression techniques cause video streams to have variable bit rates. Also, in certain specific settings, cameras may only start producing data after motion has been detected. In other cases, such as iris scans performed at airports, the amount of data to be analyzed depends on external variations.

Second, MMCA algorithms themselves are a source of variability. While many algorithms working on the pixel values in images and video streams have predictable behavior, algorithms working on derived structures, such as feature vectors describing part of the content of an image, often are data-driven. A common example is support vector machine (SVM) based classification, which tries to find an optimal separation in high-dimensional clouds of labeled data points. The identification of all support vectors that fully

describe the separation depends on the positioning of the labeled data points in the high-dimensional space. Consequently, the time required to find all support vectors is largely data dependent. In the near future we will incorporate such sources of variability in our current optimization method. In addition, we will test our method on a much larger scale for a much larger variety of state-of-the-art multimedia applications. The presented example applications merely represent two of these.

Chapter 4

Optimal Resource Allocation for Time-Reservation Systems

This chapter studies the optimal resource allocation in time-reservation systems. Jobs arrive at a service facility and receive service in two steps; in the first step information is gathered from the job, which is then sent to a pool of computing resources, and in the second step the information is processed after which the job leaves the system. A central decision maker has to decide when to reserve computing power from the pool of resources, such that the job does not have to wait for the start of the second service step and that the processing capacity is not wasted due to the job still being serviced at the first step. The decision maker simultaneously has to decide on how many processors to allocate for the second processing step such that reservation and holding costs are minimized. Since an exact analysis of the system is difficult, we decompose the system into two parts which are solved sequentially, leading to nearly optimal solutions. We show via dynamic programming that the near-optimal number of processors follows a step function with as an extreme policy the bang-bang control. Moreover, we provide new fundamental insights in the dependence of the near-optimal policy on the distribution of the information gathering times. Numerical experiments demonstrate that the near-optimal policy closely matches the performance of the optimal policy of the original problem.

4.1 Introduction

Multimedia services such as iris-scan and fingerprint systems require processing of the data to identify a person's identity. To avoid delays in queues

of people waiting, these services need to work under strict real-time restrictions. To meet such restrictions, these large-scale services on the client-side send video frames captured by a scan machine to the server, which performs the analysis in a data parallel manner. In large-scale systems, the applications can reserve a number of processing resources to process the data. This gives rise to a new class of models in which the application has to decide *when* to reserve the processing resources and *how many*. In this decision making, there is a trade-off between the lead time on one hand and the operating costs on the other hand. Making a reservation too early leads to inefficiency, since the processing resources have to wait on the data gathering/scanning process; a too late reservation leads to unnecessarily long job waiting times. Allocating too many processing resources results in a short lead time, but comes at high allocation costs; allocating too few resources leads to long processing times and blocks computing resources for the next job.

In the literature, a lot of research has been devoted to resource-allocation problems. In the context of protocol design, Daniel and Chronopoulos [26] investigate the problem for resource allocation involving selfish agents, and design a truthful mechanism for solving the load-balancing problem in heterogeneous distributed systems. Park [70] presents a scalable protocol for fast co-allocation of Internet resources that ensures deadlock and livelock freedom during the resource co-allocation process. Rana et al. [76] focus on the modeling and detection of conflicts that arise during resource discovery and application scheduling. They propose an approach that helps to resolve conflicts and enables each resource and application to respond to changes in the environment.

In a Grid computing environment, there are several papers that study architectures that enhance resources with online control, online monitoring, and decision procedures. Czajkowski et al. [25] develop implementations of two co-allocation strategies in the context of the Globus toolkit [8]. In one strategy, all the required resources are specified at the time the request is made. The request succeeds if all resources required are allocated, and otherwise, the request fails and none of the resources is acquired. The other strategy allows for application-level guidance of resource selection and failure handling prior to commitment. Foster et al. [34] describe an implementation of a mechanism that enables the coordinated use of reservation and adaptation within the GARA resource management architecture [77]. Furthermore, they develop three application-level adaptive control mechanisms: two that

use loss-rate information to adapt reservations and one that uses reservation state information to adapt the transmission rate. Wang and Luo [102] set up a layered structure of Grid QoS, which provides a reasonable essence for mapping and converting QoS parameters in Grids so that it can implement the user's QoS requirements in the process of Grid resource-allocation management.

Other research has focused on economic models in a Grid computing environment. Sandholm et al. [80] develop a suite of prediction models and tools to aid the users in deciding how much funding their jobs would need to complete within a certain deadline, or conversely, when a job would be expected to complete given a budget. In this work, one tries to find the best bidding strategy based on the so-called Best Response optimization algorithm (see Feldman et al. [32]) that aims to maximize the utility of users across a set of resources, under the constraint that the total budget of the user is given. Buyya et al. [19] give an overview of different economic models for resource trading and establishing pricing strategies, and discuss the resource trading in Grid brokering and offer an infrastructure for resource management and trading in the Grid environment (see, e.g., [21, 22, 58, 96] for more details).

The works on cost/reservation optimization in the context of resource allocation that are closest to our model are [12, 68, 67, 35]. Aziz et al. [12] present a framework for resource allocation and task scheduling, where the objective function is to minimize the job completion time and to minimize the number of resources needed for the completion of the job. Nurmi et al. [68] propose a statistical method, called VARQ [69], for job scheduling. Using QBETS [67] to compute a time bound on the delay a specific user job will experience, VARQ implements a reservation by determining when a job should be submitted to a batch queue to ensure that it will be running at a particular time in the future. Fukuda et al. [35] study the relationship between the video quality and the required number of CPUs and network resources to provide a real-time video presentation. Based on this relationship, they propose a resource-allocation scheme to share resources fairly among users by solving the utility-maximization problem, where the utility is a function of the video quality and the resource-allocation costs.

The main difference between the existing literature and our work is that we aim to (1) optimize the resource-allocation costs and reservation moments *simultaneously*, and (2) satisfy a QoS constraint on the delay of a job. This is in contrast to the aforementioned works which only have a focus on optimizing either the resource-allocation costs [12, 35] or the reservation

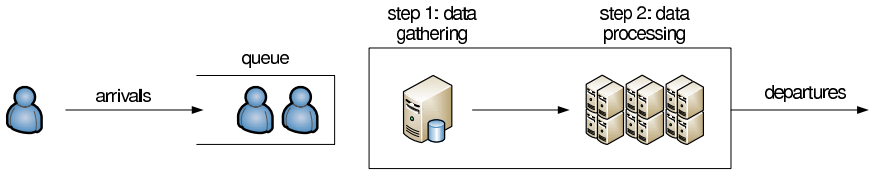


Figure 4.1. Service facility with two processing steps.

moments [67, 68]. Moreover, none of these works provides a QoS-guarantee on the sojourn time of a job in the system.

More specifically, we study the optimal resource allocation in time-reservation systems. Jobs arrive at a service facility and receive service in two steps: an information *gathering* and an information *processing* step. The system has to decide when and how much computing power to reserve for the second step such that the allocation costs are minimized while satisfying a response-time constraint. Since an exact analysis of the system is difficult, we decompose the system into two parts which are solved sequentially leading to an approximation of the original problem posed. We show via monotonicity of the dynamic programming optimality operator that the optimal number of processors for the decomposed problem follows a step function with as extreme policy the bang-bang control (see [95] for a discussion on the bang-bang control). Furthermore, we show how the optimal policy in the decomposed problem varies when the distributions of the reservation times and information gathering times and the response-time constraint are varied.

The rest of the chapter is organized as follows. In Section 4.2 we formulate the model. Next, we derive the structure of the optimal policy in the decomposed problem in Section 4.3. In Section 4.4 we illustrate these results by numerical experiments and study the impact of variability under different service distributions. Finally, in Section 4.5 we end with conclusions and discuss topics for further research.

4.2 Model formulation

Consider a service facility at which jobs arrive according to a Poisson process with rate λ. The service that the facility provides to its jobs consists of two servicing steps (see Figure 4.1): first, a job receives service at service station 1 that gathers and pre-processes data to be used in the next service step.

Then, the data is transferred to a pool of computing resources, service station 2, where the second service step takes place. After a job has received the second service step, he leaves the system. The service facility is subject to the service level constraint stating that $\mathbb{E}S \leq \alpha$, where the sojourn time S is defined as the time from the arrival of the job until the departure of the job.

We model service station 1 as a single-server station with an infinite buffer queue. A newly arriving job receives service immediately when upon arrival the station is not occupied, and he waits in the queue otherwise. The system incurs waiting costs $c_1(x)$ for having x jobs in the system, with $c_1(x)$ an increasing function in x. The service duration at the first step is modeled by the random variable R. The time to set up computing resources to process the data is modeled by the random variable T (independent of R and independent of the number of allocated computing resources [65]). We assume that the first two moments of R and T are finite. The facility can choose on the start of the service when to make a reservation for these computing resources, i.e., it wants to select the moment $s \geq 0$ at which to start the reservation after starting the service. Note that if the facility reserves the computing resources too early, then unnecessary computing resources are blocked and remain idle. Figure 4.2 represents exactly this case from the viewpoint of the job in iris-scan and fingerprint systems. The figure shows the steps a job undergoes in the system; first, a job arrival occurs after which there is a potential waiting time in the queue before he reaches the server at step 1. Upon the start of his service at step 1, a decision s is made for reserving the computing resources. This results in two competing processes; the service process R and the reservation process $s+T$. When *both* processes have completed, the job moves on to the service in step 2, while still blocking the server in step 1 (there can only be at most one job in service in both steps). After the service at step 2, the job departs the system and a new job can be served at step 1. Recall that the sojourn time S is defined as the time from the arrival of the job until the departure of the job. On the other hand, if the resources are reserved too late, then the job has to wait before the system can process his data leading to high waiting costs. This situation is reflected in Figure 4.3. The aim in this service step is to balance the end of the service time and the moment that the resources are reserved and available, e.g., the system tries to choose s such that $C(s) := \mathbb{E}(R - s - T)^2$ is minimized.

After having received service at service station 1, the job remains in front

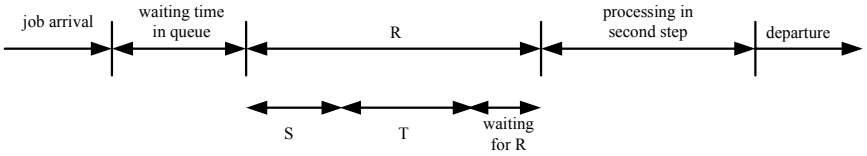


Figure 4.2. Computing resources are available before end of service at step 1.

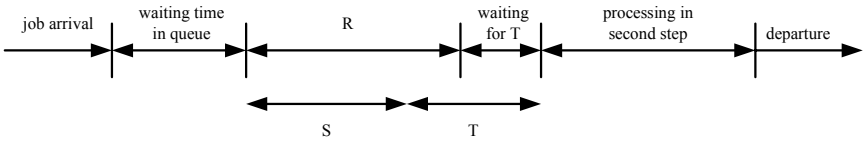


Figure 4.3. The service at step 1 is finished while computing resources are not available.

of service station 1, but now receives service from service station 2. Service station 2 is a pool of computing resources consisting of A identical parallel processors (e.g., the DAS-3 environment [2]). When the reservation time of the computing resources has finished, the job can be served at service station 2. The service facility can choose how many processors to use for the processing of the data. We assume that there are costs $c_2(a)$ for using a processors per time unit, with $c_2(a)$ an increasing function in a. Clearly, having too few processors leads to a violation of the service level, and having too many processors leads to high reservation costs. We assume that the service follows an exponential distribution with service rate $\mu(a)$ when a processors are used, with $\mu(.)$ an increasing function in a. In the ideal case one would have $\mu(a) = \mu a$, with a fixed service rate μ, however, due to communication overhead the function is sublinear in practice. Moreover, we suppose that A is large enough so that for the optimal choice of s and a, say (s^*, a^*), there is a policy that meets the service level α.

So far, we have not made any assumptions on the parameters of the system. However, to ensure a stable system one needs to add conditions that take into account both the arrival rate of jobs as well as the mean service time of a job in the system. Note that the latter depends heavily on the policy (s^*, a^*) in use for every state. Therefore, it is difficult to state general stability conditions. One can note, though, that if $(s^*, a^*) = (0, A)$ for all states, then the mean service time is minimized; the mean service time is then

given by $\mathbb{E}\max\{R,T\} + 1/\mu(A)$. Hence, if $\lambda\big[\mathbb{E}\max\{R,T\} + 1/\mu(A)\big] < 1$, then there exists at least one policy for which the system is stable.

Problem formulation
The objective is to find a policy that selects (s^*, a^*) based on the number of jobs x in the system such that *both* $C(s)$ and the long-term average costs (i.e., waiting plus reservation costs) are minimized while at the same time the service level constraint α is met. To formulate this objective in mathematical terms, the following notation is useful. Let X_t and A_t denote the random variables denoting the number of jobs in the system and the number of reserved processors at time t, respectively. Let π denote a state-dependent policy. Then the optimization problem can be stated as

$$\begin{aligned} &\min_{\pi} \big[C(s) + \mathbb{E}_\pi[c_1(X_\infty) + c_2(A_\infty)]\big] \\ &\text{subject to } \mathbb{E}_\pi[S] \leq \alpha, \end{aligned} \tag{4.1}$$

where $C(s)$ is given by $\mathbb{E}(R - s - T)^2$, X_∞ and A_∞ denote the steady-state numbers of the variables X_t and A_t, and where S is the steady-state sojourn time of an arbitrary job.

Solution approach
The service steps in the above mentioned problem have a mixture of continuous and discrete decision variables with intricate dependencies such that a classical decision theoretic approach is not computationally tractable in general. Therefore, we approximate the optimal policy by following a two-step approach: First, we solve for s^* minimizing $C(s)$ in the first service step. Second, given s^*, we find a^* that minimizes $\mathbb{E}_{(s^*,a)}[c_1(X_\infty) + c_2(A_\infty)]\big]$ subject to $\mathbb{E}_{(s^*,a)}[S] \leq \alpha$ under assumptions provided in Section 4.3. In Section 4.4 we show that the resulting approximate policy is nearly optimal in the numerical experiments.

4.3 Structural properties of the decomposed model

In order to derive an optimal policy (s^*, a^*) we assume that the optimal choice for s does not depend on the policy for the number of servers. Note that this assumption allows us to decompose the problem into two parts, whereby the optimal reservation moment s^* can be studied independently of the optimal allocation a^*. Then, given the optimal parameter s^*, one can

study the optimal reservation policy at service station 2. Also note that this assumption is not true in general in the original problem, e.g., when $c_1(x)$ and $c_2(a)$ are really large relative to $C(s)$, then one would choose $s = 0$ (in order to minimize the holding costs $c_1(x)$), whereas $s > 0$ in our decomposed problem (since the decomposed model for station 1 does take into account the holding costs). However, in Section 4.4 we numerically compare our decomposition technique with the optimal policy in the original problem, and show that for realistic parameter values the decomposition technique is close to optimal.

We first start with the optimal reservation moment s^*.

Lemma 4.3.1. *Let R and T have a general distribution with non-negative supports. Then $C(s) = \mathbb{E}(R - s - T)^2$ is minimized by $s^* = [\mathbb{E}R - \mathbb{E}T]^+$, resulting in $C(s^*) = \mathbb{V}arR + \mathbb{V}arT$ for $\mathbb{E}R \geq \mathbb{E}T$, and in $C(s^*) = \mathbb{V}arR + \mathbb{V}arT + (\mathbb{E}R - \mathbb{E}T)^2$ for $\mathbb{E}R < \mathbb{E}T$.*

Proof. Let R and T have a general distribution with non-negative supports. Then,

$$\begin{aligned}
\mathbb{E}(R-s-T)^2 &= \mathbb{E}(R^2 + s^2 + T^2 - 2sR - 2RT + 2sT) \\
&= \mathbb{E}R^2 + \mathbb{E}T^2 - 2\mathbb{E}R\,\mathbb{E}T + s^2 - 2s\mathbb{E}R + 2s\mathbb{E}T \\
&= \mathbb{V}\mathrm{ar}\,R + (\mathbb{E}R)^2 + \mathbb{V}\mathrm{ar}\,T + (\mathbb{E}T)^2 - 2\mathbb{E}R\mathbb{E}T + s^2 - 2s(\mathbb{E}R - \mathbb{E}T) \\
&= \mathbb{V}\mathrm{ar}\,R + \mathbb{V}\mathrm{ar}\,T + (\mathbb{E}R - \mathbb{E}T)^2 + [s - (\mathbb{E}R - \mathbb{E}T)]^2 - (\mathbb{E}R - \mathbb{E}T)^2 \\
&= \mathbb{V}\mathrm{ar}\,R + \mathbb{V}\mathrm{ar}\,T + [s - (\mathbb{E}R - \mathbb{E}T)]^2.
\end{aligned}$$

From this expression it follows that $C(s)$ is minimized by taking s equal to $\mathbb{E}R - \mathbb{E}T$. Since s is only allowed to be non-negative, $s^* = [\mathbb{E}R - \mathbb{E}T]^+$, with $[z]^+ = \max\{z, 0\}$. The value of $C(s^*)$ then readily follows by substitution. □

In the special case that R and T follow an exponential distribution with parameters δ and γ, respectively, we have that the optimal reservation time is given by

$$s^* = \arg\min_s \mathbb{E}(R - T - s)^2 = \left[\frac{\gamma - \delta}{\delta\gamma}\right]^+. \tag{4.2}$$

In the case of $\gamma > \delta$, i.e., the average reservation time is smaller than the average service time, we have

$$C(s^*) = \mathbb{E}(R - T - s^*)^2 = \frac{\gamma^2 + \delta^2}{\delta^2\gamma^2} = \frac{1}{\delta^2} + \frac{1}{\gamma^2}. \tag{4.3}$$

Otherwise, we have

$$C(s^*) = \mathbb{E}(R-T)^2 = \mathbb{E}(R^2)+\mathbb{E}(T^2)-2\mathbb{E}(R)\mathbb{E}(T) = \frac{2}{\delta^2}+\frac{2}{\gamma^2}-\frac{2}{\delta\gamma}. \quad (4.4)$$

Note that Equation (4.3) implies that if $\gamma \gg \delta$ or $\mathbb{E}[R] \gg \mathbb{E}[T]$, then the $\mathbb{E}R$ dominates in the expression of $C(s)$. Hence, relatively, the $\mathbb{E}T$ is not important, thus approximately, it is as good as to reserve the computing nodes directly after the scanning is completed. In this case, $s^* \approx \frac{1}{\delta}$. Equation (4.2) implies that if $\gamma < \delta$ or $\mathbb{E}[R] < \mathbb{E}[T]$, then it is optimal to reserve the computing nodes directly after the scanning starts, thus $s^* = 0$. Note that the expected time between the start of a service at station 1 and station 2 defined by $\beta := \mathbb{E}\max\{R, s^*+T\}$, is given by $\beta = s^* + \frac{1}{\gamma} + e^{-\delta s^*}\left[\frac{1}{\delta} - \frac{1}{\delta+\gamma}\right]$.

Note also that action s^* constitutes a stationary policy for all jobs. In general, the arrival process to station 2 is *not* the original Poisson process delayed by the constant β. However, for the purpose of model tractability, we assume that this does hold. To meet the service level that was specified as the requirement that the mean sojourn time of a job does not exceed α, the mean sojourn time at station 2 is allowed to be at most $\alpha' := \alpha - \beta$ time units. In Section 4.4 we will show that the error on the mean sojourn time in the system by assuming a Poisson process is not too big.

The assumption that the output process of station 1 is a Poisson process makes the derivation of the optimal resource-allocation policy for station 2 (i.e., how many processors should be assigned to a job) tractable. For this purpose, we formulate the problem as a constrained Markov decision problem. Thus, we try to minimize the holding and processing costs with the additional constraint that the mean sojourn time is below α'. The constrained Markov decision problem is, in general, hard to solve. Therefore, we first study the unconstrained problem in which we drop the sojourn time constraint. For this system, we show that the optimal policy possesses a threshold-type structure. We show that the constrained problem possesses a similar structure as the unconstrained problem. Therefore, the structure of the optimal policy of the unconstrained case carries over to the constrained problem, noting that in the latter case the optimal policy is not deterministic but randomized (see the first paragraph after Theorem 4.3.3 for details).

4.3.1 Unconstrained Markov decision problem

Let $\mathcal{X} = \mathbb{N}_0 = \{0, 1, 2, \ldots\}$ denote the state space, where $x \in \mathcal{X}$ denotes the number of jobs present at station 2. For each job the set of actions is given by $\mathcal{A} = \{0, 1, \ldots, A\}$, where $a \in \mathcal{A}$ denotes the number of processors that is allocated to a job. When action a is chosen in state x, there are two possible events in the system. First, an arrival of a job can occur with rate λ. Second, since at any moment in time there can only be one job in service, the job finishes his service with rate $\mu(a)$. Note that if multiple jobs are allowed to be served simultaneously, then the allocation policy for each job needs to be kept in the state description as well, since every job is served with a different rate.

Next, we uniformize the system (see Section 11.5 of Puterman [73]). To this end, we assume that the uniformization constant $\lambda+\mu(A) = 1$; we can always get this by scaling. Uniformizing is equivalent to adding dummy transitions (from a state to itself) such that the rate out of each state is equal to 1; then we can consider the rates to be transition probabilities. Thus, the transition probabilities p, are given by $p(x, a, x+1) = \lambda$ and $p\big(x, a, [x-1]^+\big) = \mu(a)$. Similarly, the system is subject to costs c consisting of holding costs $c_1(x)$ and costs for the resource allocation $c_2(a)$, thus $c(x, a) = c_1(x) + c_2(a)$. The tuple $(\mathcal{X}, \mathcal{A}, p, c)$ defines the Markov decision problem.

Define a deterministic policy π as a function from $\mathcal{X}$ to $\mathcal{A}$, i.e., $\pi(x) \in \mathcal{A}$ for all $x \in \mathcal{X}$. Let $u_t^\pi(x)$ denote the total expected costs up to time t when the system starts in state x under policy π. Note that for any stable and work-conserving policy, the Markov chain satisfies the unichain condition, so that the average expected costs $g(\pi) = \lim_{t\to\infty} u_t^\pi(x)/t$ is independent of the initial state x (see Proposition 8.2.1 of Puterman [73]). The goal is to find a policy π^* that minimizes the long-term average costs, thus $g = \min_\pi g(\pi)$.

Let $V(x)$ be a real-valued function defined on the state space. This function will play the role of the relative value function, i.e., the asymptotic difference in total costs that results from starting the process in state x instead of some reference state. The long-term average optimal actions are a solution of the optimality equation (in vector notation) $g+V = TV$, where T is the dynamic programming operator acting on V defined as follows:

$$\begin{aligned} TV(x) = {} & \lambda V(x+1) + c_1(x) \\ & + \min_{a\in\mathcal{A}} \big\{\mu(a)V\big([x-1]^+\big) + \big(\mu(A) - \mu(a)\big)V(x) + c_2(a)\big\}. \end{aligned} \tag{4.5}$$

The first term in the expression $TV(x)$ models the arrivals of customers to station 2. The second term denotes the holding costs. The third term denotes the departure of a customer in case action a has been chosen. The fourth term is the uniformization constant. The last term models the costs for the resource allocation. The optimality equation $g + V = TV$ is hard to solve analytically in practice. Alternatively, the optimal actions can also be obtained by recursively defining $V_{l+1} = TV_l$ for arbitrary V_0. For $l \to \infty$, the maximizing actions converge to the optimal ones (for existence and convergence of solutions and optimal policies we refer to Chapter 8 of Puterman [73]). This procedure is also called value iteration.

Note that the costs $c_1(x)$ and $c_2(a)$ model the trade-off between the server allocation and the holding costs. The first cost function drives the decision maker to be liberal with the resources, since a too low resource allocation leads to long queues and thus high holding costs. The second cost function ensures that the decision maker is not too liberal since a too high resource allocation leads to high processor costs. This is illustrated by the following two cases.

No processing costs: First, we take $c_2(a)$ to be zero for all $a \in \mathcal{A}$. From Expression (4.5) we can see that the optimal policy evidently uses all processors (also in the original system), since this maximizes the service rate and minimizes the queue length (since $c_2(a) = 0$). Therefore $a^* = A$.

No holding costs: Next, we assume that $c_1(x) = 0$ for all $x \in \mathcal{X}$. In this case, Expression (4.5) shows that the emphasis is on processor utilization. Therefore, the optimal action would be to use no processors at all. In practice, this is not a permissible action, since no services would be provided at all. If the constraint on the sojourn time is taken into account, then the optimal action changes. This will be illustrated in Section 4.3.2.

We now proceed to the general solution of the unconstrained Markov decision problem by deriving properties of the optimal dynamic policy. To this end, consider the backward recursion operator V_{n+1} given by

$$\begin{aligned} V_{n+1}(x) = \; & \lambda V_n(x+1) + c_1(x) \\ & + \min_{a \in \mathcal{A}} \big\{ \mu(a) V_n\big([x-1]^+\big) + \big(\mu(A) - \mu(a)\big) V_n(x) + c_2(a) \big\}. \end{aligned} \tag{4.6}$$

Using the backward recursion operator we can derive structural properties

of the optimal policy. The following result will play an important role for the derivation of the structure of the optimal policy.

Lemma 4.3.2. *Assume that $c_1(x)$ is a convex increasing function in x and that $c_2(a)$ is an increasing function of a. Then $V(x)$ is a convex increasing function in x.*

Proof. The proof is by induction on n. Let $V_0(x) = x^2$ for all x. Then clearly, V_0 is a convex increasing function in x. Now, assume that the statement holds for k, then we prove that the statement also holds for $k+1$. To this end, define

$$T_a^k(x) = \mu(a)V_k([x-1]^+) + \big[\mu(A) - \mu(a)\big]V_k(x) + c_2(a).$$

Then, for $x \geq 0$, we have

$$\begin{aligned} V_{k+1}(x+1) - V_{k+1}(x) = \; & [\lambda V_k(x+2) - \lambda V_k(x+1)] + [c_1(x+1) - c_1(x)] \\ & + \; [\min_a\{T_a^k(x+1)\} - \min_a\{T_a^k(x)\}] \\ & > \min_a\{T_a^k(x+1)\} - \min_a\{T_a^k(x)\}. \end{aligned}$$

The inequality holds because the first two terms are non-negative due to the induction hypothesis, and the second two terms follow by the assumption on the cost function c_1. Let $a^* \in \arg\min_a\{T_a^k(x+1)\}$, then, for $x \geq 0$, we have

$$\begin{aligned} V_{k+1}(x+1) - V_{k+1}(x) > \; & T_{a^*}^k(x+1) - \min\{T_a^k(x)\} \\ \geq \; & T_{a^*}^k(x+1) - T_{a^*}^k(x) \\ = \; & \mu(a^*)V_k(x) + \big[\mu(A) - \mu(a^*)\big]V_k(x+1) \\ + \; & c_2(a^*) - \mu(a^*)V_k([x-1]^+) \\ - \; & \big[\mu(A) - \mu(a^*)\big]V_k(x) - c_2(a^*) > \; 0. \end{aligned}$$

Therefore, by induction, we derive that $V(x+1) - V(x) > 0$. Now, we proceed to prove convexity of the relative value function. Assume that convexity holds for k, then we need to prove convexity for $k+1$. Then, for $x \geq 0$, we have

$$\begin{aligned} & V_{k+1}(x+1) - 2V_{k+1}(x) + V_{k+1}([x-1]^+) \\ & \quad = \; \big[\lambda V_k(x+2) - 2\lambda V_k(x+1) + \lambda V_k(x)\big] \\ & \quad + \; \big[c_1(x+1) - 2c_1(x) + c_1([x-1]^+)\big] \\ & \quad + \; \big[\min_a\{T_a^k(x+1)\} - 2\min_a\{T_a^k(x)\} + \min_a\{T_a^k([x-1]^+)\}\big] \\ & \quad > \; \min_a\{T_a^k(x+1)\} - 2\min_a\{T_a^k(x)\} + \min_a\{T_a^k([x-1]^+)\}. \end{aligned}$$

The inequality holds because the first expression between the brackets is non-negative due to the induction hypothesis, and the second from the assumption on c_1. Now assume that $a_1^* \in \arg\min_a\{T_a^k(x+1)\}$ and $a_2^* \in \arg\min_a\{T_a^k([x-1]^+)\}$. Then, we have

$$\begin{aligned}
&\min_a\{T_a^k(x+1)\} - 2\min_a\{T_a^k(x)\} + \min_a\{T_a^k([x-1]^+)\} \\
&\quad\geq \big[c_2(a_1^*) - c_2(a_1^*) - c_2(a_2^*) + c_2(a_2^*)\big] \\
&\quad+ \big[\mu(a_1^*)V_k(x) - \mu(a_1^*)V_k\big([x-1]^+\big) - \mu(a_2^*)V_k\big([x-1]^+\big) \\
&\quad+ \mu(a_2^*)V_k\big([x-2]^+\big)\big] \\
&\quad+ \big[\big(\mu(A) - \mu(a_1^*)\big)V_k(x+1) - \big(\mu(A) - \mu(a_1^*)\big)V_k(x) \\
&\quad- \big(\mu(A) - \mu(a_2^*)\big)V_k(x) + \big(\mu(A) - \mu(a_2^*)\big)V_k\big([x-1]^+\big)\big] \\
&\quad= \big(\mu(a_1^*) - \mu(a_2^*)\big)\big[V_k(x) - V_k\big([x-1]^+\big)\big] \\
&\quad+ \mu(a_2^*)\big[V_k(x) - 2V_k\big([x-1]^+\big) + V_k\big([x-2]^+\big)\big] \\
&\quad+ \big(\mu(A) - \mu(a_1^*)\big)\big[V_k(x+1) - V_k(x)\big] \\
&\quad- \big[\mu(A) - \mu(a_1^*) + \big(\mu(a_1^*) - \mu(a_2^*)\big)\big]\big(V_k(x) - V_k([x-1]^+)\big) \\
&\quad= \mu(a_2^*)\big[V_k(x) - 2V_k\big([x-1]^+\big) + V_k\big([x-2]^+\big)\big] \\
&\quad+ \big(\mu(A) - \mu(a_1^*)\big)\big[V_k(x+1) - 2V_k(x) + V_k\big([x-1]^+\big)\big] > 0.
\end{aligned}$$

The first inequality follows from taking a potentially suboptimal action in the second term of $\min_a\{T_a^k(x+1)\} - 2\min_a\{T_a^k(x)\} + \min_a\{T_a^k([x-1]^+)\}$. The two equalities follow by rearranging the terms. The last inequality follows by the induction hypothesis and by noting that $\mu(A) - \mu(a_1^*)$ is positive. Hence, using mathematical induction we have proved that $V(x)$ is a convex increasing function in x. □

The previous lemma shows that V is a convex increasing function in x. This property of the relative value function will play a crucial role in the derivation of the optimal policy. The next theorem shows how this lemma is used to obtain the structure of the optimal policy.

Theorem 4.3.1. *Assume $c_1(x)$ is a convex increasing function in x, $c_2(a)$ and $\mu(a)$ are strictly increasing function in a. Then the optimal resource allocation strategy is given by a non-decreasing curve, i.e., for each $a \in \arg\min_{a\in\mathcal{A}_{x+1}}\big\{T_a(x+1)\big\}$ and $b \in \arg\min_{a\in\mathcal{A}_x}\big\{T_a(x)\big\}$ we have $a \geq b$ for all $x \geq 0$.*

Proof. Let $a \in \arg\min_{a \in \mathcal{A}_{x+1}} \{T_a(x+1)\}$ and $b \in \arg\min_{a \in \mathcal{A}_x} \{T_a(x)\}$ be an arbitrary optimal allocation in states $x+1$ and x, respectively. The proof is by contradiction. Suppose that b and a such that $a < b$, then

$$\begin{aligned} T_a(x) - T_b(x) &= \big[\mu(b) - \mu(a)\big]\big[V(x) - V\big([x-1]^+\big)\big] - \big[c_2(b) - c_2(a)\big] \\ &\geq 0. \end{aligned}$$

By Property 4.3.2 we have that V is a convex increasing function in x. Together with the fact that $\mu(\cdot)$ is a strictly increasing function, and thus $\mu(b) - \mu(a) > 0$, we have

$$\begin{aligned} T_a(x+1) - T_b(x+1) &= \big[\mu(b) - \mu(a)\big]\big[V(x+1) - V\big(x\big)\big] - \big[c_2(b) - c_2(a)\big] \\ &> \big[\mu(b) - \mu(a)\big]\big[V(x) - V\big([x-1]^+\big)\big] - \big[c_2(b) - c_2(a)\big] \\ &\geq 0. \end{aligned}$$

However, this implies that a is not optimal for state $x+1$, since b has a a smaller value. Hence, $a \geq b$, and this establishes the non-decreasing curve as stated in the theorem. □

When additional assumptions are placed on the growth of processing costs c_2 and the service rate μ, one can specify the optimal policy in greater detail. The following theorem shows that if the processing costs are taken to be concave while the service rate is convex, then the optimal policy is a threshold-based bang-bang control policy.

Theorem 4.3.2. *Assume $c_1(x)$ is a convex increasing function in x, $c_2(a)$ is a concave increasing function in a, and $\mu(a)$ is a convex function in a. Then the optimal resource-allocation strategy is a threshold-based bang-bang control policy, i.e., there exists a constant τ such that for $x < \tau$ the optimal action is to allocate no processors, while for $x \geq \tau$ the optimal action is to allocate all processors.*

Proof. Suppose that $c_1(x)$ is a convex increasing function in x and that $c_2(a)$ is a concave increasing function in a. To obtain the structure of the policy, consider Z_i for $i = 0, \ldots, A-1$ given by

$$\begin{aligned} Z_i(x) = T_i(x) - T_{i+1}(x) = \ & \big[\mu(i+1) - \mu(i)\big]\big[V(x) - V\big([x-1]^+\big)\big] \\ - \ & \big[c_2(i+1) - c_2(i)\big]. \end{aligned}$$

Observe that for fixed x, the function $Z_i(x)$ is increasing in i since $\mu(i)$ is a convex function and $c_2(i)$ is concave. When $Z_i(x)$ is non-negative for some

i, this means that using one processor more is better. However, since $Z_i(x)$ is increasing in i, this means that using all processors is optimal. Similarly, when $Z_{i-1}(x)$ is non-positive, then using one processor less is better. But since $Z_i(x)$ is an increasing function in i, using no processor at all is optimal. To establish the threshold τ, consider state $x+1$. If $Z_i(x)$ is non-negative, then $Z_i(x+1)$ is non-negative as well due to Lemma 4.3.2. Thus, when in state x it is optimal to use all processors, then in all states greater than x it is also optimal to use all processors. This establishes the bang-bang control policy, i.e., there exists a $\tau \geq 0$ such that the optimal policy for $x < \tau$ is to use no processors, and for $x \geq \tau$ using all processors is optimal. □

Theorem 4.3.2 shows that the optimal policy is either to use no processing capacity at all or to use all processors. The convexity/concavity properties of c_2 and μ are essential to obtain this result. When $c_2(a)$ is convex and $\mu(a)$ is concave in a, then this policy is not optimal anymore. In fact, the optimal policy is still of threshold type, but with more threshold levels (as specified by Theorem 4.3.1). However, the precise form heavily depends on the parameters used in the problem formulation. In the numerical experiments we will show that for some parameter values the policy behaves as a bang-bang policy, but for other parameter settings the multi-threshold control policy appears to be optimal.

4.3.2 Constrained Markov decision problem

In this section we describe the constrained Markov decision problem so that optimal policies can be determined that ensure that the total mean sojourn time of a job in the whole system is less than α. Since the average time spent in the first step is β, the mean sojourn time in the second step is not allowed to exceed $\alpha' = \alpha - \beta$. In the previous subsection, we studied the unconstrained Markov decision problem in which we tried to find a policy π^* that minimized $g(\pi)$, i.e., the long-term average costs. We did not take into account the mean sojourn time constraint of a job. Let W denote the sojourn time of an arbitrary job in step 2. Then, the constrained Markov decision problem that we want to solve is:

$$\min_{\pi} g(\pi) \quad \text{subject to} \quad \mathbb{E}W \leq \alpha'.$$

Before the discussion of the general constrained case, we first examine the effect of the cost functions c_1 and c_2 on the optimal policy if one of them is

set equal to zero.

No processing costs: When the processing costs are set to zero, i.e., $c_2(a) = 0$, then we obtain a situation that is equal to the case of the unconstrained Markov decision problem. The optimal policy is $a^* = A$, which (by assumption) clearly satisfies the service level α'.

No holding costs: When we assume $c_1(x) = 0$, then we cannot use the optimal policy of the unconstrained Markov decision problem: $a^* = 0$. In the case of the constrained problem, this would violate the service level constraint α'. For a given action a, the sojourn time in the system can be given by the sojourn time in an M/M/1 queue with arrival rate λ and service rate $\mu(a)$ (see Cooper [23]). Thus, let $\rho(a) = \lambda/\mu(a)$, then the expected sojourn time $\mathbb{E}W$ is given by

$$\mathbb{E}W = \frac{\rho(a)}{\lambda\big(1 - \rho(a)\big)}.$$

The optimal action a^*, under the assumption of Poisson arrivals at station 2, is then obtained by solving the above equation for $\mathbb{E}W = \alpha'$. Note that in general this will result in a fractional value of a^*. This leads in a natural way to a randomized policy to obtain exactly α'. We will discuss the randomized policy further in the sequel.

We now proceed to the general solution of the constrained Markov decision problem. Note that the system basically resembles an M/M/1 queueing system with state-dependent service rates. For this system, due to Little's Law, we can relate the number of jobs L in the system to the sojourn time in the system W by $\mathbb{E}L = \lambda\mathbb{E}W$ (see Little [61]). In order to get $\mathbb{E}L$, one can take x as cost function in the Markov decision problem. Hence, to obtain the expected sojourn time $\mathbb{E}W$ it suffices to have as cost function $c_3(x) = x/\lambda$. To solve the constrained problem we use the Lagrange multiplier approach described in Section 12.6 of Altman [10]. Let $\mathcal{L}$ be the Lagrange multiplier,

then the dynamic programming operator T acting on $V_{\mathcal{L}}$ is defined as follows:

$$\begin{aligned} TV_{\mathcal{L}}(x) = &\ \lambda V_{\mathcal{L}}(x+1) \\ &+ \min_{a\in\mathcal{A}} \big\{\mu(a)V_{\mathcal{L}}\big([x-1]^+\big) + \big(\mu(A)-\mu(a)\big)V_{\mathcal{L}}(x) + c_2(a)\big\} \\ &+ c_1(x) + \mathcal{L}c_3(x) \\ = &\ \lambda V_{\mathcal{L}}(x+1) \\ &+ \min_{a\in\mathcal{A}} \big\{\mu(a)V_{\mathcal{L}}\big([x-1]^+\big) + \big(\mu(A)-\mu(a)\big)V_{\mathcal{L}}(x) + c_2(a)\big\} \\ &+ c_1^{\mathcal{L}}(x), \end{aligned} \tag{4.7}$$

with $c_1^{\mathcal{L}}(x) = c_1(x) + \mathcal{L}c_3(x)$ a convex increasing function in x. Note that the cost function is similar in structure to that of Equation (4.5). Hence, Lemma 4.3.2, Theorem 4.3.1, and Theorem 4.3.2 apply to this case as well. Thus, for every value of $\mathcal{L}$ the optimal policy is a non-decreasing curve. Note that when for a given $\mathcal{L}$ an optimal policy has been derived by iterating Equation (4.7), one can obtain the mean sojourn time $\mathbb{E}W$ by fixing the policy in Equation (4.7) while setting $c_1 = c_2 = 0$ and $\mathcal{L} = 1$. In Theorem 4.3.3 below, we will show that the maximizing allocation actions in Equation (4.7) increase with $\mathcal{L}$, which will imply that the mean sojourn time $\mathbb{E}W$ decreases in $\mathcal{L}$ since more processors are being allocated. In order to obtain Theorem 4.3.3, we first need to prove the following two technical lemmas.

Lemma 4.3.3. *$V_{\mathcal{L}}(x)$ is increasing in the Lagrange multiplier $\mathcal{L}$ for all $x \in \mathcal{X}$.*

Proof. To prove this lemma, we only need to prove that if $\mathcal{L}' > \mathcal{L}$ then $V_{\mathcal{L}'}(x) \geq V_{\mathcal{L}}(x)$ for all x. This is proven by induction in n. Let $V^0_{\mathcal{L}'}(x) = V^0_{\mathcal{L}}(x) = 0$ for all x. Note that $V^n_{\mathcal{L}}(x)$ is defined similarly to $V_n(x)$. Then, the lemma holds for $n = 0$. Assume it holds for $n = k$. We prove it also holds for $n = k+1$. Define $T^k_{a(\mathcal{L})}(x) = \mu(a)V_{\mathcal{L},k}([x-1]^+) + \big[\mu(A)-\mu(a)\big]V_{\mathcal{L},k}(x) + c_2(a)$. Then, for $x \geq 0$, we have

$$\begin{aligned} V_{\mathcal{L}',k+1}(x) - V_{\mathcal{L},k+1}(x) = &\ \lambda\big[V_{\mathcal{L}',k}(x+1) - V_{\mathcal{L},k}(x+1)\big] \\ &+ \min_a T^k_{a(\mathcal{L}')}(x) - \min_a T^k_{a(\mathcal{L})}(x) + (\mathcal{L}'-\mathcal{L})c_3(x) \\ \geq &\ \min_a T^k_{a(\mathcal{L}')}(x) - \min_a T^k_{a(\mathcal{L})}(x). \end{aligned}$$

The inequality above holds because the term between the brackets is non-negative due to the induction hypothesis and the last term follows by the

assumption $\mathcal{L}' > \mathcal{L}$. Let $a^* \in \arg\min_a T^k_{a(\mathcal{L}')}(x)$. Then for $x \geq 0$, we have

$$\begin{aligned}
V_{\mathcal{L}',k+1}(x) - & V_{\mathcal{L},k+1}(x) \\
&\geq T^k_{a^*(\mathcal{L}')}(x) - \min_a T^k_{a(\mathcal{L})}(x) \\
&\geq T^k_{a^*(\mathcal{L}')}(x) - T^k_{a^*(\mathcal{L})}(x) \\
&= \mu(a^*)V_{\mathcal{L}',k}([x-1]^+) + \big[\mu(A) - \mu(a^*)\big]V_{\mathcal{L}',k}(x) + c_2(a^*) \\
&- \mu(a^*)V_{\mathcal{L},k}([x-1]^+) - \big[\mu(A) - \mu(a^*)\big]V_{\mathcal{L},k}(x) - c_2(a^*) \\
&= \mu(a^*)\big[V_{\mathcal{L}',k}([x-1]^+) - V_{\mathcal{L},k}([x-1]^+)\big] \\
&+ \big[\mu(A) - \mu(a^*)\big]\big[V_{\mathcal{L}',k}(x) - V_{\mathcal{L},k}(x)\big] \geq 0.
\end{aligned}$$

Therefore, by induction we derive that $V_{\mathcal{L}'}(x) \geq V_{\mathcal{L}}(x)$. □

Lemma 4.3.3 shows that the relative value function is increasing in the Lagrange multiplier. This result is needed to show that the relative value function is also supermodular, as the following lemma shows.

Lemma 4.3.4. *If $\mathcal{L}' > \mathcal{L}$, then*

$$V_{\mathcal{L}'}(x+1) - V_{\mathcal{L}'}(x) - V_{\mathcal{L}}(x+1) + V_{\mathcal{L}}(x) > 0, \tag{4.8}$$

for all $x \in \mathcal{X}$.

Proof. This lemma is proven by induction. Let $V_{\mathcal{L}',0}(x) = \mathcal{L}'x$, and $V_{\mathcal{L},0}(x) = \mathcal{L}x$ for all x. Clearly, $V_{\mathcal{L}',0}(x+1) - V_{\mathcal{L}',0}(x) - V_{\mathcal{L},0}(x+1) + V_{\mathcal{L},0}(x) = \mathcal{L}' - \mathcal{L} > 0$. Assume it holds for $n = k$. Now, we prove it holds for $n = k+1$. Define $T^k_{a(\mathcal{L})}(x) = \mu(a)V_{\mathcal{L},k}([x-1]^+) + \big[\mu(A) - \mu(a)\big]V_{\mathcal{L},k}(x) + c_2(a)$. For all $x \geq 0$, we have

$$\begin{aligned}
&V_{\mathcal{L}',k+1}(x+1) - V_{\mathcal{L}',k+1}(x) - V_{\mathcal{L},k+1}(x+1) + V_{\mathcal{L},k+1}(x) \\
&= \lambda\big[V_{\mathcal{L}',k}(x+2) - V_{\mathcal{L}',k}(x+1) - V_{\mathcal{L},k}(x+2) + V_{\mathcal{L},k}(x+1)\big] \\
&\quad + \min_a T^k_{a(\mathcal{L}')}(x+1) - \min_a T^k_{a(\mathcal{L}')}(x) - \min_a T^k_{a(\mathcal{L})}(x+1) + \min_a T^k_{a(\mathcal{L})}(x) \\
&\quad + \mathcal{L}'c_3(x+1) - \mathcal{L}'c_3(x) - \mathcal{L}c_3(x+1) + \mathcal{L}c_3(x) \\
&> \min_a T^k_{a(\mathcal{L}')}(x+1) - \min_a T^k_{a(\mathcal{L}')}(x) - \min_a T^k_{a(\mathcal{L})}(x+1) + \min_a T^k_{a(\mathcal{L})}(x) \\
&\quad + \big[\mathcal{L}' - \mathcal{L}\big]\big[c_3(x+1) - c_3(x)\big] \\
&\geq \min_a T^k_{a(\mathcal{L}')}(x+1) - \min_a T^k_{a(\mathcal{L}')}(x) - \min_a T^k_{a(\mathcal{L})}(x+1) + \min_a T^k_{a(\mathcal{L})}(x).
\end{aligned}$$

The first inequality holds due to the induction hypothesis. The second inequality holds because $\mathcal{L}' > \mathcal{L}$ and $c_3(x)$ is an increasing function in x. Let $a^* \in \arg\min_a T^k_{a(\mathcal{L}')}(x+1)$ and $b^* \in \arg\min_a T^k_{a(\mathcal{L})}(x)$. Then we have

$$
\begin{aligned}
&V_{\mathcal{L}',k+1}(x+1) - V_{\mathcal{L}',k+1}(x) - V_{\mathcal{L},k+1}(x+1) + V_{\mathcal{L},k+1}(x) \\
&> T^k_{a^*(\mathcal{L}')}(x+1) - \min_a T^k_{a(\mathcal{L}')}(x) - \min_a T^k_{a(\mathcal{L})}(x+1) + T^k_{b^*(\mathcal{L})}(x) \\
&\geq T^k_{a^*(\mathcal{L}')}(x+1) - T^k_{b^*(\mathcal{L}')}(x) - T^k_{a^*(\mathcal{L})}(x+1) + T^k_{b^*(\mathcal{L})}(x) \\
&= \mu(a^*)\big[V_{\mathcal{L}',k}(x) - V_{\mathcal{L},k}(x)\big] + \big[\mu(A) - \mu(a^*)\big]\big[V_{\mathcal{L}',k}(x+1) - V_{\mathcal{L},k}(x+1)\big] \\
&\quad - \mu(b^*)\big[V_{\mathcal{L}',k}([x-1]^+) - V_{\mathcal{L},k}([x-1]^+)\big] - \big[\mu(A) - \mu(b^*)\big]\big[V_{\mathcal{L}',k}(x) - V_{\mathcal{L},k}(x)\big] \\
&= \mu(a^*)\big[V_{\mathcal{L}',k}(x) - V_{\mathcal{L},k}(x) - V_{\mathcal{L}',k}(x+1) + V_{\mathcal{L},k}(x+1)\big] \\
&\quad + \mu(b^*)\big[V_{\mathcal{L}',k}(x) - V_{\mathcal{L},k}(x) - V_{\mathcal{L}',k}([x-1]^+) + V_{\mathcal{L},k}([x-1]^+)\big] \\
&\quad + \mu(A)\big[V_{\mathcal{L}',k}(x+1) - V_{\mathcal{L},k}(x+1) - V_{\mathcal{L}',k}(x) + V_{\mathcal{L},k}(x)\big] \\
&> \big[\mu(A) - \mu(a^*)\big]\big[V_{\mathcal{L}',k}(x+1) - V_{\mathcal{L},k}(x+1) - V_{\mathcal{L}',k}(x) + V_{\mathcal{L},k}(x)\big] \geq 0.
\end{aligned}
$$

Hence, using mathematical induction we have proved the lemma. Additionally, at the boundaries of the state space, one also needs that the value function is increasing in $\mathcal{L}$ as shown in Lemma 4.3.3. □

We are now ready to prove the monotonicity result of the relative value function, i.e., when the Lagrange multiplier increases, and thus places more emphasis on the holding costs, the number of allocated processors also increases. The following theorem formalizes this statement.

Theorem 4.3.3. *Let* $T_{a(\mathcal{L})}(x) = \mu(a)V_{\mathcal{L}}([x-1]^+) + \big[\mu(A) - \mu(a)\big]V_{\mathcal{L}}(x) + c_2(a)$. *If* $\mathcal{L}' > \mathcal{L}$, *then for all* $a' \in \arg\min_a T_{a(\mathcal{L}')}(x)$ *and* $a \in \arg\min_a T_{a(\mathcal{L})}(x)$ *we have* $a' \geq a$ *for all* $x \in \mathcal{X}$.

Proof. The proof is by contradiction. Let $x > 0$ and suppose that an arbitrary optimal allocation $a' \in \arg\min_a T_{a(\mathcal{L}')}(x) < a \in \arg\min_a T_{a(\mathcal{L})}(x)$. Then, since $T_{a(\mathcal{L}')}(x) \geq T_{a'(\mathcal{L}')}(x)$, we have

$$
\begin{aligned}
T_{a(\mathcal{L}')}(x) - T_{a'(\mathcal{L}')}(x) &= \big[\mu(a') - \mu(a)\big]\big[V_{\mathcal{L}'}(x) - V_{\mathcal{L}'}\big([x-1]^+\big)\big] \\
&\quad - \big[c_2(a') - c_2(a)\big] \\
&\geq 0.
\end{aligned}
$$

Together with relation (4.8) and using the fact that μ is an increasing function, we have

$$\begin{aligned} T_{a(\mathcal{L})}(x) - T_{a'(\mathcal{L})}(x) &= \big[\mu(a') - \mu(a)\big]\big[V_{\mathcal{L}}(x) - V_{\mathcal{L}}\big([x-1]^+\big)\big] - \big[c_2(a') - c_2(a)\big] \\ &> \big[\mu(a') - \mu(a)\big]\big[V_{\mathcal{L}'}(x) - V_{\mathcal{L}'}\big([x-1]^+\big)\big] - \big[c_2(a') - c_2(a)\big] \\ &\geq 0, \end{aligned}$$

which implies that $a \notin \arg\min_a T_{a(\mathcal{L})}(x)$, which is in contradiction with the assumption. The case $x = 0$ results in $T_{a(\mathcal{L})}(0) - T_{a'(\mathcal{L}')}(0)$ being independent of $\mathcal{L}$ and $\mathcal{L}'$. Since there are no customers to serve in state $x = 0$, $\arg\min_a T_{a(\mathcal{L}')}(x) = \arg\min_a T_{a(\mathcal{L})}(x) = \{0\}$. □

There is a difference in the optimal policy between the one in the unconstrained case and in the constrained case. Note that due to Theorem 4.3.3 a higher value of the Lagrange multiplier $\mathcal{L}$ implies that the mean sojourn time $\mathbb{E}W$ decreases because more processors are allocated. Therefore, there is a $\mathcal{L}^*$ for which $\mathbb{E}W_{\mathcal{L}^*} \geq \alpha'$ and $\mathbb{E}W_{\mathcal{L}^*+\varepsilon} < \alpha'$ for a small $\varepsilon > 0$, where $\mathbb{E}W_{\mathcal{L}}$ is the mean sojourn time under Lagrange parameter $\mathcal{L}$. Observe that $\mathbb{E}W_{\mathcal{L}}$ is not a continuous function of $\mathcal{L}$, because the optimal policy does not change continuously with $\mathcal{L}$, but changes at specific values. In the constrained case, the optimal policy is to randomize between the associated policies $\pi(\mathcal{L}^*)$ and $\pi(\mathcal{L}^* + \varepsilon)$ so that exactly $\mathbb{E}W = \alpha'$ is achieved. The optimal policy thus randomizes between two step-function policies (see [10] for the proof).

Note that we analyzed the optimal actions for station 1 independently of the state at station 2. However, in our formulation we can see that the first station has influence on the policy in station 2. The higher the time β is spent at station 1, the tighter the constraint in station 2 becomes through $\alpha' = \alpha - \beta$. In the next section we shall illustrate how the optimal policy for the constrained problem is derived and how this tighter constraint affects the optimal policy.

4.4 Numerical experiments

In the previous sections we dealt with station 1 and station 2 separately. Although the analysis has been done separately, the performance of the system of both stations, in reality, are dependent, as was shown in Section 4.3.2. In this section we will validate the performance of the decomposed model in which we split up the real system with the simulated performance of the real

system and perform a numerical comparison of the optimality. In addition, we will illustrate how the variability in the service duration R and the set-up time T affects the policy at station 2.

4.4.1 Accuracy of the approximation

In Section 4.2 we described the model as a system in which jobs get service in two steps. First, the job receives service in step 1 in which a controller has to decide on the reservation time. Then, the job, while still blocking the service facilities in step 1, receives service in step 2 in which a controller has to decide on the number of allocated computing resources. We have split up this system into two parts and added the sojourn times of each part to each other. In doing so we make an error in the performance as compared to the real system as described in the end of Section 4.2.

First, we study how big the error is in the performance when we split up the system as compared to the real system. To this end, we derive the policy used in both cases by using s^* from Lemma 4.3.1 and π^* from the decomposed model as discussed in Section 4.3.2. Then, let $w_{\text{real}} = C(s^*) + \mathbb{E}_{s*,\pi^*}[c_3(X_\infty)]$ denote the mean sojourn time of the real system (that we obtain by simulation), and $w_{\text{model}} = C(s^*) + \mathbb{E}_{\pi^*}[c_3(X_\infty)]$ denote the mean sojourn time that a customer spends in step 1 and 2 together as obtained from the decomposed model. Note that the simulations have been run sufficiently long to ensure that the results shown in the following tables are significant. Table 4.1 depicts the relative error

$$\Delta_w := \frac{w_{\text{real}} - w_{\text{model}}}{w_{\text{real}}} \times 100\%,$$

when we split up the system, for various values of λ, μ, δ, and γ (assuming exponential distributions for R, T) using the policy that achieves $w_{\text{model}} = \alpha$ under minimal costs obtained in the decomposed model, with the cost functions specified in Experiment 1 of Table 4.3 below. The results of Table 4.1 show that the approximation works particularly well for high values of (δ, γ), but tends to degrade for smaller values. This is due to the fact that the two-step approach neglects the potential waiting time for availability of the service in the first step upon arrival epochs. This effect becomes more apparent for smaller values of (δ, γ) and for a higher utilization of the system. Note that the higher values of (δ, γ) (i.e., the service times in step 1 are relatively short in comparison) reflect realistic values of system parameters that are

used in real applications (e.g., iris scan and fingerprint applications). This underlines the applicability of the decomposed model in practice.

Second, we assess the accuracy of the approximation for our cost function defined in (4.1). To this end, let $c_{\text{real}} = \mathbb{E}_{s*,\pi^*}[c_1(X_\infty) + c_2(A_\infty)]$ denote the mean cost of the real system (that we obtain by simulation), and $c_{\text{model}} = \mathbb{E}_{\pi^*}[c_1(X_\infty) + c_2(A_\infty)]$ denote the mean cost that a customer encounters in step 1 and 2 together as obtained from the decomposed model, where s^* and π^* are obtained similarly as in the comparison of w above. Table 4.2 depicts the relative error, defined as

$$\Delta_c := \frac{c_{\text{real}} - c_{\text{model}}}{c_{\text{real}}} \times 100\%,$$

under the same parameter scenarios and policies as used in Table 4.1, and with the cost functions defined as in Table 4.3. Table 4.2 shows similar results as those in Table 4.1, namely, that the approximation works well for high values of (δ, γ) and degrades for smaller values. The different signs in the table indicate that the cost curves for the real system and the model do not completely coincide, but are slightly shifted relative to each other. The insight obtained from Table 4.1 and 4.2 is that for our problem (4.1), both the cost and the sojourn time show similar behavior, in particular for high values of (δ, γ). This insight suggests that the optimal policy in the decomposed model is a good approximation to the optimal policy in the real system. In order to validate this, we compare the performance under the optimal policy in the real system with the performance in the decomposed model when this same policy would be applied here. The derivation of the optimal policy in the real system is computationally intractable for large values of A. Therefore, we construct a small-sized problem with $A = 5$ such that enumerating over *all relevant* values of s and *all* monotone increasing step functions for the processor allocation becomes tractable.

The results of this experiment are in agreement with the results of Tables 4.1 and 4.2. To explain in greater detail where differences in the performance occur, we discuss the extreme scenarios in more detail below. We focus on $\lambda = 0.5$, $\mu = 1.2$, and the two scenarios $(\delta, \gamma) = (8, 12)$ and $(\delta, \gamma) = (44, 64)$. In the case with $(\delta, \gamma) = (8, 12)$, the unconstrained optimization leads to a relative difference of 2.6% in cost (Δ_c) and 7.7% in mean sojourn time (Δ_w). When we add a constraint with $\alpha = 1.2$, the differences are 8.0% and 7.7% for the cost and mean sojourn time, respectively. In the case with $(\delta, \gamma) = (44, 64)$, the unconstrained optimization has relative differences 0.58% and 1.85% for the cost and sojourn time, respectively. For the

$\lambda = 0.5$	(δ, γ)									
μ	(8, 12)	(12, 20)	(16, 24)	(20, 32)	(24, 40)	(28, 48)	(32, 56)	(36, 64)	(40, 64)	(44, 64)
0.4	13.0%	8.5%	6.3%	4.9%	4.1%	3.5%	2.9%	2.5%	2.1%	2.1%
1.2	7.6%	4.8%	3.6%	2.7%	2.3%	1.9%	1.6%	1.4%	1.2%	1.1%
2.0	8.6%	5.7%	4.2%	3.2%	2.6%	2.3%	2.0%	1.7%	1.5%	1.4%
4.0	8.2%	5.5%	4.4%	3.5%	2.9%	2.6%	2.3%	2.0%	1.9%	1.7%
$\lambda = 0.7$	(δ, γ)									
μ	(8, 12)	(12, 20)	(16, 24)	(20, 32)	(24, 40)	(28, 48)	(32, 56)	(36, 64)	(40, 64)	(44, 64)
0.4	22.8%	15.1%	11.3%	8.8%	7.2%	6.3%	5.3%	4.9%	4.3%	4.1%
1.2	12.0%	7.5%	5.6%	4.3%	3.5%	2.9%	2.5%	2.2%	2.0%	2.0%
2.0	10.5%	6.5%	4.9%	3.7%	3.0%	2.5%	2.2%	1.9%	1.7%	1.6%
4.0	11.6%	7.8%	6.0%	4.8%	4.1%	3.6%	3.2%	2.8%	2.5%	2.4%
$\lambda = 1.0$	(δ, γ)									
μ	(8, 12)	(12, 20)	(16, 24)	(20, 32)	(24, 40)	(28, 48)	(32, 56)	(36, 64)	(40, 64)	(44, 64)
0.4	38.0%	25.9%	19.4%	15.5%	12.6%	10.9%	9.5%	8.5%	7.6%	7.1%
1.2	20.0%	12.7%	9.4%	7.5%	6.0%	5.0%	4.4%	3.8%	3.5%	3.2%
2.0	17.8%	11.4%	8.3%	6.5%	5.3%	4.5%	3.9%	3.4%	3.1%	2.8%
4.0	17.1%	11.3%	8.6%	6.8%	5.7%	4.7%	4.1%	3.6%	3.2%	2.9%

Table 4.1. The relative error Δ_w for different parameter values with $\mu(a) = \mu a$.

constrained problem with $\alpha = 1.06$, the relative differences are 0.02% and 0.93%. In conclusion, the approximate two-step approach has a reasonable performance, and works particularly well for realistic parameter values.

4.4.2 The effect of variability in R and T

In this section, we illustrate the effect of the variability in R and T on the time spent at station 1. By Lemma 4.3.1 we know that s^* only depends on these variables through their mean, but the time spent at the station also takes into consideration the second moment of R and T. Therefore, we take for R and T distributions that increase in the coefficient of variation c^2. More specifically, we take for R and T a deterministic ($c^2 = 0$), exponential ($c^2 = 1$), and hyper-exponential distribution (with two phases and balanced means [100]) with $c^2 = 2$, and 4, respectively, while keeping the mean constant. Figure 4.4 shows $\beta := \mathbb{E}\max\{R, s^* + T\}$, as a function of c_T^2, the squared coefficient of variation of T (with mean 2/3). The four curves correspond to the different values of c_R^2, the squared coefficient of variation of R (with fixed mean 1). We also did experiments with distributions for T with means 1 and 6/5, and they gave similar curves. The graph shows that the increase in the coefficient of variation for R and/or T results in a larger mean sojourn time β that grows in a non-linear concave manner.

Now that we know how the distributions affect the β and thus the constraint

$\lambda = 0.5$	(δ, γ)									
μ	(8, 12)	(12, 20)	(16, 24)	(20, 32)	(24, 40)	(28, 48)	(32, 56)	(36, 64)	(40, 64)	(44, 64)
0.4	10.24%	6.44%	4.36%	3.18%	2.81%	2.20%	1.69%	1.23%	1.16%	1.20%
1.2	3.21%	1.45%	0.79%	0.35%	0.23%	0.09%	0.01%	0.21%	0.20%	-0.03%
2.0	1.89%	0.17%	-0.18%	-0.23%	-0.51%	-0.26%	-0.38%	-0.36%	-0.41%	-0.20%
4.0	3.79%	1.24%	0.30%	0.12%	-0.23%	-0.21%	-0.27%	-0.33%	-0.24%	-0.39%
$\lambda = 0.7$	(δ, γ)									
μ	(8, 12)	(12, 20)	(16, 24)	(20, 32)	(24, 40)	(28, 48)	(32, 56)	(36, 64)	(40, 64)	(44, 64)
0.4	19.23%	11.98%	8.58%	6.78%	5.34%	4.48%	3.99%	3.61%	3.15%	2.75%
1.2	7.66%	3.77%	2.36%	1.70%	1.48%	0.80%	0.79%	0.69%	0.63%	0.63%
2.0	5.73%	2.25%	1.25%	0.58%	0.12%	0.29%	0.21%	0.25%	0.04%	0.06%
4.0	6.11%	2.00%	0.77%	0.26%	0.01%	-0.22%	-0.26%	-0.19%	-0.28%	-0.32%
$\lambda = 1.0$	(δ, γ)									
μ	(8, 12)	(12, 20)	(16, 24)	(20, 32)	(24, 40)	(28, 48)	(32, 56)	(36, 64)	(40, 64)	(44, 64)
0.4	35.41%	22.69%	16.37%	13.38%	10.36%	8.58%	7.57%	6.58%	6.00%	5.02%
1.2	17.26%	9.15%	6.08%	4.63%	3.66%	2.95%	2.56%	1.93%	1.71%	1.77%
2.0	13.07%	6.14%	3.64%	2.15%	1.71%	1.26%	1.12%	1.01%	0.79%	0.59%
4.0	11.81%	4.32%	2.01%	1.05%	0.22%	-0.04%	-0.15%	-0.33%	-0.43%	-0.25%

Table 4.2. The relative error Δ_c for different parameter values with $\mu(a) = \mu a$.

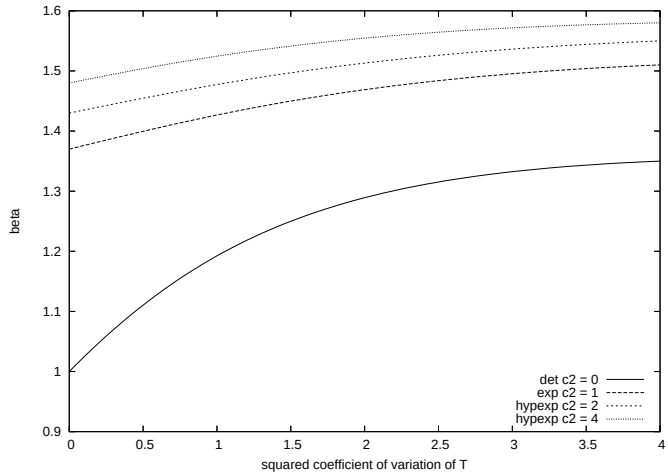
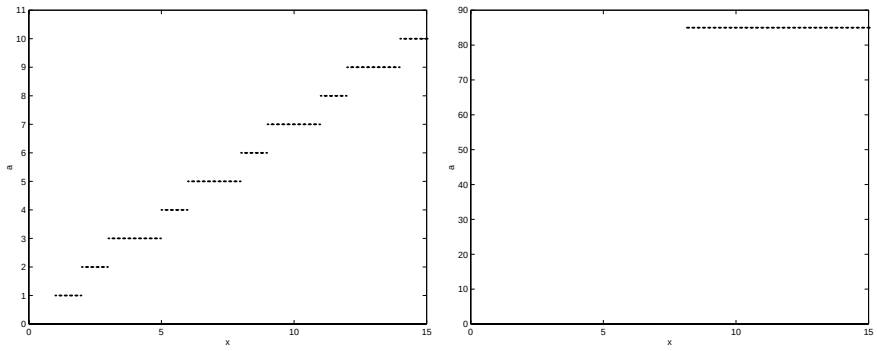


Figure 4.4. The time spent at station 1 for different distributions of R and T.

Experiment 1	$c_1(x) = 10x^2$	$c_2(a) = 10a^2$	$\mu(a) = \sqrt{a}\mu$
Experiment 2	$c_1(x) = \frac{1}{100}x^2$	$c_2(a) = 100\sqrt{a}$	$\mu(a) = a^2\mu$

Table 4.3. Functions for c_1, c_2, and μ.

level $\alpha' = \alpha - \beta$, we focus our attention to station 2 to gain further insight into the optimal resource-allocation policy in the decomposed problem. Consider

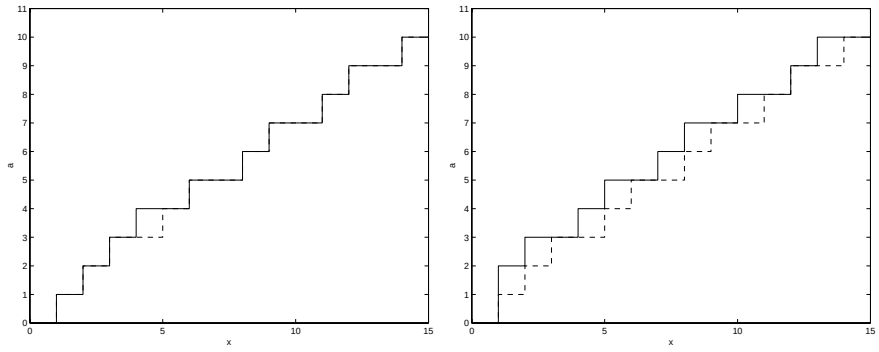


(a) Illustration of Theorem 4.3.1. (b) Illustration of Theorem 4.3.2.

Figure 4.5. Numerical experiments to illustrate the structure of the optimal policy.

the following parameters for this station: $\lambda = 0.5$, $\mu = 0.7$, and $A = 85$ (this represents the number of processors in the DAS-3 cluster that we have used, see [2]). To illustrate the structure of the policies, we vary the structure of the functions for c_1, c_2, and μ as given in Table 4.3. Experiments 1 and 2 satisfy the conditions of Theorem 4.3.1 and 4.3.2, respectively. The corresponding optimal resource policies in the decomposed problem, that are obtained through value iteration as described in Section 4.3.1, are illustrated in Figure 4.5(a) and (b). Figure 4.5(a) shows that the optimal resource strategy a^* is a non-decreasing function in x, while Figure 4.5(b) illustrates the bang-bang control policy.

Figure 4.5 illustrates the optimal policy in case of the unconstrained Markov decision problem, and the service level constraint is thus not taken into account. When the service level constraint is added, the policy changes to a more conservative strategy in which more processors are used to provide the guarantee on the sojourn time. To get insight into how the optimal policy changes, we vary $\mathcal{L}$ to study this effect for Experiment 1 in Table 4.3. For $\mathcal{L} = 0$ we get the unconstrained policy that is already depicted in Figure 4.5(a) with $g = 43.655$ and $\mathbb{E}W = 1.987$. As $\mathcal{L}$ increases, the policy changes at $\mathcal{L} = 0.94$ with corresponding values $g = 43.690$ and $\mathbb{E}W = 1.949$. Figure 4.6(a) shows how the policy differs from the previous policy; the dotted line is the policy for $\mathcal{L} = 0$ and the solid line depicts the policy for $\mathcal{L} = 0.94$. The policy differs in state 4 by using an additional server to serve the jobs. Hence,



(a) Results for $\mathcal{L} = 0.94$.

(b) Results for $\mathcal{L} = 5.00$.

Figure 4.6. Numerical experiments to illustrate the effect of changing values of $\mathcal{L}$.

to achieve $1.949 < \mathbb{E}W < 1.987$, one has to randomize between the two policies. As $\mathcal{L}$ increases, the policy becomes more conservative, in the sense that more computing resources are used in an earlier stage. Figure 4.6(b) depicts exactly this situation where $\mathcal{L} = 5.00$ is used.

To better understand the trade-off in the reduction in the average costs g and the improvement in the mean sojourn time, we depict both quantities as a function of $\mathcal{L}$ in Figure 4.7. The figure shows that in the initial increment of $\mathcal{L}$ the improvement in the mean sojourn time is the biggest, whereas the biggest increase in the average costs occurs for even bigger values of $\mathcal{L}$. Hence, one can balance both quantities by choosing an appropriate value of $\mathcal{L}$ such that the biggest improvements in the mean sojourn time are obtained while the increment in costs is relatively small. Moreover, the figure can also be used to determine the right value of $\mathcal{L}$ for a given α'. In combination with Figure 4.4 this provides a complete picture of how the variability in R and T affects the approximation with respect to the allocation of processors in station 2.

4.5 Conclusion and further research

In this chapter we have explored the optimal resource-allocation problem in time-reservation systems. In such systems one needs to optimize the

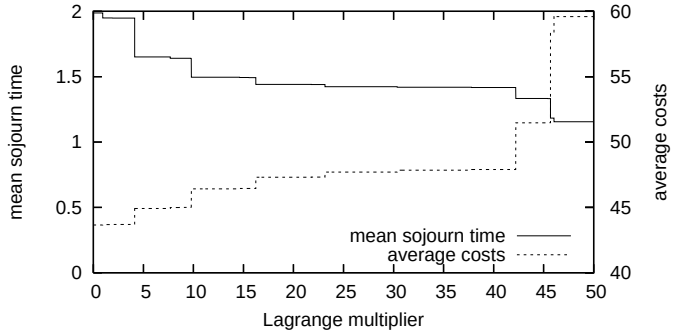


Figure 4.7. The average cost and the sojourn time as a function of the Lagrange multiplier.

resource-allocation costs and reservation moments simultaneously on one hand while satisfying a QoS-constraint on the sojourn time of a job on the other hand. We decomposed the problem into two parts and derive an optimal policy. This optimal policy serves as an approximation for the optimal policy in the original problem posed. For the decomposed problem, we first showed that the optimal reservation moment is given by the difference of the mean service time and the mean reservation time. However, the sojourn time at the first station does take into account the second moment as well. Then, we applied dynamic programming to show that the optimal resource-allocation policy in the decomposed problem follows a step function with as extreme policy the bang-bang control for given structures of the cost function and the service rate. Extensive numerical experiments showed that the optimal policy in the decomposed problem has nearly optimal performance as compared to the performance of the optimal policy in the real system.

Next, we mention several interesting avenues for further research. The work described in this chapter is part of a much larger realistic problem that exists in practice. In our current work, we focus on only one facility that uses a number of computing resources to provide a service under a QoS-constraint to its jobs, whereas in practice more than one facility may exist sharing the same computing resources. This creates dependence between the different facilities and the decision maker may not observe the state of all facilities. This warrants new stochastic control methodologies that deal with this dependence and partial information in which the insights obtained in this chapter can be very useful. Moreover, the decisions can be taken in a centralized or a decentralized manner.

In addition to having multiple facilities, the number of computing nodes may not be sufficient in all situations. In view of the limited processing capacity, new interesting problems from the server's point of view emergence. First, one could dynamically add computing nodes to the system. However, this may take a significant setup time which needs to be taken into account. The question arises "When does the system need to add/remove processing capacity? And how much?". Second, in practice system parameters may be time varying (for example the arrival rates, see also Chapter 7). This raises the need to develop algorithms that are robust against these variations. Third, the limited processing capacity also comes with a scheduling problem when many facilities share the same resource. Optimal scheduling in such situations is another challenging topic for follow-up research.

Chapter 5

Optimal Resource Allocation for Multi-queue Systems

This chapter studies optimal allocation of servers for a system with multiple service facilities and with a shared pool of servers. Each service facility poses a constraint on the maximum expected sojourn time of a job. A central decision maker can dynamically allocate servers to each facility, where adding more servers results in faster processing speeds, but against higher utilization costs. The objective is to dynamically allocate the servers over the different facilities such that the sojourn-time constraints are met at minimal costs. This situation occurs frequently in practice, e.g., in Grid systems for real-time image processing (iris scans, fingerprints). We model this problem as a Markov decision problem and derive structural properties of the relative value function. These properties, which are hard to derive for multi-dimensional systems, together with the properties derived in Chapter 6, give a full characterization of the optimal policy. We demonstrate the effectiveness of these policies by extensive numerical experiments.

5.1 Introduction

In recent years new real-time multimedia services have triggered a tremendous growth in data volumes and computational demand. Typical services include iris-scan and fingerprint systems that make high-resolution scans and require processing of the data to identify a person; these services operate in a real-time environment and run under very strict time constraints. To adhere to such constraints, these large-scale services typically use centralized computing clusters to execute on. In these service-based scenarios, a central

decision maker then allocates a number of processing resources to different service facilities to process the data. This gives rise to a class of models in which the central decision maker has to allocate the number of resources to ensure that all Quality of Service (QoS) constraints of the different facilities can be met.

In this decision making problem there is a trade-off between the processing time on the one hand and the utilization costs (lease costs, operating costs, etc.) on the other hand. Having too many resources at the server side leads to high costs and also to inefficiency, since only a part of the resources are needed to ensure that the QoS-constraint is satisfied. However, having too few resources leads to a long processing time so that the QoS-constraint of jobs can be violated. Hence, the objective is to find the allocation of the number of resources for the different facilities such that all QoS-constraints are met against minimal costs.

A few papers have been devoted to resource allocation problems closely related to our setting. Perry and Nilson [71] have studied a system in which two types of jobs are served by a single pool of resources. They associate priorities, based on an aging factor that grows proportionally with the waiting time, to these jobs and give an analytical model for computing the expected waiting. This heuristic, was first analyzed by Kleinrock [51, 52]. Borst and Seri [15] apply more complex heuristics in a multi-skilled queueing system with as performance metric the tail probabilities of the waiting time. They compare the number of jobs in each facility that actually has been served to the number that, nominally, should have been served under a long-run average allocation scheme. The "further behind" the actual number of services, the higher the resulting priority. Bhulai and Koole [14] and Gans and Zhou [36] study a variant with fully cross-trained servers in which only one queue has a QoS-constraint. They use Markov decision processes and Linear Programming to obtain (nearly) optimal control strategies. Stanford and Grassmann [93] simplify the problem by using fixed, static priority policies using matrix-geometric methods. Shumsky [86] divides the state space into regions, and uses an approximate analysis for the conditional system performance within each region.

In this chapter, we investigate and compare the optimal server allocation for the following three related models: (1) each service facility is viewed in isolation having its dedicated servers, (2) a system in which a chosen allocation cannot be changed during a service of a job, and (3) a fully flexible system in which it is allowed to change the allocation during the service of a job. The

main difference between the existing literature and our work is that we show by studying monotonicity properties of the dynamic programming relative value function that the optimal strategy has appealing structural properties; it is the multi-dimensional analog of a non-decreasing step function. This structure enables one to find optimal policies relatively easily as compared to solving the dynamic program which suffers from computational tractability. These methods will be illustrated in extensive numerical experiments.

The contribution of this work is two-fold. First, on the methodological side we provide an essential characterization of the optimal policy of a high-dimensional system, which is numerically intractable. This is quite exceptional, since there is no standard approach to derive monotonicity properties for multi-dimensional systems other than componentwise / directional monotonicity. This is the reason why the literature overview mainly deals with models having only two service facilities [71, 14, 36], or use heuristics [51, 52, 15, 93, 86] (sometimes even without constraints). Second, on the application side we have readily available policies that are easy to implement in systems that are highly relevant in practice. The comparison of the different models provides fundamental insights into the operational flexibility that is needed in the design of these systems.

The chapter is organized as follows. In Section 5.2 we formulate the models for the different cases. Next, we derive the structure of the optimal policy in Section 5.3. In Section 5.4 we illustrate these results by numerical experiments. Finally, in Section 5.5 we conclude with addressing a number of challenging topics for further research.

5.2 Model formulation

Consider N parallel service facilities at which jobs arrive according to a Poisson process with rate λ_i for facility i, $i = 1, \ldots, N$. There is a common pool of $A \geq 1$ resources to serve the jobs in the system. When upon arrival of a job at facility i there are no other jobs present, the arriving job is taken into service. However, if there are other jobs present, then the arriving job joins an infinite-capacity queue at facility i and awaits its service in an FCFS manner. When facility i has been allocated a_i resources, the job that is in service has a service duration that is exponentially distributed with parameter $\mu(a_i)$, where $\mu(.)$ is an increasing function. In the ideal case one would have $\mu(a_i) = \mu a_i$ for some fixed service rate μ. However, in practice,

there is communication overhead between multiple resources, and therefore the function μ is typically sublinear. In some cases, resources can cache results so that its effect is that the function μ is superlinear. After a job has received its service, it leaves the system.

Each facility provides a QoS-guarantee on the mean delay to the jobs served at that facility. Although, in practice, the QoS-constraints are usually expressed in terms of tail probabilities, we choose to express the constraints in terms of the mean sojourn time. This choice keeps the already complex model tractable for analysis and serves as a first step towards the analysis with tail probabilities as QoS-constraints. For this purpose, let S_i denote the steady state sojourn time of an arbitrary job at facility i. Then facility i is constrained by $\mathbb{E}S_i \leq \alpha_i$ for a preset value of α_i. There is a central decision maker that needs to allocate the resources to the different facilities such that the QoS-constraints are met. This gives rise to a problem in which the optimal allocation strategy needs to be determined. However, when facility i uses a_i resources a cost of $c_i(a_i)$ is incurred by the system with c_i an increasing function in a_i. Therefore, we are simultaneously interested in meeting the N constraints against lowest average costs. The optimal solution provides the value of A for which the optimal allocation strategy meets all the constraints, but fails to meet them when the optimal allocation strategy under $A-1$ resources is used.

We study the optimal number of resources A^* from three different viewpoints. First, we consider the case in which all service facilities operate independently of each other. In this case, the resources are not shared among the different facilities but are dedicated to each facility. Second, we study the case in which the resource pool is shared among different facilities. However, we make the assumption that the resource allocation cannot be changed when a job is served; only upon the start of the service of the next job the resource allocation can be changed. This is typically the case in systems where resources need to be reserved in advance. The third case deals with the fully flexible case in which the system can take full advantage of the economies of scale by allowing the resource allocation to change even during the service of a job. Since we can directly observe that going from case 1 to case 3 increases the flexibility, we can expect that $A_1^* \geq A_2^* \geq A_3^*$, with A_i^* the optimal number of resources needed in case i for $i = 1, 2, 3$. However, it is of interest to determine how big the gap between the three cases is and to study how the policy changes from case to case.

5.2.1 Service facilities with dedicated resources

In this section we assume that the service facilities do not share the resources among each other and thus have their own dedicated resources. This makes service facility i independent of the other facilities and turns the facility into a regular $M/M/1$ queue with arrival rate λ_i and service rate $\mu(A_i)$ when A_i servers have been allocated. In that case it is well-known that the expected sojourn time is given by $1/\big(\mu(A_i) - \lambda_i\big)$. Hence,

$$A_1^* = \sum_{i=1}^{N} A_i^* = \sum_{i=1}^{N} \Big\lceil \mu^{-1}\big(\lambda_i + 1/\alpha_i\big) \Big\rceil,$$

with $\lceil x \rceil$ the smallest integer greater than or equal to x.

5.2.2 Service facilities with limited resource sharing

In this subsection we focus on the case in which service facilities are allowed to share resources among each other. However, resources become free to be reassigned only at service completion instants. Hence, we make the assumption that the resource allocation for a service facility can only be changed upon the start of the service of a new job. Hence adding or removing resources during a service is not allowed. To study this case, we cast the problem as a Markov decision problem. Define the state space

$$\mathcal{X} = \{(x_1, \ldots, x_N, a_1, \ldots, a_N) \in \mathbb{N}_0^N \times \mathbb{N}_0^N \mid \sum_{i=1}^{N} a_i \leq A\},$$

where $(x, a) \in \mathcal{X}$ denotes that there are x_i customers at facility i with a_i resources allocated to it for $i = 1, \ldots, N$, where $a_i > 0$ also means that a service is ongoing and $a_i = 0$ means that no job is in service at facility i. When the system is in state $(x, a) \in \mathcal{X}$ the decision maker can choose actions from the action space

$$\mathcal{A}_{(x,a)} = \{(b_1, \ldots, b_N) \in \mathbb{N}_0^N \mid \sum_{i=1}^{N} (a_i + b_i) \leq A \text{ and } a_i b_i = 0 \text{ for } i = 1, \ldots, N\},$$

where action $b \in \mathcal{A}_{(x,a)}$ denotes the number of resources that one can allocate. Here, the restriction $a_i b_i = 0$ models the fact that when a service is ongoing (i.e., $a_i > 0$), the service allocation cannot be changed (i.e., $b_i = 0$).

However, after a service completion at facility i, we have that $a_i = 0$ and hence an allocation $b_i > 0$ is allowed. The transition rates when the system is in state $(x, a) \in \mathcal{X}$ and action $b \in \mathcal{A}_{(x,a)}$ is chosen are given by

$$p\big((x,a),b,(x',b')\big) = \begin{cases} \lambda_i, & x' = x + e_i, b' = a + b, \\ \mu(a_i + b_i), & x' = [x - e_i]^+, b' = a + b - a_i e_i - b_i e_i, \\ 0, & \text{otherwise}, \end{cases}$$

for $i = 1, \ldots, N$, with e_i the zero vector with a one at the i-th entry, and $[x]^+$ the componentwise maximum (i.e., $(\max\{x_1, 0\}, \ldots, \max\{x_N, 0\})$). The first line in the expression above models arrivals, the second line models service completions, and the third line prohibits any other state transitions. Note that when a service completion takes place, the resource allocation for that facility is set to zero. Finally, when the system is in state $(x, a) \in \mathcal{X}$ and action $b \in \mathcal{A}_{(x,a)}$ has been chosen, the direct costs are

$$c\big((x,a),b\big) = \sum_{i=1}^{N} c_i(a_i + b_i).$$

The quadruple $(\mathcal{X}, \mathcal{A}, p, c)$ completely describes the Markov decision process. Define a decision rule $\pi_{(x,a)}$ as a probability distribution on $\mathcal{A}_{(x,a)}$, i.e., when the system is in state $(x, a) \in \mathcal{X}$, the decision maker chooses action $b \in \mathcal{A}_{(x,a)}$ with probability $\pi_{(x,a)}(b)$. Let the policy π denote the collection of decision rules for all states. Let $u_t^\pi(x, a)$ denote the total expected costs up to time t when the system starts in state (x, a) under policy π. Note that for any stable and work-conserving policy, the Markov chain satisfies the unichain condition, so that the average expected costs $g(\pi) = \lim_{t\to\infty} u_t^\pi(x, a)/t$ is independent of the initial state (x, a) (see Proposition 8.2.1 of Puterman [73]). The goal is to find a policy π^* that minimizes the long-term average costs per time unit under the constraints, thus

$$\min_\pi g(\pi) \quad \text{subject to} \quad \mathbb{E}S_i \le \alpha_i \text{ for } i = 1, \ldots, N.$$

Note that due to Little's Law the number of jobs L_i in facility i can be related to the sojourn time S_i in facility i by $\mathbb{E}L_i = \lambda_i \mathbb{E}S_i$. Using this formula, the constrained Markov decision problem can be rewritten as an unconstrained Markov decision problem using Lagrange multipliers (see Section 12.6 of Altman [10]). To this end, we uniformize the system (see Section 11.5 of Puterman [73]). Therefore, assume that the uniformization constant $\sum_{i=1}^{N} \lambda_i + N\mu(A) = 1$; we can always get this by scaling. Uniformizing

is equivalent to adding dummy transitions (from a state to itself) such that the rate out of each state is equal to 1; then we can consider the rates to be transition probabilities. Now, let $V(x,a)$ be a real-valued function defined on the state space. This function will play the role of the relative value function, i.e., the asymptotic difference in total costs that results from starting the process in state (x,a) instead of some reference state. The long-term average optimal actions are a solution of the optimality equation (in vector notation) $g+V = TV$, where T is the dynamic programming operator acting on V defined as follows:

$$\begin{aligned} TV(x,a) = & \sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} c_i(a_i) + \sum_{i=1}^{N} \lambda_i H(x+e_i,a) \\ & + \sum_{i=1}^{N} \mu(a_i) H([x-e_i]^+, a-a_i e_i) \\ & + \Big(1 - \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \mu(a_i)\Big) V(x,a) \\ = & \sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} c_i(a_i) + \sum_{i=1}^{N} \lambda_i H(x+e_i,a) \\ & + \sum_{i=1}^{N} \mu(a_i) H([x-e_i]^+, a-a_i e_i) \\ & + \Big(N\mu(A) - \sum_{i=1}^{N} \mu(a_i)\Big) V(x,a), \end{aligned} \tag{5.1}$$

where τ_i are Lagrange multipliers, and the function H is given by

$$H(x,a) = \min_{b \in \mathcal{A}_{(x,a)}} \{V(x,a+b)\}.$$

The first term in the dynamic programming operator corresponds to the QoS-constraints of the several facilities. The second term represents the cost of using a resources. The third term is involved with the decision making upon arrival of a job. The fourth term deals with the decision making when a job has finished its service. The final term is the dummy term due to uniformization. Note that the decision making is modeled uniformly through the function H.

Note that when facility i has no holding costs x_i/λ_i, then no resources will be allocated to facility i, since it does not incur any costs from the buildup

in jobs. Therefore, when $\tau = e_i$, i.e., $\tau_i = 1$ and $\tau_j = 0$ for $j \neq i$, the optimal strategy will not allocate any resources to service facility $j \neq i$. Hence, one can find a value z_i such that the QoS-constraint for facility i with $\tau = z_i e_i$ is met under the assumption that there are infinitely many resources. By repeating this procedure for all facilities, one finds a box $\prod_{i=1}^{n}[0, z_i]$ in which the value of τ should lie under the optimal allocation that satisfies all constraints. Now, we can divide this box into a grid $\mathcal{G}$ which serve as our search space for τ. Then the following approach will find A_2^*.

1. Set $A := A_1^*$.
2. Solve the Markov decision process for all values of $\tau \in \mathcal{G}$ until all QoS-constraints are met or all grid points have been searched.
3. If for the value of τ all constraints are met, set $A := A - 1$ and return to step 2.
4. Return $A_2^* := A + 1$.

Note that in Step 2 of the algorithm, one needs to solve an infinite-dimensional Markov decision problem. In our numerical experiments, we truncate the state space such that we get a finite-dimensional problem that is numerically tractable. In doing so, the truncation is done such that the difference in the outcomes do not differ significantly when the state space is somewhat enlarged by shifting the truncation boundary. We will illustrate this algorithm in Section 4.4.

The algorithm to find A_2^* relies on evaluating the Markov decision problem for all $\tau \in \mathcal{G}$. One might formulate an unconstrained Markov decision problem, in which costs are associated with the queue length, in order to circumvent these evaluations. However, this would lead to formulation (6.1) with $\tau_i = \lambda_i$ for all i. Since the alternative unconstrained model is a special case of (6.1), the structural results that are obtained in the next section for the constrained Markov decision problem also hold for the unconstrained problem.

5.2.3 Service facilities with full flexibility in resource sharing

In this subsection we study the case in which service facilities have full flexibility in the resource-allocation policies. Thus, the resource facilities

can change the resource allocation during a service of a job, and do not have to wait for the job to finish. Since our system has Poisson arrivals and exponential service times, such a situation need only occur at moments an event occurs. Therefore, the only difference with the previous case is that this system allows to change the allocation at arrival instants.

In this case, the state space is given by $\mathcal{X} = \mathbb{N}_0^N$, where $x \in \mathcal{X}$ denotes that there are x_i customers at facility i for $i = 1, \ldots, N$. The action space is given by $\mathcal{A}_x = \{a \in \mathbb{N}_0^N \mid \sum_{i=1}^N a_i \leq A\}$, where action $a \in \mathcal{A}_x$ denotes the number of resources that one can allocate in state $x \in \mathcal{X}$. The transition rates when the system is in state $x \in \mathcal{X}$ and action $a \in \mathcal{A}_x$ is chosen are given by

$$p(x, a, x') = \begin{cases} \lambda_i, & x' = x + e_i \text{ for } i = 1, \ldots, N, \\ \mu(a_i), & x' = [x - e_i]^+ \text{ for } i = 1, \ldots, N, \\ 0, & \text{otherwise.} \end{cases}$$

Finally, when the system is in state $x \in \mathcal{X}$ and action $a \in \mathcal{A}_x$ has been chosen, the direct costs are $c(x, a) = \sum_{i=1}^N c_i(a_i)$. The quadruple $(\mathcal{X}, \mathcal{A}, p, c)$ completely describes the Markov decision process for this problem.

Let $V(x)$ denote the relative value function in this case. Then, the dynamic programming operator acting on V is defined as follows:

$$\begin{aligned} & TV(x) \\ &= \sum_{i=1}^N \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^N \lambda_i V(x + e_i) \min_{a \in \mathcal{A}_x} \Bigg[\sum_{i=1}^N \mu(a_i) V([x - e_i]^+) \\ &\quad + \Big(1 - \sum_{i=1}^N \lambda_i - \sum_{i=1}^N \mu(a_i)\Big) V(x) + \sum_{i=1}^N c_i(a_i) \Bigg] \\ &= \sum_{i=1}^N \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^N \lambda_i V(x + e_i) \min_{a \in \mathcal{A}_x} \Bigg[\sum_{i=1}^N \mu(a_i) V([x - e_i]^+) \\ &\quad + \Big(N\mu(A) - \sum_{i=1}^N \mu(a_i)\Big) V(x) + \sum_{i=1}^N c_i(a_i) \Bigg]. \end{aligned} \tag{5.2}$$

Note that the algorithm described in the previous subsection also applies to this case to obtain A_3^*. In fact, in step 1 one can choose $A := A_2^*$ for faster convergence.

5.3 Structural properties of the optimal policy

In the previous section, we described the three models and a solution technique to obtain the optimal policy. However, the optimal policy also possesses structural properties that can speed up the solution technique. Instead of searching a full grid for the optimal solution, the structural properties can reduce the search space considerably. Therefore, in this section, we focus on the structural properties of the described systems.

5.3.1 Allocation strategy for service facilities with dedicated resources

In case the service facilities have their own dedicated servers, each facility i should be equipped with A_i^* servers (as defined in Section 5.2.1) to meet the QoS-constraint. Since the servers are not shared, the optimal resource-allocation strategy is quite simple. When there are $x > 0$ customers at the facility, then A_i^* servers are allocated, and when $x = 0$ then no servers are allocated. This policy is also known as a bang-bang control policy (i.e., everything or nothing).

5.3.2 Allocation strategy for service facilities with limited resource sharing

The structure of the optimal policy for a service facility with limited resource sharing is more intricate than the case with dedicated resources. In order to study the structure, in principle, one needs to solve the optimality equation $g + V = TV$ with TV given by Equation (5.1). As stated in Chapter 2, the optimality equation is hard to solve analytically in practice. Therefore, we consider the backward recursion equation that is given by

$$\begin{aligned} V_{n+1}(x,a) = &\sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} c_i(a_i) + \sum_{i=1}^{N} \lambda_i H_n(x + e_i, a) \\ &+ \sum_{i=1}^{N} \mu(a_i) H_n([x - e_i]^+, a - a_i e_i) \\ &+ \Big(N\mu(A) - \sum_{i=1}^{N} \mu(a_i) \Big) V_n(x,a), \end{aligned} \tag{5.3}$$

where the function H_n is given by

$$H_n(x,a) = \min_{b \in \mathcal{A}_{(x,a)}} \{V_n(x, a+b)\}.$$

For ease of notation in the proofs in the sequel, we also define $\arg H_n(x,a)$ by

$$\arg H_n(x,a) = \arg\min_{b \in \mathcal{A}_{(x,a)}} \{V_n(x, a+b)\}.$$

The backward recursion equation allows us to prove structural properties of the relative value function V through induction on n in V_n. It is clear that the objective of the system is to strive for the fewest number of customers in the system as more customers mean higher waiting costs. Therefore, it is intuitive that V is increasing in each component of x, i.e., adding customers to facility i results in higher costs for the system. The following lemma makes this statement more precise.

Lemma 5.3.1 (Increasingness)**.** *The relative value function V is strictly increasing in the number of customers, i.e.,*

$$V(x + e_j, a) - V(x,a) > 0,$$

for all $x \in \mathcal{X}$ for some a with $\sum_{i=1}^N a_i \leq A$, and for $j = 1, \ldots, N$.

Proof. The proof is by induction in n in V_n. Define $V_0(x,a) = \sum_{i=1}^N x_i$ for all actions a. Then, clearly, $V_0(x,a)$ is strictly increasing in all components of x. Now, assume that $V_n(x+e_j,a) - V_n(x,a) > 0$ for some $n \in \mathbb{N}$, and for $j = 1, \ldots, N$. Now, we prove that $V_{n+1}(x,a)$ satisfies the increasingness property as well. Therefore, fix $j \in \{1, \ldots, N\}$, then

$$\begin{aligned}
&V_{n+1}(x+e_j,a) - V_{n+1}(x,a) \\
&= \frac{\tau_j}{\lambda_j} + \sum_{i=1}^N \lambda_i \Big[H_n(x + e_j + e_i, a) - H_n(x+e_i,a) \Big] \\
&\quad + \sum_{i=1}^N \mu(a_i) \Big[H_n([x+e_j-e_i]^+, a - a_i e_i) - H_n([x-e_i]^+, a - a_i e_i) \Big] \\
&\quad + \Big[N\mu(A) - \sum_{i=1}^N \mu(a_i) \Big] \Big[V_n(x+e_j,a) - V_n(x,a) \Big].
\end{aligned} \tag{5.4}$$

Note that the first term (i.e., τ_j/λ_j) and the last term with $V_n(x+e_j,a) - V_n(x,a)$ are positive. Hence, based on the induction hypothesis, we have

$$\begin{aligned} & V_{n+1}(x+e_j,a) - V_{n+1}(x,a) \\ & > \sum_{i=1}^{N} \lambda_i \Big[H_n(x+e_j+e_i,a) - H_n(x+e_i,a) \Big] \\ & \quad + \sum_{i=1}^{N} \mu(a_i) \Big[H_n([x+e_j-e_i]^+, a-a_ie_i) - H_n([x-e_i]^+, a-a_ie_i) \Big]. \end{aligned} \tag{5.5}$$

Let $b \in \arg H_n(x+e_j+e_i,a)$ and $c \in \arg H_n([x+e_j-e_i]^+, a-a_ie_i)$. Then,

$$\begin{aligned} & V_{n+1}(x+e_j,a) - V_{n+1}(x,a) \\ & > \sum_{i=1}^{N} \lambda_i \Big[V_n(x+e_j+e_i,a+b) - V_n(x+e_i,a+b) \Big] \\ & \quad + \sum_{i=1}^{N} \mu(a_i) \Big[V_n([x+e_j-e_i]^+, a-a_ie_i+c) \\ & \quad - V_n([x-e_i]^+, a-a_ie_i+c) \Big] \geq 0. \end{aligned} \tag{5.6}$$

Clearly, the inequality above holds because of the induction hypothesis. Hence, we conclude, by taking the limit as $n \to \infty$, that $V(x,a)$ is increasing in x_j for all $j = 1, \ldots, N$. □

Lemma 5.3.1 shows that the costs incurred by the system increases as the number of customers in the system increases. In fact, more can be said about the rate at which the costs increase; the increase in costs is higher when more customers are in the system. Hence, this implies that the relative value function is a convex function. In the sequel we will show that this is indeed true. We do this by studying the case with one-dimensional case (i.e., $N = 1$) first. Note that we will adjust the notation for $N = 1$ straightforwardly by omitting the indices of all variables. However, before doing so, we need two preparative lemma's.

Lemma 5.3.2. *The value function satisfies the following property:*

$$H(x+1,0) - H(x,0) - V(x,0) + V(x-1,0) < 0,$$

for all $x \geq 1$.

Proof. Let $x \geq 1$. Then

$$\begin{aligned} &V(x,0) - V(x-1,0) \\ &\quad = \frac{\tau}{\lambda} + \lambda\big[H(x+1,0) - H(x,0)\big] + \mu(A)\big[V(x,0) - V(x-1,0)\big]. \end{aligned}$$

Since $\lambda + \mu(A) = 1$, the equation above implies

$$\lambda\big[V(x,0) - V(x-1,0)\big] = \frac{\tau}{\lambda} + \lambda\big[H(x+1,0) - H(x,0)\big].$$

Therefore,

$$\lambda\big[H(x+1,0) - H(x,0) - V(x,0) + V(x-1,0)\big] = -\frac{\tau}{\lambda}.$$

Thus, $H(x+1,0) - H(x,0) - V(x,0) + V(x-1,0) < 0$, since $-\tau/\lambda < 0$. □

Lemma 5.3.2 is almost the inequality that represents convexity of the value function. This would be the case if H were to be replaced by V. However, for the proof of convexity, we need three additional properties to hold as well. The following lemma makes these properties explicit.

Lemma 5.3.3 (Convexity)**.** *For $N = 1$, the following properties hold:*

(i) $V(x+1,a) - 2V(x,a) + V(x-1,a) \geq 0$ *for all* $x \geq 1$ *and* $a \geq 0$,

(ii) $V(x,a) - V([x-1]^+,a) - H([x-1]^+,0) + H([x-2]^+,0) > 0$ *for all* $x \geq 0$ *and* $a > 0$,

(iii) The minimal element of $\arg H(x,0)$ is strictly positive for all $x \geq 2$,

(iv) $H(x+1,0) - 2H(x,0) + H(x-1,0) \geq 0$ *for all* $x \geq 1$.

Proof. The proof is by induction on n in V_n. Define $V_0(x,a) = 0$ for all states x and actions $a > 0$ and $V_0(x,0) = \varepsilon > 0$ for all x. Then, clearly, $V_0(x,a)$ satisfies all properties. Now suppose that the properties hold for some $n \in \mathbb{N}$. We prove that the properties also hold for $n+1$. Therefore, we start with convexity first.

Property (i). Let $x \geq 1$ and suppose that $a = 0$. Then,

$$\begin{aligned} &V_{n+1}(x+1,0) - 2V_{n+1}(x,0) + V_{n+1}(x-1,0) \\ &\quad = \lambda\big[H_n(x+2,0) - 2H_n(x+1,0) + H_n(x,0)\big] \\ &\qquad + \mu(A)\big[V_n(x+1,0) - 2V_n(x,0) + V_n(x-1,0)\big] \\ &\quad \geq \lambda\big[H_n(x+2,0) - 2H_n(x+1,0) + H_n(x,0)\big] \geq 0. \end{aligned} \tag{5.7}$$

The equality following by expanding V_{n+1} into V_n. The first inequality follows by using Property (i) of the induction hypothesis. The second inequality follows by using Property (iv) of the induction hypothesis.

Now let $x \geq 1$ and suppose that $a > 0$. Then

$$\begin{aligned} V_{n+1}(x+1,a) &- 2V_{n+1}(x,a) + V_{n+1}(x-1,a) \\ &= \lambda\big[V_n(x+2,a) - 2V_n(x+1,a) + V_n(x,a)\big] \\ &\quad + \mu(a)\big[H_n(x,0) - 2H_n([x-1]^+,0) + H_n([x-2]^+,0)\big] \\ &\quad + \big[\mu(A) - \mu(a)\big]\Big[V(x+1,a) - 2V(x,a) + V(x-1,a)\big] \\ &\geq \mu(a)\big[H_n(x,0) - 2H_n([x-1]^+,0) + H_n([x-2]^+,0)\big] \geq 0. \end{aligned} \tag{5.8}$$

The equality follows by expanding V_{n+1} into V_n. The first inequality follows by using Property (i) of the induction hypothesis. The second inequality follows by using Property (iv) of the induction hypothesis. Thus, for all $x \geq 1$ and $a \geq 0$, $V_{n+1}(x+1,a) - 2V_{n+1}(x,a) + V_{n+1}(x-1,a) \geq 0$.

Property (ii). Let $x \geq 0$ and suppose $a > 0$. Then, based on the optimality equation, we have

$$\begin{aligned} V_{n+1}(x,a) - V_{n+1}([x-1]^+,a) &= \frac{\tau}{\lambda} + \lambda\big[V_{n+1}(x+1,a) - V_{n+1}(x,a)\big] \\ &\quad + \mu(a)\big[H_{n+1}([x-1]^+,0) - H_{n+1}([x-2]^+,0)\big] \\ &\quad + \big[\mu(A) - \mu(a)\big]\big[V_{n+1}(x,a) - V_{n+1}([x-1]^+,a)\big]. \end{aligned}$$

Recall that the uniformization constant $\lambda + \mu(A) = 1$. Thus, the equation above is equivalent to

$$\begin{aligned} \lambda\big[V_{n+1}(x,a) - V_{n+1}([x-1]^+,a)\big] &= \frac{\tau}{\lambda} + \lambda\big[V_{n+1}(x+1,a) - V_{n+1}(x,a)\big] \\ &\quad + \mu(a)\big[H_{n+1}([x-1]^+,0) - H_{n+1}([x-2]^+,0)\big] \\ &\quad - \mu(a)\big[V_{n+1}(x,a) - V_{n+1}([x-1]^+,a)\big]. \end{aligned}$$

The equation above implies that

$$\begin{aligned} &\mu(a)\big[V_{n+1}(x,a) - V_{n+1}([x-1]^+,a) - H_{n+1}([x-1]^+,0) + H_{n+1}([x-2]^+,0)\big] \\ &\quad = \frac{\tau}{\lambda} + \lambda\big[V_{n+1}(x+1,a) - 2V_{n+1}(x,a) + V_{n+1}([x-1]^+,a)\big]. \end{aligned}$$

Hence, by using Property (i) of the induction hypothesis, the righthand side of the equation is positive. Hence,

$$V_{n+1}(x,a) - V_{n+1}([x-1]^+,a) - H_{n+1}([x-1]^+,0) + H_{n+1}([x-2]^+,0) > 0.$$

Property (iii). We prove the property by means of contradiction. Assume that there exists a $x \geq 2$ such that the minimal element of $\text{arg}H(x,0)$ is 0, i.e., $\min\{\text{arg}H(x,0)\} = 0$. This, by definition, implies that $V_{n+1}(x,0) = H_{n+1}(x,0)$. Therefore,

$$\begin{aligned} &H_{n+1}(x,0) - H_{n+1}(x-1,0) - V_{n+1}(x-1,0) + V_{n+1}(x-2,0) \\ &\quad \geq V_{n+1}(x,0) - 2V_{n+1}(x-1,0) + V_{n+1}(x-2,0) \geq 0. \end{aligned}$$

The first inequality follows by taking action $a = 0$ in the second term $H_{n+1}(x-1,0)$. The second inequality follows by Property (i) of the induction hypothesis. However, based on Lemma 5.3.2 we know that $H_{n+1}(x,0) - H_{n+1}(x-1,0) - V_{n+1}(x-1,0) + V_{n+1}(x-2,0) < 0$. Therefore, we conclude that $\min\{\text{arg}H(x,0)\} > 0$ for $x \geq 2$.

Property (iv). Let $x \geq 1$. Since $x - 1 \geq 0$, we have $x + 1 \geq 2$. Thus by using Property (iii) of the induction hypothesis, we have $a^*(x) := \min\{\text{arg}H(x,0)\} > 0$. Therefore,

$$\begin{aligned} &H_{n+1}(x+1,0) - 2H_{n+1}(x,0) + H_{n+1}(x-1,0) \\ &\quad \geq V_{n+1}(x+1, a^*(x+1)) - V_{n+1}(x, a^*(x+1)) - H_{n+1}(x,0) \\ &\qquad + H_{n+1}(x-1,0) \geq \ 0. \end{aligned}$$

The first inequality follows by taking action $a^*(x+1)$ in $H_{n+1}(x,0)$. The second inequality follows by Property (ii) of the induction hypothesis. We conclude the proof by taking the limit as $n \to \infty$. □

Lemma 5.3.3 shows that the relative value function is convex. However, in proving this one needed three additional properties simultaneously in the proof by induction (Property (i) depends on (iv), which depends on (ii) and (iii)). Now, we are ready to study monotonicity properties of the optimal policy. The convexity of the relative value function is crucial in this step. Due to the convexity, we have that the optimal policy is a step function. The following theorem formalizes this statement.

Theorem 5.3.1 (Monotonicity)**.** *For $N = 1$, if the service rate $\mu(a)$ and cost function $c(a)$ are strictly increasing functions in a, then for all $a \in \text{arg}H(x+k,0)$ and $b \in \text{arg}H(x,0)$, we have $a \geq b$ for all $k \geq 0$.*

Proof. For $x = 0$, the only feasible action in $\text{arg}H(0,0)$ is $a_0 = 0$, since there are no customers to serve. For $x = 1$, if $a_1 = 0 \in \text{arg}H(1,0)$, then all

$a_2 \in \arg H(2,0)$, we have $a_2 > a_1$ [property (iii) of Property 6.3.2]. Hence, $a_2 > a_1 \geq a_0$. Now, for $x \geq 1$ and $a_1 > 0$ it suffices to show that the relative value function satisfies an extension of submodularity, namely

$$[V(x, a+k) - V(x,a)] - [V(x+1, a+k) - V(x+1,a)] > 0, \tag{5.9}$$

for all $k > 0$. If this property holds, then since $V(x+1, a_{x+1}+k) - V(x+1, a_{x+1}) \geq 0$, we have that $V(x, a_{x+1}+k) - V(x, a_{x+1}) > V(x+1, a_{x+1}+k) - V(x+1, a_{x+1}) \geq 0$ with a_{x+1} the minimal element of $\arg H(x+1,0)$. Hence, this implies that all minimizing actions $a_x \in \arg H(x,0)$ in state x satisfy $a_x \leq a_{x+1}$.

We prove the submodularity property by induction on n in V_n. Let $V_0(x,a) = 0$. Clearly, the submodularity property holds. Now assume that the property holds for for some $n \in \mathbb{N}$ and for all $x \geq 0$. We proceed to prove that $V_{n+1}(x,a)$ satisfies the property as well. Therefore, fix $x \geq 1$ and $a > 0$, then

$$\begin{aligned}
&V_{n+1}(x, a+k) - V_{n+1}(x,a) - V_{n+1}(x+1, a+k) + V_{n+1}(x+1,a) \\
&= \lambda\big[V_n(x+1, a+k) - V_n(x+1,a) - V_n(x+2, a+k) + V_n(x+2,a)\big] \\
&\quad + \big[\mu(a+k) - \mu(a)\big]\big[H_n(x-1,0) - H_n(x,0)\big] \\
&\quad + \mu(A)\big[V_n(x, a+k) - V_n(x,a) - V_n(x+1, a+k) + V_n(x+1,a)\big] \\
&\quad - \mu(a+k)V_n(x, a+k) + \mu(a)V_n(x,a) \\
&\quad + \mu(a+k)V_n(x+1, a+k) - \mu(a)V_n(x+1,a) \\
&= \lambda\big[V_n(x+1, a+k) - V_n(x+1,a) - V_n(x+2, a+k) + V_n(x+2,a)\big] \\
&\quad + \big[\mu(a+k) - \mu(a)\big]\big[H_n(x-1,0) - H_n(x,0)\big] \\
&\quad + \mu(A)\big[V_n(x, a+k) - V_n(x,a) - V_n(x+1, a+k) + V_n(x+1,a)\big] \\
&\quad - \big[\mu(a+k) - \mu(a)\big]V_n(x, a+k) - \mu(a)V_n(x, a+k) + \mu(a)V_n(x,a) \\
&\quad + \big[\mu(a+k) - \mu(a)\big]V_n(x+1, a+k) \\
&\quad + \mu(a)V_n(x+1, a+k) - \mu(a)V_n(x+1,a) \\
&= \lambda\big[V_n(x+1, a+k) - V_n(x+1,a) - V_n(x+2, a+k) + V_n(x+2,a)\big] \\
&\quad + \big[\mu(A) - \mu(a)\big]\big[V_n(x, a+k) - V_n(x,a) \\
&\quad - V_n(x+1, a+k) + V_n(x+1,a)\big] \\
&\quad + \big[\mu(a+k) - \mu(a)\big]\big[V_n(x+1, a+k) - V_n(x, a+k) \\
&\quad - H_n(x,0) + H_n(x-1,0)\big].
\end{aligned} \tag{5.10}$$

The first equality follows from expanding V_{n+1} into V_n. The second equality follows from adding and subtracting $\mu(a)V_n(x, a+k)$ and $\mu(a)V_n(x+1, a+k)$. The third equality follow from standard algebraic manipulations. Based on the induction hypothesis, we have

$$\begin{aligned} &V_{n+1}(x, a+k) - V_{n+1}(x, a) - V_{n+1}(x+1, a+k) + V_{n+1}(x+1, a) \\ &\geq \big[\mu(a+k) - \mu(a)\big] \\ &\quad \times \big[V_n(x+1, a+k) - V_n(x, a+k) - H_n(x, 0) + H_n(x-1, 0)\big]. \end{aligned} \tag{5.11}$$

By using Property (ii) of Lemma 5.3.3, we obtain

$$V_n(x+1, a+k) - V_n(x, a+k) - H_n(x, 0) + H_n(x-1, 0) > 0. \tag{5.12}$$

Thus, we have shown that $[V(x, a+k) - V(x, a)] - [V(x+1, a+k) - V(x+1, a)] > 0$. Consequently, this completes the proof as this property implies that a_x is an increasing function in x. □

Theorem 5.3.1 shows that in the one-dimensional case (i.e., $N = 1$), the optimal policy a_x is a step function in the variable x. The multi-dimensional case with arbitrary N has the same structure. The proof of this statement fundamentally boils down to the one-dimensional case. The following theorem shows the argument.

Theorem 5.3.2 (Step function)**.** *Consider an arbitrary number $N \in \mathbb{N}$ of queues, and let $a_i^*(x, a) \in [\arg H(x, a)]_i = [\arg\min_{b \in \mathcal{A}_{(x,a)}} \{V_n(x, a+b)\}]_i$. If the service rate $\mu(a)$ and cost function $c(a)$ are strictly increasing functions in a, then $a_i^*(x, a) \leq a_i^*(x + e_i, a)$.*

Proof. Fix $(x, a) \in \mathcal{X}$ and suppose that $(x + e_i, a) \in \mathcal{X}$ as well. If $a^*(x, a)$ is such that $\sum_{n=1}^N \big(a_n + a_n^*(x, a)\big) < A$, then there is spare capacity to assign. Hence, in state $(x + e_i, a)$ all queues except queue i need no more capacity, since the number of customers in their system did not increase. Hence, queue i can be viewed in isolation due to the spare capacity. Therefore, Theorem 5.3.1 applies and $a_i^*(x, a) \leq a_i^*(x + e_i, a)$. In case $\sum_{n=1}^N \big(a_n + a_n^*(x, a)\big) = A$, there is no spare capacity left. Now, there are two cases. Either the performance of queue i becomes so stringent that capacity is taken away from a different queue, or the capacity allocation does not change at all. In both cases we have $a_i^*(x, a) \leq a_i^*(x + e_i, a)$. □

5.3.3 Allocation strategy for service facilities with full flexibility in resource sharing

In this subsection we focus our attention to optimal allocation strategies for service facilities with fully flexible resource sharing capabilities. The difference with the previous section is that the allocation of the number of servers is allowed to change when a job is already in service. We shall adopt the same techniques in deriving the structure of the optimal policy as in the previous section. Therefore, we focus on the main theorems that characterize the structure of the policy, and move the lengthiest proof to Appendix A.

We start by rewriting Equation (5.2) for the fully flexible system as a set of backward recursion equations. This set of equations is given by

$$V_{n+1}(x) = \sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} \lambda_i V_n(x + e_i) \; + \min_{a \in \mathcal{A}_x} T_a^n(x), \tag{5.13}$$

where $T_a^n(x)$ is given by

$$T_a^n(x) = \sum_{i=1}^{N} \mu(a_i) V_n([x - e_i]^+) + \Big[N\mu(A) - \sum_{i=1}^{N} \mu(a_i) \Big] V_n(x) + \sum_{i=1}^{N} c_i(a_i).$$

Rewriting the optimality equations in this way has its advantage in showing structural properties of the relative value function V. We start by showing that the relative value function is increasing in all components of the state. The following lemma makes this statement more precise.

Lemma 5.3.4 (Increasingness)**.** *The relative value function $V(x)$ is increasing in all components of the state x, i.e., $V(x + e_j) - V(x) > 0$ for $j = 1, \ldots, N$.*

Proof. The proof is by induction on n in $V_n(x)$. Let $V_0(x) = \sum_{i=1}^{N} x_i$ for all states $x \in \mathcal{X}$. Then, clearly, $V_0(x)$ satisfies the increasingness property. Now, assume that $V_n(x)$ is an increasing function in x. We proceed to prove

that $V_{n+1}(x)$ is also increasing in x. First, we have

$$\begin{aligned} V_{n+1}(x+e_j) - V_{n+1}(x) &= \frac{\tau_j}{\lambda_j} + \sum_{i=1}^{N} \lambda_i \Big[V_n(x+e_i+e_j) - V_n(x+e_i) \Big] \\ &\quad + \min_{a \in \mathcal{A}_x} \Big\{ T_a^n(x+e_j) \Big\} - \min_{a \in \mathcal{A}_x} \Big\{ T_a^n(x) \Big\} \\ &> \min_{a \in \mathcal{A}_x} \Big\{ T_a^n(x+e_j) \Big\} - \min_{a \in \mathcal{A}_x} \Big\{ T_a^n(x) \Big\}. \end{aligned}$$

The inequality holds because the first term is positive, and the second expression between the brackets is also positive due to the induction hypothesis. Let $a^* \in \arg\min_{a \in \mathcal{A}_x} \{T_a^n(x+e_j)\}$. Then we have

$$\begin{aligned} V_{n+1}(x+e_j) - V_{n+1}(x) &> T_{a^*}^n(x+e_j) - \min_{a \in \mathcal{A}_x} \Big\{ T_a^n(x) \Big\} \\ &\geq T_{a^*}^n(x+e_j) - T_{a^*}^n(x) \\ &= \sum_{i=1}^{N} \mu(a_i^*) \Big[V_n(x-e_i+e_j) - V_n(x-e_i) \Big] \\ &\quad + \Big[N\mu(A) - \sum_{i=1}^{N} \mu(a_i^*) \Big] \Big[V_n(x+e_i) - V_n(x) \Big] \geq 0. \end{aligned}$$

Therefore, by induction, we have shown that $V_{n+1}(x+e_j) - V_{n+1}(x) > 0$. Hence, by taking the limit as n to infinity, we get that $V(x)$ is increasing in x_j for $j = 1, \ldots, N$. □

We are now ready to pose our main theorem for the model with full flexibility. The theorem characterizes the structure of the optimal policy to be a non-decreasing function in the components of the state, i.e., if the number of customers in queue i increases, then the allocation of the number of servers to queue i is non-decreasing. The following theorem provides a rigorous proof to this statement.

Theorem 5.3.3. *Consider an arbitrary number $N \in \mathbb{N}$ of queues, and let*

$$\begin{aligned} a_i^*(x) &\in \Big[\arg\min_{a \in \mathcal{A}_x} \Big\{ T_a(x) \Big\} \Big]_i \\ &= \Big[\arg\min_{a \in \mathcal{A}_x} \Big\{ \sum_{i=1}^{N} \mu(a_i) V([x-e_i]^+) \\ &\quad + \Big[N\mu(A) - \sum_{i=1}^{N} \mu(a_i) \Big] V(x) + \sum_{i=1}^{N} c_i(a_i) \Big\} \Big]_i. \end{aligned}$$

If the service rate $\mu(a)$ and cost function $c(a)$ are increasing functions in a, then $a_i^*(x+e_i) \geq a_i^*(x)$ and $a_j^*(x+e_i) \leq a_j^*(x)$ for $j \neq i$.

Proof. Fix $x \in \mathcal{X}$ and suppose that $x + e_i \in \mathcal{X}$ as well. If $a^*(x)$ is such that $\sum_{i=1}^N a_i^*(x) < A$, then there is sufficient spare capacity to assign. Hence, in state $x + e_i$ all queues except queue i need no more capacity, since the number of customers in their system did not increase. Hence, queue i can be viewed in isolation due to the spare capacity. The queue in isolation satisfies that conditions of Theorem 1 of [112]. From this theorem it follows that $a_i^*(x+e_i) \geq a_i^*(x)$ and $a_j^*(x+e_i) = a_j^*(x)$ for $j \neq i$.

Now suppose that $\sum_{i=1}^N a_i^*(x) = A$, i.e., all servers have been allocated. Denote $a = a^*(x+e_i)$ and $b = a^*(x)$. Note that $T_a(x) - T_b(x) \geq 0$ and $T_a(x+e_i) - T_b(x+e_i) \leq 0$. Therefore, $Z = T_a(x) - T_b(x) - T_a(x+e_i) + T_b(x+e_i) \geq 0$. Since

$$\begin{aligned} T_a(x) - T_b(x) = & \Big[\sum_{j=1}^N c_j(a_j) - \sum_{j=1}^N c_j(b_j)\Big] \\ & + \Big[\mu(b_i) - \mu(a_i)\Big]\Big[V(x) - V(x-e_i)\Big] \\ & + \sum_{j\neq i}^N \Big[\mu(b_j) - \mu(a_j)\Big]\Big[V(x) - V(x-e_j)\Big], \end{aligned} \tag{5.14}$$

and

$$\begin{aligned} T_a(x+e_i) - T_b(x+e_i) = & \Big[\sum_{j=1}^N c_j(a_j) - \sum_{j=1}^N c_j(b_j)\Big] \\ & + \Big[\mu(b_i) - \mu(a_i)\Big]\Big[V(x+e_i) - V(x)\Big] \\ & + \sum_{j\neq i}^N \Big[\mu(b_j) - \mu(a_j)\Big]\Big[V(x+e_i) - V(x+e_i-e_j)\Big], \end{aligned} \tag{5.15}$$

we have

$$\begin{aligned} Z &= T_a(x) - T_b(x) - T_a(x+e_i) + T_b(x+e_i) \\ &= \Big[\mu(a_i) - \mu(b_i)\Big]\Big[V(x-e_i) - 2V(x) + V(x+e_i)\Big] \\ &\quad + \sum_{j\neq i}^{N} \Big[\mu(a_j) - \mu(b_j)\Big] \\ &\quad \times \Big[V(x+e_i) - V(x+e_i-e_j) - V(x) + V(x-e_j)\Big]. \end{aligned} \tag{5.16}$$

Now, we proceed to prove the structure of the policy. We distinguish between four cases:

1. $a_j > b_j$ for $j = 1, \ldots, N$,
2. $a_i < b_i$ and $a_j > b_j$ for $j \neq i$,
3. $a_j < b_j$ for $j = 1, \ldots, N$, and
4. $a_i \geq b_i$ and $a_j \leq b_j$ for $j \neq i$.

Note that case 1 cannot occur, since we assumed that our starting point was a state in which all capacity was assigned already. Hence, one cannot assign even more capacity. Cases 2 and 3 do not occur either. Intuitively, assigning fewer servers to queue i while increasing the number of jobs in queue i leads to degraded performance. To improve readability, the rigorous proofs are given by Lemma's A.1 and A.2 in the Appendix. The proofs are based on the method of contradiction: assuming that the statement of case 2 or 3 are true, we derive that $Z < 0$. However, above we have shown that $Z \geq 0$. Hence, we are left with case 4, which completes the proof. □

5.4 Numerical experiments

In this section, we will illustrate the monotonicity results of the previous sections. First, we will show how variability in the time constraints affects the processor allocation for the three different models: I) service facilities with dedicated resources, II) service facilities with limited resource sharing, and III) service facilities with full flexibility in resource sharing. Then, we

will study the differences between the optimal policies for the three different models, in particular, we will study the effect of having more flexibility in the system versus the reduction in the number of allocated processors. We will run our experiments under two systems that consist of two and three facilities, respectively. The parameters used in the experiment are defined as follows:

- λ_i: the arrival rate at facility i;
- $\mu(a_i)$: the service rate of using a_i processors at facility i;
- $c_i(a_i)$: the costs of using a_i processors at facility i;
- α_i: the time constraint of facility i.

First, we show the experimental results of the system consisting of two facilities. The parameter values are set as follows:

- $\lambda_1 = \lambda_2 = 0.5$;
- $\mu(a_i) = \sqrt{a_i}\mu$ and $\mu(a_i) = a_i^2\mu/5$ with $\mu = 1.2$;
- $c_i(a_i) = a_i$, $c_i(a_i) = a_i^2$, and $c_i(a_i) = \sqrt{a_i}$;
- $\alpha_1 = 0.5$ and $\alpha_2 \in \{0.25, 0.35, 0.5, 0.75, 1, 1.25, 1.5\}$.

Based on these values, the minimum number of processors required by the system to meet all time constraints of the different facilities are illustrated in Tables 5.1(a) and 5.1(b). From these tables, we observe that 1) when the time constraint is less strict, the minimum number of processors required is non-increasing, and 2) as the system is more flexible, the number of processors needed decreases.

The properties stated above also hold for the experiment with three facilities, and its results are shown in Tables 5.1(c) and 5.1(d). The results of the systems with three facilities are based on the parameter values below.

- $\lambda_1 = \lambda_2 = \lambda_3 = 0.5$;
- $\mu(a_i) = \sqrt{a_i}\mu$ and $\mu(a_i) = a_i^2\mu/5$ with $\mu = 1.2$;
- $c_i(a_i) = a_i$, $c_i(a_i) = a_i^2$, $c_i(a_i) = \sqrt{a_i}$;

α_2	0.25	0.35	0.5	0.75	1	1.25	1.5
dedicated resources	20	13	10	8	7	7	6
limited resource sharing	16	10	6	6	6	6	5
full resource sharing	15	9	6	5	5	5	5

(a) System with two facilities and $\mu(a_i) = \sqrt{a_i}\mu$ with $\mu = 1.2$.

α_2	0.25	0.35	0.5	0.75	1	1.25	1.5
dedicated resources	9	8	8	7	7	7	7
limited resource sharing	5	4	4	4	4	4	4
full resource sharing	5	4	4	4	4	4	4

(b) System with two facilities and $\mu(a_i) = a_i^2\mu/5$ with $\mu = 1.2$.

α_3	0.25	0.35	0.5	0.75	1	1.25	1.5
dedicated resources	25	18	15	13	12	12	11
limited resource sharing	17	11	8	7	7	7	7
full resource sharing	15	9	6	6	6	6	6

(c) System with three facilities and $\mu(a_i) = \sqrt{a_i}\mu$ with $\mu = 1.2$.

α_3	0.25	0.35	0.5	0.75	1	1.25	1.5
dedicated resources	13	12	12	11	11	11	11
limited resource sharing	5	5	4	4	4	4	4
full resource sharing	5	5	4	4	4	4	4

(d) System with three facilities and $\mu(a_i) = a_i^2\mu/5$ with $\mu = 1.2$.

Table 5.1. Minimum number of processors required to meet service level constraints.

- $\alpha_1 = \alpha_2 = 0.5$, and $\alpha_3 \in \{0.25, 0.35, 0.5, 0.75, 1, 1.25, 1.5\}$.

From Table 5.1 we observe that the difference in the required number of processors between the system with dedicated servers and the system with limited resource sharing is quite large (model I versus model II). However, the system with limited and full resource sharing have quite similar performance

experiment 1	$c_i(a_i) = a_i$	$\mu(a_i) = \sqrt{a_i}\mu$
experiment 2	$c_i(a_i) = a_i^2$	$\mu(a_i) = \sqrt{a_i}\mu$
experiment 3	$c_i(a_i) = \sqrt{a_i}$	$\mu(a_i) = a_i^2\mu/5$

Table 5.2. Parameter choices for $c_i(\cdot)$ and $\mu(\cdot)$ with $\mu = 1.2$.

(model II versus model III). Note that we did not specify the cost functions that have been used in these experiments. This is due to the fact that the calculation of the minimum required number of processors A_i^*, such that the time constraints are satisfied, is independent of the cost functions.

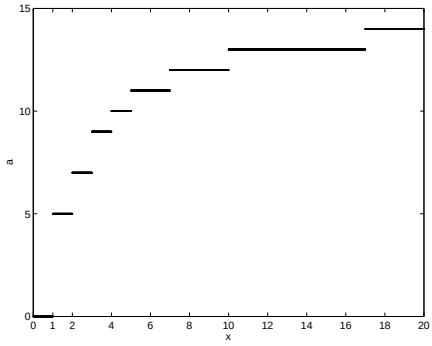
Now we focus our attention to the structure of the optimal resource-allocation policy for a system consisting of two and three facilities, respectively. To illustrate the structure of the policies, we vary the structure of the functions for $c_i(\cdot)$, and $\mu_i(\cdot)$ as given in Table 5.2. We start by showing the experimental results of the system with two facilities. The parameter values used in the three experiments are set as follows.

- $\lambda_1 = \lambda_2 = 0.5$;
- $\alpha_1 = 0.5$ and $\alpha_2 = 0.25$.

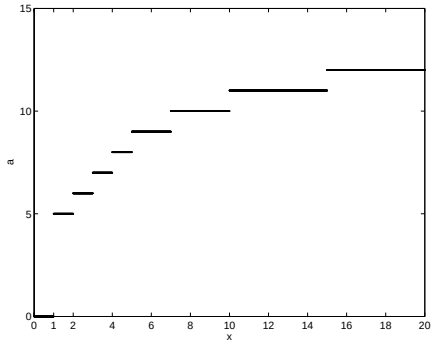
In model I of service facilities with dedicated resources, the number of resources A_i^* allocated to facility i in the system is a constant, given by $A_i^* = \lceil \mu^{-1}(\lambda_i + 1/\alpha_i) \rceil$. For example, for experiment 1, the expression gives that five processors should be used for facility 1 and fifteen for facility 2.

We now turn our attention to model II of service facilities with limited resource sharing. Based on Theorem 5.3.3, if both $\mu(a_i)$ and $c_i(a_i)$ are increasing functions in the number of allocated resources a_i, then the optimal allocation policy to any facility j is a non-decreasing function in the number of customers at that facility, given that the number of customers at all other facilities and the number of resources allocated to all other facilities are fixed. Given that the number of customers at facility 2 is two and the number of resources assigned to facility 2 is one, Figure 5.1 illustrates the structure of the optimal policy for facility 1. These three figures correspond to the three experiments shown in Table 5.2.

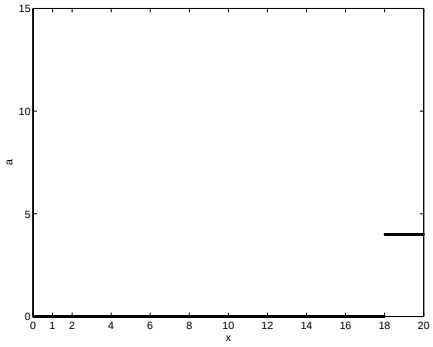
Finally, we discuss model III with full flexibility in resource sharing. The structure of the optimal policy is quite similar to that of the previous model (with limited resource sharing). Consider again the optimal policy for facility 1. The corresponding experimental results are shown in Figure 5.2.



(a) Optimal action a as a function of x_1 for experiment 1.



(b) Optimal action a as a function of x_1 for experiment 2.

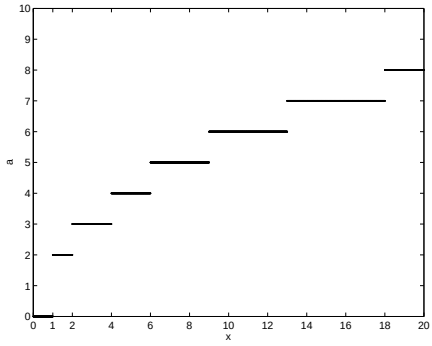


(c) Optimal action a as a function of x_1 for experiment 3.

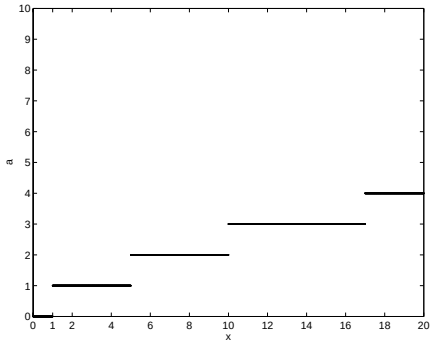
Figure 5.1. Experiments of the system with two facilities: optimal policy of facility 1 for the limited resource sharing model, given $(x_2, a_2) = (2, 1)$.

All figures illustrate that the optimal policy is a non-decreasing function in number of customers at the facility, given that the number of customers at facility 2 is $x_2 = 2$.

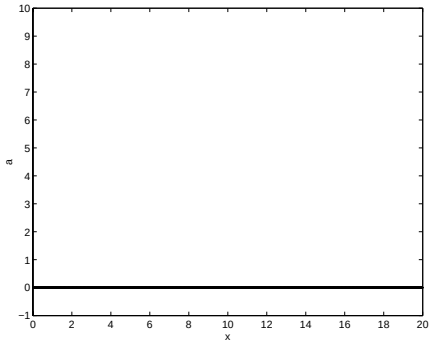
Figures 5.1 and 5.2 show that the structure of the optimal policy for the system with two facilities. The experiments with three (and more) facilities show similar structure. Combining the experimental results of the system with two and three facilities, we conclude that the optimal policy for an



(a) Optimal action a as a function of x_1 for experiment 1.



(b) Optimal action a as a function of x_1 for experiment 2.



(c) Optimal action a as a function of x_1 for experiment 3.

Figure 5.2. Experiments of the system with two facilities: optimal policy to facility 1 in case of service facility with full flexibility in resource sharing, given $x_2 = 2$.

arbitrary facility in model I with dedicated resources is a constant; in case of model II with limited resource sharing and model III with full flexibility in resource sharing, the optimal policy for each facility is a non-decreasing function in the number of customers at that facility.

5.5 Conclusion and further research

In this chapter, we have derived an essential characterization of the optimal policy in three different models. We have shown that directional monotonicity is not sufficient to derive the structure of the optimal policy. In addition to directional convexity, the structure of the problem also requires submodularity of the relative value function. In general, this is hard to derive for multi-dimensional systems, since one needs to compare different states that differ in multiple components simultaneously. The extensive numerical experiments reveal several fundamental insights of the relative effectiveness the optimal policies.

There are several interesting avenues for further research. First, one may suspect that the Poisson assumption of the arrival process can be relaxed. The proof of submodularity shows that the service rates are the dominant factor for the properties of the relative value function. This suggests that there is room for more generality in the arrival process for which the structure of the policies remain valid. Second, from an application point of view, generalization to non-exponential service times is practically relevant. It is an open question to what extent the policies are still optimal for, e.g., phase-type service distributions. Finally, user-perceived service quality often requires more detailed information than the mean processing time only. The level of quality can be highly dependent on, e.g., the variance and/or the tail probabilities in the processing times. This requires a new approach to handle such service requirements, opening up new challenging areas for research.

Chapter 6

Full Structural Properties of Optimal Allocation Policies for Single-queue Systems

In previous chapters, we have studied resource-allocation problems in different settings. In Chapter 4, we studied a resource-allocation problem in which resources need to be reserved in advance before they are available. In Chapter 5, we studied a resource-allocation problem in a multi-queue setting in which multiple queues compete for the same pool of shared resources. In this model, the optimal resource-allocation policy partitions the resource pool in subsets each of which is dynamically allocated to the different queues. The model studied in this chapter is a special case of the model in Chapter 5 in which there is only a single queue. We show via dynamic programming that (1) the optimal allocation policy has a work-conservation property, and (2) the optimal number of servers follows a step function with as extreme policy the bang-bang control policy. Moreover, (3) we provide conditions under which the bang-bang control policy takes place. These characterizations of the optimal policy are not directly applicable in the multi-queue setting because of techniques used to prove the results which are not generalizable to multiple queues.

6.1 Introduction

In this chapter, we focus on a model with a single FCFS queue with a common pool of computing resources in which for each job a number of computing resources has to be allocated dynamically such that a mean sojourn time requirement is met against minimal allocation costs. In this model, one has to deal with the following trade-off: if the number of computing resources

allocated to a facility is too low, then the processing power is insufficient to meet strict processing time requirements; if the number of computing nodes allocated is too high, then it brings forth high resource-allocation costs.

In the literature, a lot of research has been devoted to resource-allocation problems. In [26, 70, 76], the authors investigate resource-allocation problems in the context of protocol design. The problem from an architectural point of view is studied in [25, 34, 102]. Security assurance in Grid/Cluster job scheduling is studied in [91, 108]. Other research is focused on economic models in a Grid computing environment, e.g., [80, 19, 43]. Several papers are focused on optimization problems in the context of resource allocation. In [12], a framework for resource allocation and task scheduling is presented, where the objective function is to minimize the job completion time. Nurmi and co-authors [68, 67] propose a statistical method determining when a job should be submitted to a batch queue to ensure that it will be running at a particular time in the future. Fukuda et al. [35] propose a resource-allocation scheme to share resources fairly among users by solving the utilization-maximization problem where the utilization ratio is a function of the video quality and the resource-allocation costs.

In this chapter, we study structural properties of the optimal resource-allocation policy for single-queue systems. In this context, we study the two models described in Section 5.2: (1) a system in which a chosen allocation cannot be changed during a service of a job, and (2) a system in which it is allowed to change the allocation during the service of a job. For both models, we show via dynamic programming that (i) the optimal allocation policy has a work-conservation property that implies when the system is not empty, the optimal policy is not allowed to keep all computing resources idle, (ii) the optimal number of servers follows a step function with as extreme policy the bang-bang control policy, which means a facility receives all computing resources or none at all, and moreover (iii) we also provide the conditions under which the bang-bang control policy is optimal. The techniques to prove such results are based on monotonicity properties of the dynamic programming relative value function (see, e.g., [55, 54, 78]).

The contribution of this chapter is two-fold. First, on the methodological side we provide a full characterization of the optimal policy for a single-queue system. Interestingly, the derivation of these results is not obtained via standard induction-based arguments, but is based on a combination of direct arguments and induction. The results provide new and valuable insight into optimal resource-allocation problems with time constraints. Second, on the

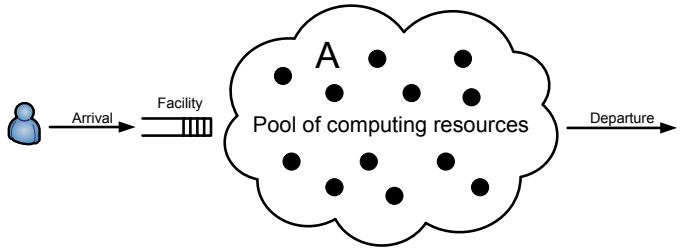


Figure 6.1. Arrival and departure of the jobs.

application side we have structured policies that are easy to be implemented in systems that are highly relevant in practice.

This chapter is organized as follows. In Section 6.2 we formulate the models for the different cases. Next, we derive the special properties of the optimal policy in Section 6.3. In Section 6.4 we illustrate these results by numerical experiments. Finally, in Section 6.5 we make some concluding remarks and address a number of challenging topics for further research.

6.2 Model formulation

Consider a service facility at which jobs arrive according to a Poisson process with arrival rate λ and have exponentially distributed service requirements. The facility has a single infinite-sized FCFS queue and a pool of $A \geq 1$ parallel and identical servers that can work together to process a single job. There is a central decision maker that can assign a number of servers to a job. The service rate of a job depends on the number of servers assigned to that job. More precisely, when a job has been allocated a servers, the service duration of that job is exponentially distributed with parameter $\mu(a)$, which is strictly increasing in a. Without loss of generality, we assume that $\mu(0) = 0$. After a job has completed its service, it leaves the system. Throughout the chapter, it is assumed that the stability condition $\lambda < \mu(A)$ is met, and that the system is in steady state. Figure 6.1 gives an illustration of the system.

We consider the following cost structure in the facility. When the facility uses a resources a cost of $c(a)$ is incurred by the system per unit time. Here $c(\cdot)$ is an strictly increasing function of a. Without loss of generality, we

assume that $c(0) = 0$. Let S denote the sojourn time of an arbitrary job at the facility. The problem in the system is to find a server assignment policy that minimizes that long-run average costs subject to $\mathbb{E}S \leq \alpha$ for some $\alpha > 0$.

When the system works at full speed, all A computing resources are allocated at each moment in time. In this case, the average sojourn time is given by the mean sojourn in an $M|M|1$ queue with arrival rate λ and service rate $\mu(A)$. Therefore, by using all A computing resources the average sojourn time is equal to $\frac{1}{\mu(A)-\lambda}$. Hence, to ensure that there exists at least one allocation strategy that meets the time constraint α, the total number of compute resources A should satisfy the following condition:

$$A \geq \left\lceil \mu^{-1}(\lambda + 1/\alpha) \right\rceil,$$

where $\mu^{-1}(\cdot)$ is the inverse function of $\mu(\cdot)$. Throughout the chapter, it is assumed that this condition is met.

The objective of the decision maker is to derive an optimal policy based on the number of jobs in the system. More precisely, when the decision maker observes x jobs in the system, he can decide to allocate $0 \leq i(x) \leq A$ servers to the first job in the queue. We emphasize that at any moment in time, there is at most one job in service.

We study two models. First, we study the case in which the resource-allocation policy cannot be changed when a job is being served; only *upon the start of the service* of the next job the resource-allocation policy can be modified. The second case deals with the fully flexible case in which the system can change the resource-allocation policy *at any moment in time*, thus also during the service of a job.

6.2.1 Limited resource allocation policy

In this subsection we focus on the case in which the resource allocation for a service facility can only be changed upon the start of the service of a new job. Hence, adding or removing resources during a service is not allowed. To study this case, we cast the resource-allocation problem as a Markov decision problem.

Define the state space $\mathcal{X} = \mathbb{N}_0 \times \{0, \ldots, A\}$, where $(x, a) \in \mathcal{X}$ denotes the state in which there are x jobs at the facility with a resources allocated to the first job. When the system is in state $(x, a) \in \mathcal{X}$ the decision maker can

choose actions from the action space $\mathcal{A}_{(x,a)} = \{b \in \mathbb{N}_0 \,\big|\, a + b \leq A \text{ and } ab = 0\}$, where action $b \in \mathcal{A}_{(x,a)}$ denotes the available number of resources for allocation. Here, the restriction $ab = 0$ models the fact that when a service is ongoing (i.e., $a > 0$), the service allocation cannot be changed (i.e., $b = 0$). The transition rates when the system is in state $(x, a) \in \mathcal{X}$ and action $b \in \mathcal{A}_{(x,a)}$ is chosen are given by

$$p\big((x,a),b,(x',b')\big) = \begin{cases} \lambda, & x' = x+1 \text{ and } b' = a+b, \\ \mu(a+b), & x' = [x-1]^+ \text{ and } b' = 0, \\ 0, & \text{otherwise}, \end{cases}$$

with $[x]^+ = \max\{x, 0\}$. The first line in the expression above models arrivals, the second line models service completions, and the third line prohibits any other state transitions. Note that when a service completes, the resource allocated for that facility is released completely. Finally, when the system is in state $(x, a) \in \mathcal{X}$ and action $b \in \mathcal{A}_{(x,a)}$ has been chosen, the direct cost is $r\big((x,a),b\big) = c(a+b)$ per time unit. The quadruple $(\mathcal{X}, \mathcal{A}, p, r)$ completely describes the Markov decision process. The goal is to find a policy π^* that minimizes the long-term average costs under the time constraint, thus

$$\min_{\pi} g(\pi) \quad \text{subject to} \quad \mathbb{E}S \leq \alpha.$$

As stated in the previous chapters, the constrained Markov decision problem can be rewritten as an unconstrained Markov decision problem using Lagrange multipliers. Assume that the uniformization constant $\lambda + \mu(A) = 1$. Then we can consider the rates to be transition probabilities. Let $V(x, a)$ be a real-valued function defined on the state space. The long-term average optimal actions are a solution of the optimality equation (in vector notation) $g + V = TV$, where T is the dynamic programming operator acting on V, defined as follows

$$\begin{aligned} TV(x,a) &= \tau\frac{x}{\lambda} + c(a) + \lambda H(x+1,a) + \mu(a)H([x-1]^+,0) \\ &\quad + \big(1 - \lambda - \mu(a)\big)V(x,a) \\ &= \tau\frac{x}{\lambda} + c(a) + \lambda H(x+1,a) + \mu(a)H([x-1]^+,0) \\ &\quad + \big(\mu(A) - \mu(a)\big)V(x,a), \end{aligned} \tag{6.1}$$

where τ is the Lagrange multiplier, and where the function $H(\cdot,\cdot)$ is given by

$$H(x,a) = \min_{b \in \mathcal{A}_{(x,a)}} \{V(x, a+b)\}.$$

The first term in the dynamic programming operator (6.1) corresponds to the service requirement $\mathbb{E}S \leq \alpha$. When τ increases, the value of $\mathbb{E}S$ decreases. Therefore, there exists a value of τ^* such that $\mathbb{E}S^{\tau^*} > \alpha$ for the facility and there exists another τ' such that $\mathbb{E}S^{\tau'} < \alpha$, where $\tau' = \tau^* + \varepsilon$ for a small $\varepsilon \geq 0$. Note that we can obtain the expected sojourn time at the facility for a given policy by setting $r\big((x,a),b\big) = 0$ for all states (x,a) in the optimality equation. The optimal policy is to randomize between the associated policies π^{τ^*} and $\pi^{\tau'}$ so that the equality $\mathbb{E}S = \alpha$ is achieved for the facility. This is consistent with the fact that the optimal policy will be randomized in exactly one state (see Section 12.6 of Altman [10]). The second term represents the cost of using a resources. The third term is involved with the decision making upon arrival of a job. The fourth term deals with the decision making when a job has completed its service. The final term is the dummy term due to uniformization. Note that the decision making is modeled uniformly through the function $H(\cdot,\cdot)$.

6.2.2 Fully flexible resource allocation policy

In this section we study the case in which the service facility has full flexibility in the resource-allocation policies. The resource facility can change the resource allocation during a service of a job, and it does not have to wait for the job to be finished. Since our system has Poisson arrivals and exponential service times, it suffices to consider only the moments that an event occurs. Therefore, the only difference with the previous case, discussed in Section 6.2.1, is that this system allows to change the allocation at arrival instants.

In the fully flexible case, the state space is given by $\mathcal{X} = \mathbb{N}_0$, where $x \in \mathcal{X}$ denotes that there are x jobs at the facility. The action space is given by $\mathcal{A}_x = \{0,\ldots,A\}$, where action $a \in \mathcal{A}_x$ denotes the number of resources that one has allocated in state $x \in \mathcal{X}$. The transition rates when the system is in state $x \in \mathcal{X}$ and action $a \in \mathcal{A}_x$ is chosen are given by

$$p(x,a,x') = \begin{cases} \lambda, & x' = x+1, \\ \mu(a), & x' = [x-1]^+, \\ 0, & \text{otherwise.} \end{cases}$$

Finally, when the system is in state $x \in \mathcal{X}$ and action $a \in \mathcal{A}_x$ has been chosen, the direct cost is $r(x,a) = c(a)$ per time unit. The tuple $(\mathcal{X},\mathcal{A},p,r)$ completely describes the Markov decision process for this problem.

Let $V(x)$ denote the relative value function in this case. Then, the dynamic programming operator acting on V, defined as follows:

$$TV(x) = \tau\frac{x}{\lambda} + \lambda V(x+1) + \min_{a\in\mathcal{A}_x} T_a(x), \tag{6.2}$$

where

$$\begin{aligned} T_a(x) &= \mu(a)V([x-1]^+) + \big(1-\lambda-\mu(a)\big)V(x) + c(a) \\ &= \mu(a)V([x-1]^+) + \big(\mu(A)-\mu(a)\big)V(x) + c(a). \end{aligned}$$

The first term in the expression $TV(x)$ corresponds to the constraint $\mathbb{E}S \leq \alpha$ of the facility, similar to the limited resource-allocation model, see Equation (6.1). The second term models the arrivals of jobs to the facility. The first term in the expression $T_a(x)$ denotes the departure of a job in case action a has been chosen. The second term in $T_a(x)$ is the uniformization constant. The last term in $T_a(x)$ models the direct cost.

6.3 Structural properties of the optimal policy

In the previous section, we described two models and a solution technique to obtain the optimal policy. However, the optimal policy also possesses structural properties that provide fundamental insight. Moreover, this also enables one to determine the optimal policy with less computational effort due to a reduction of the solution search space. Therefore, in this section, we derive a full characterization of the optimal policy for both models.

6.3.1 Limited resource allocation policy

The structure of the optimal policy for a service facility with the limited resource-allocation policy is more intricate than the case with dedicated resources. As mentioned in previous chapter the optimal actions can be obtained by recursively defining $V_{l+1} = TV_l$ for arbitrary V_0. For $l \to \infty$, the maximizing actions converge to the optimal ones (for existence and convergence of solutions and optimal policies we refer to Puterman [73] and Aviv and Federgruen [11]). The backward recursion equation is given by

$$\begin{aligned} V_{n+1}(x,a) = \tau\frac{x}{\lambda} + c(a) + H_n(x+1,a) + \mu(a)H_n([x-1]^+,0) \\ + \big(\mu(A)-\mu(a)\big)V_n(x,a), \end{aligned} \tag{6.3}$$

where the function $H_n(\cdot,\cdot)$ is given by

$$H_n(x,a) = \min_{b\in\mathcal{A}_{(x,a)}} \{V_n(x,a+b)\}.$$

For ease of notation, we also define $\arg H_n(x,a)$ by

$$\arg H_n(x,a) = \arg\min_{b\in\mathcal{A}_{(x,a)}} \{V_n(x,a+b)\}.$$

The backward recursion equation (6.3) allows us to prove structural properties of the relative value function V through induction on n in V_n. By using this, we can show that the optimal allocation policy has the following properties:

(*i*) **monotonicity**: this implies that the optimal policy is a step function. Thus for all $a \in \arg H(x+k,0)$ and $b \in \arg H(x,0)$, we have $a \geq b$ for all $k \geq 0$.

(*ii*) **work-conservation**: this implies that if the system is not empty, then the optimal policy is not keeping all computing resources idle. Thus the minimal element of $\arg H(x,0)$ is strictly positive for all $x > 0$.

(*iii*) **bang-bang control**: if the condition $c(a)\mu(a+1) - \mu(a)c(a+1) \geq 0$ holds for all $a \in \mathcal{A}_{(x,b)}$, then the optimal policy is a bang-bang control policy, which means using all servers or no server at all.

Monotonicity property (i) has been studied in [110] in a multi-queue setting. For our model, this property follows as a special case for a single server queue. The following properties of the relative value function (proven in [110]) that we need to prove Properties (ii) and (iii) are listed below for completeness of the chapter.

Property 6.3.1 (non-decreasingness)**.** *The relative value function V is increasing in the number of jobs, i.e.,*

$$V(x+1,a) - V(x,a) \geq 0,$$

for all $x \in \mathcal{X}$ and $0 \leq a \leq A$.

Property 6.3.2 (convexity-related properties)**.** *For a single-queue system, the following properties hold*

(*i*) $V(x+1,a) - 2V(x,a) + V(x-1,a) \geq 0$ for all $x \geq 1$ and $a \geq 0$,

(*ii*) $V(x,a) - V([x-1]^+,a) - H([x-1]^+,0) + H([x-2]^+,0) > 0$ for all $x \geq 0$ and $a > 0$,

(*iii*) The minimal element of $\arg H(x,0)$ is strictly positive for all $x \geq 2$,

(*iv*) $H(x+1,0) - 2H(x,0) + H(x-1,0) \geq 0$ for all $x \geq 1$.

Property 6.3.3 (submodularity)**.** *The relative value function satisfies a version of submodularity, namely,*

$$V(x,a+k) - V(x,a) - V(x+1,a+k) + V(x+1,a) > 0,$$

for all $x \geq 1$, $a > 0$, and $k > 0$.

Property 6.3.4 (monotonicity)**.** *If the service rate $\mu(a)$ and cost function $c(a)$ are strictly increasing functions in a, then for all $a \in \arg H(x+k,0)$ and $b \in \arg H(x,0)$, we have $a \geq b$ for all $k \geq 0$.*

Now, we proceed to prove the work-conservation property (ii) and the bang-bang control property (iii) of the optimal policy. Recall that we assumed $c(0) = \mu(0) = 0$. First, we start with the proof of the work-conservation property.

Theorem 6.3.1 (work-conservation)**.** *If the service rate function $\mu(\cdot)$ and cost function $c(\cdot)$ are strictly increasing functions, then the minimal element of $\arg H(x,0)$ is strictly positive for all $x > 0$.*

Proof. Based on part (iii) of Property 6.3.2, we have that the minimal element of $\arg H(x,0)$ is strictly positive for all $x \geq 2$. Therefore, we only need to prove that each element of $\arg H(1,0)$ is strictly positive. This can be proven by the contradiction method. Assume that $0 \in \arg H(1,0)$. Then it follows that $H(1,0) = V(1,0)$. Based on the optimality equation and the assumption that $c(0) = \mu(0) = 0$, we have

$$V(1,0) - V(0,0) = \frac{\tau}{\lambda} + \lambda\big[H(2,0) - H(1,0)\big] + \mu(A)\big[V(1,0) - V(0,0)\big].$$

Since $\mu(A) + \lambda = 1$, it follows

$$\lambda\big[V(1,0) - V(0,0) - H(2,0) + H(1,0)\big] = \frac{\tau}{\lambda}.$$

Because we have $\tau > 0$, it holds that

$$V(1,0) - V(0,0) - H(2,0) + H(1,0) > 0. \tag{6.4}$$

However, based on our assumption $H(1,0) = V(1,0)$ and the fact that $V(0,0) \geq H(0,0)$, it holds that

$$V(1,0) - V(0,0) - H(2,0) + H(1,0) \leq 2H(1,0) - H(0,0) - H(2,0).$$

Based on part (iv) of Property 6.3.2, it follows that $V(1,0) - V(0,0) - H(2,0) + H(1,0)$ is negative. This is in contradiction with Equation (6.4). Therefore, we conclude that each element of $\arg H(1,0)$ is strictly positive, which completes the proof. □

Next, we prove that the optimal allocation policy has the bang-bang control property. To this end, the following result is needed.

Lemma 6.3.1. *The following inequality holds:*

$$V(x,a) - H(x-1,0) > \frac{c(a)}{\mu(a)},$$

for all $x \geq 1$ *and* $a > 0$.

Proof. Since $a > 0$, it holds that $V(x,a) = H(x,a)$ for all x. Therefore, based on the optimality equation, we have

$$\begin{aligned} V(x,a) - V(0,0) = {} & \tau\frac{x}{\lambda} + c(a) + \lambda\big[V(x+1,a) - H(1,0)\big] \\ & + \mu(a)H([x-1]^+,0) + \big[\mu(A) - \mu(a)\big]V(x,a) - \mu(A)V(0,0). \end{aligned}$$

Since $\lambda + \mu(A) = 1$, the equation above implies that

$$\begin{aligned} \lambda\big[H(1,0) - V(0,0) + V(x,a) - V(x+1,a)\big] = {} & \tau\frac{x}{\lambda} + c(a) \\ & + \mu(a)\big[H([x-1]^+,0) - V(x,a)\big]. \end{aligned} \tag{6.5}$$

Because of the definition of the value function, it holds that $V(0,0) \geq H(0,0)$. This implies that

$$\begin{aligned} & H(1,0) - V(0,0) + V(x,a) - V(x+1,a) \\ & \leq H(1,0) - H(0,0) + V(x,a) - V(x+1,a). \end{aligned} \tag{6.6}$$

Based on parts (i) and (ii) of Property 6.3.2, we obtain that

$$V(x+1,a) - V(x,a) \geq V(2,a) - V(1,a) \geq H(1,0) - H(0,0). \qquad (6.7)$$

Combining Equations (6.6) and (6.7), we conclude that

$$H(1,0) - V(0,0) + V(x,a) - V(x+1,a) \leq 0.$$

Because $\tau > 0$, Equation (6.5) implies that

$$c(a) + \mu(a)\big[H(x-1,0) - V(x,a)\big] < 0,$$

and hence $V(x,a) - H(x-1,0) > \frac{c(a)}{\mu(a)}$. □

We are now ready to prove the optimality of the bang-bang control policy.

Theorem 6.3.2 (bang-bang control)**.** *For a single-queue system, the following properties hold*

(*i*) $\arg H(0,0) = \{0\}$.

(*ii*) if $c(a)\mu(a+1) - \mu(a)c(a+1) \geq 0$ for all $a \in \mathcal{A}_{(x,b)}$, then $\arg H(x,0) = \{0, A\}$ for all $x > 0$.

Proof. We use the contradiction method to prove this theorem. To start, let $a \in \arg H(0,0)$. Then $H(0,0) = V(0,a)$. Now assume that $a > 0$, then based on the definition of the value function, we have

$$\begin{aligned} V(0,0) - V(0,a) = & -c(a) + \lambda\big[H(1,0) - H(1,a)\big] - \mu(a)H(0,0) \\ & + \mu(A)\big[V(0,0) - V(0,a)\big] + \mu(a)V(0,a) \\ = & -c(a) + \lambda\big[H(1,0) - H(1,a)\big] \\ & + \mu(A)\big[V(0,0) - V(0,a)\big]. \end{aligned}$$

Since $\lambda + \mu(A) = 1$, we have that

$$\lambda\big[V(0,0) - V(0,a)\big] = -c(a) + \lambda\big[H(1,0) - H(1,a)\big].$$

Because $H(1,0) - H(1,a) \leq 0$ and $c(a) > 0$ for $a > 0$, we have $V(0,0) - V(0,a) < 0$, which is in contradiction with the assumption that $0 < a \in \arg H(0,0)$. Therefore, we conclude that $\arg H(0,0) = \{0\}$, which proves part (i).

To prove part (ii), we assume that there exists $x \geq 1$ and $a \in \arg H(x,0)$ such that $0 < a < A$. Then we have $V(x,a) = H(x,0)$ for all $0 < a < A$. Because of the definition of the value function, $V(x,a) = H(x,a)$ for all x. Based on the optimality equation, we have

$$\begin{aligned} V(x,a+1) - V(x,a) = \; & c(a+1) - c(a) + \lambda\big[V(x+1,a+1) - V(x+1,a)\big] \\ & + \big[\mu(a+1) - \mu(a)\big]H(x-1,0) + \mu(A)\big[V(x,a+1) - V(x,a)\big] \\ & - \mu(a+1)V(x,a+1) + \mu(a)V(x,a). \end{aligned}$$

This implies that

$$\begin{aligned} & \lambda\big[V(x,a+1) - V(x,a) - V(x+1,a+1) + V(x+1,a)\big] \\ & \quad = c(a+1) - c(a) + \big[\mu(a+1) - \mu(a)\big]H(x-1,0) \\ & \qquad - \mu(a+1)V(x,a+1) + \mu(a)V(x,a). \end{aligned}$$

Since $V(x,a) = H(x,0)$, we have $V(x,a+1) \geq V(x,a)$, and hence

$$\begin{aligned} & \lambda\big[V(x,a+1) - V(x,a) - V(x+1,a+1) + V(x+1,a)\big] \\ & \quad \leq c(a+1) - c(a) + \big[\mu(a+1) - \mu(a)\big]H(x-1,0) \\ & \qquad - \mu(a+1)V(x,a) + \mu(a)V(x,a) \\ & \quad = c(a+1) - c(a) + \big[\mu(a+1) - \mu(a)\big]\big[H(x-1,0) - V(x,a)\big]. \end{aligned} \tag{6.8}$$

Based on Property 6.3.3, it holds that $V(x,a+1) - V(x,a) - V(x+1,a+1) + V(x+1,a) \geq 0$. Therefore, Inequality (6.8) implies that

$$c(a+1) - c(a) + \big[\mu(a+1) - \mu(a)\big]\big[H(x-1,0) - V(x,a)\big] \geq 0,$$

which is equivalent to

$$V(x,a) - H(x-1,0) \leq \frac{c(a+1) - c(a)}{\mu(a+1) - \mu(a)}. \tag{6.9}$$

Based on the condition $c(a)\mu(a+1) - \mu(a)c(a+1) \geq 0$ of part (ii), we have

$$\begin{aligned} & c(a)\mu(a+1) - c(a)\mu(a) + c(a)\mu(a) - \mu(a)c(a+1) \\ & = \; c(a)\big[\mu(a+1) - \mu(a)\big] - \mu(a)\big[c(a+1) - c(a)\big] \geq \; 0, \end{aligned}$$

which implies that

$$\frac{c(a+1) - c(a)}{\mu(a+1) - \mu(a)} \leq \frac{c(a)}{\mu(a)}, \text{ and hence } V(x,a) - H(x-1,0) \leq \frac{c(a)}{\mu(a)},$$

which is in contradiction with Lemma 6.3.1. This proves part (ii). □

The following result follows from combining Theorems 6.3.1 and 6.3.2.

Corollary 6.3.1. *Under the assumption* $c(a)\mu(a+1) - \mu(a)c(a+1) \geq 0$ *it holds that* $\arg H(x,0) = \{A\}$ *for all* $x > 0$ *and* $a \in \mathcal{A}_{(x,b)}$.

6.3.2 Fully flexible resource allocation policy

In this section we focus our attention to optimal allocation strategies for the system in which the allocation of the number of servers is allowed to change when a job is already in service. We shall adopt the same techniques in deriving the structure of the optimal policy as in the previous section.

We start by rewriting Equation (6.2) for the fully flexible system as a set of backward recursion equations. This set of equations is given by

$$V_{n+1}(x) = \tau\frac{x}{\lambda} + \lambda V_n(x+1) \ + \min_{a\in\mathcal{A}_x} T_a^n(x), \tag{6.10}$$

where $T_a^n(x)$ is given by

$$T_a^n(x) = \mu(a)V_n([x-1]^+) + \big[\mu(A) - \mu(a)\big]V_n(x) + c(a).$$

By performing backward recursion, the optimal allocation policy can be obtained, which has the following properties:

(i) **monotonicity**: this implies that the optimal policy is a step function. Thus for all $a \in \arg\min_{a\in\mathcal{A}_{x+1}} \big\{T_a(x+1)\big\}$ and $b \in \arg\min_{a\in\mathcal{A}_x} \big\{T_a(x)\big\}$ we have $a \geq b$ for all $x \geq 0$.

(ii) **work-conservation**: this implies that if the system is not empty, then the optimal policy is not keeping all compute resources idle. Thus, the minimal element of $\arg\min_{a\in\mathcal{A}_x} \big\{T_a(x)\big\}$ is strictly positive for all $x > 0$.

(iii) **bang-bang control**: if $c(a)\mu(a+1) - \mu(a)c(a+1) \geq 0$ holds for all $a \in \mathcal{A}_x$, then the optimal policy is a bang-bang control policy, which means using all servers or no server at all.

Monotonicity property (i) has been proven in [112]. Additionally, we need the following properties of the relative value function from the same chapter to prove Properties (ii) and (iii).

Property 6.3.5 (convexity)**.** *Assume that the functions $\mu(\cdot)$ and $c(\cdot)$ are strictly increasing functions, then $V(x)$ is a convex increasing function in x, i.e., $V(x) > V(x-1)$ and $V(x+1) - 2V(x) + V(x-1) > 0$ for all $x \geq 1$.*

Property 6.3.6 (monotonicity)**.** *Assume that the functions $\mu(\cdot)$ and $c(\cdot)$ are strictly increasing functions. Then the optimal resource allocation strategy is given by a non-decreasing curve, i.e., for each $a \in \arg\min_{a \in \mathcal{A}_{x+1}} \{T_a(x+1)\}$ and $b \in \arg\min_{a \in \mathcal{A}_x} \{T_a(x)\}$ we have $a \geq b$ for all $x \geq 0$.*

We proceed to derive parts (ii) and (iii). We start with the work-conservation property. Recall that without loss of generality, it is assumed that $c(0) = \mu(0) = 0$. The following result shows that if the system is not empty, then the optimal policy will not keep all computing resources idle.

Theorem 6.3.3 (work-conservation)**.** *If the system is not empty, then the optimal policy is not keeping all compute resources idle. Thus, the minimal element of $\arg\min_{a \in \mathcal{A}_x} \{T_a(x)\}$ is strictly positive for all $x > 0$.*

Proof. We use the contradiction method to prove this theorem. Assume there exists $x > 0$ such that the minimal element of $\arg\min_{a \in \mathcal{A}_x} \{T_a(x)\}$ is 0. Then, Property 6.3.6 implies that $\arg\min_{a \in \mathcal{A}_x} \{T_a(0)\} = \{0\}$. Therefore, by applying the optimality equation, we obtain

$$V(x) - V(0) = \tau \frac{x}{\lambda} + \lambda\big[V(x+1) - V(1)\big] + \mu(A)\big[V(x) - V(0)\big].$$

Since $\lambda + \mu(A) = 1$, this is equivalent to

$$\lambda\big[V(1) - V(0) + V(x) - V(x+1)\big] = \tau \frac{x}{\lambda}.$$

Because the time constraint of the facility is finite, we have $\tau > 0$. Hence, it holds that $V(1) - V(0) + V(x) - V(x+1) > 0$. However, Property 6.3.5 directly implies that $V(1) - V(0) + V(x) - V(x+1) \leq 0$, which contradicts the convexity of Property 6.3.5. This completes the proof. □

To prove that the optimal allocation policy has the bang-bang control property, we need the following lemma.

Lemma 6.3.2. *If $b \in \arg\min_{a \in \mathcal{A}_x} \{T_a(x)\}$ for all $x \geq 1$, then the following inequality holds:*

$$V(x) - V(x-1) \geq \frac{c(b)}{\mu(b)}.$$

Proof. From the definition of the value function, we have

$$T_a(0) = \mu(a)V(0) + \big[\mu(A) - \mu(a)\big]V(0) + c(a) = \mu(A)V(0) + c(a),$$

which implies that $\arg\min_{a\in\mathcal{A}_x}\big\{T_a(0)\big\} = \{0\}$. For $x > 0$, let us define $b \in \arg\min_{a\in\mathcal{A}_x}\big\{T_a(x)\big\}$. Then Theorem 6.3.3 implies that $b > 0$. Moreover, based on the optimality equation, it holds that

$$\begin{aligned} V(x) - V(0) = \tau\frac{x}{\lambda} + \lambda\big[V(x+1) - V(1)\big] + \mu(b)V(x-1) \\ + \mu(A)\big[V(x) - V(0)\big] - \mu(b)V(x) + c(b). \end{aligned}$$

Since $\lambda + \mu(A) = 1$, this is equivalent to

$$\lambda\big[V(1) - V(0) + V(x) - V(x+1)\big] = \tau\frac{x}{\lambda} + c(b) + \mu(b)\big[V(x-1) - V(x)\big].$$

Based on $\tau > 0$ and Property 6.3.5, it holds that

$$V(1) - V(0) + V(x) - V(x+1) \leq 0,$$

and hence

$$c(b) + \mu(b)\big[V(x-1) - V(x)\big] < 0,$$

which is equivalent to $V(x) - V(x-1) > \frac{c(b)}{\mu(b)}$. This completes the proof. □

We are now ready to prove the optimality of the bang-bang control policy.

Theorem 6.3.4 (Optimality of bang-bang control)**.** *For a single-queue system, the following properties hold:*

(*i*) $\arg\min_{a\in\mathcal{A}_0}\big\{T_a(0)\big\} = \{0\}$.

(*ii*) If $c(a)\mu(a+1) - \mu(a)c(a+1) \geq 0$ holds for all $a \in \mathcal{A}_x$, then $\arg\min_{a\in\mathcal{A}_x}\big\{T_a(x)\big\} = \{0, A\}$ for all $x > 0$.

Proof. First, we prove that $\arg\min_{a\in\mathcal{A}_0}\big\{T_a(0)\big\} = \{0\}$. To this end, the definition of the value function implies

$$T_a(0) = \mu(a)V(0) + \big[\mu(A) - \mu(a)\big]V(0) + c(a) = \mu(A)V(0) + c(a),$$

which immediately shows that $\arg\min_{a\in\mathcal{A}_0}\big\{T_a(0)\big\} = 0$. This concludes the proof of part (i).

To prove part (ii), we use the contradiction method. Assume that there exists $x > 0$ and $b = \arg\min_{a \in \mathcal{A}_x} \{T_a(x)\}$ such that $0 < b < A$. Then $0 < b < A$, which implies that $T_{b+1}(x) - T_b(x) \geq 0$. Since

$$T_{b+1}(x) - T_b(x) \geq 0.$$

Since

$$T_{b+1}(x) - T_b(x) = c(b+1) - c(b) + \big[\mu(b+1) - \mu(b)\big]\big[V(x-1) - V(x)\big],$$

we have

$$V(x) - V(x-1) \leq \frac{c(b+1) - c(b)}{\mu(b+1) - \mu(b)}.$$

Based on the condition of the lemma, we have $c(a)\mu(a+1) - \mu(a)c(a+1) \geq 0$ for all $a \in \mathcal{A}_x$. Therefore, it holds that $\frac{c(b+1)-c(b)}{\mu(b+1)-\mu(b)} \leq \frac{c(b)}{\mu(b)}$. Thus

$$V(x) - V(x-1) \leq \frac{c(b)}{\mu(b)}.$$

However, based on Lemma 6.3.2 it holds that $V(x) - V(x-1) > \frac{c(b)}{\mu(b)}$ for all $x > 0$, which is in contradiction. This completes part (ii) of the proof. □

By combining Theorems 6.3.3 and 6.3.4 we obtain the following result.

Corollary 6.3.2. $\arg\min_{a \in \mathcal{A}_x} \{T_a(x)\} = \{A\}$ *for all* $x > 0$.

6.4 Numerical results

In this section, we illustrate the structural properties of the optimal allocation policies. To this end, we have conducted numerical experiments for a variety of cost functions $c(\cdot)$ and service rates $\mu(\cdot)$ having different characteristics. These parameter choices are given in Table 6.1 for seven experiments. The other parameters in our experiments are set as follows: $A = 85$ (this represents the number of servers in the DAS-3 cluster that we used, see [2]), $\lambda = 2$, $\mu = 0.7$, and $\tau = 1$. Note that μ is defined as $\mu(1)$, the service rate of using a single server (used for normalization of the server speed). Moreover, we fixed the value of τ, since we are only interested in the structural properties of the optimal allocation policy. In Experiments 1–4, the functions $c(\cdot)$ and $\mu(\cdot)$ satisfy the conditions of the bang-bang control property

Experiment 1	$c(a) = a$	$\mu(a) = a\mu$
Experiment 2	$c(a) = \sqrt{a}$	$\mu(a) = a^2\mu$
Experiment 3	$c(a) = \sqrt[3]{a}$	$\mu(a) = \sqrt{a}\mu$
Experiment 4	$c(a) = a^{1.5}$	$\mu(a) = a^2\mu$
Experiment 5	$c(a) = a^2$	$\mu(a) = \sqrt{a}\mu$
Experiment 6	$c(a) = \sqrt{a}$	$\mu(a) = \sqrt[3]{a}\mu$
Experiment 7	$c(a) = a^2$	$\mu(a) = a^{1.5}\mu$

Table 6.1. Parameter choices for $c(\cdot)$ and $\mu(\cdot)$.

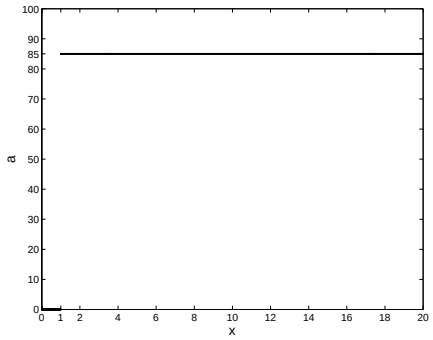
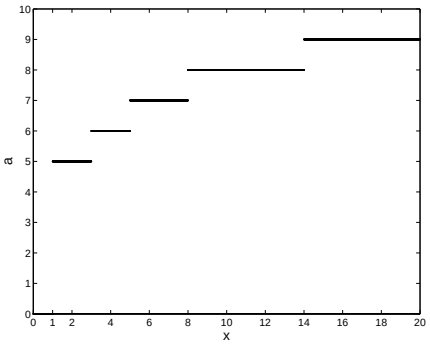


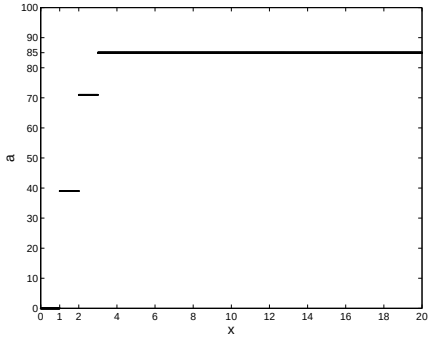
Figure 6.2. Optimal action a as a function of x for Experiment 1 to 4.

(see Theorems 6.3.2 and 6.3.4). For these experiments, the results for both models are shown in Figure 6.2. In this figure, we see that the results are in agreement with Corollaries 6.3.1 and 6.3.2. In Experiments 5–7, the conditions of the bang-bang control property are not satisfied. Since $c(\cdot)$ and $\mu(\cdot)$ are increasing functions, the optimal allocation policy satisfies the monotonicity property, which implies that the optimal policy follows a step function. The experimental results of Experiments 5–7 for the limited resource allocation model and the fully flexible resource allocation model are shown in Figures 6.3 and 6.4, respectively.

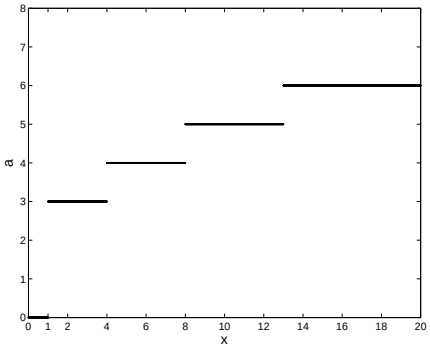
Note that in the experiments we did not consider the service time requirement that is imposed in the system. However, the optimal policy under minimal costs when meeting the service requirement still has the same structural properties. The only difference with the results presented here is that this policy will be randomized in exactly one state (see Altman [10]).



(a) Experiment 5

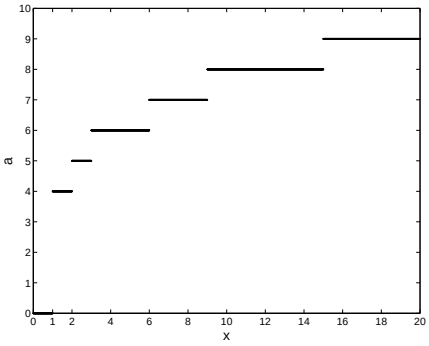


(b) Experiment 6

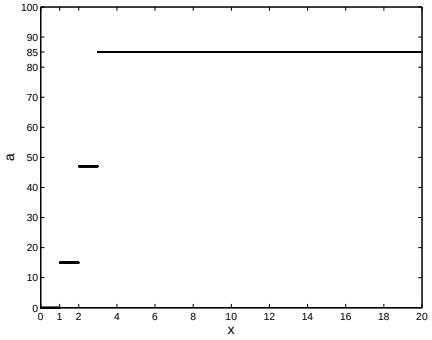


(c) Experiment 7

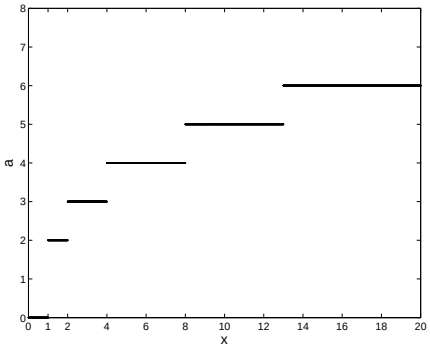
Figure 6.3. Optimal action a as a function of x for the limited resource allocation model.



(a) Experiment 5



(b) Experiment 6



(c) Experiment 7

Figure 6.4. Optimal action a as a function of x for the fully flexible resource allocation model.

6.5 Conclusion and further research

In this chapter we have explored the structural properties of the optimal allocation policy in two different systems. Both systems are capable to change the number of computing resources dynamically. However, one system deals with updating the number of computing resources only upon the start of the service of a new job whilst in the other system that can be done during the service of a job. In both systems, one needs to optimize the resource-allocation costs on the one hand while satisfying a service requirement on the

sojourn time of a job on the other hand. We applied dynamic programming to show that the optimal resource-allocation policy has a work-conservation property. It follows a step function with as extreme policy the bang-bang control policy. Also, we provide the conditions under which the bang-bang control policy is optimal. The techniques to show these results are not obtained by standard induction arguments, and thus provide a foundation for studying generalized systems in which these techniques can be applied.

There are several possible trends of the research in the future. First, one may suspect that the optimal policy in the multi-queue setting may also have kind of work-conservation property and bang-bang control policy under certain circumstances. This suggests that there is space for extending the structural properties of the optimal allocation policy for multi-queue systems described in Chapter 5. Second, the work described in this chapter is part of a larger strive to bring the benefits of high-performance computing to the multimedia community. In the near future, we will test our method by implementing them in state-of-the-art multimedia applications.

Chapter 7

Dynamic Resource Allocation for Systems with Time-varying Arrivals

In Chapter 5, we studied a resource-allocation problem for multi-queue systems with homogeneous Poisson arrivals. In this chapter, we extend the problem to the situation with time-varying arrival rates. Determining the optimal policy in this case is complicated for the following reasons: (1) each application imposes different QoS-constraints, (2) each application has its specific parameters that may vary over time, and (3) the allocation should be adjusted in real-time to properly respond to changing environments. This raises the need for dynamic policies such that the strict response-time requirements are met with a limited number of compute processors in the server pool.

We solve the problem by casting it as Markov decision problem, as we did in the previous chapters. In the case that the time-varying arrival rates are known beforehand, the optimal policy is numerically obtained. In the other case, we use both a prediction method and a stochastic approximation method to track the time-varying parameters to obtain near-optimal policies. Extensive experimental validation on a simulated distributed system shows that our techniques are highly effective.

7.1 Introduction

As stated in Chapter 1, the methods for multimedia data applications, such as iris and fingerprint recognition systems, need high-resolution scans and data processing to identify individuals in groups that are under surveillance.

This kind of applications normally has very strict response-time requirements, and hence, require real-time control of the system. To meet the strict time requirements, a good solution is to distribute the data and computation over compute nodes that are interconnected over a network.

At the server side the number of compute nodes is limited and all of these nodes are shared by multiple applications. For different applications, the operating costs may not be the same, the service-time constraints (i.e., a maximum to the average time that a job spends in the system) may be different, and the job arrival rates for the applications may vary over time. Therefore, there is an urgent need for methods that provide an optimal allocation policy such that the time constraints of all applications can be met whilst the utilization costs are minimized. Also, the method should be easily adaptable to the dynamic changes in the time-varying arrival rates in the distributed environment and be effective in terms of time duration for generating the optimal allocation policy because the allocation of resources should be adjusted in real-time. Besides that, researchers in the multimedia content analysis (MMCA) domain generally require that the optimal allocation method should also be simple and easily implementable.

There are various ways to extend the resource allocation models that have been studied in the literature to deal with time-varying arrivals. One way is to solve the Chapman-Kolmogorov forward equations (see [49, 118]). This is done by approximating the varying parameters by small, discrete intervals and use the randomization method (see [38]) to explicitly calculate the change in system occupancy from one interval to the next. Another way is the point-wise stationary approximation (PSA) of [39] that reduces the interval of changes over which a stationary measure is applied. The PSA, however, does not explicitly consider non-stationary behavior that may be induced by abrupt changes in the arrival rate, and it appears to perform less well in these cases. In [48] the accuracy and computational requirements of a number of approaches, including the exact calculation of the Chapman-Kolmogorov forward equations and the method of randomization have been evaluated. The results show that the method of randomization generally produces results that are close to exact, however the computation is quite burdensome. The PSA method is quicker but more approximate. The computational complexity increases and accuracy diminishes when the approximation methods to control the system are combined with methods to deal with time-varying parameters making the problem numerically intractable.

In this chapter, we study two different cases of the problem. The first case considers the optimal allocation problem with full knowledge of the job arrival rates to the applications. Therefore, there is no need to estimate the arrival rate. This is the simplest situation, in which we cast the problem as a Markov Decision Problem (MDP). The optimal policy can be derived numerically from the optimality equations. That policy satisfies all time constraints of the applications and minimizes the average costs. The second case considers the optimal allocation problem without full knowledge of the job arrival rates, they are unknown beforehand. The simplest approach is then to allocate a fixed number of processors to the applications irrespective of the job arrival rates. This approach is quite simple to implement in real systems. However, the average operating costs may be too high in order to meet the QoS or time-constraints as compared to smarter dynamic methods. A smarter way to solve the problem with unknown arrival rates is a method that consists of two steps. First, we use a prediction method to estimate the value of the arrival rate based on historical information. Then, we derive and apply the optimal allocation policy corresponding to the estimated arrival rate using the MDP formulation. However, a drawback of this approach is that a sudden increase or decrease in the arrival rate can only be perceived afterwards even when a very accurate prediction method is applied. This becomes worse if the arrival rate fluctuates significantly and quickly over time, which would result in constant underestimation or overestimation of the arrival rate. By applying the policy based on the underestimated arrival rate means that jobs are getting a longer waiting time, while for the overestimation of the arrival rate this means that too many compute nodes are allocated, which comes at high allocation costs. To alleviate this problem, we provide a stochastic approximation algorithm that keeps adjusting the deviation between the time constraint and the (weighted) average sojourn time (i.e., the total time spend in the system for an arbitrary job) observed from previous time periods caused by the prediction model from time to time. In this way, large deviations can be prevented yielding policies that are nearly optimal.

The main added value of this work comparing to that in the previous chapters is that we propose and validate readily available algorithms that are highly relevant for practical applications fed by inherently time-varying job-arrival processes.

The chapter is organized as follows. The formulation of the resource-allocation problem is presented in Section 7.2. The solution of the problem in case

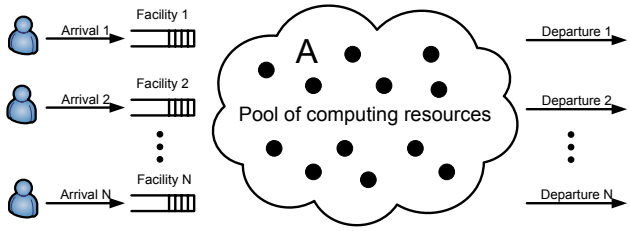


Figure 7.1. Arrivals and departures of the model.

of homogeneous arrival rates and time-varying arrival rates is presented in Section 7.3 and Section 7.4, respectively. In Section 7.5 we validate the allocation methods by numerical experiments. Finally, in Section 7.6 we make concluding remarks and address a number of challenging topics for further research.

7.2 Model formulation

To model the shared server pool with different applications running on it, we interpret each application as a facility to which jobs arrive requesting, e.g., an iris or fingerprint scan. Therefore, let the system with N facilities (applications) be depicted as in Figure 7.1 and assume that jobs arrive there according to a non-homogeneous Poisson process. The arrival rate to facility $i = 1, \ldots, N$ is denoted by $\lambda_i(t)$. Figure 7.2 gives an example of the pattern of the arrival rate function. This pattern of the arrival rate function reflects the pattern observed in practical systems, in which the highest load to the system is observed during working hours.

Upon serving a job, the image of the iris or fingerprint of a job is collected. Thereafter, it is processed by a cluster of compute nodes in parallel, provided by a pool of $A \geq 1$ compute resources. The total number of compute nodes that can be allocated among all N facilities is thus limited by the number A. If a job arrives to an empty facility, the job is taken into service immediately. Otherwise, the job waits in a buffer of infinite size. Upon a service completion at a facility, the longest waiting job is taken out of the buffer and starts its service. A decision maker decides on how to allocate the compute resources among the facilities. If a_i resources have been allocated to facility i, the service rate is a function of the number of compute

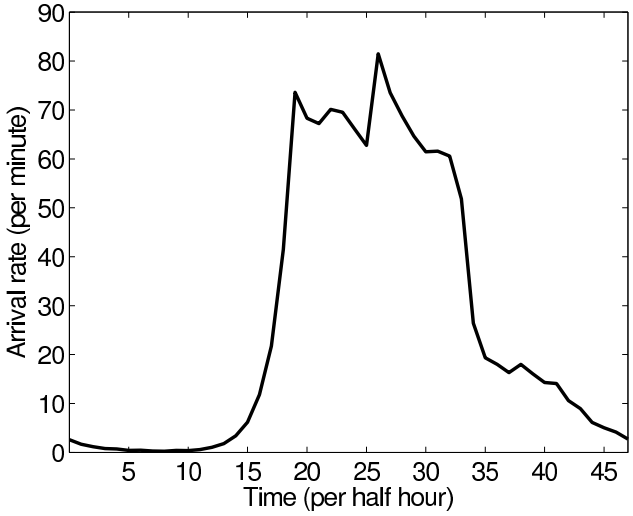


Figure 7.2. Arrival rate varies over time.

resources, represented by $\mu(a_i)$, where $\mu(\cdot)$ is an increasing function. After processing the images, the job leaves the system. Using a_i resources for facility i costs $c_i(a_i)$, and we assume that $c_i(a_i)$ is increasing in a_i for all i. Furthermore, each facility provides a QoS-guarantee on the expected delay of the jobs served at that facility. This is referred to as the time constraint. Let S_i denote the sojourn time (i.e., the sum of the waiting time and the processing time) of an arbitrary job at facility i. Then, facility i should satisfy the constraint that $\mathbb{E}S_i \leq \alpha_i$ for some predefined value of the delay criteria α_i. The goal is to find a method that can adjust the allocation policy dynamically according to changes in the arrival rates so as to satisfy all delay criteria for the N facilities whilst minimizing the average costs. We make the assumption that the total number of the compute resources A is large enough to obtain at least one policy that satisfies all time constraints.

In the sequel we consider two distinct situations of the allocation problem stated in Chapter 5. In the first situation, the decision maker can only decide on how to allocate the resources upon taking the job into service. It is called as the "limited resource sharing" case. This is typically the case in systems where resources need to be reserved in advance. The second case is more flexible, in which the system can obtain full benefits of the economies of scale by adjusting the resource allocation even during the service time of a job.

This is referred to as the "flexible resource sharing" case. For both cases, we model the resource-allocation problem as a Markov decision problem in which the problem with a constant arrival rate (a homogeneous Poisson process) is already studied in Chapter 5 that would be used as preparation for the more complex problem dealing with time-varying arrival rates (a non-homogeneous Poisson process).

7.3 Optimal allocation policy for homogeneous Poisson arrival processes

In this section, we recall the model described in Section 5.2 for the resource-allocation problem under the limited resource sharing case and the flexible resource sharing case, assuming that the arrivals in both cases are according to homogeneous Poisson processes. Note that $\lambda_i(t)$ denotes the arrival rate to facility i at time t. Since we now deal with a constant arrival rate, we have that $\lambda_i(t) = \lambda_i$. We start with short description of the model formulation for the case of limited resource sharing.

7.3.1 Service facilities with limited resource sharing

Define the state space $\mathcal{X} = \{(x_1, \ldots, x_N, a_1, \ldots, a_N) \in \mathbb{N}_0^N \times \mathbb{N}_0^N \,|\, 0 \leq \sum_{i=1}^N a_i \leq A\}$, where $(x, a) \in \mathcal{X}$ denotes that there are x_i jobs at facility i with a_i resources allocated to it for $i = 1, \ldots, N$. When the system is in state $(x, a) \in \mathcal{X}$ the decision maker can choose actions from the action space $\mathcal{A}_{(x,a)} = \{(b_1, \ldots, b_n) \in \mathbb{N}_0^N \,|\, \sum_{i=1}^N (a_i + b_i) \leq A \text{ and } a_i b_i = 0 \text{ for } i = 1, \ldots, N\}$, where action $b \in \mathcal{A}_{(x,a)}$ denotes the number of resources that one can allocate. The transition rates when the system is in state $(x, a) \in \mathcal{X}$ and action $b \in \mathcal{A}_{(x,a)}$ is chosen are given by

$$p\big((x,a),b,(x',b')\big) = \begin{cases} \lambda_i, & x' = x + e_i, b' = a + b \\ \mu(a_i + b_i), & x' = [x - e_i]^+, \\ & b' = a + b - a_i - b_i \\ 0, & \text{otherwise,} \end{cases}$$

for $i = 1, \ldots, N$, with e_i the zero vector with a one at the i-th entry, and $[x]^+$ the componentwise maximum $(\max\{x_1, 0\}, \ldots, \max\{x_N, 0\})$. Finally, when the system is in state $(x, a) \in \mathcal{X}$ and action $b \in \mathcal{A}_{(x,a)}$ has been chosen,

the direct costs $c\big((x,a),b\big) = \sum_{i=1}^{N} c_i(a_i + b_i)$. The quadruple $(\mathcal{X}, \mathcal{A}, p, c)$ completely describes the Markov decision problem.

Define a decision rule $\pi_{(x,a)}$ as a probability distribution on $\mathcal{A}_{(x,a)}$. The objective is to find a policy π^* that minimizes the long-term average costs while meeting the time constraints, thus

$$\min_{\pi} g(\pi) \quad \text{subject to} \quad \mathbb{E}S_i \leq \alpha_i \text{ for } i = 1, \ldots, N.$$

Due to Little's Law, the constrained Markov decision problem can be rewritten as an unconstrained Markov decision problem. To this end, we uniformize the system. Therefore, assume that the uniformization constant $\sum_{i=1}^{N} \lambda_i + N\mu(A) = 1$. Let $V(x,a)$ be a real-valued function defined on the state space. The long-term average optimal actions are a solution of the optimality equation (in vector notation) $g + V = TV$, where T is the dynamic programming operator acting on V defined as follows

$$\begin{aligned} TV(x,a) = &\sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} c_i(a_i) + \sum_{i=1}^{N} \lambda_i H(x + e_i, a) \\ &+ \sum_{i=1}^{N} \mu(a_i) H([x - e_i]^+, a - a_i e_i) \\ &+ \Big(1 - \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \mu(a_i)\Big) V(x,a), \end{aligned} \tag{7.1}$$

where τ_i are Lagrange multipliers, and the function H is given by

$$H(x,a) = \min_{b \in \mathcal{A}_{(x,a)}} \{V(x, a+b)\}.$$

Alternatively, the optimality equation can be solved recursively. The backward recursion equation is given by

$$\begin{aligned} V_{n+1}(x,a) = &\sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} c_i(a_i) + \sum_{i=1}^{N} \lambda_i H_n(x + e_i, a) \\ &+ \sum_{i=1}^{N} \mu(a_i) H_n([x - e_i]^+, a - a_i e_i) \\ &+ \Big(N\mu(A) - \sum_{i=1}^{N} \mu(a_i)\Big) V_n(x,a), \end{aligned} \tag{7.2}$$

where the function H_n is given by

$$H_n(x,a) = \min_{b \in \mathcal{A}_{(x,a)}} \{V_n(x, a+b)\}.$$

The algorithm to solve the optimality equation through recursion on V_n is known as value iteration (see [73]).

7.3.2 Service facilities with flexible resource sharing

In the case of service facilities with flexible resource sharing, the state space is given by $\mathcal{X} = \mathbb{N}_0^N$, where $x \in \mathcal{X}$ denotes that there are x_i jobs at facility i for $i = 1, \ldots, N$. The action space is given by $\mathcal{A}_x = \{a \in \mathbb{N}_0^N \,|\, 0 \leq \sum_{i=1}^N a_i \leq A\}$, where action $a \in \mathcal{A}_x$ denotes the number of resources that one can allocate in state $x \in \mathcal{X}$. The transition rates when the system is in state $x \in \mathcal{X}$ when action $a \in \mathcal{A}_x$ is chosen are given by

$$p(x,a,x') = \begin{cases} \lambda_i, & x' = x + e_i \text{ for } i = 1, \ldots, N, \\ \mu(a_i), & x' = [x - e_i]^+ \text{ for } i = 1, \ldots, N, \\ 0, & \text{otherwise.} \end{cases}$$

Finally, when the system is in state $x \in \mathcal{X}$ and action $a \in \mathcal{A}_x$ has been chosen, the direct costs are given by $c(x,a) = \sum_{i=1}^N c_i(a_i)$. The quadruple $(\mathcal{X}, \mathcal{A}, p, c)$ completely describes the Markov decision problem for this case. As in the previous case, the goal is to find a policy π^* that minimizes the long-term average costs under the time constraints. Let $V(x)$ denote the relative value function in this case. Then, the dynamic programming operator acting on V is defined as follows:

$$TV(x) = \sum_{i=1}^N \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^N \lambda_i V(x + e_i) + \min_{a \in \mathcal{A}_x} T_a(x), \tag{7.3}$$

where $T_a(x)$ is given by

$$\begin{aligned} T_a(x) = &\sum_{i=1}^N \mu(a_i) V([x - e_i]^+) \\ &+ \Big[N\mu(A) - \sum_{i=1}^N \mu(a_i) \Big] V(x) + \sum_{i=1}^N c_i(a_i), \end{aligned}$$

and where τ_i is the corresponding Lagrange multiplier. The backward recursion to obtain the optimal actions is given by

$$V_{n+1}(x) = \sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} \lambda_i V_n(x + e_i) \; + \min_{a \in \mathcal{A}_x} T_a^n(x), \tag{7.4}$$

where $T_a^n(x)$ is given by

$$\begin{aligned} T_a^n(x) = & \sum_{i=1}^{N} \mu(a_i) V_n([x - e_i]^+) \\ & + \Big[N\mu(A) - \sum_{i=1}^{N} \mu(a_i) \Big] V_n(x) + \sum_{i=1}^{N} c_i(a_i). \end{aligned}$$

Note that the action space for the flexible resource sharing problem is an extension of the action space for the limited resource sharing problem. Instead of formulating a new MDP, we also could have used the previous formulation with a larger action space. However, our specific formulation for the flexible resource sharing problem reduces the state space significantly, and is therefore faster to solve for the optimal allocation policy.

7.3.3 Randomized policy for a constant arrival rate

In the case of limited resource sharing and flexible resource sharing, by solving the corresponding optimality equations recursively, the optimal policy that minimizes the costs can be obtained for a fixed vector τ. When the i-th component of τ increases, the value of $\mathbb{E}S_i$ decreases. Therefore, there exists a vector τ^* such that $\mathbb{E}S_i^{\tau^*} > \alpha_i$ for all facilities and there exists another τ' such that $\mathbb{E}S_i^{\tau'} < \alpha_i$, where $\tau_i' = \tau_i^* + \varepsilon_i$ for all i and $\varepsilon_i > 0$ and small. Note that we can obtain the expected sojourn time at facility i for a given policy by setting $c(a) = 0$ (or $c(x, a) = 0$) for all actions a in the optimality equations and $\tau_j = 0$ for all $j \neq i$ and $\tau_i = 1$. The optimal policy is to randomize between the associated policies π^{τ^*} and $\pi^{\tau'}$ such that the equality $\mathbb{E}S_i = \alpha_i$ is achieved for each facility.

To randomize between two policies π^{τ^*} and $\pi^{\tau'}$, for which $ES_{\tau^*} > \alpha$ and $ES_{\tau'} < \alpha$, we introduce the randomization factor p, i.e., with probability p one chooses policy π^{τ^*} and with probability $1 - p$ one chooses policy $\pi^{\tau'}$. The probability p is chosen such that $\mathbb{E}S$ approaches α. The algorithm to determine p is described below.

```
Step 1.  Let step_size := 0.01.
Step 2.  Let p := 1.
Step 3.  If p ≥ 0, then calculate the average
         sojourn time for all facilities.
Step 4.  If ∃i such that 𝔼S_i > α_i, then let
         p := p − step_size and goto Step 3.
Step 5.  Return p and stop.
```

Algorithm 7.1. Pseudo code for generating randomization factor p.

For a fixed value of p, the average sojourn time at facility i in case of a service facility with limited resource sharing can be derived by the following equation:

$$\begin{aligned} TV(x,a) = \tau_i \frac{x_i}{\lambda_i} &+ p \sum_{i=1}^{N} \lambda_i H^*(x+e_i,a) \\ &+ (1-p) \sum_{i=1}^{N} \lambda_i H'(x+e_i,a) \\ &+ p \sum_{i=1}^{N} \mu(a_i) H^*([x-e_i]^+, a-a_i e_i) \\ &+ (1-p) \sum_{i=1}^{N} \mu(a_i) H'([x-e_i]^+, a-a_i e_i) \\ &+ \Big(1 - \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \mu(a_i)\Big) V(x,a), \end{aligned} \tag{7.5}$$

where

$$H^*(x,a) = V\big(x, a + \pi^{\tau^*}(x,a)\big),$$

$$H'(x,a) = V\big(x, a + \pi^{\tau'}(x,a)\big).$$

The average sojourn time in case of a service facility with flexible resource sharing by a given p can be calculated by the following equation:

$$TV(x) = \sum_{i=1}^{N} \tau_i \frac{x_i}{\lambda_i} + \sum_{i=1}^{N} \lambda_i V(x+e_i) + pT^*(x) + (1-p)T'(x), \tag{7.6}$$

where

$$T^*(x) = \sum_{i=1}^{N} \mu\big(\pi^{\tau^*}(x) \cdot e_i\big) V([x - e_i]^+) + \Big[1 - \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \mu\big(\pi^{\tau^*}(x) \cdot e_i\big)\Big] V(x),$$

and

$$T'(x) = \sum_{i=1}^{N} \mu\big(\pi^{\tau'}(x) \cdot e_i\big) V([x - e_i]^+) + \Big[1 - \sum_{i=1}^{N} \lambda_i - \sum_{i=1}^{N} \mu\big(\pi^{\tau'}(x) \cdot e_i\big)\Big] V(x).$$

Note that the algorithm described above is usable for a constant arrival rate. In the next section we extend this algorithm to the situation with time-varying arrival rates.

7.4 Extension to time-varying arrival rates

The method described in the previous section generates an optimal allocation policy for the problem in which the arrival rates to all facilities are constant over time. In this section, we show how to extend the method to the case of time-varying arrival rates. We study the problem of time-varying arrival rates from two different points of view. In the first case, full information on the time-varying arrival rates is known beforehand. On the contrary, in the second case we have no information on the arrival rates. Therefore, it should be estimated over time. We study for both points of view how to derive optimal allocation policies that are suitable for both the service facilities with limited resource sharing and flexible resource sharing.

7.4.1 The case of full information on the time-varying arrival rates

First, we focus on the optimal allocation policy for the situation in which we have full information on the time-varying arrival rates. In this case, it is not

necessary to predict (or keep track of) the actual arrival rate. Algorithm 7.1 described in Section 7.3.3 provides an optimal allocation policy for a homogeneous Poisson arrival process. In this section, we extend this algorithm so that it is suitable to generate an optimal policy in case that the arrival rate varies over time.

The idea is to generate optimal randomized policies for a set of different values for the arrival rates. Based upon the current arrival rate, we control the system according to the optimal policy belonging to the corresponding constant arrival rate. In order to control the system online, the decisions on how to allocate the compute processors should be as fast as possible. Since generating the optimal policy for a constant arrival rate by solving the optimality equations is time consuming, we do not want this to happen online, but offline, and store the policies into memory. Upon a change of the current arrival rate, we take the corresponding optimal policy out of the memory, and apply the corresponding action upon a decision moment. The algorithm to determine the optimal policies for a constant arrival rate is already mentioned in the previous section, but in Algorithm 7.2 we recall the different steps.

For each arrival rate combination λ do
   Step 1. Solve the optimality equations and obtain the optimal policy for τ^* and $\tau^* + \varepsilon$ such that $\mathbb{E}S_{\tau^*} > \alpha$ and $\mathbb{E}S_{\tau^*+\varepsilon} < \alpha$ for a small $\varepsilon > 0$.
           Denote the two policies by $\pi_\lambda(\tau^*)$ and $\pi_\lambda(\tau')$, respectively.
   Step 2. Calculate the randomization factor p using Algorithm 7.1.
   Step 3. Store the randomization factor p and the policies into memory.
end do.
Step 4. Upon a decision moment, apply policy $\pi_\lambda(\tau^*)$ with probability p and policy $\pi_\lambda(\tau')$ with probability $1 - p$.

Algorithm 7.2. Pseudo code for randomization between two policies.

To apply our algorithm, a day of 24 hours is divided into 48 intervals of 30

minutes as shown in Figure 7.2. Note that the solution to the optimality equations is for the steady-state situation. By dividing the day into 48 intervals, with each having a possibly different λ, we assume that the system behaves as in the steady-state situation in each interval. However, this is normally not the case in practice since for the first minutes in each interval the system does not behave as in steady-state. However the steady state is reached very quickly due to the structure of the policy and the small changes in the arrival rate from interval to interval. Thus, applying the optimal policy for a fixed λ yields very good results. This approach is also used in the field of call centers, where the SIPP (stationary, independent, period by period) approach assumes that each period of the day is independent of other periods, and the arrival process is considered to be stationary in that period (see, e.g., [40]).

7.4.2 The case of no information on the time-varying arrival rates

In this section, we focus on how to find an optimal way to allocate resources in the case that information about the time-varying arrival rate is not known beforehand. A simple way to deal with this problem is to use a prediction algorithm. Once we have an estimate for the current arrival rate, we can take the corresponding policy and apply actions upon a decision moment. The way to derive the optimal policies is the same as in Section 7.4.1. The only difference here is that we do not have the actual arrival rate, but an estimate. This method is referred to as the (traditional) "predictive method" in the sequel.

Applying the policy corresponding to the predicted arrival rate has several difficulties. First of all, one wants to update the prediction of the arrival rate regularly to be able to track rapid increases or decreases in the arrival rates, but this is time consuming. Second, an underestimate or overestimate of the arrival rate will lead to a different policy that cannot guarantee the time constraints or that yields very small sojourn times that comes with additional costs. We shall give an example in Section 7.5 that illustrates this. To cope with both issues, we need an allocation algorithm that is not only able to work efficiently (in terms of computation time), but is also capable to react quickly to underestimates or overestimates of the arrival rate. We do this by combining a prediction algorithm with a stochastic approximation approach, as described in Algorithm 7.3 below, and this is referred to as the "adapted prediction method".

Step 1. Let λ_{min} and λ_{max} be the minimum and maximum arrival rate, respectively. We set $\lambda_{min} = 0$, and discretize the possible values between λ_{min} and λ_{max} by steps of size λ_u, i.e., we obtain a vector λ with $\lambda_i = \lambda_{min} + (i-1) \cdot \lambda_u$ as the i-th entry ($i \geq 1$). For every λ_i compute the optimal policy using the algorithm described in Section 7.4.1 and keep the results in the background.

Step 2. Upon arrival of a job, we predict the arrival rate by taking the average of the observed interarrival times, and denote it by $\hat{\lambda}$.

Step 3. Upon departure of a job we calculate the average sojourn times $\mathbb{E}S_i(j)$ for each facility in time period j. Recall that we divide a day into 48 periods of length 30 minutes. We then take the observed weighted average sojourn time $\mathbb{E}\bar{S}_i$ for each facility $i = 1, \ldots, N$, with the weighting factors given by the number of jobs arrived within each period of 30 minutes for each facility, denoted by $N(i,j)$. Thus $\mathbb{E}\bar{S}_i = (\sum_j N(i,j)\mathbb{E}S_i(j))/(\sum_j N(i,j))$.

Step 4. We periodically check if $\mathbb{E}\bar{S}_i < \alpha_i - \varepsilon_i$ or $\mathbb{E}\bar{S}_i > \alpha_i$, for small ε_i. If so, then let $a_i = 1 + (\mathbb{E}\bar{S}_i - \alpha_i)/\alpha_i$, else let $a_i = 1$.

Step 5. We apply the optimal policy corresponding to λ_{i+1} if $a \cdot \hat{\lambda}_i$ is in the interval $[\lambda_i, \lambda_{i+1}]$.

Algorithm 7.3. Adapted prediction model: stepwise approach.

In the algorithm, Step 1 is used to store a set of optimal policies in memory. Step 2 uses an arrival event to update the estimate of the arrival rate; here we use the exponential smoothing method. Steps 3 and 4 are concerned with the stochastic approximation approach, in which the expected sojourn times are compared to the time constraints α. A positive difference between $\mathbb{E}\bar{S}_i$ and α_i implies that the time constraint is not satisfied, and hence the system should be controlled using a policy corresponding to a higher arrival rate. If the difference is negative, this means that there are too many compute processors allocated so that it is better to control the system according to a policy that belongs to a lower value of the arrival rate. Step 5 describes which policy to choose.

7.5 Numerical results

In this section, we first validate Algorithm 7.2 that is used for generating the optimal allocation policy in case of time-varying arrival rates with full information. Thereafter, we validate the allocation methods for the case that full information is not available. We compare the fixed allocation method, the traditional prediction method, and our adapted prediction method with each other. To this end, we simulate a scenario of the flexible resource sharing problem for two different systems; one system with only one facility, and one system with two facilities. In both systems, the arrival rate to a facility changes every 30 minutes according to the graph shown in Figure 7.2. The other parameter values are set as follows:

- A: total number of resources, set to 35;
- μ: service rate per unit, set to 25;
- $\mu(a)$: service rate function, set to $\sqrt{a}\mu$;
- $c(a)$: cost function, varied over $\{a, a^{3/2}, a^2\}$;
- α: time constraint for each facility, set to 0.04 minutes.

7.5.1 Experimental results for a system with one facility

In this section, we show the experimental results for a system with one facility. To compare the results of the implementable techniques of assigning

	Assigned number of resources	Average sojourn time	Average costs
$c(a) = a$	15	0.031	4.13
	14	0.035	3.99
	13	0.042	3.84
$c(a) = a^{3/2}$	15	0.031	16.00
	14	0.035	14.91
	13	0.042	13.84
$c(a) = a^2$	15	0.031	61.95
	14	0.035	55.81
	13	0.042	49.93

Table 7.1. Performance for the fixed allocation policy.

resources to a job, first we are going to observe the average costs and the average sojourn time in the case of allocating a fixed number of resources. Table 7.1 shows the results for the fixed allocation policy.

Note that the time constraint is set to $\alpha = 0.04$. In order to satisfy the constraint, we should use at least 14 resources. If we allow to randomize between two policies, then we obtain slightly better results. The experienced average sojourn time under the fixed policy with 13.23 resources happens to be approximately 0.04 (actually it was 0.0399). The average costs for the three cost functions are 3.87, 14.12, and 51.23, respectively. These results are used to compare the performance derived by the prediction method and our adapted prediction method.

7.5.1.1 Validation of the allocation method for time-varying arrival rates with full information

Algorithm 7.2 provides a solution for obtaining the optimal allocation policy for time-varying arrival rates with full information. By using the policy in our simulation, we obtain the optimal average costs for three different cost functions. The results are shown in Table 7.2. The corresponding average sojourn time equals 0.04, which is found by simulation. The MDP formulation for the problem gives the same results, and therefore we conclude that the simulation and the MDP formulation correspond to each other. Furthermore, for a fixed value of λ the derived policy is also the optimal policy.

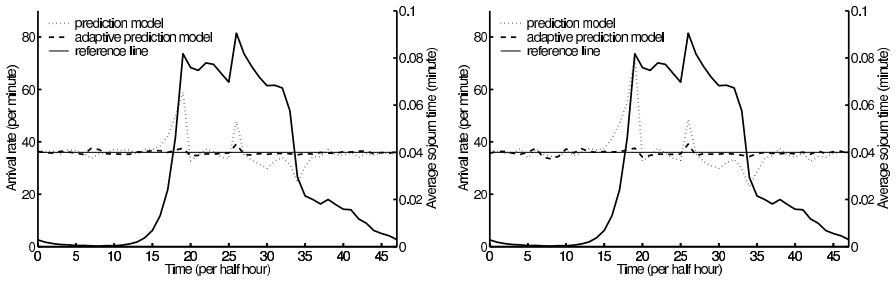
	$c(a) = a$		$c(a) = a^{3/2}$		$c(a) = a^2$	
Algorithm	$\mathbb{E}S$	$\mathbb{E}U$	$\mathbb{E}S$	$\mathbb{E}U$	$\mathbb{E}S$	$\mathbb{E}U$
real optimum	0.040	3.25	0.040	10.83	0.040	38.14
prediction	0.041	3.27	0.042	10.90	0.043	38.19
adapted prediction	0.0398	3.26	0.0396	10.98	0.0396	39.28

Table 7.2. Performance for the one-dimensional system.

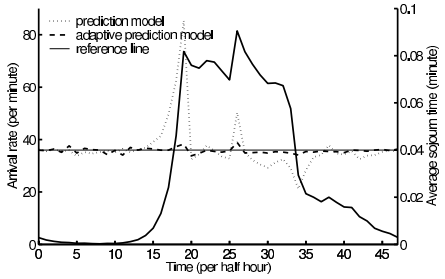
Compared to the performance obtained by using the fixed allocation policy, the relative improvements in the costs are 16%, 23%, and 26%, respectively.

7.5.1.2 Validation of the allocation method for time-varying arrival rates with no information

In case of time-varying arrival rates with no information, first we apply the optimal strategy based on the predicted arrival rate. Since it is not known if there is a trend or seasonal influence in the arrival rates, the adaptive exponential smoothing method is chosen to estimate the arrival rate. In the best case, we act as if we know that the arrival rate changes per half hour and the time of generating the optimal policy can be omitted. Then, by applying the optimal policy corresponding to the estimated arrival rate, we get that the optimal average costs for the three different cost functions are equal to 3.27, 10.90, and 38.19, respectively, as shown in Table 7.2. The corresponding average sojourn times are slightly above the time constraints (> 0.040), and the average costs in the three cases are close to the optimal case. Although the average sojourn times of the entire day are larger than the time constraints, they still do not deviate too much from the time constraints. This is related to the fact that the performance by underestimating the arrival rate compensates for the situations in which the arrival rate is overestimated, which results in a good average result for the entire day. However, if we look at the average sojourn time per half an hour as shown in Figure 7.3, we notice that the average sojourn time can be two times higher than the time constraint by underestimating and 40% lower than the time constraint by overestimating the arrival rate. By applying our model, the optimal average costs for the three different cost functions are equal to $3,26$, 10.98, and 39.28, respectively, as shown in Table 7.2. The corresponding average sojourn time of the entire day are slightly below the time constraints (< 0.040). This shows that by applying our model, the time



(a) $c(a) = a$ (b) $c(a) = a^{3/2}$



(c) $c(a) = a^2$

Figure 7.3. Experiments of the system with one facility.

constraints are met, whilst the average costs is not more than 3% higher than the optimal case. Besides that, Figure 7.3 shows that using our model, the average sojourn time of every half hour is very close to the time constraint as compared to the average sojourn time for the traditional prediction model. Therefore, we conclude that our model yields more stable results.

7.5.2 Experimental results for a system with two facilities

Now, we show the experimental results for a system with two facilities. The parameters for both facilities are set as in the beginning of this section. The arrival rate to each facility follows the same pattern as shown in Figure 7.2. Since the parameter values of both facilities are identical, the average sojourn time at both facilities by applying the prediction method and the adapted prediction method are close to each other. Therefore, in this section only the

numerical results of the average sojourn time at facility 1 are shown. First, we apply the fixed allocation policy. This policy allocates the same number of resources to both facilities. Therefore, the two facilities can be seen as two independent systems. Thus to meet the time constraint, $c = 13.23$ compute resources should be allocated to each facility as was found earlier in Section 7.5.1 (using a randomized policy). The average costs for the entire system for the three different cost functions are twice the average costs for the system with one facility. They are equal to 7.74, 28.24, and 102.46, respectively. These results are used to compare the performance of the prediction method and our adapted prediction method.

7.5.2.1 Validation of the allocation method for time-varying arrival rates with full information

In case the time-varying arrival rates are known beforehand, we get that the optimal average costs ($\mathbb{E}U$) for the three different cost functions are equal to 6.30, 20.74, and 71.10, respectively, as can be seen in Table 7.3. The corresponding average sojourn time ($\mathbb{E}S$) is 0.04. Note that the allocation method also guarantees that the average sojourn time in every time interval is equal to 0.04, and that the allocation policy derived here is also optimal. Comparing its performance to the performance by using the fixed allocation policy, the relative improvements in the costs are 22%, 36%, and 44%, respectively.

7.5.2.2 Validation of the allocation method for time-varying arrival rates with no information

In case of time-varying arrival rates with no information, first we apply the optimal strategy based on the predicted arrival rate. Then, by applying the optimal policy of the estimated arrival rate, we get that the optimal average costs for the three different cost functions are equal to 6.32, 20.73, and 70.87, respectively (see Table 7.3). The corresponding average sojourn times of the entire day are 0.040, 0.041, and 0.042, respectively. This shows that the average costs in the three cases are close to the optimal case or even slightly better than the average costs of the optimal case. However, in case of the lower average costs, the average sojourn times of the entire day are higher than the time constraint. If we look at the average sojourn time per half hour as shown in Figure 7.4, we notice that the average sojourn

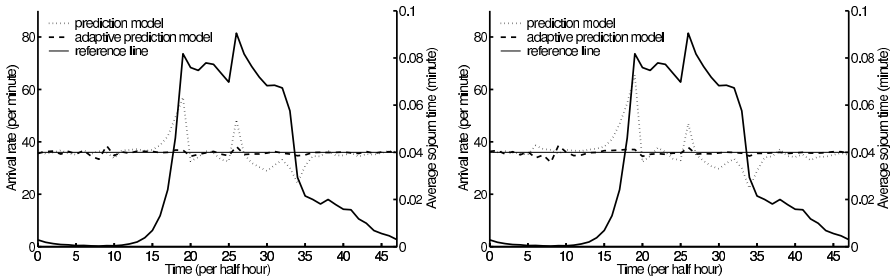
	$c(a) = a$		$c(a) = a^{3/2}$		$c(a) = a^2$	
Algorithm	$\mathbb{E}S$	$\mathbb{E}U$	$\mathbb{E}S$	$\mathbb{E}U$	$\mathbb{E}S$	$\mathbb{E}U$
real optimum	0.040	6.30	0.040	20.74	0.040	71.10
prediction	0.040	6.32	0.041	20.73	0.042	70.87
adapted prediction	0.0398	6.32	0.0398	20.96	0.0398	72.26

Table 7.3. Performance for the two-dimensional system.

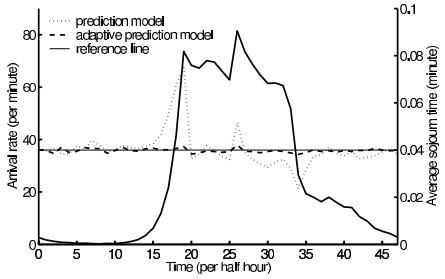
time can be much higher than the time constraint by underestimating the arrival rate and much lower than the time constraint by overestimating the arrival rate. By applying our model, the optimal average costs for the three different cost functions are equal to 6.32, 20.96, and 72.26, respectively (see Table 7.3). The corresponding average sojourn time of the entire day are all around 0.0398. This shows that by applying our model, the average constraints are met, whilst the average costs is no more than 2% higher than the optimal case. Besides that, Figure 7.4 shows that using our model, the average sojourn time of every half hour is very close to the time constraint. Hence, we conclude by comparing our model to the prediction model, that the variation of our model in the average sojourn time per half hour is much smaller whilst the related average costs is very close to the optimal one.

7.5.3 Experimental results of system with two facilities using other time-varying arrival rates

Now, we are interested in the difference between the performance by applying the prediction model and our method to another pattern of the time-varying arrival rates. Therefore, we take the pattern of the time-varying arrival rates according to Figure 7.5. The cost function is set to $c(a) = a^{3/2}$. The other parameter values remain the same. In the first case, we assume that the arrival rate is known. Then we obtain that the real optimal costs for $\lambda_2(t)$ and $\lambda_3(t)$ are 8.80 and 13.00, respectively, as shown in Table 7.4. By using the prediction model, the average sojourn times for both cases are 0.0405 and 0.042, and the corresponding costs are 8.85 and 12.73, respectively. The average sojourn times per half hour are shown in Figure 7.5(a) and Figure 7.5(b). By applying our adapted model, we achieve 0.040 and 0.0398 as the average sojourn times for the two cases and the corresponding costs are 8.77 and 13.14, respectively. Based on the figure we see that by applying our adapted prediction model the variation in the average sojourn time per half hour is very small and the time constraint (average sojourn time of the



(a) $c(a) = a$ (b) $c(a) = a^{3/2}$



(c) $c(a) = a^2$

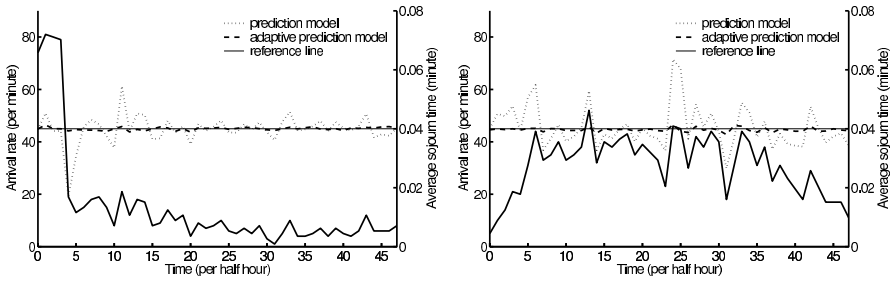
Figure 7.4. Experiments of the system with two facilities.

	λ_2		λ_3	
Algorithm	$\mathbb{E}S$	$\mathbb{E}U$	$\mathbb{E}S$	$\mathbb{E}U$
real optimum	0.040	8.77	0.040	13.00
prediction	0.0405	8.85	0.042	12.73
adapted prediction	0.040	8.77	0.0398	13.14

Table 7.4. Performance for the two-dimensional system with $c(a) = a^{3/2}$.

entire day) is met.

So far, we have shown that the performance of our model for the system with time-varying arrival rates and fully flexible resource sharing. For the case of a system with limited resource sharing, we suffice to mention that we obtain similar results.



(a) λ_2 (b) λ_3

Figure 7.5. Experiments of the system with two facilities and with different arrival rates.

7.6 Conclusion and further research

In this chapter we have studied the optimal allocation problem in case of time-varying arrival rates. We discussed two cases: a system with limited resource sharing and a system with flexible resource sharing. For both cases, we derived the optimal allocation policy under two situations; one with enough prior knowledge of the time-varying arrival rates and one without the prior knowledge.

We have compared the results of our adapted prediction method to a fixed allocation method and traditional prediction methods. We observed that the adapted prediction method can save 16% in the average cost compared to the fixed allocation method while satisfying the same time constraint in a system with one facility with a linear cost functions, and 44% in a system with two facilities with a square-law increasing cost function.

Although the average costs for the adapted prediction method are quite similar to the costs for the traditional prediction methods, we notice that traditional methods fail to satisfy the time constraints, while the adapted prediction method does meet the time constraints. Furthermore, for different arrival rate patterns, the variation in average sojourn time of the adapted prediction method is much smaller than the variation of the traditional methods.

Finally, we conclude that the adaptive prediction method is very effective and outperforms traditional methods, and moreover, this method is simple

and easy to implement.

Based on the results presented in this chapter, a number of possible directions for further research can be considered. First, the evaluation of the methods in this study are based on the results obtained by computer simulations. The study can be extended by performing extensive experiments in a real Grid environment by implementing the methods described in this chapter in state-of-the-art multimedia applications. Second, in this chapter we focus on the fluctuations in the arrival rate, however, in a real Grid environment the fluctuations in processor speed may also be important. Hence, the condition of the problem can be generalized to non-exponential service times, which is practically relevant.

Chapter 8

Experimental Validation of Optimal Allocation Policies

In previous chapters, we have studied resource-allocation problems and structural properties of optimal allocation methods in different settings. In this chapter, we validate the optimal resource-allocation methods for single-queue systems in the following two models: (1) service facilities with dedicated resources, and (2) service facilities with limited resource sharing, which are discussed extensively in Chapters 5 and 6. In the first model, the number of resources allocated to each job is fixed for the duration of the application. In the second model, the number of allocated resources is allowed to change upon the start of the processing of each new job. The validation of the optimal allocation methods for both models is performed by extensive experiments on a real-world cluster system. The experimental results show that: (i) the results for the average sojourn time and the average cost measured in the application indeed converge to the results predicted by the theoretical models, (ii) the method proposed for the second model, which is adaptive to system variations, is much more effective than the method for the first model, and (iii) the optimal allocation methods for both models are simple and easily implementable, which demonstrates the practicality of these methods.

8.1 Introduction

The number of compute resources that can be applied concurrently by each separate compute service is limited, amongst others because of utilization

cost and occupation efficiency. In Chapter 5 we developed three simple, easily implementable and efficient methods to determine the optimal resource-allocation policy for three models: a model with service facilities with dedicated resources, a model with service facilities with limited resource sharing, and a model with service facilities with full flexibility in resource sharing, respectively. In the decision making, there is a trade-off between the service processing time and resource utilization costs (e.g., lease costs, operation costs, etcetera). Occupying too many resources at the server side may lead to high costs and low efficiency, in particular when the use of a low number of resources is sufficient to meet the set time constraints. Using insufficient resources, however, may cause the time constraints of the application to be violated. Hence the goal of the allocation methods is to minimizes resource utilization costs while satisfying the time constraint at the same time.

In this chapter, we validate and compare the optimal resource-allocation methods for the first two models. We assume that the resources are available for the entire duration of the application at hand and are ready for use immediately after a reservation has been made [63, 97]. In both models homogeneous sets of resources are either statically or dynamically allocated to a single compute service. Adding or removing resources in these models during a service is not allowed. In the first model, the number of resources that allocated to each job is fixed for the duration of the application. In the second model, the number of allocated resources is allowed to change upon the start of the processing of each new job. To prevent long-term overprovisioning of resources and to further reduce costs, the second model incorporates a randomization factor in determining the allocated number of nodes. Depending on the number of jobs queued, a different number of resources may be allocated. The allocation methods for both models are equally suited for application in clusters and cloud systems where resources need to be reserved in advance. The effectiveness of the allocation policies proposed for the two models is validated by extensive experiments on a state-of-the-art cluster system.

The remainder of this chapter is organized as follows. In Section 8.2 we recall briefly the allocation models in case of service facilities with dedicated resources and service facilities with limited resource sharing (see Chapters 5 and 6 for more details). The optimal allocation methods for both models are also presented shortly in this section. Section 8.3 presents and discusses the validation results. Finally, in Section 8.4 we present our conclusions and address topics for future research.

8.2 Model formulation

Consider a service facility at which jobs arrive according to a Poisson process with rate λ. There is a common pool of $A \geq 1$ resources to serve the jobs in the system. When upon arrival of a job at the facility there are no other jobs present, the arriving job is taken into service directly. However, if there are other jobs present, the arriving job joins an infinite-sized queue at the facility and awaits its service in a First-Come First-Served (FCFS) manner. When the facility has been allocated a resources, a job that is being serviced has a service duration that is exponentially distributed with parameter $\mu(a)$, where $\mu(.)$ is an increasing function. In the ideal case one would have $\mu(a) = \mu a$ for some fixed service rate μ. However, in practice, there is communication between the resources, to the effect that the function μ is typically sublinear. In some cases, caching effects may cause the function μ to be superlinear. After a job has been serviced, it leaves the system. See also Figure 8.1.

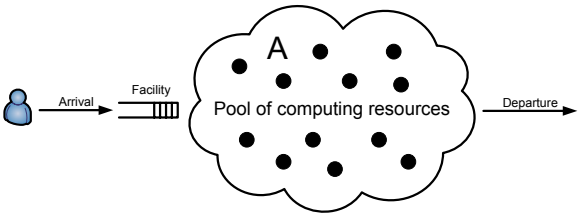


Figure 8.1. Arrival and departure of jobs.

The facility provides a QoS-guarantee on the delay of the jobs served at that facility. Let S denote the sojourn time of an arbitrary job at the facility. Then the facility is constrained by $\mathbb{E}S \leq \alpha$ for a preset value of α. There is a central decision maker that needs to determine how many resources should be allocated to the facility such that the QoS-constraints are met. However, when using a resources a cost of $c(a)$ is incurred by the system with $c(.)$ an increasing function. The objective is to find an optimal policy that selects a^* resources to allocate based on the number of jobs x in the system such that the long-term average costs (i.e., utilization costs) are minimized while at the same time the service constraint is met.

8.2.1 Service facilities with dedicated resources

The model in which the resource allocation is decided upon beforehand and remains unchanged for the duration of the application turns the facility into

a regular $M/M/1$ queue with arrival rate λ and service rate $\mu(a)$ when a resources have been allocated. In this case it is well-known that the mean sojourn time is given by $1/\big(\mu(a)-\lambda\big)$. Hence, to ensure that there exists at least one allocation policy that meets the time constraint $\mathbb{E}S \leq \alpha$, the total number of compute resources A should satisfy the following condition,

$$A \geq \lceil \mu^{-1}\big(\lambda + 1/\alpha\big) \rceil, \tag{8.1}$$

with $\lceil x \rceil$ the smallest integer greater than or equal to x. For the $M/M/1$ queue, the average service duration by using a compute resources in parallel is given by $1/\mu(a)$, which implies that the long-run average cost is $\lambda c(a)/\mu(a)$. Therefore, the optimal number of processors to allocation A^* is given by

$$A^* = \arg\min_{a \in N} \Big\{ \frac{c(a)}{\mu(a)} \Big| \lceil \mu^{-1}\big(\lambda + 1/\alpha\big) \rceil \leq a \leq A \Big\}. \tag{8.2}$$

8.2.2 Service facilities with limited resource sharing

The single-queue model in which the decision maker is allowed to change the allocation whenever a new job is to be served is described extensively in Sections 6.2.1 and 6.3.1. In those sections, we cast the problem as a constrained Markov decision problem that can be rewritten as an unconstrained Markov decision problem using Lagrange multipliers, denoted τ. Then for a fix value of τ the optimal actions can be obtained by solving the backward recursion equation (6.3). As the Lagrange multiplier τ increases, the value of EW decreases. Therefore, there exists a τ^* such that $\mathbb{E}S^{\tau^*} > \alpha$ for the facility and there exists another τ' such that $\mathbb{E}S^{\tau'} < \alpha$, where $\tau' = \tau^* + \epsilon$ for a small $\epsilon > 0$. The optimal policy is to randomize between the associated policies π^{τ^*} and $\pi^{\tau'}$ such that $\mathbb{E}S = \alpha$ for that facility.

8.3 Experimental results

In this section we validate and compare the optimal allocation methods for the two resource-allocation models described in Section 8.2. All experiments are performed on the DAS-4 cluster, located at the VU University in Amsterdam [9].

In our experiments we use a simple *sleep* application to validate the allocation methods. The jobs in the application do not perform any processing,

but simply wait for a predetermined amount of time. Jobs arrive according to a Poisson process. The required sleep time for each arriving job is randomly assigned and has an exponential distribution. When a job is executed in parallel, the *service rate* determines the required sleep time based on the number of processors used, and a user defined speedup function. This approach allows us to experiment with different speedup characteristics. Similarly, different cost functions can be selected to reflect the cost of the processors used. The parameters used in the experiments are defined as follows:

- A: the maximum number of processors available,
- a: the number of processors in use,
- λ: the arrival rate,
- μ: the service rate of using one processor,
- $\mu(a)$: the service rate of using a processors,
- $c(a)$: the costs of using a processors,
- α: the time constraint requirement.

The parameter values used in the models are set as follows:

- $A = 8$,
- $\mu = 0.75$,
- $\alpha = 1$.

The cost function $c(a)$ and service-rate function $\mu(a)$ are varying according to Table 8.1 for $a \geq 1$. Without loss of generality, we set $c(0) = \mu(0) = 0$. In Experiment 1, both the cost function and the service-rate function are linear. In Experiment 2, both of these two functions are concave (i.e., sub-linear speedup and cost). Experiment 3 represents the situation with a concave cost function and convex service-rate function (i.e., sub-linear cost, and super linear speedup), whereas Experiment 4 represents the opposite case. For all these experiments, we vary the arrival rate as given in Table 8.1 to achieve results in low, medium and high load scenarios.

	$c(a)$	$u(a)$	λ for low load	λ for medium load	λ for high load
Experiment 1	$0.9a + 0.1$	$a\mu$	0.1	2.5	4
Experiment 2	$\sqrt{a}$	$\sqrt{a}\mu$	0.1	0.5	1
Experiment 3	$\sqrt{a}$	$a^{\frac{3}{2}}\mu$	0.1	8	15.5
Experiment 4	a^2	$\sqrt{a}\mu$	0.1	0.5	1

Table 8.1. Parameter values for the different experiments.

For Experiments 1 to 4, the optimal allocation methods derived by the first resource-allocation model are shown in Table 8.2. By applying these policies, the related average cost per time unit and the average sojourn time of a job can be calculated and are presented in Table 8.3.

	A^* for low load	$A^*$for medium load	$A^*$for high load
Experiment 1	8	8	8
Experiment 2	8	8	8
Experiment 3	8	8	8
Experiment 4	3	5	8

Table 8.2. Optimal allocation policy derived by the fixed model.

	low load		medium load		high load	
	cost	sojourn	cost	sojourn	cost	sojourn
Experiment 1	0.12	0.17	3.04	0.29	4.87	0.50
Experiment 2	0.13	0.495	0.67	0.62	1.3	0.89
Experiment 3	0.0167	0.059	1.33	0.11	2.58	0.68
Experiment 4	0.69	0.834	7.45	0.85	30.17	0.892

Table 8.3. The average costs and sojourn time when using the fixed model.

The optimal allocation methods for the second model provide strategies that randomize between two policies π^{τ^*} and $\pi^{\tau'}$. The policies for Experiments 1 to 4 are shown in Table 8.4. This table presents the number of processors to allocate when a job is started, given the queue length at that time. For brevity, the table only shows the policies up to a queue length of 10. Each strategy randomizes between two policies. The randomization factor p for each of the experiments is shown in Table 8.5.

For the first three experiments, only a single strategy is provided, albeit with a different randomization factor for each experiment. For these experiments, the strategy randomizes between the first policy given by $\pi^{\tau^*}(x, a) = 0$ for

Queue length	Experiment 1, 2, 3 all loads		Experiment 4 low load		Experiment 4 medium load		Experiment 4 high load	
	π^{τ^*}	$\pi^{\tau'}$	π^{τ^*}	$\pi^{\tau'}$	π^{τ^*}	$\pi^{\tau'}$	π^{τ^*}	$\pi^{\tau'}$
0	0	0	0	0	0	0	0	0
1	0	8	2	3	3	4	6	7
2	0	8	3	4	4	5	8	8
3	0	8	4	5	5	5	8	8
4	0	8	4	6	6	6	8	8
5	0	8	4	6	6	7	8	8
6	0	8	5	7	7	7	8	8
7	0	8	5	7	7	8	8	8
8	0	8	6	8	8	8	8	8
9	0	8	6	8	8	8	8	8
10	0	8	8	8	8	8	8	8

Table 8.4. The adaptive allocation policies.

for all (x, a) and the second policy given by $\pi^{\tau^*}(x, a) = 8$ for $x > 0$ and $a = 0$. The first policy implies that no resources are allocated for any states. The second policy implies that all $A = 8$ resources would be allocated upon the start of the processing of every new job. When the first policy is selected, no processors will be used and the current job will re-inserted into the queue, only to be de-queued again when a new job arrives. When this happens, a policy will again be chosen at random.

For the fourth experiment three different strategies are shown for low, medium and high load, respectively. As Table 8.4 shows, each strategy will apply an increasing number of processors as the queue length increases. The maximum number of processors (in this case 8) is applied sooner if the arrival rate of the jobs (e.g., the load of the system) is higher. The average costs

	p for low load	p for medium load	p for high load
Experiment 1	0.9300	0.7500	0.8700
Experiment 2	0.9600	0.9000	0.9700
Experiment 3	0.9220	0.6950	0.9750
Experiment 4	0.1900	0.8300	0.5700

Table 8.5. Randomization factor of the adaptive allocation model.

based on these randomized policies can be calculated by the second model and are shown in Table 8.6. Note that the sojourn time is not shown, as it

approaches the time constraint ($\mathbb{E}S = \alpha = 1$) for all experiments applying the adaptive policies.

	low load	medium load	high load
Experiment 1	0.12	3.04	4.87
Experiment 2	0.13	0.67	1.3
Experiment 3	0.0167	1.33	2.58
Experiment 4	0.44	5.29	24.44

Table 8.6. The average costs by using the adapted allocation model.

Comparing the average costs stated in Tables 8.3 and 8.6, we observe that the cost by using the optimal resource-allocation methods derived by the first and the second model for the first three experiments is the same irrespective of the load. However, for the last experiment, the optimal allocation method of the second model significantly outperforms that of the first model. Specifically, in comparison to the first allocation method, the second one reduces the average costs by 36% in case of low load, and 19% for high load. This observation can be explained by the fact that the first model pays no attention to system variations whilst the second one makes an allocation decision based on the system information. As the number of jobs in the system grows, or if the load of the system increases, the number of resources to allocate based on the second model would remain the same or increase. In the opposite case, it would be reduced.

Next, we proceed to validate two models by applying their policies to the *sleep* application described above. Figures 8.2 to 8.5 illustrate the experimental results of the four experiments with the low, medium and high loads, respectively. The first column of the figures (on the left) shows the average sojourn time as a function of the time. The red and blue solid lines show the application results when applying the allocation methods of the first model (red) and the second method (blue) to the *sleep* application. The read and blue dotted lines show the average sojourn time derived by the theoretical models. Similarly, the second column shows the average cost as a function of the time for the fixed and adaptive methods, both as measured in the application and as predicted by the theoretical models. Note that for the first three experiments, the costs are the same for both models. In all cases, the time unit is 1000 seconds.

These figures demonstrate that the simulation results of the average sojourn time and the average cost are approaching the results derived by the models

in steady state. For all experiments, both sojourn time and costs converge to the results provided by the model.

For the first three experiments, the resource cost is the same for both resource-allocation methods. This is explained by Tables 8.2 and 8.4. As these tables show, both the fixed and adaptive strategies always use 8 processors, regardless of the number of queued jobs. As a result, the resource-utilization cost is the same for both policies. The first allocation method provides a lower average sojourn time for each of these experiments. The second allocation method occasionally decides to postpone a job when policy π^{τ^*} is chosen (which is always 0). As a result, the sojourn time of this job (and all other jobs queued at that time) will increase, thereby also increasing the average sojourn time. In spite of that, Figures 8.2 and 8.3 demonstrate clearly that in the first two experiments the optimal allocation methods for the second model meet the time constraint. In Figure 8.4, we notice that in Experiment 3 in case of high and medium load the averages sojourn times by applying the optimal allocation methods of the second model seem slightly above the time constraint. This is due to the fact that the application has not been run long enough to reach steady state. Since the time to reach steady state is quite long and the patterns of both trend lines of the average sojourn time converge to the time constraint, we conclude that also in Experiment 3 the average sojourn time under optimal allocation methods of the second model indeed converges to the time constraint.

As expected, the optimal allocation method of the second model used in the fourth experiment does provide an advantage compared to the fixed strategy. As Figure 8.5 shows, the average cost is reduced by 19% for the high-load scenario, to 36% for the low-load scenario. Although the average sojourn time increases in all scenarios, leading to a slightly higher queue utilization, it still meets the sojourn-time constraint.

8.4 Conclusion and further research

In this chapter, we validated and compared the optimal allocation methods for two resource-allocation problems in distributed systems that require to meet the QoS time constraint on one hand, and minimize the average utilization costs on the other hand. Our validation experiments show that the average sojourn time and the average cost measured in the application are in good agreement with the prediction results obtained by the theoret-

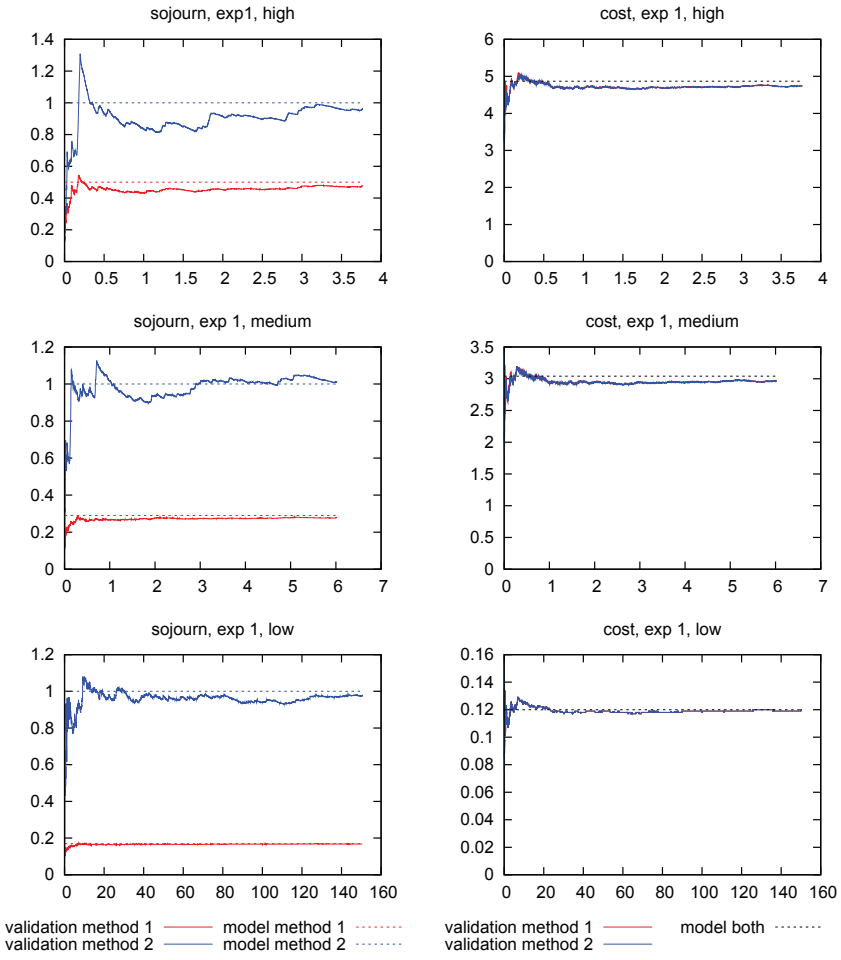


Figure 8.2. Validation results for Experiment 1.

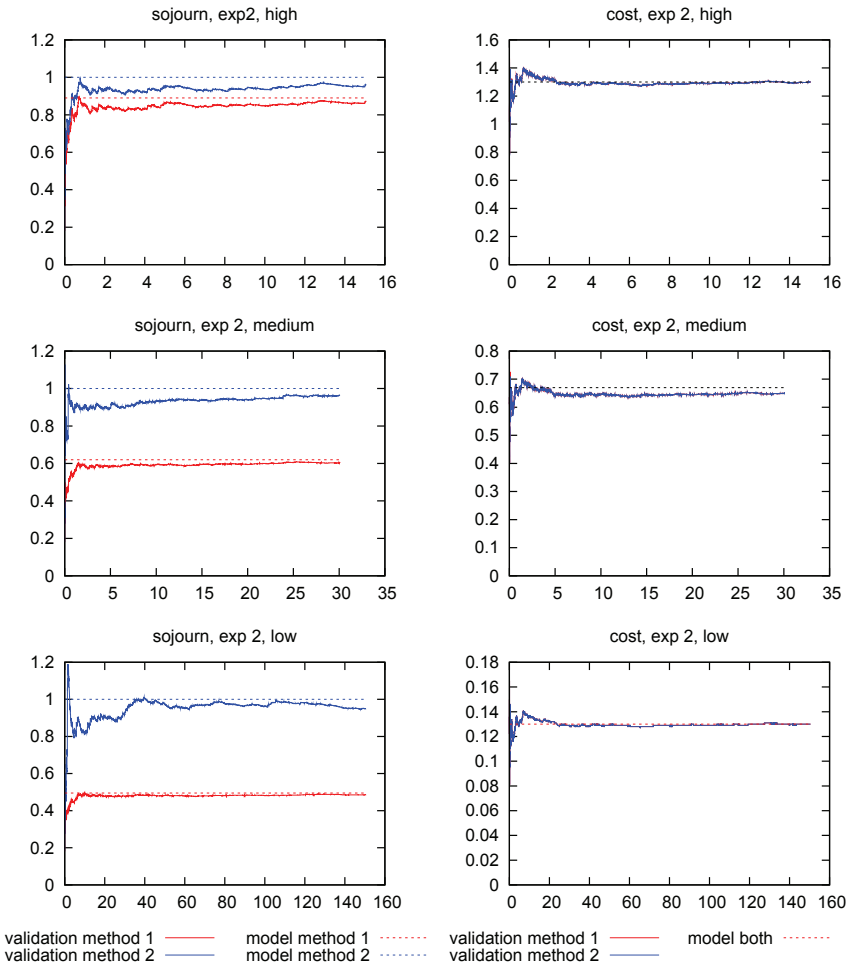


Figure 8.3. Validation results for Experiment 2.

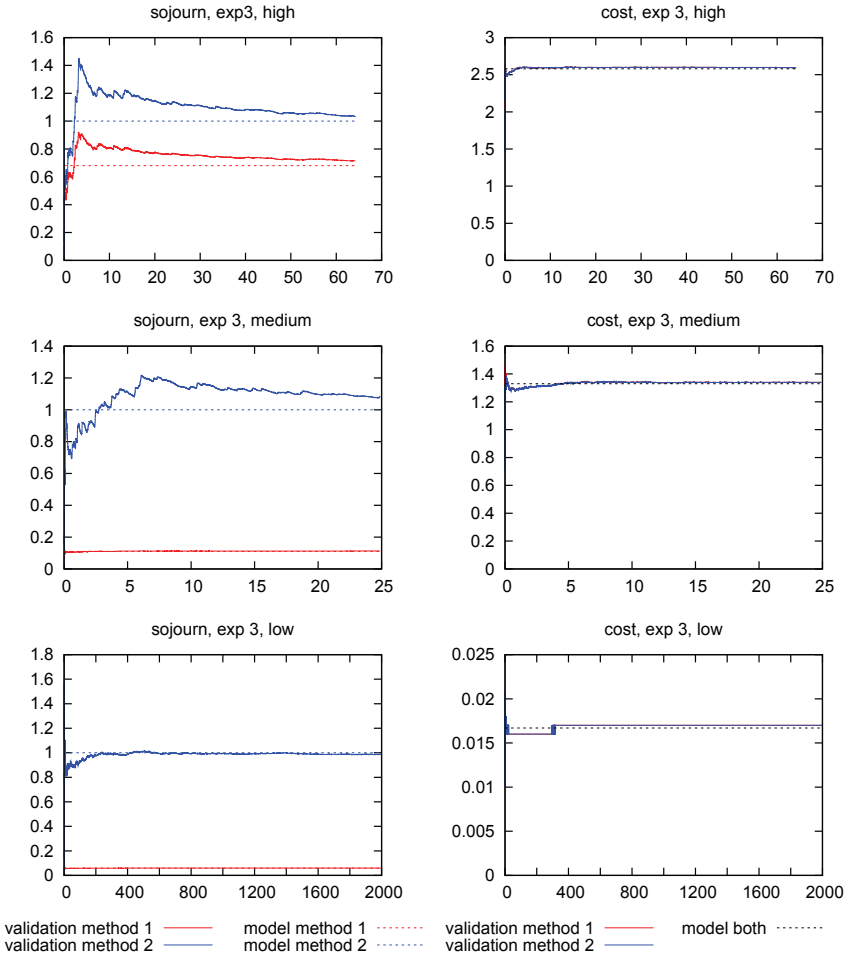


Figure 8.4. Validation results for Experiment 3.

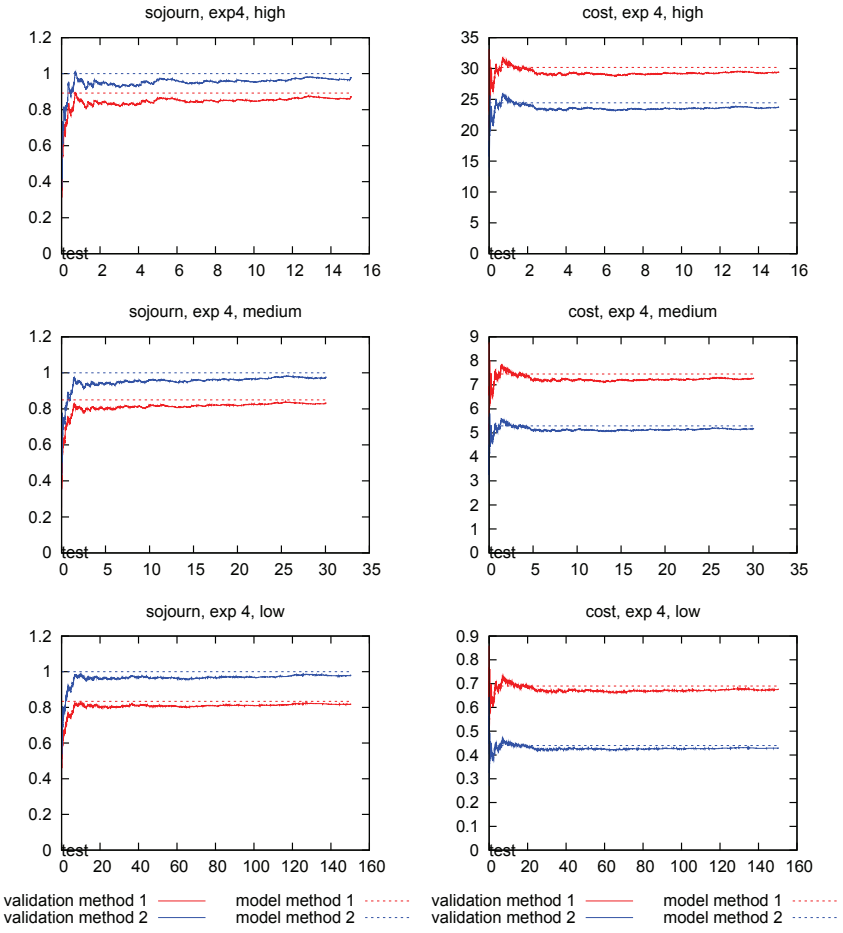


Figure 8.5. Validation results for Experiment 4.

ical models. We observed as well that the method proposed for the second model, which is adaptive to system variations, is much more effective than the method for the first model. Moreover, the optimal allocation methods for both models are simple and can be easily implemented, which demonstrates the practicality of these methods.

Based on the results presented in this chapter, a number of directions for further research can be considered. First, the model validation in this study is performed by running a *sleep* application on the DAS-4 cluster. The study can be extended by implementing the methods in more complex multimedia applications, and by validating the methods in a real large-scale Grid environment. Second, the models validated do not take the reservation time of compute resources into account (as discussed in Chapter 4). Therefore, it is interesting to know how accurate the results would be in case the reservation time is significant.

Appendix

A Proofs of Section 5.3.3

In this section we provide proofs of statements made on two cases that appear in Theorem 5.3.3. The first case deals with the situation in which $a_i < b_i$ and $a_j > b_j$ for $j \neq i$, i.e., the case in which the queue with one additional customer is allocated strictly fewer processors, whereas the other queues are potentially allocated more processors. The other case concerns $a_j < b_j$ for $j = 1, \ldots, N$, i.e., all queues, including the one with an additional customer, are allocated strictly fewer processors. In Theorem 5.3.3 the claim is that both cases cannot occur (which is intuitively plausible). In Lemma's A.1 and A.2 we show that this claim indeed holds.

Lemma A.1. *Let $a = a^*(x + e_i)$ and $b = a^*(x)$. If $a_i < b_i$ and $a_j > b_j$ for $j \neq i$ then the following properties hold:*

1. *$V(x)$ is convex in i: $V(x + e_i) - 2V(x) + V(x - e_i) > 0$,*
2. *$V(x)$ is submodular in (x_i, x_j):*
   $V(x) - V(x - e_j) - V(x + e_i - \sum_{p=1,p\neq j,p\neq i}^{N} d_p \cdot e_p) + V(x + e_i - e_j - \sum_{p=1,p\neq j,p\neq i}^{N} d_p \cdot e_p) \geq 0$ for $j \in \{1, 2, \ldots, N\} \setminus \{i\}$ and $d_p \geq 0$.

Proof. The proof is by induction on V_n. First, for $n = 0$, define $V_0(x) = \sum_{i=1}^{n} x_i^2$. Clearly, the function $V_0(x)$ is convex in x_i and submodular in (x_i, x_j). Second, assume that the two properties hold for $n = k$. Then, we proceed to prove that $V_{n+1}(x)$ is also convex in x_i and submodular in (x_i, x_j) under the conditions stated in the theorem. Let $c = a^*(x - e_i)$ and recall that $a = a^*(x + e_i)$ and $b = a^*(x)$. By the conditions of the theorem, we

have that $a_i < b_i < c_i$ and $a_j > b_j > c_j$ for all $j \neq i$. Now, the convexity follows from the following set of inequalities:

$$\begin{aligned}
&V_{n+1}(x+e_i) - 2V_{n+1}(x) + V_{n+1}(x-e_i) \\
&\quad = \Big[\sum_{l=1}^{N} \lambda_l V_n(x+e_l+e_i) - 2\sum_{l=1}^{N} \lambda_l V_n(x+e_l) + \sum_{l=1}^{N} \lambda_l V_n(x+e_l-e_i)\Big] \\
&\qquad + \Big[\min_{a\in\mathcal{A}_x}\{T_a^k(x+e_i)\} - 2\min_{a\in\mathcal{A}_x}\{T_a^k(x)\} + \min_{a\in\mathcal{A}_x}\{T_a^k(x-e_i)\}\Big] \\
&\quad = \sum_{l=1}^{N} \lambda_l \Big[V_n(x+e_l+e_i) - 2V_n(x+e_l) + V_n(x+e_l-e_i)\Big] \\
&\qquad + \Big[\min_{a\in\mathcal{A}_x}\{T_a^k(x+e_i)\} - 2\min_{a\in\mathcal{A}_x}\{T_a^k(x)\} + \min_{a\in\mathcal{A}_x}\{T_a^k(x-e_i)\}\Big] \\
&\quad > \min_{a\in\mathcal{A}_x}\{T_a^k(x+e_i)\} - 2\min_{a\in\mathcal{A}_x}\{T_a^k(x)\} + \min_{a\in\mathcal{A}_x}\{T_a^k(x-e_i)\}.
\end{aligned}$$

The inequality holds because the first expression between the brackets is non-negative due to the induction hypothesis. Now, by expanding the operator T, we derive

$$\begin{aligned}
&V_{n+1}(x+e_i) - 2V_{n+1}(x) + V_{n+1}(x-e_i) \\
&\quad > T_a^k(x+e_i) - T_a^k(x) - T_c^k(x) + T_c^k(x-e_i) \\
&\quad \geq \Big[\sum_{l=1}^{N} c_l(a_l) - \sum_{l=1}^{N} c_l(a_l) - \sum_{l=1}^{N} c_l(c_l) + \sum_{l=1}^{N} c_l(c_l)\Big] \\
&\qquad + \Big[\sum_{l=1}^{N} \mu(a_l) V_n(x-e_l+e_i) - \sum_{l=1}^{N} \mu(a_l) V_n(x-e_l) \\
&\qquad - \sum_{l=1}^{N} \mu(c_l) V_n(x-e_l) + \sum_{l=1}^{N} \mu(c_l) V_n(x-e_l-e_i)\Big] \\
&\qquad + \Big[\Big(N\mu(A) - \sum_{l=1}^{N} \mu(a_l)\Big) V_n(x+e_i) - \Big(N\mu(A) - \sum_{l=1}^{N} \mu(a_l)\Big) V_n(x) \\
&\qquad - \Big(N\mu(A) - \sum_{l=1}^{N} \mu(c_l)\Big) V_n(x) + \Big(N\mu(A) - \sum_{l=1}^{N} \mu(c_l)\Big) V_n\Big(x-e_i\Big)\Big].
\end{aligned}$$

Note that the first expression between the brackets is equal to 0 (with a slight abuse of notation, where we use $c_l(\cdot)$ for the cost function, and c_l as

variable for the optimal allocation). Now, we rearrange the terms and put all terms together. Then we get,

$$V_{n+1}(x+e_i) - 2V_{n+1}(x) + V_{n+1}(x-e_i) > S_i + \sum_{l=1, l\neq i}^{N} S_l,$$

where

$$\begin{aligned} S_l := \Big[&\mu(a_l)V_n(x-e_l+e_i) - \mu(a_l)V_n(x-e_l) - \mu(c_l)V_n(x-e_l) \\ &+ \mu(c_l)V_n(x-e_l-e_i)\Big] + \big(\mu(A)-\mu(a_l)\big)V_n(x+e_i) \\ &- \big(\mu(A)-\mu(a_l)\big)V_n(x) - \big(\mu(A)-\mu(c_l)\big)V_n(x) \\ &+ \big(\mu(A)-\mu(c_l)\big)V_n(x-e_i), \end{aligned}$$

for $l = 1, \ldots, N$. Clearly, if all $S_l \geq 0$, then

$$V_{n+1}(x+e_i) - 2V_{n+1}(x) + V_{n+1}(x-e_i) > 0.$$

Thus, as the next step we prove that this statement holds. We discuss this for S_i first, and then we deal with the case S_l for $l \neq i$. We first add $-\mu(c_i)\big[V_n(x) - V_n(x-e_i)\big] + \mu(c_i)\big[V_n(x) - V_n(x-e_i)\big]$ and $\big[-\mu(a_i) + \mu(a_i)\big]\big[V_n(x) - V_n(x-e_i)\big]$ to S_i. Then, we get

$$\begin{aligned} S_i = &\big[\mu(a_i) - \mu(c_i)\big]\big[V_n(x) - V_n(x-e_i)\big] \\ &+ \mu(c_i)V_n(x) - 2\mu(c_i)V_n(x-e_i) + \mu(c_i)V_n(x-2e_i) \\ &+ \big[\mu(A) - \mu(a_i)\big]\big[V_n(x+e_i) - V_n(x)\big] \\ &- \big[\mu(A) - \mu(a_i) + \mu(a_i) - \mu(c_i)\big]\big[V_n(x) - V_n(x-e_i)\big] \\ = &\big[\mu(a_i) - \mu(c_i)\big]\big[V_n(x) - V_n(x-e_i)\big] \\ &+ \mu(c_i)V_n(x) - 2\mu(c_i)V_n(x-e_i) + \mu(c_i)V_n(x-2e_i) \\ &+ \big[\mu(A) - \mu(a_i)\big]\big[V_n(x+e_i) - V_n(x)\big] \\ &- \big[\mu(A) - \mu(a_i)\big]\big[V_n(x) - V_n(x-e_i)\big] \\ &- \big[\mu(a_i) - \mu(c_i)\big]\big[V_n(x) - V_n(x-e_i)\big] \\ = &\,\mu(c_i)\big[V_n(x) - 2V_n(x-e_i) + V_n(x-2e_i)\big] \\ &+ \big[\mu(A) - \mu(a_i)\big]\big[V_n(x+e_i) - 2V_n(x) + V_n(x-e_i)\big] \geq 0. \end{aligned}$$

The second and third equalities above follow from the standard calculus. The last inequality holds because of the induction hypothesis. Next, we

proof that $S_l \geq 0$ for $l \neq i$. By rearranging the terms of S_l, we derive

$$\begin{aligned} S_l = {} & \mu(a_l)\big[V_n(x) + V_n(x - e_l + e_i) - V_n(x + e_i) - V_n(x - e_l)\big] \\ & + \mu(c_l)\big[V_n(x) + V_n(x - e_l - e_i) - V_n(x - e_i) - V_n(x - e_l)\big] \\ & + \mu(A)\big[V_n(x + e_i) - 2V_n(x) + V_n(x - e_i)\big]. \end{aligned}$$

Recall that $a_l > c_l$ for $l \neq i$. In addition, due to the induction hypothesis, $V_n(x)$ is submodular in (x_i, x_l) for all $d_p \geq 0$ and for $l \neq i$. This implies that

$$\begin{aligned} & V_n(x) - V_n(x - e_l) - V_n\Big(x + e_i - \sum_{p=1, p\neq i, p\neq l}^{N} d_p \cdot e_p\Big) \\ & \quad + V_n\Big(x + e_i - e_l - \sum_{p=1, p\neq i, p\neq l}^{N} d_p \cdot e_p\Big) \geq 0. \end{aligned}$$

Set $d_p = 0$ for all $p \in \{1, \ldots, N\} \setminus \{i, l\}$. Then it follows: $V_n(x) + V_n(x - e_l + e_i) - V_n(x + e_i) - V_n(x - e_l) \geq 0$ for $l \neq i$. Therefore, using $a_l > c_l$ we derive that

$$\begin{aligned} S_l \geq {} & \mu(c_l)\big[V_n(x) + V_n(x - e_l + e_i) - V_n(x + e_i) - V_n(x - e_l) \\ & + V_n(x) + V_n(x - e_l - e_i) - V_n(x - e_i) - V_n(x - e_l)\big] \\ & + \mu(A)\big[V_n(x + e_i) - 2V_n(x) + V_n(x - e_i)\big] \\ \geq {} & \mu(c_l)\big[V_n(x - e_l + e_i) - 2V_n(x - e_l) + V_n(x - e_l - e_i)\big] \\ & + \big[\mu(A) - \mu(c_l)\big]\big[V_n(x + e_i) - 2V_n(x) + V_n(x - e_i)\big] \geq 0. \end{aligned}$$

Hence, we conclude that $V_{n+1}(x + e_i) - 2V_{n+1}(x) + V_{n+1}(x - e_i) > 0$. Now, we proceed to prove that $V_{n+1}(x)$ is submodular in (x_i, x_j). To this purpose, define $P = \{1, \ldots, N\} \setminus \{i, j\}$.

Next, to prove submodularity, let $d_p \geq 0$ for all $p \in P$. Then,

$$
\begin{aligned}
& V_{n+1}(x) - V_{n+1}(x - e_j) - V_{n+1}(x + e_i - \sum_{p \in P} d_p \cdot e_p) \\
& + V_{n+1}(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p) \\
= & \sum_{l=1}^{N} \lambda_l \Big[V_n(x + e_l) - V_n(x - e_j + e_l) - V_n(x + e_i - \sum_{p \in P} d_p \cdot e_p + e_l) \\
& + V_n(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p + e_l) \Big] \\
& + \Big[\min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x)\} - \min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x - e_j)\} - \min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x + e_i - \sum_{p \in P} d_p \cdot e_p)\} \\
& + \min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p)\} \Big] \\
\geq & \min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x)\} - \min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x - e_j)\} - \min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x + e_i - \sum_{p \in P} d_p \cdot e_p)\} \\
& + \min_{a' \in \mathcal{A}_x} \{T^k_{a'}(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p)\}.
\end{aligned}
$$

The inequality holds because the first expression between the brackets is non-negative due to the induction hypothesis. Let $\tilde{a} = a^*(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p)$ and recall that $b = a^*(x)$. Then it follows that

$$
\begin{aligned}
&V_{n+1}(x) - V_{n+1}(x - e_j) - V_{n+1}(x + e_i - \sum_{p\in P} d_p \cdot e_p) \\
&\quad + V_{n+1}(x + e_i - e_j - \sum_{p\in P} d_p \cdot e_p) \\
&\geq T_b^k(x) - T_b^k(x - e_j) - T_{\tilde{a}}^k(x + e_i - \sum_{p\in P} d_p \cdot e_p)\} \\
&\quad + T_{\tilde{a}}^k(x + e_i - e_j - \sum_{p\in P} d_p \cdot e_p) \\
&= \Big[\sum_{l=1}^{N} \mu(b_l) V_n(x - e_l) + \big(N\mu(A) - \sum_{l=1}^{N} \mu(b_l)\big) V_n(x)\Big] \\
&\quad - \Big[\sum_{l=1}^{N} \mu(b_l) V_n(x - e_j - e_l) + \big(N\mu(A) - \sum_{l=1}^{N} \mu(b_l)\big) V_n(x - e_j)\Big] \\
&\quad - \Big[\sum_{l=1}^{N} \mu(\tilde{a}_l) V_n(x + e_i - \sum_{p\in P} d_p \cdot e_p - e_l) \\
&\quad + \big(N\mu(A) - \sum_{l=1}^{N} \mu(\tilde{a}_l)\big) V_n(x + e_i - \sum_{p\in P} d_p \cdot e_p)\Big] \\
&\quad + \Big[\sum_{l=1}^{N} \mu(\tilde{a}_l) V_n(x + e_i - e_j - \sum_{p\in P} d_p \cdot e_p - e_l) \\
&\quad + \big(N\mu(A) - \sum_{l=1}^{N} \mu(\tilde{a}_l)\big) V_n(x + e_i - e_j - \sum_{p\in P} d_p \cdot e_p)\Big].
\end{aligned}
$$

Now we rearrange the terms to simplify the inequality. Then,

$$
\begin{aligned}
&V_{n+1}(x) - V_{n+1}(x - e_j) - V_{n+1}(x + e_i - \sum_{p \in P} d_p \cdot e_p) \\
&\quad + V_{n+1}(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p) \\
&\geq \sum_{l=1}^{N} \mu(b_l) \Big[V_n(x - e_l) - V_n(x) - V_n(x - e_j - e_l) + V_n(x - e_j) \Big] \\
&\quad - \sum_{l=1}^{N} \mu(\tilde{a}_l) \Big[V_n(x + e_i - \sum_{p \in P} d_p \cdot e_p - e_l) - V_n(x + e_i - \sum_{p \in P} d_p \cdot e_p) \\
&\quad - V_n(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p - e_l) + V_n(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p) \Big] \\
&\quad + N\mu(A) \Big[V_n(x) - V_n(x - e_j) - V_n(x + e_i - \sum_{p \in P} d_p \cdot e_p) \\
&\quad + V_n(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p) \Big].
\end{aligned}
$$

Subsequently, we write the final expression above by $S_i + S_j + \sum_{l=1,\ l \neq i,\ l \neq j}^{N} S_l$, where

$$
\begin{aligned}
S_k = &\ \mu(b_k) \Big[V_n(x - e_k) - V_n(x) - V_n(x - e_j - e_k) + V_n(x - e_j) \Big] \\
&- \mu(\tilde{a}_k) \Big[V_n(x + e_i - \sum_{p \in P} d_p \cdot e_p - e_k) - V_n(x + e_i - \sum_{p \in P} d_p \cdot e_p) \\
&- V_n(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p - e_k) + V_n(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p) \Big] \\
&+ \mu(A) \Big[V_n(x) - V_n(x - e_j) - V_n(x + e_i - \sum_{p \in P} d_p \cdot e_p) \\
&+ V_n(x + e_i - e_j - \sum_{p \in P} d_p \cdot e_p) \Big],
\end{aligned}
$$

for $k = 1, \ldots, N$. In the remainder of the proof we will show that all $S_k \geq 0$. First, we focus on S_i. Consider the first term between brackets in S_i. By setting $x' = x - e_i$, we see that this term is equal to $V_n(x') - V_n(x' - e_j) - V_n(x' + e_i) + V_n(x' + e_i - e_j) \geq 0$, since $V_n(x)$ is submodular. In addition,

because of the conditions of the theorem, we have $b_i \geq \tilde{a}_i$. Therefore,

$$
\begin{aligned}
S_i \geq\ & \mu(\tilde{a}_i)\Big[V_n(x-e_i) - V_n(x) - V_n(x-e_j-e_i) + V_n(x-e_j)\Big] \\
& - \mu(\tilde{a}_i)\Big[V_n(x - \sum_{p\in P} d_p\cdot e_p) - V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p) \\
& - V_n(x-e_j-\sum_{p\in P} d_p\cdot e_p) + V_n(x+e_i-e_j-\sum_{p\in P} d_p\cdot e_p)\Big] \\
& + \mu(A)\Big[V_n(x) - V_n(x-e_j) - V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p) \\
& + V_n(x+e_i-e_j-\sum_{p\in P} d_p\cdot e_p)\Big] \\
=\ & \mu(\tilde{a}_i)\Big[V_n(x-e_i) - V(x-e_j-e_i) - V_n(x-\sum_{p\in P}^{N} d_p\cdot e_p) \\
& + V_n(x-e_j-\sum_{p\in P}^{N} d_p\cdot e_p)\Big] \\
& + \Big[\mu(A)-\mu(\tilde{a}_i)\Big]\Big[V_n(x) - V_n(x-e_j) - V_n(x+e_i-\sum_{p\in P}^{N} d_p\cdot e_p) \\
& + V_n(x+e_i-e_j-\sum_{p\in P}^{N} d_p\cdot e_p)\Big]\Big] \geq 0.
\end{aligned}
$$

The second equality above follows from standard calculus and the last inequality follows from submodularity of the induction hypothesis. Now, we proceed to study S_j. Since $V_n(x)$ is convex, the first expression between the brackets of S_j is non-positive. Combining this result with the condition of

the theorem, $\tilde{a}_j \geq b_j$ (which is equivalent to $-b_j \geq -\tilde{a}_j$), we get

$$
\begin{aligned}
S_j \geq\ & \mu(\tilde{a}_j)\Big[V_n(x-e_j) - V_n(x) - V_n(x-2e_j) + V_n(x-e_j)\Big] \\
& - \mu(\tilde{a}_j)\Big[V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p - e_j) - V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p) \\
& - V_n(x+e_i-2e_j-\sum_{p\in P} d_p\cdot e_p) + V_n(x+e_i-e_j-\sum_{p\in P} d_p\cdot e_p)\Big] \\
& + \mu(A)\Big[V_n(x) - V_n(x-e_j) - V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p) \\
& + V_n(x+e_i-e_j-\sum_{p\in P} d_p\cdot e_p)\Big] \\
\geq\ & \mu(\tilde{a}_j)\Big[V_n(x-e_j) - V_n(x-2e_j) - V_n(x+e_i-e_j-\sum_{p\in P}^{N} d_p\cdot e_p) \\
& + V_n(x+e_i-2e_j-\sum_{p\in P}^{N} d_p\cdot e_p)\Big] \\
& + \Big[\mu(A)-\mu(\tilde{a}_j)\Big]\Big[V_n(x) - V_n(x-e_j) - V_n(x+e_i-\sum_{p\in P}^{N} d_p\cdot e_p) \\
& + V_n(x+e_i-e_j-\sum_{p\in P}^{N} d_p\cdot e_p)\Big] \geq 0.
\end{aligned}
$$

Finally, we prove that S_l for $l \neq i$ and $l \neq j$ is also non-negative. Let $x' = x - e_j$. Then the first term of S_l is equal to $V_n(x') - V_n(x' - e_l) - V_n(x' + e_j) + V_n(x' + e_j - e_l)$. Since $V_n(x)$ is submodular in all (x_i, x_j) with $j \neq i$, $V_n(x')$ is submodular in (x_j, x_l). Therefore, using $\mu(A) \geq \mu(\tilde{a}_l)$, we

derive that

$$
\begin{aligned}
S_l \geq\ & \mu(b_l)\Big[V_n(x-e_j) - V_n(x-e_j-e_l) - V_n(x) + V_n(x-e_l)\Big] \\
& - \mu(\tilde{a}_l)\Big[V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p - e_l) - V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p) \\
& - V_n(x+e_i-e_j-\sum_{p\in P} d_p\cdot e_p - e_l) + V_n(x+e_i-e_j-\sum_{p\in P} d_p\cdot e_p)\Big] \\
& + \mu(\tilde{a}_l)\Big[V_n(x) - V_n(x-e_j) - V_n(x+e_i-\sum_{p\in P} d_p\cdot e_p) \\
& + V_n(x+e_i-e_j-\sum_{p\in P} d_p\cdot e_p)\Big] \\
\geq\ & \mu(\tilde{a}_l)\Big[V_n(x) - V_n(x-e_j) - V_n(x+e_i-\sum_{p\in P}^{N} d_p\cdot e_p - e_l) \\
& + V_n(x+e_i-e_j-\sum_{p\in P}^{N} d_p\cdot e_p - e_l)\Big].
\end{aligned}
$$

Now, let $d' = d + e_l$. Then

$$
\begin{aligned}
S_l \geq\ & \mu(\tilde{a}_l)\Big[V_n(x) - V_n(x-e_j) - V_n(x+e_i-\sum_{p\in P}^{N} d'_p\cdot e_p) \\
& + V_n(x+e_i-e_j-\sum_{p\in P}^{N} d'_p\cdot e_p)\Big] \geq 0.
\end{aligned}
$$

Hence, $V_{n+1}(x)$ is submodular in (x_i, x_j). Thus, by taking the limit as $n \to \infty$, we conclude that $V(x)$ is convex and submodular under the conditions given in the theorem. □

Lemma A.1 shows that the relative value function V satisfies convexity and submodularity in case $a_i < b_i$ and $a_j > b_j$ for $j \neq i$. Note that this leads to a contradiction in Theorem 5.3.3, since this implies that $Z = T_a(x) - T_b(x) - T_a(x+e_i) + T_b(x+e_i) < 0$. However, it was shown in the theorem that $Z \geq 0$. Hence, this case cannot occur. Similarly, when $a_j < b_j$ for all j cannot occur. The next lemma provides a rigorous statement for this.

Lemma A.2. *Let $a = a^*(x+e_i)$ and $b = a^*(x)$. If $a_j < b_j$ for $j = 1, \ldots, N$ then the following properties hold:*

1. $V(x)$ *is convex in* i: $V(x+e_i) - 2V(x) + V(x-e_i) \geq 0$,

2. $V(x)$ *is supermodular* in (x_i, x_j):
   $V(x+e_i - \sum_{p=1,p\neq j,p\neq i}^{N} d_p \cdot e_p) - V(x+e_i-e_j - \sum_{p=1,p\neq j,p\neq i}^{N} d_p \cdot e_p) - V(x) + V(x-e_j) \geq 0$ *for* $j \in \{1,2,\ldots,N\} \setminus \{i\}$ *and* $d_p \geq 0$.

Proof. The proof of the lemma is by induction on V_n. Note that the proof of convexity and supermodularity is completely analogous to the proof of convexity and submodularity in Lemma A.1. The only difference is that in Lemma A.1 we have $a_l > c_l$ for $l \neq i$, whereas in this case $a_l < c_l$. However, with supermodularity (instead of submodularity) the signs of the inequalities turn out to be correct for proving convexity and supermodularity. □

Bibliography

[1] http://www.cellular-news.com/story/47245.php.

[2] http://www.cs.vu.nl/das3, 2007.

[3] http://www.artima.com/underthehood/gc.html, 2007.

[4] http://privacy.cs.cmu.edu/dataprivacy/projects/explosion, 2009.

[5] http://www.nasa.gov/home/index.html, 2010.

[6] http://www.multimedian.nl/en/home.php, 2010.

[7] http://www.starplane.org, 2010.

[8] http://www.globus.org, 2010.

[9] http://www.cs.vu.nl/das4, 2010.

[10] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1999.

[11] Y. Aviv and A. Federgruen. The value-iteration method for countable state Markov decision processes. *Operations Research Letters*, 24(5):223–234, 1999.

[12] A. Aziz and H. El-Rewini. Grid resource allocation and task scheduling for resource intensive applications. In *Proceedings of the 2006 International Conference on Parallel Processing Workshops*, Columbus, OH, USA, 2006.

[13] F. Berman, R. Wolski, H. Casanova, and W. Cirne. Adaptive computing on the grid using apples. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.

[14] S. Bhulai and G.M. Koole. A queueing model for call blending in call centers. *IEEE Transactions on Automatic Control*, 48:1434–1438, 2003.

[15] S.C. Borst and P. Seri. Robust algorithms for sharing agents with multiple skills. Working paper, Bell Laboratories, Murray Hill, NJ, 2000.

[16] R.G. Brown. *Statistical Forecasting for Inventory Control.* McGraw-Hill New York, 1959.

[17] R.G. Brown. *Smoothing, Forecasting and Prediction of Discrete Time Series.* Prentice-Hall, 1963.

[18] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *Proceedings of the Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, pages 283–289, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[19] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service oriented grid computing. In *Proceedings of the 15th International Symposium on Parallel and Distributed Processing*, pages 776–790, San Francisco, CA, USA, April 2001.

[20] C. Cascaval, L. DeRose, D. Padua, and D. Reed. Compile-time based performance prediction. *Languages and Compilers for Parallel Computing*, pages 365–379, 2000.

[21] B. Chun and D. Culler. Market-based proportional resource sharing for clusters. Technical report, University of California, Berkeley, September 1999.

[22] R. Cocchi, S. Shanker, D. Estrin, and L. Zhang. Pricing in computer networks: motivation, formulation, and example. *IEEE/ACM Transactions on Networking*, 1(6):614–627, December 1993.

[23] R.B. Cooper. *Introduction to Queueing Theory.* North Holland, 1981.

[24] H.J. Curnow and B.A. Wichmann. A synthetic benchmark. *The Computer Journal*, 19(1):43–49, February 1976.

[25] K. Czajkowski, I.T. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, pages 219–228, Redondo Beach, CA, USA, 1999.

[26] G. Daniel and A.T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. *Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 34:77–84, 2004.

[27] M.A Dobber, G.M. Koole, and R.D. van der Mei. Dynamic load balancing for a grid application. In *Proceedings of the 11th International Conference on High Performance Computing*, volume 1, pages 342–352, Bangalore, India, 2004. Springer.

[28] M.A. Dobber, R.D. van der Mei, and G.M. Koole. A prediction method for job running times on shared processors: survey, statistical analysis and new avenues. *Performance Evaluation*, 64:755–781, 2007.

[29] J. Dongarra, J.L. Martin, and J. Worlton. Computer Benchmarking: Paths and Pitfalls. *IEEE Spectrum*, 24(7):38–43, jul 1987.

[30] T. Fahringer. Evaluation of benchmark performance estimation for parallel fortran programs on massively parallel simd and mimd computers. In *IEEE Proceedings of the 2nd Euromicro Workshop on Parallel and Distributed Processing*, pages 449–456, 1994.

[31] J. Farmer and J. Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59(8):845–848, 1987.

[32] M. Feldman, K. Lai, and L. Zhang. A price-anticipating resource allocation mechanism for distributed shared clusters. In *Proceedings of the ACM Conference on Electronic Commerce*, Vancouver, British Columbia, Canada, 2005.

[33] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[34] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the 8th International Workshop on Quality of Service*, pages 181–188, Pittsburgh, PA, USA, June 2000.

[35] K. Fukuda, N. Wakamiya, M. Murata, and H. Miyahara. *Integrated resource allocation for real-time video transfer to maximize user's utility.* PhD thesis, School of Engineering Science, Osaka University, 2000.

[36] N. Gans and Y. Zhou. A call-routing problem with service-level constraints. *Operations Research*, 51:255–271, 2003.

[37] J.M. Geusebroek, G.J. Burghouts, and A.W.M. Smeulders. The amsterdam library of object images. *International Journal on Computer Vision*, 61(1):103–112, 2005.

[38] W.K. Grassmann. Transient solutions in Markovian queueing systems. *Computers and Operations Research*, 4:47–53, 1977.

[39] L. Green and P. Kolesar. The pointwise stationary approximation for queues with nonstationary arrivals. *Management Science*, 37:84–97, 1991.

[40] L.V. Green, P.J. Kolesar, and J. Soares. Improving the SIPP approach for staffing service systems that have cyclic demands. *Operations Research*, 49:549–564, 2001.

[41] C. Grelck. Array padding in the functional language sac. *High-Level Parallel Programming Paradigm*, 5:2553–2560, 2000.

[42] F. Guim, A. Goyeneche, J. Corbalan, J. Labarta, and G. Terstyansky. Grid computing performance prediction based in historical information. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, Barcelona, Spain, 2006.

[43] M. Guo and V. Conitzer. Optimal in expectation redistribution mechanisms. *Artificial Intelligence*, 174:363–381, 2010.

[44] M. Harchol-Balter and A.B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, 1997.

[45] A. Hauptmann et al. Informedia at trecvid 2003: analyzing and searching broadcast news video. In *Proceeding of TRECVID*, Gaithersburg, USA, 2003.

[46] R. Hockney and M. Berry. Public International Benchmarks for Parallel Computers. Technical report, PARKBENCH Committee: Report-1, feb 1994.

[47] C.C. Holt. Forecasting trends and seasonals by exponentially weighted moving averages. *ONR Memorandum*, 52, 1957.

[48] A. Ingolfsson, E. Akhmetshina, S. Budge, Y. Li, and X. Wu. A survey and experimental comparison of service level approximation methods for non-stationary m/m/s queueing systems. *INFORMS Journal on Computing*, 19:201–214, 2007.

[49] A. Ingolfsson, M.A. Haque, and A. Umnikov. Accounting for time-varying queueing effects in workforce scheduling. *European Journal on Operations Research*, 139:585–597, 2002.

[50] R.K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* John Wiley and Sons, 1991.

[51] L. Kleinrock. A delay-dependent queue discipline. *Naval Research Logistics Quarterly*, 11:59–73, 1964.

[52] L. Kleinrock and R.P. Finkelstein. Time dependent priority queues. *Operations Research*, 15:104–116, 1967.

[53] D.E. Knuth. *The Art of Computer Programming, volume 3: Sorting and Searching.* Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1998.

[54] G.M. Koole. Structural results for the control of queueing systems using event-based dynamic programming. *Queueing Systems*, 30:323–339, 1998.

[55] G.M. Koole. Monotonicity in Markov reward and decision chains: theory and applications. *Foundations and Trends in Stochastic Systems*, 1:1–76, 2006.

[56] K. Kurowski, A. Oleksiak, J. Nabrzyski, A. Kwiecien, M. Wojtkiewicz, M. Dyczkowski, F. Guim, J. Corbalan, and J. Labarta. Multi-criteria grid resource management using performance prediction techniques. In *Proceeding of the CoreGrid Integration Workshop*, Pisa, Italy, 2005.

[57] H.J. Kushner and G.G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, 2003.

[58] A. Lazar and N. Semret. Auctions for network resource sharing. Technical Report TR 468-97-02, Columbia University, February 1997.

[59] J.W. Lee and K. Asanovic. Meterg: Measurement-based end-to-end performance estimation technique in qos-capable multiprocessors. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 135–147, 2006.

[60] S. Lee, Y. Han, H. Song, C. Han, and C. Youn. Grid policy administrator with mdp-based resource management scheme. In *Proceeding of the 2006 International Conference on Grid Computing and Applications*, Las Vegas, NV, USA, 2006.

[61] J.D.C. Little. A proof of the queueing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[62] B.M. Maggs, L.R. Matheson, and R.E. Tarjan. Models of parallel computation: a survey and synthesis. In *Proceedings of 28th Hawaii International Conference on System Sciences*, volume 2, pages 61–70, 1995.

[63] S. McGough, A. Afzal, J. Darlington, N. Furmento, A. Mayer, and L. Young. Making the grid predictable through reservations and performance modelling. *The Computer Journal*, 48:358–368, 2005.

[64] M.S. Moore, J. Sztipanovitz, G. Karsai, and J. Nichols. A Model-Integrated Program Synthesis Environment for Parallel/Real-Time Image Processing. In *Parallel and Distributed Methods for Image Processing, Proceedings of SPIE*, volume 3166, pages 31–45, 1997.

[65] A. Mutz, R. Wolski, and J. Brevik. Eliciting honest value information in a batch-queue environment. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, pages 291–297, Austin, Texas, 2007.

[66] F. Nadeem, R. Prodan, T. Fahringer, and A. Iosup. A Framework For Resource Availability Characterization And Online Prediction in the Grids. In *Grid Computing: Achievements and Prospects*, pages 209–224. Springer, 2008.

[67] D. Nurmi, J. Brevik, and R. Wolski. Qbets: Queue bounds estimation from time series. In *Proceedings of Job Scheduling Strategies for Parallel Processing*, Seattle, WA, USA, June 2007.

[68] D. Nurmi, R. Wolski, and J. Brevik. Probabilistic advanced reservations for batch-scheduled parallel machines. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 289–290, Salt Lake City, UT, USA, February 2008.

[69] D. Nurmi, R. Wolski, and J. Brevik. Varq: virtual advance reservations for queues. In *Proceedings of the 17th international symposium on High Performance Distributed Computing*, pages 75–86, Boston, MA, USA, 2008.

[70] J. Park. A scalable protocol for deadlock and livelock free co-allocation of resources in internet computing. In *Proceeding of the Symposium on Applications and the Internet*, pages 66–73, Orlando, FL, USA, January 2003.

[71] M. Perry and A. Nilsson. Performance modeling of automatic call distributors: Assignable grade of service staffing. In *Proceedings of the XIV International Switching Symposium*, pages 294–298, Yakohama, Japan, October 1992.

[72] A. Pimentel. *A Computer Architecture Workbench*. PhD thesis, University of Amsterdam, The Netherlands, December 1998.

[73] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., 1994.

[74] M.L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[75] Y. Qiao and P. Dinda. Network traffic analysis, classification, and prediction. Technical report, Northwestern University, 2003.

[76] O.F. Rana, M. Winikoff, L. Padgham, and J. Harland. Applying conflict management strategies in BDI agents for resource management in computational grids. In *Proceedings of the Fifth UK Workshop on Multi-Agent Systems*, pages 205–214, Liverpool, UK, 2002. IEEE Computer Society Press.

[77] A. Roy and V. Sander. Grid resource management: state of the art and future trends. In *GARA: a uniform quality of service architecture*, pages 373–394. Kluwer Academic Publishers, 2004.

[78] V.V. Rykov. Monotone control of queueing systems with heterogeneous servers. *Queueing Systems*, 37(4):391–403, 2001.

[79] R.H. Saavedra-Barrera, A.J. Smith, and E. Miya. Machine characterization based on an abstract high-level language machine. *IEEE Transactions on Computers*, 38(12):1659–1679, 1989.

[80] T. Sandholm, K. Lai, J. Andrade, and J. Odeberg. Market-based resource allocation using price prediction in a high performance computing grid for scientific applications. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, pages 19–23, Paris, France, 2006.

[81] C.H. Sauer and K. Mani Chandi. *Computer Systems Performance Modeling*. Prentice-Hall Series in Advances in Computing Science and Technology. Prentice-Hall, 1981.

[82] K. Schutte and G.M.P. van Kempen. Optimal cache usage for separable image processing algorithms on general purpose workstations. *Signal Processing*, 59(1):113–122, 1997.

[83] F.J. Seinstra, J. Geusebroek, D. Koelma, C.G.M. Snoek, M. Worring, and A.W.M. Smeulders. High-performance distributed video content analysis with parallel-horus. *IEEE MultiMedia*, 14(4):64–75, 2007.

[84] F.J. Seinstra, D. Koelma, and J.M. Geusebroek. A software architecture for user transparent parallel image processing. *Parallel Computing*, 28(7-8):967–993, 2002.

[85] F.J. Seinstra, C.G.M. Snoek, D. Koelma, J.M. Geusebroek, and M. Worring. User transparent parallel processing of the 2004 nist trecvid data set. In *Proceedings of the International Parallel and Distributed Processing Symposium*, Denver, Colorado, USA, 2005.

[86] R.A. Shumsky. Approximation and analysis of a queueing system with flexible and specialized servers. *Operations Research Spektrum*, 26:307–330, 2004.

[87] R.H. Shumway and D.S. Stoffer. *Time Series Analysis and Its applications: with R Examples*. Springer Texts in Statistics, 2006.

[88] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. *Journal on Parallel and Distributed Computing*, 64:1007–1016, 2004.

[89] G.G.M. Snoek, M. Worring, J.M. Geusebroek, D.C. Koelma, F.J. Seinstra, and A.W.M. Smeulders. The semantic pathfinder: Using an authoring metaphor for generic multimedia indexing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1678–1689, 2006.

[90] C.G.M. Snoek et al. The mediamill trecvid 2005 semantic video search engine. In *Proceedings of the 3rd TRECVID Workshop*, Gaithersburg, MD, USA, 2005.

[91] S. Song, K. Hwang, and Y.K. Kwok. Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling. *IEEE Transactions on Computers*, 55:703–719, 2006.

[92] O. Sonmez, N. Yigitbasi, D. Epema, and A. Iosup. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *Proceeding of the International Symposium on High Performance Distributed Computing*, Munich, Germany, 2009.

[93] D.A. Stanford and W.K. Grassmann. Bilingual server call centres. In D.R. McDonald and S.R.E. Turner, editors, *Call Centres, Traffic and Performance*, volume 28, pages 31–48. Fields Institute Communications, 2000.

[94] A.J. van der Steen. Is it really possible to benchmark a supercomputer? A graded approach to performance measurement. In A.J. van der Steen, editor, *Evaluating Supercomputers: Strategies for Exploiting, Evaluating and Benchmarking Computers with Advanced Architectures*, chapter 14, pages 190–212. Chapman and Hall, 1990.

[95] S. Stidham and R. Weber. A survey of Markov decision models for control of networks of queues. *Queueing Systems*, 13:291–314, 1993.

[96] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing

and data migration in mariposa. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, pages 28–30, Austin, TX, USA, September 1994. IEEE Comput. Soc. Press.

[97] A. Sulistio, W. Schiffmann, and R. Buyya. Advanced reservation-based scheduling of task graphs on clusters. In *Proceedings of the 13th International Conference on High Performance Computing*, 2006.

[98] R.S. Sutton and A.G. Barto. *Reinforcement Learning: an introduction*. MIT Press, Cambridge, 1998.

[99] C. Tham and R. Buyya. Sensorgrid: integrating sensor networks and grid computing. *Invited Paper in CSI Communications, Special Issue on Grid Computing, Computer Society of India*, pages 24–29, 2005.

[100] H.C. Tijms. *Stochastic Models: an Algorithmic Approach*. Wiley, 1986.

[101] B. Veeravalli, L. Chen, H.Y. Kwoon, G.K. Whee, S.Y. Lai, L.P. Hian, and H.C. Chow. Design, analysis, and implementation of an agent driven pull-based distributed video-on-demand system. *Multimedia Tools and Applications*, 28(1):89–118, 2006.

[102] X. Wang and J. Luo. Architecture of grid resource allocation management based on qos. *Grid and Cooperative Computing*, pages 81–88, 2004.

[103] R.P. Weicker. Dhrystone: A synthetic systems programming benchmark. *Communications of the ACM*, 27(10):1013–1030, 1984.

[104] P.R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342, 1960.

[105] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*, pages 316–325, Portland, OR, USA, 1997.

[106] R. Wolski, N.T. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal on Future Generation Computing Systems*, 15:757–768, 1999.

[107] Y. Wu, Y. Yuan, G. Yang, and W. Zheng. Load prediction using hybrid model for computational grid. In *Proceddings of 8th IEEE/ACM International Conference on Grid Computing*, pages 235–242, Austin, TX, USA, 2007. IEEE Computer Society.

[108] L. Xie and R. Yan. Extracting semantics from multimedia content: challenges and solutions. *Multimedia Content Analysis*, pages 35–65, 2008.

[109] Z. Xu, X. Zhang, and L. Sun. Semi-Empirical Multiprocessor Performance Predictions. *Journal on Parallel and Distributed Computing*, 39(1):14–28, 1996.

[110] R. Yang, S. Bhulai, and R.D. van der Mei. Optimal resource allocation for multi-queue systems with a shared server pool. To appear in *Queueing Systems*, 2011.

[111] R. Yang, S. Bhulai, and R.D. van der Mei. Structural properties of the optimal resource allocation policy for single-queue systems. To appear in *Annals of Operations Research*, 2011.

[112] R. Yang, S. Bhulai, R.D. van der Mei, and F.J. Seinstra. Optimal resource allocation for time-reservation systems. To appear in *Performance Evaluation*, 2011.

[113] R. Yang, S. Bhulai, R.D. van der Mei, F.J. Seinstra, and H.E. Bal. Resource optimization in distributed real-time multimedia applications. To appear in *Multimedia Tools and Applications*, 2011.

[114] R. Yang, R.D. van der Mei, F.J. Seinstra, and J. Maassen. Experimental validation of optimal allocation policies for multimedia content analysis. Paper in preparation, 2010.

[115] R. Yang, D. Roubos, S. Bhulai, and R.D. van der Mei. Time-constrained resource allocation for multi-queue service systems with non-stationary arrival rates. Submitted, 2010.

[116] R. Yang, R.D. van der Mei, D. Roubos, F.J. Seinstra, and G.M. Koole. On the optimization of resource utilization in distributed multimedia applications. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, pages 358–365, Lyon, France, 2008. IEEE Computer Society Press.

[117] R. Yang, R.D. van der Mei, D. Roubos, F.J. Seinstra, G.M. Koole, and H. Bal. Modeling "just-in-time" communication in distributed real-time multimedia applications. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, pages 518–525, Lyon, France, 2008. IEEE Computer Society Press.

[118] J. Yoo. *Queueing Models for Staffing Service Operations*. PhD thesis, University of Maryland, 1996.

[119] J. Yu, R. Buyya, and C.K. Tham. QoS-based scheduling of workflow applications on service grids. In *Proceeding of the 1st IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia, December 2005.

[120] Y. Zhang, W. Sun, and Y. Inoguchi. Predict task running time in grid environments based on CPU load predictions. *Future Generations Computer Systems*, 24(6):489–497, 2008.

List of publications

The scientific contents of this thesis are based on the following papers.

1. R. Yang, R.D. van der Mei, D. Roubos, F.J. Seinstra, G.M. Koole, and H.E. Bal. Modeling "Just-in-Time" communication in distributed real-time multimedia applications. *Proceedings of IEEE international symposium on Cluster Computing and the Grid*, Lyon, pages 518–525, IEEE Computer Society Press, 2008.

2. R. Yang, R.D. van der Mei, D. Roubos, F.J. Seinstra, and G.M. Koole. On the optimization of resource utilization in distributed multimedia applications. *Proceedings of IEEE international symposium on Cluster Computing and the Grid*, Lyon, pages 358–365, IEEE Computer Society Press, 2008.

3. R. Yang, R.D. van der Mei, D. Roubos, F.J. Seinstra, and H.E. Bal. Resource optimization in distributed real-time multimedia applications. To appear in *Multimedia Tools and Applications*, 2011.

4. R. Yang, S. Bhulai, and R.D. van der Mei, F.J. Seinstra. Optimal resource allocation for time-reservation systems. To appear in *Performance Evaluation*, 2011.

5. R. Yang, S. Bhulai, and R.D. van der Mei. Optimal resource allocation for multi-queue systems with a shared server pool. To appear in *Queueing Systems*, 2011.

6. R. Yang, S. Bhulai, and R.D. van der Mei. Structural properties of the optimal resource allocation policy for single-queue systems. To appear in *Annals of Operations Research*, 2011.

7. R. Yang, D. Roubos, S. Bhulai, and R.D. van der Mei. Time-constrained resource allocation for multi-queue service systems with non-stationary arrival rates. Submitted.

8. R. Yang, R.D. van der Mei, F.J. Seinstra, and J. Maassen. Experimental validation of optimal allocation policies for multimedia content analysis, in preparation, 2010.

Other collaborated work that is not included in this thesis is listed below:

1. R. Yang, R.E. Kooij and R.D. van der Mei. TCP performance modeling in the case of bi-directional packet loss. In: D. Kouvatsos ed., *Traffic and Performance Engineering for Heterogeneous Networks* (River Publishers), pages 1-18, 2009.

2. R.E. Kooij, R.D. van der Mei and R. Yang. TCP and Web browsing performance in case of bi-directional packet loss. *Computer Communications* 33, S50-S57, 2010.

Samenvatting

Adaptieve toewijzing van processoren in gedistribueerde computersystemen voor multimediale toepassingen

Vandaag de dag bestaat digitale informatie in toenemende mate uit multimediale elementen, een combinatie van audiovisuele en linguïstische gegevens. Het maatschappelijke belang van dit soort informatie is de laatste jaren sterk toegenomen door de opkomst van publiek toegankelijke televisie-archieven, videobeelden van bewakingscamera's in openbare ruimten en het automatisch vergelijken van forensisch bewijsmateriaal in biomedische identificatiesystemen (zoals iris-scans en vingerafdrukken). In de nabije toekomst zal geautomatiseerde toegang tot multimediale gegevens een probleem zijn van fenomenale omvang. Immers, digitale video kan data produceren op een zeer hoge snelheid en de opslag van multimediale gegevens loopt dan al gauw in de Petabytes (10^{15} bytes). Omdat de rekencapaciteit van een enkele computer bij lange na niet kan voldoen aan de eisen die veel multimediale applicaties stellen, is het nodig om de processor- en opslagcapaciteit van grootschalige gedistribueerde computersystemen aan te wenden.

Een probleem bij het analyseren van de multimediale gegevens is dat er vaak strikte tijdsbeperkingen gelden. Bijvoorbeeld, om lange wachttijden van klanten te voorkomen bij de iris-scanners op een internationaal vliegveld zal het systeem de passagier moeten kunnen identificeren binnen enkele seconden. Een groot deel van de autonome toepassingen, zoals het automatisch herkennen en volgen van verdachte individuen in videobeelden van bewakingscamera's, moet zelfs in real-time plaatsvinden. Daarom is het in steeds meer industriële toepassingen noodzakelijk uit een kolossale hoeveelheid van dergelijke gegevens automatisch binnen een beperkte tijd de juiste informatie te extraheren die voor mensen van betekenis is. Om dat te realiseren is de processing-capaciteit voor het analyseren van multimediale data van essentieel belang.

In ons onderzoek beperken we ons tot de beeldbewerking van de multimediale elementen. In dit soort service-gebaseerde scenario's sluit een klantapplicatie zich aan bij een of meerdere multimediale servers op afstand. Elke server voert de opdracht uit bij een (mogelijk verschillend) computercluster. Tijdens de uitvoering stuurt de applicatie van een klant videobeelden naar de beschikbare servers om te analyseren, waarbij een job kan worden uitgevoerd op meerdere parallelle servers.

Een van de fundamentele uitdagingen bij de parallelle berekening in gedistribueerde computersystemen is het bereiken van een hoge efficiëntie in het bewerken van de data, ongeacht de variabiliteit in de beschikbare hardware- en software-resources. Er zijn een aantal oorzaken voor de variabiliteit in gedistribueerde computersystemen. Ten eerste, de computerkracht in gedistribueerde computersystemen wordt gedeeld door diverse applicaties. Dat maakt de beschikbare capaciteit vaak schaars en onderhevig aan veranderingen van tijd tot tijd. Ten tweede, de hoeveelheid data van de multimediale applicaties die geanalyseerd moet worden verschillen ook van elkaar. Bijvoorbeeld een applicatie waarin objecten die in een videostroom verkregen zijn door bewakingscamera's moeten worden herkend bevat een gigantische hoeveelheid data om te analyseren. Echter, bij de iris-scan applicatie is het aankomstproces van de data die geanalyseerd moet worden veel minder voorspelbaar. Deze variabiliteit verhoogt de behoefte aan stochastische methoden die de multimediale applicaties minder gevoelig maken voor de fluctuaties van beschikbare resources in gedistribueerde computersystemen. Naast de tijdsbeperkende eis die de applicaties hebben, is het ook van belang om de gemiddelde gebruikskosten zo klein mogelijk te houden vanwege het feit dat de rekencapaciteit van gedistribueerde computersystemen vaak schaars en kostbaar is. Daarom moeten de stochastische methoden zorgen voor optimale balans tussen de beschikbare rekencapaciteit en de gebruikskosten, en tegelijkertijd voldoen aan strikte tijdsbeperkingen. In dit proefschrift staan het ontwikkelen, analyseren en optimaliseren van dit soort methoden centraal.

Na een uitgebreide introductie in hoofdstuk 1 en een formele beschrijving van de methoden in hoofdstuk 2 bestuderen we in hoofdstuk 3 de optimale inzet van beschikbare resources voor multimediale applicaties waarbij de beeldbewerking sequentieel wordt uitgevoerd. Om dit probleem op te lossen is het eerst van belang om het optimale aantal processoren gebruikt bij elke multimediale server te bepalen door de juiste afweging te maken tussen de rekentijd *van* en de communicatietijd *tussen* de processoren. Dit pro-

bleem noemen we het "resource utilisatie" (RU) probleem. Vervolgens is het van belang om het beste tijdsmoment te bepalen voor het versturen van nieuwe data naar de servers, zodanig dat het gebruik van de processoren wordt gemaximaliseerd en tegelijkertijd de (kostbare) tijd dat de data in de buffer staat te wachten wordt geminimaliseerd. Dit wordt ook wel het "just-in-time" (JIT) communicatie probleem genoemd. Voor het RU probleem ontwikkelen we een simpele en eenvoudig implementeerbare methode om het optimale aantal processoren te gebruiken bij een multimediale server te bepalen. Deze methode is gebaseerd op de klassieke binary-search methode voor niet-lineaire optimalisering en het is onafhankelijk van de specificaties van een systeem. Bij het JIT-probleem ontwikkelen we een slimme adaptieve controlemethode die snel en adequaat kan reageren op veranderingen in gedistribueerde computersystemen.

In hoofdstuk 4 bestuderen we allocatieproblemen in systemen met tijdsreservering. In deze systemen komen klanten aan bij een service-faciliteit volgens een Poisson-proces en ontvangen service in twee stappen. In de eerste stap wordt de klantinformatie verzameld, die vervolgens wordt verzonden naar de servers voor het analyseren. In de tweede stap wordt de informatie verwerkt, waarna de klant het systeem verlaat. Hier moeten twee beslissingen worden genomen: Ten eerste, *wanneer* moeten de processoren worden gereserveerd, zodanig dat (1) de klant niet hoeft te wachten voor de start van de service van de tweede stap, en (2) de gereserveerde processoren niet wordt verspild als gevolg van het wachten op de informatie bij de eerste stap. Ten tweede, *hoeveel* processoren moeten er worden toegewezen voor de tweede processtap zodat de gemiddelde utilisatie- en de holdingkosten per tijdseenheid zijn geminimaliseerd onder de gegeven tijdsbeperking. Aangezien de exacte analyse van het systeem moeilijk is, splitsen we het probleem op in twee delen die sequentieel worden opgelost en leiden tot bijna-optimale oplossingen. In het eerste deel tonen we aan dat het optimale reserveringsmoment wordt bepaald door het verschil van de gemiddelde servicetijd van de eerste stap en de gemiddelde reserveringstijd. Daarna passen we dynamisch programmeren toe om de beste allocatie-strategie op de tweede stap van het systeem te bepalen, en tegelijkertijd tonen we aan dat de optimale allocatie-strategie een stapfunctie is en met als extreem geval een bang-bang controle policy (dat wil zeggen: alle of geen van de processoren toewijzen).

In hoofdstuk 5 leggen we het accent op de tweede stap van het system beschreven in hoofdstuk 4. We breiden het model uit tot systemen met meerdere wachtrijen waarin elke service-faciliteit een eigen tijdsbeperking heeft

van de gemiddelde verblijfsduur van een job in het systeem. De beslissing moet worden gemaakt voor het dynamisch toewijzen van processoren over de verschillende faciliteiten, zodanig dat aan alle tijdsbeperkingen wordt voldaan tegen de minimale kosten. Structurele eigenschappen van de optimale strategie, die moeilijk te bewijzen zijn voor multi-dimensionale systemen, worden bewezen in dit hoofdstuk. Deze eigenschappen samen met die in hoofdstuk 6 geven volledige karakterisering van de optimale strategie voor dit type systeem.

Het model in hoofdstuk 6 is een speciaal geval van het model beschreven in hoofdstuk 5, waarin sprake is van slechts één enkele wachtrij. We laten zien met gebruik van dynamisch programmeren dat de optimale allocatie-strategie (1) work-conserving is, en (2) een stapfunctie volgt met als extreem geval de bang-bang controle strategie. Bovendien, (3) leiden we een conditie af waaronder de bang-bang controle strategie optimaal is. Deze karakteriseringen van de optimale strategie zijn niet rechtstreeks van toepassing in de multi-wachtrij systemen vanwege het feit dat de technieken die hier worden gebruikt om de resultaten te tonen niet generaliseerbaar zijn naar meerdere wachtrijen.

In hoofdstuk 7 bestuderen we de optimale allocatie-strategie voor multi-wachtrij systemen met een tijdsvariërend Poisson-aankomstproces. We bestuderen twee gevallen. In het eerste geval zijn de tijdsafhankelijke parameters van te voren bekend. In dit geval laten we zien hoe de optimale strategie numeriek kan worden bepaald. In het tweede geval bekijken we het optimale allocatie probleem *zonder* volledige kennis van de aankomstintensiteit van jobs. Daartoe gebruiken we zowel een voorspellingsmethode als een stochastische benaderingsmethode om de tijdsvariërende parameters op te sporen en vervolgens een strategie te bepalen die bijna-optimaal is.

Ten slotte wordt in hoofdstuk 8 de optimale allocatiemethode gevalideerd voor een eenvoudige applicatie in een experimentele testomgeving. De resultaten laten zien dat onze modellen inderdaad effectief, simpel en eenvoudig implementeerbaar zijn.