UNIVERSITY OF TORINO

DOCTORAL SCHOOL IN SCIENCE AND HIGH TECHNOLOGY
SPECIALIZATION IN COMPUTER SCIENCE

XXIII PHD CYCLE

Power and Performance Management in Cloud Computing Systems

by

MARCO GUAZZONE

Advisor: Prof. COSIMO ANGLANO

To Lorenza, my forever love . . .

Abstract

Cloud computing is an emerging computing paradigm which is gaining popularity in IT industry for its appealing property of considering "Everything as a Service". The goal of a cloud infrastructure provider is to maximize its profit by minimizing the amount of violations of Quality-of-Service (QoS) levels agreed with service providers, and, at the same time, by lowering infrastructure costs. Among these costs, the energy consumption induced by the cloud infrastructure, for running cloud services, plays a primary role. Unfortunately, the minimization of QoS violations and, at the same time, the reduction of energy consumption is a conflicting and challenging problem. In this thesis, we propose a framework to automatically manage computing resources of cloud infrastructures in order to simultaneously achieve suitable QoS levels and to reduce as much as possible the amount of energy used for providing services. We show, through simulation, that our approach is able to dynamically adapt to time-varying workloads (without any prior knowledge) and to significantly reduce QoS violations and energy consumption with respect to traditional static approaches.

Acknowledgements

First and foremost, I want to thank my advisor *Prof. Cosimo Anglano*. I am very grateful to him for providing me the opportunity to conduct my research and constantly supported my work with invaluable advice and comments. He was always there with his help and encouragement whenever I needed it. I was truly fortunate to have had the opportunity to learn from and work with him. His enthusiasm and passion were a source of constant inspiration to me.

I am indebted to *Dr. Massimo Canonico* for his guidance throughout my graduate education. I learned a lot from interactions with Massimo; he kept me focused by always asking sharp and challenging questions and showed me the need to be persistent to accomplish any goal.

I would like to thank my colleagues and especially *Dr. Marco Beccuti*, *Dr. Alessio Bottrighi*, *Andrea Bussi*, *Dr. Davide Cerotti*, *Dr. Giorgio Leonardi*, *Alberto Livio-Beccaria*, *Roberto Pinna*, *Dr. Daniele Codetta-Raiteri*, *Dr. Roberta Terruggia*, and *Matteo Zola* for a very pleasant work environment.

Last but not least, I am extremely grateful to my amazing partner *Lorenza*, for her support along all over the time. She stood by me during the good times and the hard times. With her support, encouraging, and love, she made my journey to the finish line much more enjoyable and unique.

Contents

Mathematical Notation

$\mathbf{0}$	The zero (null) vector
$\mathbf{1}$	The vector or matrix with all elements set to 1
$\mathbf{A}$	A matrix
$\mathbf{A}^T$	The transpose of matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	The inverse of matrix $\mathbf{A}$
$\mathbf{A} > 0$	A positive-definite matrix
$\mathbf{A} \geq 0$	A positive-semidefinite matrix
$\mathbf{A} \diamond \mathbf{B}$	The element-by-element product of two matrices
$\lvert B \rvert$	The cardinality of set B
$\mathbb{C}$	The set of complex numbers
$\mathrm{E}[X]$	The expected value of the random variable X
$f(\cdot)$	A function where the independent variable is left unspecified
$f \star g$	The convolution of f and g
$\mathbf{I}$	The identity matrix
$\max\{x, y\}$	The maximum between x and y
$\min\{x, y\}$	The minimum between x and y
$\mathbb{N}$	The set of natural numbers
$\mathbb{N}^+$	The set of positive natural numbers
$\mathrm{pmin}\{\mathbf{v}, \mathbf{u}\}$	The element-by-element minimum between two vectors
$\mathbb{R}$	The set of real numbers
$\mathbf{v}$	A column vector
$\mathbf{v}^T$	The transpose of vector $\mathbf{v}$
$\mathbf{v} \diamond \mathbf{u}$	The element-by-element product of two vectors
x	A scalar
$\{x(k)\}$	A discrete-time series
$x \triangleq y$	x is defined as y

$\mathbb{Z}$	The set of integer numbers
$\mathbb{Z}^*$	The set of nonnegative integer numbers
$\mathbf{Z}$	The zero (null) matrix
δ_{ij}	Kronecker's delta

Acronyms

AIC	Akaike's Information Criterion
ARMAX	AutoRegressive Moving Average with eXogenous variables
ARX	AutoRegressive with eXogenous variables
BIBO	Bounded-Input Bounded-Output
CaaS	Communication as a Service
CVA	Canonical Variate Analysis
DaaS	Data as a Service
DARE	Discrete-time Algebraic Riccati Equation
DES	Discrete-Event Simulation
DSP	Digital Signal Processing
DVFS	Dynamic Voltage and Frequency Scaling
EW-RLS	Exponential Weighting RLS
IaaS	Infrastructure as a Service
HaaS	Hardware as a Service
LQ	Linear Quadratic
LQI	Linear Quadratic Integral
LQR	Linear Quadratic Regulator
LQRY	Linear Quadratic Regulator with Output Weighting
LS	Least-Squares
LTI	Linear Time-Invariant system
LTV	Linear Time-Varying system
MAPE-K	Monitor, Analysis, Plan, Execute, and Knowledge
MDL	Minimum Description Length
MIMO	Multiple-Input Multiple-Output
MINLP	Mixed-Integer NonLinear Programming
MISO	Multiple-Input Single-Output

MOESP	Multivariable Output-Error State-sPace
MPC	Model Predictive Control
MSE	Mean-Squared Error
N4SID	Numerical algorithm for Subspace State-Space IDentification
OE	Output-Error
PaaS	Platform as a Service
PEM	Prediction-Error Minimization
PID	Proportional-Integral-Derivative
QoS	Quality of Service
RLS	Recursive Least-Squares
RLS-DF	RLS with Directional Forgetting
RLS-DF*	RLS-DF with Bittanti's correction
RLS-EF	RLS with Exponential Forgetting
SaaS	Software as a Service
SASO	Stability, Accuracy, Settling time, and Overshoot
SISO	Single-Input Single-Output
SLA	Service Level Agreement
SLO	Service Level Objective
SOA	Service Oriented Architecture
STR	Self-Tuning Regulation
SVD	Singular-Value Decomposition
TCO	Total Cost of Ownership
VM	Virtual Machine
VMM	Virtual Machine Monitor

Chapter 1

Introduction

Cloud computing [82, 168] is an emerging computing paradigm which is rapidly gaining consideration in the IT industry [133]. Since cloud computing still is in its infancy, there are many open research challenges. In this thesis, we address the problem of power-aware resource management in cloud systems (i.e., in systems that use the cloud computing paradigm) where hosted applications are subjected to performance constraints.

In this chapter, we provide an overview of the problem we tackle in this thesis and the approach we take to solve it. First, we introduce the key motivation that guided us to research on this topic. Second, we present the original contribution provided in this thesis. Finally, we illustrate the outline of this thesis.

1.1 Motivation

Cloud computing is growing in popularity among computing paradigms for its appealing property of considering "Everything as a Service". The building block of cloud computing is the cloud infrastructure, that enables service providers to provision the infrastructure they need (in terms of processing, storage, networks, and other fundamental computing resources) for the delivery of their services without having to buy the resources necessary for running them. Usually, the service and infrastructure providers agree on a prescribed set of service levels, commonly referred to as *Service Level Agreement* (SLA), that is a formal description of tempo-

ral, performance and economical constraints under which hosted services need to operate. Under this agreement, the service provider accepts to pay a certain amount of money for using the infrastructure, and, in turn, the infrastructure provider accepts either to provide enough resources to meet SLAs or to pay a money penalty for each SLA miss.

From the point-of-view of the infrastructure provider, the decision of the amount of computing resources to allocate to a specific service provider may have a critical impact on its revenue. On the one hand, resource under-provisioning may lead to the increment of SLA violations and thus to the reduction of the profit, while, on the other hand, resource over-provisioning may contribute to increase the *Total Cost of Ownership* (TCO), which comprises capital and administrative costs [119]. Thus, the ultimate goal for an infrastructure provider is to maximize its profit by minimizing the number of SLA violations and, at the same time, by reducing the TCO. This is a challenging goal because of the conflicting nature of the two aspects. Indeed, on the one hand the achievement of SLAs would lead the provider to over-provision hosted services (in order to cope with possible peak workload demand), thus increasing the TCO by investment, operating and energy consumption costs. On the other hand, the decrease of TCO (e.g., in terms of energy consumption costs) would lead the provider to under-provision hosted services, thus increasing the possibility to violate some SLA.

Recent studies have reported that energy costs are among the most important factors impacting on TCO, and that this influence will grow in the near future due to the increase of electricity costs [60]. Therefore, the reduction of operating costs is usually pursued through the reduction of the amount of energy absorbed by the physical resources of the infrastructure. Various techniques already exist that aim at reducing the amount of electrical energy consumed by the physical infrastructure of a cloud system, ranging from energy-efficient hardware and energy-aware design strategies, to *server consolidation*, whereby multiple virtual machines run on the same physical resource [166]. Unfortunately, these techniques alone are not enough to guarantee service performance requirements because of the complexity of cloud computing systems, where (1) system resources have to be dynamically shared among several independent services, (2) the requirements of each service must be met in order to avoid economical penalties, (3) the workload of each service

generally changes over time, (4) services may span multiple computing nodes, and (5) system resources may be possibly distributed world-wide. Moreover, the inherently conflicting nature of energy and performance management, along with the complexity of cloud computing systems, makes a manual or semi-automatic approach unsuitable, so that much of current research work is looking for coordinated and fully automated solutions.

1.2 Contributions

In this thesis, we tackle the problem of providing a fully automated solution to the problem of dynamically managing physical resources of a cloud infrastructure, able to achieve the SLA of hosted services and, at the same time, to reduce the energy consumption (and hence to reduce the TCO) induced by the infrastructure for running these services.

Original contributions we provide in this thesis comprise:

1. A framework able to automatically manage physical and virtual resources of a cloud infrastructure in such a way to maximize the profit of the infrastructure provider by minimizing SLA violations while, at the same time, reducing the energy consumed by the physical infrastructure. Specifically, we rely on mechanisms provided by machine virtualization technologies (to achieve a more efficient and flexible use of physical resources) and control-theoretic techniques (as a way for enabling the cloud infrastructure to automatically manage performance and power consumption) to create a decentralized and time-hierarchical framework that combines short-to-medium term allocation decisions (by which the capacity of a given physical resource is assigned to the various virtual machines running on it) with long-term ones (by which a given virtual machine may be migrated from one physical resource to another), and that is able to deal with multi-tier services under very variable and unpredictable workloads.

   Basically, we accomplish this goal by providing each service with the minimum amount of physical resource capacity needed to meet its SLAs, and by dynamically adjusting it according to various parameters, that include

the intensity of its workload, the number of competing virtual machines allocated on the same physical resource, and their time-varying behavior induced by variations in the respective workloads. The rationale underlying this approach is that, in order to balance energy consumption and SLAs satisfaction, each service needs exactly the fraction of physical resource capacity as the one dictated by current operating conditions of the cloud infrastructure. Indeed, on one hand, a greater amount of physical resource capacity would imply an increase of energy consumption without any benefit to the profit. On the other hand, a smaller fraction of physical resource capacity would increase the probability of incurring in a SLA violation.

2. A *Discrete-Event System* (DES) simulator written in C++ (developed as part of this thesis) used to evaluate the performance of our solution. Results, obtained from simulation experiments, show that our solution is able to dynamically adapt to time-varying workloads (without any prior knowledge) and to significantly reduce SLA violations and energy consumption with respect to traditional static approaches, where resource capacity is statically allocated to cloud services in order to either satisfy SLAs or reduce energy consumption induced by the cloud infrastructure.

1.3 Outline of the Thesis

This thesis is divided into five parts, which are structured as follows.

Background Part. In the first part, called "Background", we provide a theoretical background of the most important concepts used in this thesis. Specifically, in Chapter 2 we present main concepts and definitions of the cloud computing paradigm. In Chapter 3, we provide an overview of the theory of linear systems, which are the type of systems that we use in this thesis. In Chapter 4, we present an introduction to the theory of (linear) system identification, which is the theory we use to estimate models of services hosted by the cloud infrastructure. Finally, in Chapter 5, we present an overview of (linear) control theory, with an emphasis on (linear) feedback control, which is the type of control systems we mainly use in

this thesis.

> It is important to note that we tried to keep this part as self-contained as possible, in order to let readers that are unfamiliar with these concepts to be acquainted. Thus, readers that already have experience with this background, can simply skip this part.

Methodology Part. In the second part of the thesis, called "Methodology", we provide insights about the resource management framework proposed in this thesis. This part only contains one chapter, that is Chapter 6, which accurately describes the internal design of our framework.

Experimental Evaluation Part. In the third part of the thesis, called "Experimental Evaluation", we present the experimental evaluation that we perform to assess the performance of the resource management framework proposed in this thesis. Specifically, in Chapter 7, we describe the experimental setup that is common to all of the experiments and we provide an overview of the C++ simulator which we use to perform the experiments. In Chapter 8, we present the performance evaluation of the resource management framework when only the Application and Physical Machine Managers are used. In Chapter 9, we provide the performance evaluation of our framework when all of its components are used; this experimental evaluation is used to assess the impact of the Migration Manager (and, hence, of the virtual machines migration) on application performance and power consumption.

Conclusion Part. In the fourth part of the thesis, called "Conclusion", there is only one chapter, Chapter 11, where we summarize the work described in this thesis, discuss features and limitations of our framework, and present possible future works.

Appendices Part. Finally, at the end of the thesis, there is the "Appendices" part, where we provide a series of appendices in order to collect topics that have been only marginally considered in this thesis.

Part I

Background

Chapter 2

Cloud Computing

Cloud computing has recently emerged as a new form of the utility-based computing paradigm for hosting and delivering hardware and software "as services". It provides its users with the illusion of infinite computing and storage resources which are potentially available on-demand from anywhere and anytime. Cloud computing is attractive since it eliminates the requirement for its users to plan ahead for provisioning, by allowing IT enterprises to start from the small and to increase resources only when there is a rise in service demand. However, despite of this, the development of techniques to make cloud computing effective is currently at its infancy, with many issues still to be addressed.

In this chapter, we provide an overview of main concepts and definitions underlying the cloud computing paradigm. First, in Section 2.1, we define what is cloud computing, we describe its architecture and present the most important technologies that are behind this computing paradigm. Then, in Section 2.2, we concentrate on resource management of cloud computing systems, which is the main research topic of this thesis.

2.1 Overview of Cloud Computing

For years now, there is a trend to make the computing paradigm to behave like traditional public utilities (such as water, electricity, telephony, and gas), whereby the user takes advantage of services provided by these utilities on the basis of its

needs, without taking care of where those services are located or how they are delivered. *Grid computing* [68] (whose name is a metaphor for making computer power as easy to access as an electric power grid), *utility computing* [136], and the more recent *cloud computing* [82, 168] are all examples of such tendency. In particular, cloud computing is recently growing in popularity in the IT industry for its appealing and unique properties whereby, following a "utility" pricing model, users are charged based on their use of computational resources, storage, and transfer of data [133]. A number of organizations are already benefiting of it by hosting and/or offering cloud computing services. Examples include Google Apps [7], Amazon Web Services [3], Microsoft Windows Azure [12], IBM Smart Cloud [8], and Salesforce.com [13].

In this section, we provide an overview of fundamental principles of cloud computing and we present some open research challenges.

2.1.1 Definition of Cloud Computing

The main idea behind cloud computing is not new. It has been around since the 1960's, when John McCarthy, while speaking at the MIT Centennial in 1961, already envisioned that computing facilities will be provided like a public utility [16]:

> "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."

However, it was in 2006, after Eric Schmidt (CEO of Google) used the word "cloud" to describe the provision of services across the Internet as a new business model, that the term began to gain popularity. Ever since then, it has been used mainly as an IT marketing buzzword in a variety of contexts to represent many different ideas. Thus, what is really new in cloud computing is the new operations model that brings together a set of existing technologies to run business in a different way.

Currently, in the scientific literature, there is still no universally agreed-upon definition of what cloud computing is and what unique characteristics it should offer. For instance, Buyya et al. [40] provide the following definition:

> "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers."

Armbrust et al. [26] summarized the main characteristics of cloud computing as:

> "(1) The illusion of infinite computing resources available on demand... (2) The elimination of an up-front commitment by cloud users... (3) The ability to pay for use of computing resources on a short-term basis as needed..."

Mell et al. [117], from NIST, defined cloud computing as:

> "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

While there are a large number of other definitions [161], the majority of them seems to agree on two key characteristics:

- *on-demand self-service*: a consumer can autonomously provision computing capabilities (e.g., computing power, storage space, network bandwidth), that is without requiring human interaction with the respective provider(s);
- *rapid elasticity*: the above capabilities may be dynamically resized in order to quickly scale up (to potentially unlimited size) or down in according to the specific needs of the consumer.

Together, these two properties, which are probably the most prominent reasons behind the success of cloud computing, are transforming the traditional way in which computing resources are used, shifting the focus from the classical vision under which (software and hardware) resources must be purchased before they can be used to run the applications of interest, to a novel one under which "Everything

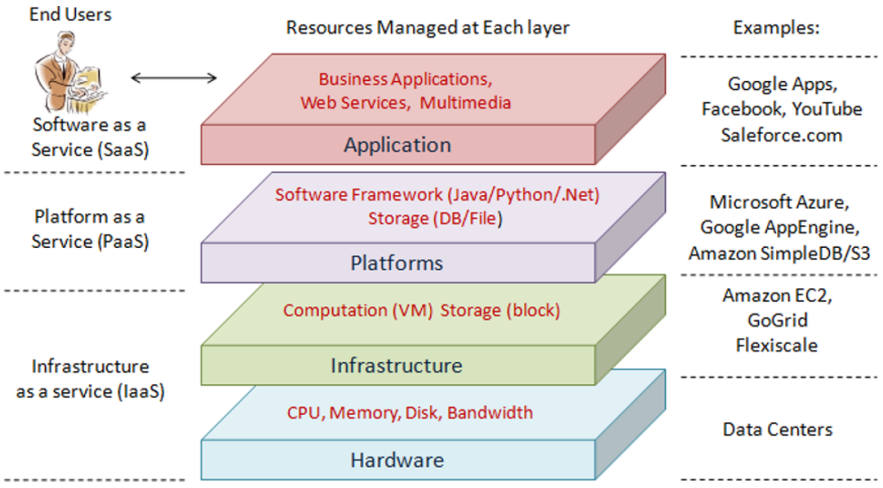


Figure 2.1: The architecture of a cloud system (from [173]).

is a Service", and can be used in a "pay-as-you-go" modality. This service-centric vision permeates all the various components of a computing system, ranging from software to computing platforms, and, finally, to computing infrastructure.

2.1.2 Architecture of a Cloud System

A *cloud system*, that is a system which adopts the cloud computing paradigm, can be characterized by its architecture and the services it offers. The architecture of a cloud computing system is usually structured as a set of layers. A typical architecture of a cloud system is shown in Fig. 2.1 (from [173]). At the lowest level of the hierarchy there is the *hardware layer*, which is responsible for managing the physical resources of the cloud system, such as servers, storage, network devices, power and cooling systems. On the top of the hardware layer, resides the *infrastructure layer*, which provides a pool of computing and storage resources by partitioning the physical resources of the hardware layer by means of virtualization technologies. Built on top of the infrastructure layer, the *platform layer* consists of operating systems and application frameworks. The purpose of this layer is to minimize the burden of deploying applications directly onto infrastructure resources by providing support for implementing storage, database and business logic of cloud applications. Finally, at the highest level of the hierarchy there is the

application layer, which consists of cloud applications.

For what regards services implemented on top of a cloud computing system, they can be provided in three modality, according to the abstraction level of the capability provided and the service model of providers [117]:

- *Infrastructure as a Service* (IaaS), which comprises services to allow its consumers to request computational, storage and communication resources on-demand, thus enabling the so called "pay-per-use" paradigm whereby consumers can pay for exactly the amount of resource they use (like for electricity or water). The consumers can use the provided resources to deploy and run arbitrary software; however, the management and control of the underlying cloud infrastructure is possible only by the provider. An example is Amazon EC2 [1].

- *Platform as a Service* (PaaS), which comprises high-level services providing an independent platform to manage software infrastructures, where consumers (i.e., developers) can build and deploy particular classes of applications using programming languages, libraries, and tools supported by the provider. Usually, consumers don't manage or control the underlying infrastructure (such as servers, network, storage, or operating systems), which can only be accessed by means of the high-level services provided by the provider. An example is Google App Engine [6].

- *Software as a Service* (SaaS), which comprises specific end-user applications running on a cloud infrastructure. Such applications are delivered to consumer as a network service (accessible from various client devices, ranging from desktop computers to smartphones), thus eliminating the need to install and run the application on the consumer's own computers and simplifying maintenance and support. Consumers don't manage or control the underlying infrastructure and application platform; only limited user-specific application configurations are possible. An example is Salesforce.com [13].

Additional service models have been proposed in literature, but generally they can be regarded as a specialization of the above three service models. For instance, in [171] two extensions are proposed: (1) an additional model called *Hardware as a*

Service (HaaS), which comprises services for operating, managing and upgrading the hardware, and (2) a specialization of the IaaS model into three categories:

- *Infrastructure as a Service* (IaaS), which, unlike the above definition, is only concerned to services related to computational resources. An example is Amazon EC2 [1].
- *Data as a Service* (DaaS), which includes services to allow consumers to store their data at remote disks and access them anytime and anywhere. An example is Amazon S3 [2].
- *Communication as a Service* (CaaS), which provides services related to network communication such as *Quality-of-Service* (QoS) management and network security. An example is Microsoft Connected Service Framework [11].

The traditional approach to deploy a cloud system is a public computing system. However, other deployment models are possible which differentiate each others by variations in physical location and distribution. For instance, the following models are taken from NIST [117]:

- *public cloud*: the cloud infrastructure is provisioned for open use by the general public and is made available in a "pay-per-use" manner;
- *private cloud*: the cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple users;
- *community cloud*: the cloud infrastructure is provisioned for exclusive use by a specific community of users from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations);
- *hybrid cloud*: the cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by technology that enables data and application portability. A typical example is when a private cloud is temporarily supplemented with computing capacity from public clouds, in order to manage peaks in load (also known as "cloud-bursting").

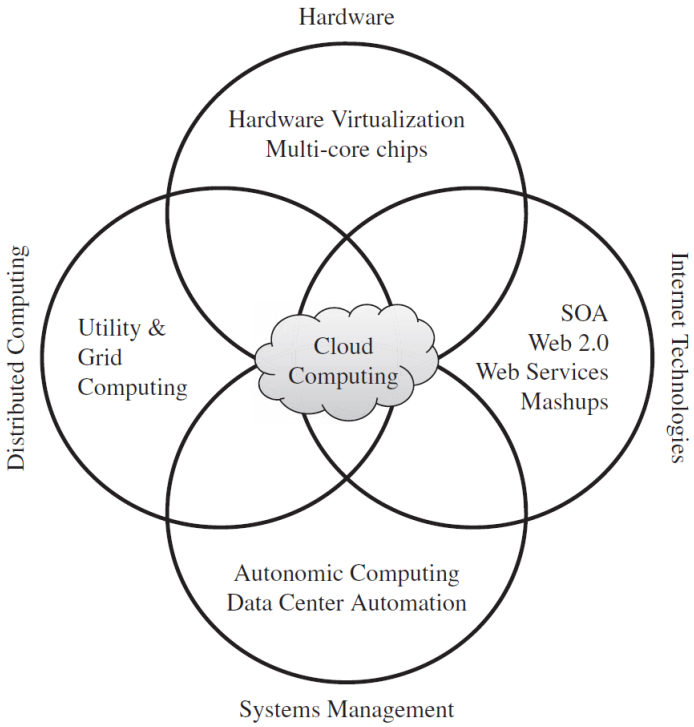


Figure 2.2: Most influential technologies converging in cloud computing (from [38]).

2.1.3 Enabling Technologies

As depicted in Fig. 2.2 (from [38]), cloud computing brings together a set of existing technologies to run business in a different way. In the rest of this section we provide an overview of the most influential technologies enabling cloud computing.

Virtualization

Virtualization is dating back to the early 1960's, when pioneering experimental projects developed it as a way of time-sharing very expensive mainframe computers [17, 87, 149]. Later, in 1972, IBM announced the first release of the *Virtual Machine Facility/370* system, better known as VM/370, an operating system which provided its users with seemingly separate and independent IBM System/360 or System/370 computing systems; this system opened the way to other successful

virtualized systems [55, 74, 132].

Starting from the 1980's, virtualization saw a decrease in popularity primarily due to a change in computing needs; indeed, the decrease in hardware costs favoured the transition from large centralized mainframes towards a collection a several mini- and micro-computers, making the primary motivation to use virtualization (i.e., to increase the level of sharing and utilization of expensive computing resources) disappear. Subsequently, with the introduction of new computing paradigms, like client-server, grid and peer-to-peer computing, new challenging problems, like the need for reliability, security and reduction of management costs, caused the rebirth of the use of virtualization techniques as a way to address theses problems [63, 144]. Virtualization returned to be attractive only recently, when innovations in hardware, like the Intel-VT [155] and the AMD-V [20] architectures, and their wide availability in both server and client platforms, enabled near bare-metal performance for virtualized operating systems.

Nowadays, virtualization has been proven to be an effective way for driving server consolidation [166] since it allows the same physical resource to be shared among multiple applications by deploying each of them into one or more *Virtual Machines* (VMs), each of which representing an isolated runtime environment. The resulting benefit are thus an increase of server utilization and a decrease of management cost.

In the context of cloud computing, resource virtualization is the key ingredient to enable the "on-demand" physical resource provisioning model in the IaaS substrate.

Web Services and Service Oriented Architecture

A cloud computing service is usually exposed as a *Web Service* (WS), which can be defined as [34]:

> "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically *Web Services Description Language* (WSDL) [48, 47]). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages

> [77, 78], typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Thus, WSs provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.

To organize and orchestrate such services, one usually implements a *Service-Oriented Architecture* (SOA) [61]. The purpose of a SOA is to address requirements of loosely coupled, standards-based, and protocol-independent distributed computing. The building blocks of a SOA are "services", which are well-defined, loosely coupled, self-contained modules that provide standard business functionality and are independent of the state or context of other services. Services, rather than embedding calls to each other, use defined protocols that describe how services pass and parse messages using description metadata.

The potential of WSs and SOA is in the possibility they offer for system integration. As a matter of fact, cloud applications can be built as compositions of other services (also referred to as *service mashups*) from the same or different providers.

Utility Computing

The *utility computing* paradigm [136, 145] brings to the IT world the concept of "pay-as-you-go" business model, typical of traditional public utility (like gas, electricity, water and telephony). Under this paradigm, computing resources themselves are considered an utility service as much like water and public telephony. According to [145], utility computing is

> "a collection of technologies and business practices that enables computing to be delivered seamlessly and reliably across multiple computers. Moreover, computing capacity is available as needed and billed according to usage, much like water and electricity are today."

Thus, in the context of utility computing, on one hand, service consumers assign to their jobs a "utility" value, representing the amount they are willing to pay a service provider to get their service level constraints satisfied; while, on the other

hand, service providers attempt to maximize their own utility (which may include their profit).

Cloud and utility computing shares several concepts. Specifically, utility computing can be considered the precursor of cloud computing, which in turn can be viewed as a broader concept which encompasses the utility computing one.

Autonomic Computing

The *autonomic computing* paradigm [98], a term derived from human biology, has been introduced in order to make complex systems able to regulate and maintain themselves without human intervention. Specifically, as the autonomic human nervous system monitors our main vital activities with any our conscious effort, in much the same way a self-managing autonomic system should monitor and manage itself without human involvement. This can be achieved by following the *Monitor, Analysis, Plan, Execute, and Knowledge* (MAPE-K) control loop model [90], whereby the system relies on monitoring probes and gauges (e.g., sensors), on an adaptation engine (e.g., autonomic manager) for computing optimizations based on observed data, and on effectors to act on the system itself. A system, in order to reach the self-managing behavior, should exhibit the so called *Self-** properties: *Self-Configuration* (for automatically configuring its components), *Self-Healing* (for automatically discover and correct system faults), *Self-Optimization* (for automatically controlling and monitoring performance goals) and *Self-Protection* (for automatically protecting and discovering from malicious attacks).

In the context of cloud computing, concepts inspiring autonomic computing could be borrowed to automate the management of the IaaS substrate. As a matter of fact, IaaS is usually incarnated in the form of large and possibly world-wide distributed data centers, that need to managed efficiently. Thus, in this sense, the concepts of autonomic computing may inspire software technologies for data center automation, which may perform tasks such as management of service levels of running applications, management of data center capacity, proactive disaster recovery, and automation of VM provisioning.

2.2 Resource Management in Cloud Systems

Cloud computing is a rather new paradigm and as such there are still a significant number of challenges that need to be addressed. One important challenge faced by cloud infrastructure providers is the effective management of physical resources hosted by the infrastructure. In order to be effective, resource management should take into consideration several factors, such as:

- the sharing of physical resources of the infrastructure among different and independent services;
- the service level agreement between the infrastructure and service provider;
- the cost induced by running hosted services on virtualized resources.

In the following, we provide an overview of each issue. Specifically, first we discuss the first issue in terms of *server consolidation*, then we present the second issue in terms of *quality and availability of services*, and finally we discuss the last issue in terms of *cost management*.

2.2.1 Server Consolidation

The term *server consolidation* is generally used to indicate an approach to maximize physical resource utilization. Currently, the most effective way to drive server consolidation is resource virtualization, by running each service inside the isolated environment offered by a VM. This brings to the problem to find a good placement of VMs onto available physical machines. In its most general form, this is a multi-dimensional mapping problem including as many dimensions as the number of components that need to be virtualized for each resource (e.g., the number of CPU cores, the amount of main memory, the amount of disk space, and the network bandwidth).

Traditional approaches usually determine a unique VMs placement before deploying a service in the production environment. However, such approaches are not suitable to cloud system due to the heterogeneity and the time-variability of the workload under which hosted services are subjected, which cause changes in the operative conditions of the services and, if not taken into consideration, may

increase the probability of a SLA violation. To this end, VM *live migration* [50] can be used to dynamically consolidate VMs onto more suitable physical machines.

The problem of optimally consolidating servers in a data center is often formulated as a variant of the *multiple multidimensional variable-size bin-packing problem* [45], which is an NP-hard optimization problem and thus, since cannot be solved in a reasonable amount of time, it is not suited for very large-sized problems like the ones arising in cloud computing.

2.2.2 Service Quality and Availability

One of the benefit brought by cloud computing is the possibility offered to its users to adopt the "pay-per-use" business model for renting resources where running own services. As a consequence, users will expect to have certain assurances about the service level to be provided once their applications run in the cloud system. These expectations include the availability of the service, its end-to-end performance, and the measures that are to be taken in case of a system failure. To this end, the service consumer and provider typically agree on a prescribed set of service levels, commonly referred to as *Service Level Agreement* (SLA), under which service provider agrees on paying a fee to the service provider for using the service, and the service provider agrees on providing enough resources to meet the service levels defined by the SLA.

Thus, SLA constraints have to be taken into consideration in the VMs placement problem, which is usually restated as a utility-maximization problem. Since the VMs placement is NP-hard, the restated problem remains in this class of problems, and hence the globally optimal solution cannot be computed in a reasonable amount of time for systems of realistic size.

2.2.3 Cost Management

The ultimate goal of a cloud service provider is the maximization of its profit. In addition to reduce the number of SLA violations (and hence to reduce the amount of monetary penalty to possibly pay to service consumer), one way to increase the profit is to reduce the *Total Cost of Ownership* (TCO), which comprises capital and administrative costs [119].

It has been argued [60] that energy costs are among the most important factors impacting on TCO, and that this influence will grow in the near future due to the increase of electricity costs. Therefore, the reduction of operating costs is usually pursued through the reduction of the amount of energy absorbed by the physical resources of the data center. Various techniques already exist that aim at reducing the amount of electrical energy consumed by the physical infrastructure underlying the IaaS substrate, ranging from energy-efficient hardware and energy-aware design strategies, to server consolidation, whereby multiple virtual machines run on the same physical resource [166]. Moreover, VM live migration can be used to dynamically consolidate VMs residing on multiple under-utilized servers onto a single server, so that the remaining servers can be set to an energy-saving state. Unfortunately, these techniques alone are not enough to guarantee application performance requirements because of the complexity of cloud computing systems, where (1) system resources have to be dynamically and unpredictably shared among several independent applications, (2) the requirements of each application must be met in order to avoid economical penalties, (3) the workload of each application generally changes over time, (4) applications may span multiple computing nodes, and (5) system resources may be possibly distributed world-wide.

Chapter 3

Linear Systems Theory

Linear System Theory [146] investigates the behavior and the response of a linear system to arbitrary input signals. In this thesis, we use linear system theory as a mean to characterize the behavior of computing systems. Specifically, the class of dynamical systems we are interested in are *discrete-time linear time-invariant causal systems*, which are the ones that are usually employed to describe current computing systems from the control-theoretic point-of-view [84].

> In this thesis, we focus on discrete-time linear time-invariant casual systems.

In the rest of this chapter, we present an overview of some of the key concepts and results related to discrete-time linear systems, especially from the perspective of identification and control (topics that will be covered in the subsequent chapters). First, in Section 3.1 we provide a characterization of dynamical systems, with a particular emphasis on linear systems. Then, in Section 3.2, we discuss some technique to approximate nonlinear systems with linear ones. In Section 3.3, we present main representations of linear systems. In Section 3.4, we introduce the realization theory, which provides a bridge between the state-space and the input-output representations. Finally, in Section 3.5, we discuss about the stability of linear systems.

3.1 Characterization of Dynamical Systems

A *system* is a collection of interacting components. An electric motor, an airplane, and a biological unit such as the human arm are examples of systems. A *dynamical system* is a system consisting of a set of possible states, together with a rule that determines the present state in terms of past and future states [18]. A more formal definition is provided by [94]:

Definition 3.1.1 (Dynamical System)**.** A *dynamical system* is composed of three parts:

- the *state* of a system, which is a representation of all the information about the system at some particular value of an *independent variable* (e.g., the time);
- the *state-space* of a system, which is a set that contains all of the possible states to which a system can be assigned;
- the *state-transition* function that is used to update and change the state from one moment to another.

Usually the independent variable denotes the *time*; however, it can represent other type of information, as is the case of image processing, where the independent variable denotes the *space*.

Dynamical systems can be characterized by the interaction of different type of *signals*, which are quantities that change their value as a function of an independent variable (which, in this thesis, is the time).

Definition 3.1.2 (Signal)**.** A *signal* g is a uniquely defined mathematical function (single-valued function) of an independent variable k. The set for which the independent variable k is defined, is called the *domain* of the signal.

To denote the "whole signal", that is the sequence of values $g(k)$, for every k, we use the notation $g(\cdot)$ or g; while, to denote the value of the signal at a specific k, we use the notation $g(k)$. A vector-valued signal $\mathbf{g}(\cdot)$ is a signal whose values are vectors. If the domain of the signal represents discrete time, then $g(\cdot)$ defines a *discrete-time signal*. [1] In Fig. 3.1 is shown an example of discrete-time signal.

[1] Often, especially in the digital signal processing (DSP) community, discrete signals are denoted

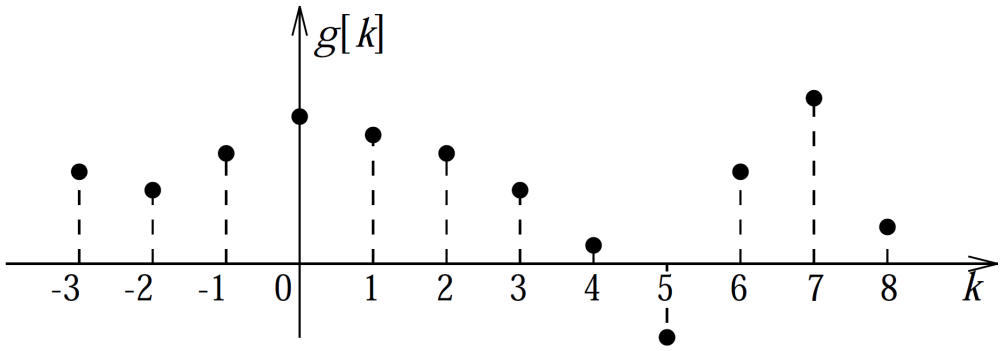


Figure 3.1: A discrete signal.

In this thesis, we assume that the independent variable represents the time and that it takes values on the set $\mathbb{N}$; thus, we are only concerned to discrete-time signals.

Signals interacting in a dynamical system can be classified into:

- *Inputs* (or exogenous variables) $\mathbf{u}(\cdot)$, which consist of variables from the environment that influence the system. They are the only variables that can be manipulated by the user in order to have effect on the system.
- *Outputs* (or endogenous variables) $\mathbf{y}(\cdot)$, that quantify effects of inputs on the system.
- *States* $\mathbf{x}(\cdot)$, which represent the internal state of the system and provide a way to characterize the effects of the input on the produced output and on the future state of the system.

When the input signal $\mathbf{u}(\cdot)$ takes scalar values, the system is called *single-input* (SI); otherwise, it is called *multiple-input* (MI). When the output signal $\mathbf{y}(\cdot)$ takes scalar values the system is called *single-output* (SO); otherwise, it is called *multiple-output* (MO). Thus, a SISO system is a single-input single-output system, a MISO system is a multiple-input single-output system, and a MIMO system is a multiple-input multiple-output system.

with $g[\cdot]$, in order to distinguish them from continuous signals, which are instead denoted with $g(\cdot)$.

Dynamical systems can evolve either in *continuous-time* or in *discrete-time*; in the former, the independent variable is defined over some continuous time interval (e.g., the set $\mathbb{R}$), while in the latter, is defined over a discrete interval (e.g., the set $\mathbb{N}$). A dynamical system is *causal* (or *nonanticipatory*) if its current output depends on past and current inputs but not on future input; conversely, a *noncausal* (or *anticipatory*) system can predict or anticipate what will be applied in the future. Physical systems are usually considered causal systems; noncausal systems can be found in off-line post-processing systems, like image or audio processing. We say that a dynamical system is *memoryless* (or without memory) if its output, for each value of the independent variable, is dependent only on the input evaluated at the same value of the independent variable. A system that is not memoryless is said to *have memory*. All memoryless systems are also causal, since they depend only on its current input; the converse, in general, is not true. A system is said to be *time-invariant* if a time shift in the input causes a corresponding time shift in the output. If a system is not time-invariant, it is said to be *time-varying*.

A system is *linear* if it obeys to the *principle of superposition*, whereby the response to a weighted sum of any two inputs is the (same) weighted sum of the responses to each individual input. More formally, the system $y = T(u)$ is linear, if T is a *linear operator* that is, for any valid inputs u_1 and u_2, and for any $\alpha, \beta \in \mathbb{R}$:

$$\begin{aligned} y_3 &= T(\alpha u_1 + \beta u_2) \\ &= \alpha T(u_1) + \beta T(u_2) \\ &= \alpha y_1 + \beta y_2 \end{aligned} \tag{3.1}$$

A system that does not obey to the superposition principle is called *nonlinear*.

In this thesis, we are only concerned to casual discrete-time linear systems.

3.2 Linearization

Linear systems are especially appealing for their mathematical properties that make the solution of modelling equations simpler than the ones available for nonlinear

systems. However, it is important to note that many physical systems (included computing systems) are usually best described by nonlinear models, mainly due to their complexity and to saturation phenomena [84]. Unfortunately, nonlinear models generally make complex the system design. A standard way to cope with this situation is the use of *linearization* in order to approximate a nonlinear relationship with a linear one. The most common approaches to linearization include [24, 86, 146]:

- *Local linearization around an equilibrium point*, whereby the original nonlinear system is restated in terms of small perturbations with respect to such nominal point. In this case, the idea is that the solution of the slightly perturbed point should differ only slightly from the solution of the equilibrium point. Thus, the obtained linear system is an approximation of the behavior of the nonlinear system for inputs that are "close" to such equilibrium point.

- *Local linearization around a trajectory*, which is similar to the linearization around an equilibrium point, but perturbations are computed with respect to a known solution (i.e., a know input, output and initial conditions) of the nonlinear system equations. In this case, the resulting linear system is an approximation of the behavior of the nonlinear system for inputs (and initial conditions) that are "close" to such solution;

- *Feedback linearization*, which is a way of transforming the original nonlinear system model into an equivalent one of a simpler form; unlike local linearization, it achieves an exact transformation of the dynamics of the original nonlinear system, rather than a linear approximation of them. The central idea is to algebraically transform nonlinear systems dynamics into (fully or partly) linear ones, so that linear control techniques can be applied. This is achieved by simplifying the form of the nonlinear system by choosing a representation that leads to a linear system. The main disadvantage of this approach is that it has a limited applicability.

In this thesis, we use the technique of local linearization around an equilibrium point.

3.3 System Representation

A dynamical system can be analyzed both with graphical and with analytical models. In the following, we present some commonly used analytical and graphical modeling tools. First, we present two commonly used analytical modeling tool: the *input-output* and the *state-space* representations. Then, we introduce the *block diagram algebra* as an example of graphical modeling.

3.3.1 Input-Output Representation

If the internal structure (i.e., the state) of the system is unknown, it may still be possible to use an external description of the system, called *input-output representation*, that relates the next system output $\mathbf{y}(k)$ (at time k) to present and past system inputs $\mathbf{u}(k)$, for $k = k - n_b, \ldots, k$, and to past system outputs $\mathbf{y}(k)$, for $k = k - n_a, \ldots, k - 1$. Specifically:

$$\mathbf{y}(k) = f\left(k; \mathbf{y}(k-n_a), \ldots, \mathbf{y}(k-1); \mathbf{u}(k-n_b), \ldots, \mathbf{u}(k)\right) \tag{3.2}$$

For linear systems, (i.e., when the function f is linear), such representation is an algebraic recursive equation called *linear recurrence equation*, which is usually stated as a *difference equation*:

$$\sum_{i=0}^{n_a} \mathbf{A}_i(k)\mathbf{y}(k-i) = \sum_{i=0}^{n_b} \mathbf{B}_i(k)\mathbf{u}(k-i) \tag{3.3}$$

where the independent variable $k \in \mathbb{N}$ is the model *sampling time* (or *discrete time step*). Vectors $\mathbf{u}(\cdot) \in \mathbb{R}^{n_u}$ and $\mathbf{y}(\cdot) \in \mathbf{R}^{n_y}$ are the *input vector* and the *output vector* of the system, respectively. Matrices $\mathbf{A}_i(\cdot) \in \mathbb{R}^{n_y \times n_y}$, for $i = 1, \ldots, n_a$, and $\mathbf{B}_i(\cdot) \in \mathbb{R}^{n_y \times n_u}$, for $i = 1, \ldots, n_b$, are the *parameters* of the system. The integer value n_a, which is the order of the difference equation, is called the *system order* and it reflects the extent to which previous inputs and outputs affect the current output. The use of difference equations is motivated by the fact that they are useful for relating the evolution of variables (or parameters) from one discrete instant of time (or other independent variable) to another.

An alternative notation for Eq. (3.3) is sometimes used in order to separate the

sampling time and the discrete time step:

$$\sum_{i=0}^{n_a} \mathbf{A}_i(k\tau)\mathbf{y}((k-i)\tau) = \sum_{i=0}^{n_b} \mathbf{B}_i(k\tau)\mathbf{u}((k-i)\tau) \tag{3.4}$$

where the independent variable $k \in \mathbb{N}$ is the model *sampling time* multiplied by the *discrete time step* τ. This notation is used, for instance, to represent *sampled data systems* [69], which are continuous-time systems that are controlled with a digital device.

When all the parameter matrices of Eq. (3.3) are constant $\forall k \in \mathbb{N}$, the system is called a *linear time-invariant* (LTI) system and is written as:

$$\sum_{i=0}^{n_a} \mathbf{A}_i\mathbf{y}(k-i) = \sum_{i=0}^{n_b} \mathbf{B}_i\mathbf{u}(k-i) \tag{3.5}$$

Otherwise, the system is called a *linear time-varying* (LTV) system.

To obtain a unique solution for $\mathbf{y}(k)$, two additional items must be specified, the *initial time* $k_0 \in \mathbb{N}$ (which defines the time sequence over which a solution is desired), and a set of n_a *initial conditions* for $\mathbf{y}(k)$.

$$\mathbf{y}(k_0), \mathbf{y}(k_0+1), \ldots, \mathbf{y}(k_0+n_a-1) \tag{3.6}$$

The problem defined over this initial time and with these initial conditions is called an *initial-value problem*. For convenience, in the rest of this section, we assume that the initial time $k_0 = 0$.

Given a linear system and an initial-value problem, we can study the system response (i.e., its solution) to different input signals. In the same way, given a linear system and an input signal, we can study the system response to different initial conditions. From the principle of superposition, the solution of the linear system Eq. (3.3) can be studied by analyzing separately the solutions:

1. *Forced response*: the solution of the system corresponding to a given input $\mathbf{u}(\cdot)$ and with zero initial conditions.

2. *Free response*: the solution of the system corresponding to a given initial-value problem and with the input set to zero.

The *total response*, that is the solution of Eq. (3.3) for a given initial-value problem and input signal, is thus given by the sum of the two separate solution.

The *forced response* of a linear system can be stated in terms of the *impulse response* function, which, intuitively, represents the response of the system to a brief input signal, called an *impulse*.

Definition 3.3.1 (Impulse)**.** A *discrete-time impulse* $\delta(k)$ (or *unit pulse*, or *unit sample*) corresponds to a unit discrete-time pulse, that is:

$$\delta(k) = \begin{cases} 0, & k \in \mathbb{N}\backslash\{0\}, \\ 1, & k = 0 \end{cases} \tag{3.7}$$

Definition 3.3.2 (Impulse Response)**.** Given a discrete-time linear system in input-output representation, there exists a matrix-valued signal $\mathbf{h}(n,k) \in \mathbb{R}^{n_y \times n_u}$ called discrete-time *impulse response matrix* (or simply *impulse response*), such that for every input $\mathbf{u}(\cdot)$, a corresponding output $\mathbf{y}(k)$ is given by the following *superposition sum*:

$$\mathbf{y}(k) = \sum_{n \in \mathbb{N}} \mathbf{h}(k,n)\mathbf{u}(n), \quad k \in \mathbb{N} \tag{3.8}$$

Each element $h_{ij}(k,n)$ of the impulse response can be viewed as the ith entry of an output at time n, corresponding to a unit discrete-time pulse applied at the jth input at time k.

Definition 3.3.3 (Forced Response)**.** Given a discrete-time linear system in input-output representation, the *forced response* is given by Eq. (3.8), and represents the solution of the system when all of its initial conditions are zero.

It is worth noting that, the impulse response is the forced response of the system to a unit impulse. Often, it is of particular interest to compute to forced response to well-known input signals. For instance, in control-theoretic system analysis, a particular family of *singularity functions* is used extensively. The three most widely used singularity functions are the *unit step*, the *unit impulse*, and the *unit ramp* (see Fig. 3.2).

The impulse response has several important properties. Among them, it is important to mention the following ones:

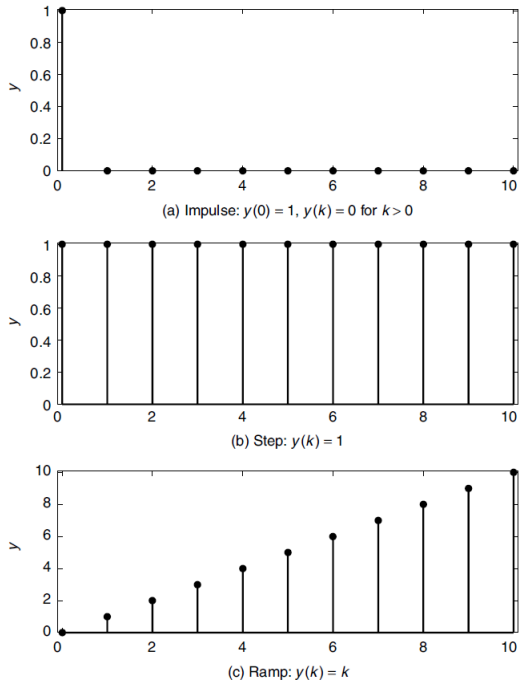


Figure 3.2: Common discrete-time signals (singularity functions).

1. For causal systems, one can choose the impulse response to satisfy:

$$\mathbf{h}(k,i) = 0, \quad \forall k,i \in \mathbb{N} : i > k \tag{3.9}$$

2. For LTI systems, one can choose the impulse response to satisfy:

$$\begin{aligned} \mathbf{h}(k,i) &= \mathbf{h}(k-i,0) \\ &\triangleq \mathbf{h}(k-i), \quad \forall k,i \in \mathbb{N} : k \geq i \end{aligned} \tag{3.10}$$

3. For causal LTI systems, we can write Eq. (3.8) as the *convolution sum* $\star$ of the impulse response matrix and the input:

$$\begin{aligned} \mathbf{y}(k) &\triangleq (\mathbf{h} \star \mathbf{u})(k) \\ &= \sum_{i=0}^{k} \mathbf{h}(k-i)\mathbf{u}(i), \quad i \in \mathbb{N} \end{aligned} \tag{3.11}$$

For causal LTI systems, the last property provides a convenient way to compute the impulse response.

In addition to the forced response, the other term needed to compute the total response is the *free response*.

Definition 3.3.4 (Free Response)**.** The *free response* (also known as *natural response*) of an linear system is the solution of Eq. (3.3) when the input sequence is identically zero, that is the solution of the following *characteristic equation*:

$$\sum_{i=0}^{n_a} \mathbf{A}_i(k)\mathbf{y}(k-i) = 0 \tag{3.12}$$

Finally, for the superposition principle, we can compute the total response of a system in the following way.

Definition 3.3.5 (Total Response)**.** The *total response* of a linear system, for a given initial-value problem and input signal, is the sum of the forced and the free responses:

$$\text{total response} = \text{free response} + \text{forced response} \tag{3.13}$$

The total response can also be characterized by two other quantities: the *transient response* and the *steady-state response*.

Definition 3.3.6 (Transient Response)**.** The *transient response* is the part of the total response that does not approach zero as time approaches infinity.

Definition 3.3.7 (Steady-state Response)**.** The *Steady-state response* is the part of the total response that approaches zero as time approaches infinity.

The results discussed so far are related to the characterization of linear systems on the *time domain*; the fundamental result in this representation is that a linear system can be characterized entirely by its initial-value problem and by a single function called the impulse response. For LTI systems, an equivalent characterization is available in the *frequency domain*, whereby any LTI system can be characterized by a single function called *transfer function*, which is the *z-transform* (see Appendix A) of system's impulse response. [2]

[2] An operational equivalent of the transfer function for the LTV case is more problematical and is still an open research problem; some result is available in literature (e.g., see [19, 95, 126, 172]).

Definition 3.3.8 (Transfer Function)**.** The *transfer function* $\mathbf{H}(z)$ of an LTI system for a given input signal is the z-transform of the system's impulse response $\mathbf{h}(k)$.

From this definition, it follows that, as the impulse function completely characterizes the system response $\mathbf{y}(\cdot)$ for a given input signal $\mathbf{u}(\cdot)$ at *zero initial conditions* in the time domain, in a similar way, the transfer function $\mathbf{H}(z)$ completely characterizes the system response $\mathbf{Y}(z)$ for a given input $\mathbf{U}(z)$ at *zero initial conditions* in the frequency domain, that is:

$$\mathbf{Y}(z) = \mathbf{H}(z)\mathbf{U}(z) \tag{3.14}$$

It is worth noting that the assumption of *zero initial conditions* is not a limitation because of the principle of superposition, whereby the total system response can be computed as the sum of the system response at zero initial conditions and the one with input signal at zero, as stated by Eq. (3.13).

For the special case of SISO LTI systems, the transfer function $H(z)$ of the system (at zero initial condition) is given by the ratio between the z-transform $Y(z)$ of the output signal $y(\cdot)$ and the z-transform $U(z)$ of the input signal $u(\cdot)$:

$$H(z) = \frac{Y(z)}{U(z)} \tag{3.15}$$

Thus, intuitively, a SISO transfer function describes how an input $U(z)$ is transformed into the output $Y(z)$, or, put differently, it describes the response to a unit impulse. The denominator of $H(z)$ is called the *characteristic polynomial*, which, when set to zero, takes the name of *characteristic equation*; its degree is called the *degree of the transfer function* and its roots are called the *poles of the transfer function*. According to the *fundamental theorem of algebra*, the number of poles of the transfer function is equal to its degree. The roots of the numerator of $H(z)$ are called the *zeros of the transfer function*. When the polynomials $Y(z)$ and $U(z)$ are *coprime*, [3] then the zeros are simply the roots of $Y(z)$ and the poles are the roots of $U(z)$.

[3] Two polynomials are *coprime* if they have no common roots.

To illustrate, the transfer function of the SISO LTI system:

$$\sum_{i=0}^{n_a} a_i y(k-i) = \sum_{i=0}^{n_b} b_i u(k-i)$$

is given by:

$$H(z) = z^{n_a-n_b} \frac{b_0 z^{n_b} + b_1 z^{n_b-1} + \cdots + b_{n_b}}{a_0 z^{n_a} + a_1 z^{n_a-1} + \cdots + a_{n_a}}$$

The definition of transfer function can be extended to the MIMO case, where now the effect of every input on every output has to be taken into consideration. Specifically, for MIMO LTI systems, the *transfer function matrix* (or, simply, the *transfer matrix*) $\mathbf{H}(z) \in \mathbb{C}^{n_y \times n_u}$ relates input $\mathbf{U}(z)$ with output $\mathbf{Y}(z)$

$$\mathbf{Y}(z) = \mathbf{H}(z)\mathbf{U}(z) \tag{3.16}$$

such that each element $H_{ij}(z)$ of $\mathbf{H}(z)$ is a SISO transfer function relating the j-th component $U_j(z)$ of the input to the i-th component $Y_i(z)$ of the output. A possible generalization of poles and zeros concepts for a MIMO transfer function $\mathbf{H}(z)$ turn out to be as follows:

- the poles of $\mathbf{H}(z)$ are the values of $z \in \mathbb{C}$ for which at least one of the entries of $\mathbf{H}(s)$ becomes unbounded;
- the *rank* [4] of $\mathbf{H}(z)$ takes the same value for almost all values of $z \in \mathbb{C}$, but for some $z \in \mathbb{C}$, the rank of $\mathbf{H}(z)$ drops; these values are called the *transmission zeros* of $\mathbf{H}(z)$.

For a formal treatment of MIMO transfer functions, the interested reader can refer to [24, 46, 86].

The frequency-domain representation is particularly convenient for the analysis and the manipulation of linear systems. Given a frequency-domain representation $\mathbf{Y}(z)$ of the system response $\mathbf{y}(\cdot)$ for a given input signal $\mathbf{u}(\cdot)$ at zero initial conditions, it is always possible to return to the time-domain counterpart by taking its inverse z-transform $\mathscr{Z}^{-1}\{\mathbf{Y}(z)\}$. In particular, as shown in Fig. 3.3, when all the

[4] The rank of a matrix $\mathbf{A}$ is equal to the number of linearly independent columns of $\mathbf{A}$, which is also equal to the number of linearly independent rows of $\mathbf{W}$.

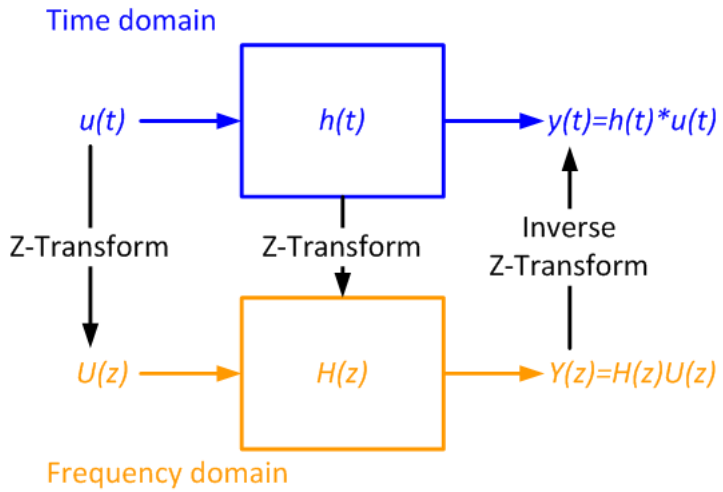


Figure 3.3: Relationship between the *time-domain* and the *frequency-domain* representation.

terms due to initial conditions are zero, the output of a linear system is given by the convolution of the impulse response and the input signal functions in the time domain, while it is simply the product of the transfer function and the transform of the input in the frequency domain. This equivalence, depicted in Fig. 3.3, can be expressed by the following relationship:

$$\begin{aligned} \mathbf{y}(k) &= \mathscr{Z}^{-1}\{\mathbf{Y}(z)\} \\ &= \mathscr{Z}^{-1}\{\mathbf{H}(z)\mathbf{U}(z)\} \\ &= (\mathbf{h} \star \mathbf{u})(k) \end{aligned} \tag{3.17}$$

3.3.2 State-Space Representation

When the internal structure (i.e., the state) of the system is known, the most general form to represent a dynamical system is the *state-space representation*:

$$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u}(k), \tag{3.18a}$$

$$\mathbf{y}(k) = \mathbf{C}(k)\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k) \tag{3.18b}$$

where Eq. (3.18a) is called *state (difference) equation*, and Eq. (3.18b) is called *output equation*. The independent variable $k \in \mathbb{N}$ is the model *sampling time* (or

discrete time step). Vectors $\mathbf{x}(\cdot) \in \mathbb{R}^{n_x}$, $\mathbf{u}(\cdot) \in \mathbb{R}^{n_u}$, and $\mathbf{y}(\cdot) \in \mathbf{R}^{n_y}$ are the *state vector*, the *input vector*, and the *output vector* of the system, respectively. Matrices $\mathbf{A}(\cdot) \in \mathbb{R}^{n_x \times n_x}$, $\mathbf{B}(\cdot) \in \mathbb{R}^{n_x \times n_u}$, $\mathbf{C}(\cdot) \in \mathbb{R}^{n_y \times n_x}$, and $\mathbf{D}(\cdot) \in \mathbb{R}^{n_y \times n_u}$ are the *parameters* of the system and are called the *state matrix*, the *input matrix*, the *output matrix*, and the *feedforward matrix*, respectively. The integer value n_x, that is the dimension of the state vector, is called the *system order*.

According to this representation, for a given input $\mathbf{u}(\cdot)$, we need to solve the state (difference) equation to determine the state $\mathbf{x}(\cdot)$ and then replace it in the output equation to obtain the output $\mathbf{y}(\cdot)$. In general, one input corresponds to several possible outputs; in order to fix it, one has to provide the *initial value* (or *initial condition*): $\mathbf{x}(k_0) = \mathbf{x}_0$. In rest of this section, we assume that $k_0 = 0$.

When all the parameter matrices of Eq. (3.18) are constant $\forall k \in \mathbb{N}$, the above system is called a *linear time-invariant* (LTI) system and is written as:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k), \tag{3.19a}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k), \tag{3.19b}$$

Otherwise, the system is called a *linear time-varying* (LTV) system.

LTI state-space models are usually denoted with the tuple of their system matrices; thus, the model Eq. (3.19) can be replaced by the following notation:

$$(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}) \tag{3.20}$$

In a similar way to the input-output representation, the *total response* of a linear system in the state-space representation can be computed as the sum of two independent terms, namely the *zero-state response* and the *zero-input response*, and can expressed both in the time and in the frequency domain [24]. In order to obtain a unique solution from Eq. (3.18), initial conditions $\mathbf{x}(k_0)$ and the initial time $k_0 \in \mathbb{N}$ must be specified; this is known as the *initial-value problem*. In the rest of this section, we assume that $k_0 = 0$.

Definition 3.3.9 (Zero-input Response)**.** The *zero-input response* is the response (solution) of the system when the input $\mathbf{u}(\cdot)$ is identically zero.

For instance, for the LTI system Eq. (3.19), the zero-input response in time-

domain is given by:

$$\mathbf{y}_{\text{zi}} = \mathbf{C}\mathbf{A}^k\mathbf{x}(0) \tag{3.21}$$

Definition 3.3.10 (Zero-state Response)**.** The *zero-state response* is the response (solution) of the system when all of its initial conditions $\mathbf{x}(\cdot)$ are zero.

For instance, for the LTI system Eq. (3.19), the zero-state response in time-domain is given by:

$$\mathbf{y}_{\text{zs}}(k) = \mathbf{C}\sum_{i=0}^{k-1}\mathbf{A}^{k-(i+1)}\mathbf{B}\mathbf{u}(i) + \mathbf{D}\mathbf{u}(k) \tag{3.22}$$

The zero-state response can also be stated in terms of the discrete-time *unit impulse response matrix* $\mathbf{h}(\cdot)$:

$$\mathbf{y}_{\text{zs}}(k) = \sum_{i=0}^{k}\mathbf{h}(k,i)\mathbf{u}(i) \tag{3.23}$$

which for the system Eq. (3.19) is given by:

$$\begin{aligned}\mathbf{h}(k,n) &\triangleq \mathbf{h}(k-n)\\ &= \begin{cases}\mathbf{C}\mathbf{A}^{k-(n+1)}\mathbf{B}, & k > n,\\ \mathbf{D}, & k = n\end{cases}\end{aligned} \tag{3.24}$$

Definition 3.3.11 (Total Response)**.** For a given initial-value problem $\mathbf{x}(0)$ and an input $\mathbf{u}(\cdot)$ the *total response* of Eq. (3.18) is given by:

$$\text{total response} = \text{zero-input response} + \text{zero-state response} \tag{3.25}$$

The total response of Eq. (3.19) in time-domain, with initial conditions $\mathbf{x}(0)$, is given by:

$$\begin{aligned}\mathbf{y}(k) &= \mathbf{y}_{\text{zi}}(k) + \mathbf{y}_{\text{zs}}(k)\\ &= \mathbf{C}\Big(\mathbf{A}^k\mathbf{x}(0) + \sum_{i=0}^{k-1}\mathbf{A}^{k-1-i}\mathbf{B}\mathbf{u}(i)\Big) + \mathbf{D}\mathbf{u}(k)\end{aligned} \tag{3.26}$$

The total response can also be expressed in the frequency domain by means of the z-transform. For instance, the total response in frequency domain of a LTI

system is obtained by taking the z-transform of Eq. (3.19):

$$z\mathbf{X}(z) - z\mathbf{x}(0) = \mathbf{A}\mathbf{X}(z) + \mathbf{B}\mathbf{U}(z), \tag{3.27a}$$

$$\mathbf{Y}(k) = \mathbf{C}\mathbf{X}(z) + \mathbf{D}\mathbf{U}(z), \tag{3.27b}$$

and then solving for the output $\mathbf{Y}(z)$:

$$\mathbf{Y}(z) = \mathbf{C}\big(z\mathbf{I} - \mathbf{A}\big)^{-1} z\mathbf{x}(0) + \Big[\mathbf{C}\big(z\mathbf{I} - \mathbf{A}\big)^{-1}\mathbf{B} + \mathbf{D}\Big]\mathbf{U}(z) \tag{3.28}$$

The output signal $\mathbf{y}(\cdot)$ in the time domain can be recovered from its one-sided z-transform $\mathbf{Y}(z)$, by applying the inverse z-transform $\mathscr{Z}^{-1}\{\mathbf{Y}(\mathbf{z})\}$.

Just like in the time domain, where the impulse response relates the output signal to a particular input signal (at zero initial conditions), in the frequency domain we can introduce an equivalent concept, namely the transfer function.

Definition 3.3.12 (Transfer Function)**.** The *transfer function matrix* $\mathbf{H}(z)$ of system Eq. (3.19) relates the z-transform of the output $\mathbf{y}(\cdot)$ to the z-transform of the input $\mathbf{u}(\cdot)$ *under the assumption that* $\mathbf{x}(0) = 0$, such that:

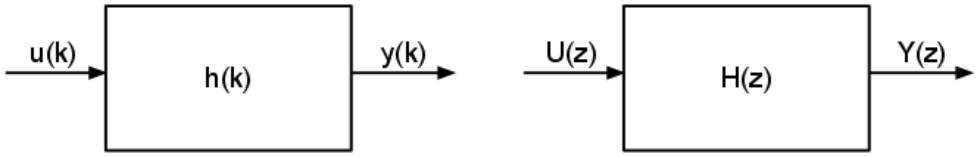
$$\mathbf{Y}(z) = \mathbf{H}(z)\mathbf{U}(z) \tag{3.29}$$

where:

$$\begin{aligned} \mathbf{H}(z) &\triangleq \mathscr{Z}\{\mathbf{h}(k)\} \\ &= \mathbf{C}\big(z\mathbf{I} - \mathbf{A}\big)^{-1}\mathbf{B} + \mathbf{D} \end{aligned} \tag{3.30}$$

is the z-transform of the impulse response matrix $\mathbf{h}(\cdot)$ (see Eq. (3.24)).

With respect to the input-output representation, the state-space representation provides a convenient and compact way to model and analyze MIMO systems. With n_u inputs and n_y outputs, we would otherwise have to write down $n_u \times n_y$ transfer functions to encode all the information about a system. Moreover, unlike the frequency-domain approach, the use of the state-space representation is not limited to systems with linear components and zero initial conditions. Another advantage of state-space representation, is that system dynamics are described by first-order difference equations; instead, with the input-output representation,



(a) Time-domain block. (b) Frequency-domain block.

Figure 3.4: Transfer function block.

system dynamics can require high-order difference equations, thus making the analysis of the system harder.

3.3.3 Block Diagrams Algebra

Block diagrams are a shorthand, graphical representation of a physical system, illustrating the functional relationships among its components in terms of cause-and-effect relationships between its inputs and its outputs.

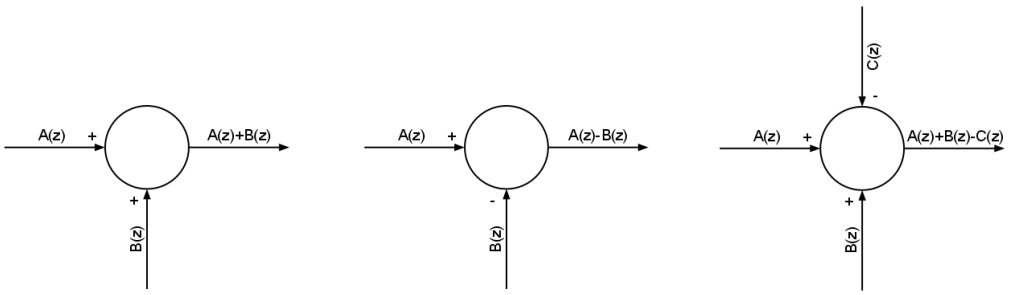
The basic building blocks of a block diagram include:

- *Transfer function block*, which represents the impulse response $\mathbf{h}(k)$ (see Fig. 3.4a) or the transfer function $\mathbf{H}(z)$ (see Fig. 3.4b) of an LTI system in the time-domain or frequency-domain, respectively. The latter (i.e., the frequency-domain representation) is the preferred form because of ease of manipulation. Transfer function blocks can represents different components of a physical system. Examples of transfer function block include the *reference sensor* (or *input sensor*), the *output sensor*, the *actuator*, the *controller*, and the *plant* (i.e., the component whose variable are to be controlled).

- *Junction point* (or *summing point* or *comparator*), which are used to sum or subtract two or more signals (see Fig. 3.5).

- *Take-off point* (or *branch point*), which represents a signal branching, like a feedback loop (see Fig. 3.6).

- *Arrows*, which represent unidirectional signal flows.



(a) Additive junction point.

(b) Subtractive junction point.

(c) Multiple input junction point.

Figure 3.5: Junction point.

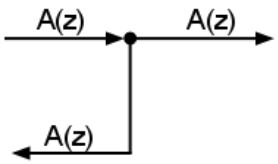


Figure 3.6: Take-off point.

One of the main advantage offered by the block diagram algebra is the ease of block interconnection and manipulation. The most common block interconnections include:

- *Series* (or *cascade*): for two systems in series with transfer functions $H_1(z)$ and $H_2(z)$, the overall transfer function is obtained as the product of the two single transfer functions:

$$H(z) = H_1(z)H_2(z) \tag{3.31}$$

- *Parallel*: for two systems in parallel with transfer functions $H_1(z)$ and $H_2(z)$, the overall transfer function is obtained as the sum of the two single transfer functions:

$$H(z) = H_1(z) + H_2(z) \tag{3.32}$$

- *Negative feedback*: for two systems in a negative feedback path with transfer

functions $H_1(z)$ and $H_2(z)$, the overall transfer function is obtained as:

$$H(z) = \frac{H_1(z)}{1 + H_1(z)H_2(z)} \tag{3.33}$$

Other common block manipulation rules are shown in Fig. 3.7.

3.4 Realization of a Transfer Function

The *realization theory* [24] provides a connection between the state-space and the input-output representations. In the following, we present some basic result for discrete-time LTI systems.

Definition 3.4.1 (Realization)**.** Given a transfer function $\mathbf{H}(z)$, we say that a discrete-time LTI state-space model $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ is a *realization* of $\mathbf{H}(z)$ if:

$$\mathbf{H}(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \tag{3.34}$$

We call Eq. (3.34), the *transfer function* of the state-space system.

In general, many state-space systems may realize the same transfer function, which motivates the following definition.

Definition 3.4.2 (Zero-state equivalence)**.** Two state-space systems are said to be *zero-state equivalent* if they realize the same transfer function, which means that they exhibit the same forced response to every input.

Definition 3.4.3 (Realizability)**.** A transfer function $\mathbf{H}(z)$ is said to be *realizable* if it there exists a state space model $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ with transfer function $\mathbf{H}(z)$.

Not all LTI state-space systems can realize a transfer function. It turns out that only *proper rational* transfer functions can be realized by an LTI system. Informally, a *proper rational* transfer function $\mathbf{H}(z)$ is a transfer function of the form

$$\mathbf{H}(z) = \frac{\mathbf{N}(z)}{\mathbf{D}(z)}$$

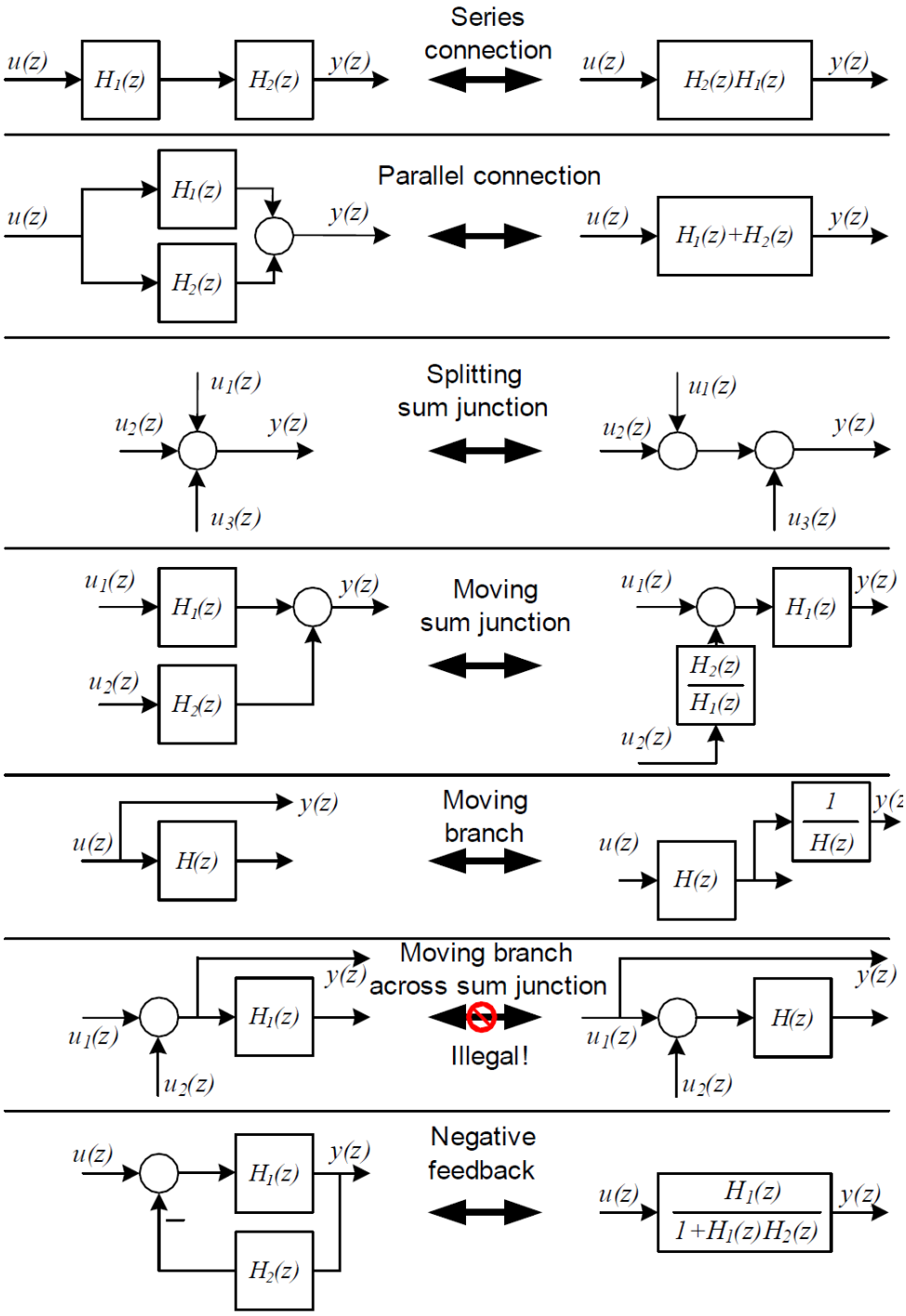


Figure 3.7: Common rules for block diagram manipulation.

in which the degree of the numerator $\mathbf{N}(z)$ does not exceed the degree of the denominator $\mathbf{D}(z)$. A transfer function $\mathbf{H}(z)$ is *strictly proper rational* is the degree of the number $\mathbf{N}(z)$ is less than the degree of the denominator $\mathbf{D}(z)$. A proper transfer function will never grow unbounded as the frequency approaches infinity.

Theorem 3.4.1 (MIMO Realization)**.** *A transfer function matrix* $\mathbf{H}(z)$ *can be realized by an LTI state-space system* if and only if $\mathbf{H}(z)$ *is a proper rational matrix.*

It is worth noting that, if a realization of a given $\mathbf{H}(s)$ exists, then there exists an infinite number of realizations (e.g., by performing an equivalence transformation of the LTI state-space system). In particular, a *minimal* realization of the transfer function $\mathbf{H}(z)$ is a state space representation of a system that has the minimum possible number of states, that is a state-space system where the dimension of the state vector (i.e., the number of states) is the same as the degree of the numerator of $\mathbf{H}(z)$ (which, for proper rational transfer functions, does not exceed the one of the denominator).

Definition 3.4.4 (Minimal Realization)**.** A realization $(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D})$ of the transfer function $\mathbf{H}(z)$ of least-order n, where $\mathbf{A} \in \mathbf{R}^{n\times n}$, is called a *least-order*, or a *minimal*, or an *irreducible* realization of $\mathbf{H}(z)$.

There are several way to construct an LTI system that realizes an arbitrary given proper rational function $\mathbf{H}(z)$. Among them, *canonical realizations* are a very commonly way to realize a strictly proper transfer function [5] in order to construct a state-space system with specific properties, like *controllability* and *observability*. In particular, with a *controllable canonical form* the resulting state-space system is guaranteed to be *controllable*, while with an *observable canonical form* the resulting state-space system is guaranteed to be *observable*. The controllability and observability of a system are mathematical duals, which play a central role in the study of state feedback controllers and state observers [69], and in establishing the relations between internal and external system representations. The concept

[5] Transfer functions which are only proper can also be realised, by expressing them as the sum of two terms: a strictly proper part $\hat{\mathbf{H}}(z)$ and a constant one $\mathbf{H}(\infty)$. The strictly proper transfer function can then be transformed into a canonical state space realization, while the state space realization of the constant is trivially $\mathbf{Y}(z) = \mathbf{H}(\infty)\mathbf{U}(z)$.

of *(state) controllability* refers to the ability to manipulate the state by applying appropriate inputs, while *(state) observability* observability refers to the ability to determine the state vector of the system from knowledge of the input and the corresponding output over some finite time interval.

It is important to realize that, although a model of a linear time invariant system can be a transfer function as well as a state space model, these two models do not give the same insight into the system's properties. The transfer function is a very efficient description of the relationship between the input and the output. The state space model provides the same information but, in addition, it also gives detailed information on the internal state variables. Moreover, when a transfer function is constructed one can be sure that it is a unique model. In contrast to this, the state-space model is not unique. One has considerable freedom when selecting the state variables and the resulting model obviously depends on the specific choice of state variables.

3.5 Stability

One of the most important properties of a system is *stability*. Intuitively, a system can be considered unstable if its output is unbounded. There are different definitions of system stability. For analysis and design purposes, we can classify stability as *absolute stability* and *relative stability*.

Absolute stability refers to whether the system is stable or unstable; it is a "yes" or "no" answer. The absolute stability of a system is determined by its response to inputs or disturbances. Intuitively, a system is *absolutely stable* if it remains at rest (in equilibrium) unless excited by an external source and returns to rest if all excitations are removed. If action persists indefinitely after excitation is removed, the system is considered *absolutely unstable*. Once the system is found to be absolutely stable, it is of interest to determine how stable it is, and this degree of stability is a measure of relative stability.

Absolute stability can be divided into *internal stability* (or *Lyapunov stability*), that expresses how the (internal) state of the system evolves with time, and *external stability* (or *input-output stability*), that expresses how the magnitude of the output relates to the magnitude of the input in the absence of initial conditions [24].

Internal stability concerns with the stability of solutions near to a point of equilibrium [114]. Informally, if all solutions of the system that start out near an equilibrium point $\bar{\mathbf{x}}$ stay near $\bar{\mathbf{x}}$ forever, then the system is *stable* (in the Lyapunov sense). More strongly, if a system is Lyapunov stable and all solutions that start out near the equilibrium point $\bar{\mathbf{x}}$ converge to $\bar{\mathbf{x}}$, then the system is *asymptotically stable* (in the Lyapunov sense). In addition, the notion of *exponential stability* guarantees a minimal rate of decay (i.e., an estimate of how quickly the solutions converge).

For what concerns external stability, of particular interest is the concept of *Bounded-Input Bounded-Output* (BIBO) stability, whereby a system is stable if, for zero initial conditions, the output will be bounded for every input to the system that is bounded. [6]

Definition 3.5.1 (BIBO stability)**.** A linear discrete-time system is said to be (uniformly) *Bounded-Input Bounded-Output (BIBO) stable* if its impulse response approaches zero as time approaches infinity, that is if there exists a finite constant c such that, for every input $\mathbf{u}(\cdot)$, its forced response $\mathbf{y}_f(\cdot)$ satisfies:

$$\sup_{k\in\mathbb{N}} \|\mathbf{y}_f(k)\| \leq \sup_{k\in\mathbb{N}} \|\mathbf{u}(k)\| \leq c, \quad c > 0 \tag{3.35}$$

Examples of measures of relative stability are *phase margin*, which indicates the tendency of a system to oscillate during its transient response to an input change such as a step function, and *gain margin*, which provides the degree to which the system will oscillate without limit given any disturbance [43].

A key role for the stability of a system is played by the poles of the transfer function. As depicted in Fig. 3.8 (from [84]), the stability of a system can be assessed by the location of the poles of the transfer function:

- poles inside the unit circle ($|z| < 1$) result in an unbounded signal,
- poles outside the unit circle ($|z| > 1$) result in a bounded signal that converges to zero,
- poles on the unit circle ($|z| = 1$) result in a bounded signal that not converges to zero.

[6] A discrete-time signal $x(\cdot)$ is bounded if there is a finite value $c > 0$ such that the signal magnitude never exceeds c, that is $|x(n)| \leq c \quad \forall n \in \mathbb{Z}$.

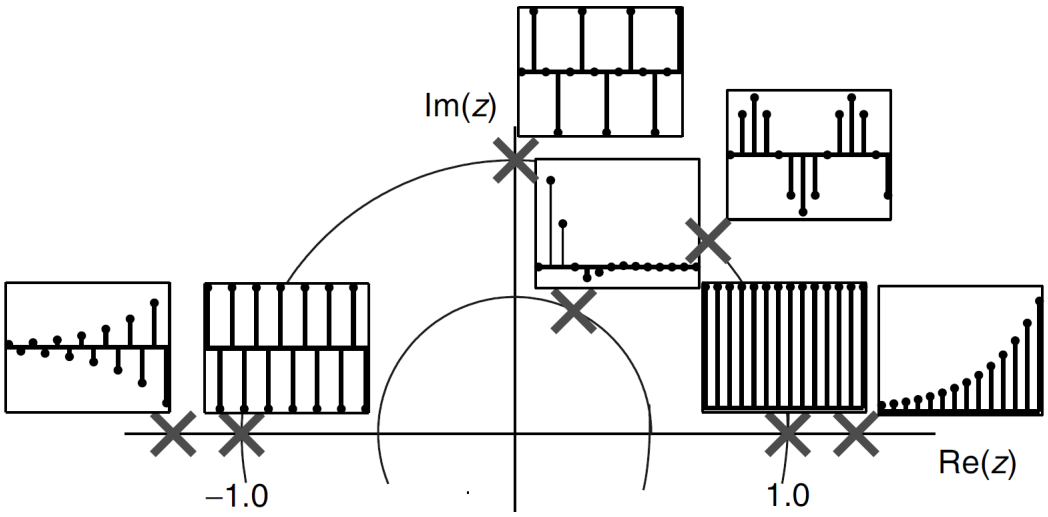


Figure 3.8: Relationship between location of the poles of the transfer function and the time-domain behavior of the signal with those poles (from [84]).

When poles are inside the unit circle, the rate of convergence to zero depends on the magnitude of the poles: the closer the pole is to the origin, the faster the convergence. Moreover, the location of poles in the positive or negative real axis determine the oscillatory behavior and the frequency of oscillation depends on the angle of each pole relative to the origin.

A direct consequence of pole location analysis is the following theorem.

Theorem 3.5.1 (BIBO Stability)**.** *A system is BIBO stable if and only if all the poles of its transfer function are inside the unit circle.*

Chapter 4

Linear System Identification

System identification [112] is the process of building mathematical models of dynamical system from measured data, representing inputs and outputs of the system, and from prior knowledge. Theory behind system identification is vast and covers both parametric and nonparametric techniques than can be applied to both time- and frequency-domain system representation.

> In this thesis, we focus on the black-box modeling approach as a mean to describe linear systems. We use online parametric system identification to estimate parameters of such black-box models, and, specifically, we employ variants of the recursive least-squares algorithm.

In the rest of this chapter, we provide an overview of the identification process of linear systems, with a particular focus to the techniques used in this thesis.

4.1 The System Identification Process

A *system model* is a mathematical description (in either the time or frequency domain) of the dynamic behavior of a real system, which helps to capture and study relevant aspects of the system in a tractable mathematical form. Several approaches and techniques are available for deriving system models. Standard modeling approaches include two streams: (1) the *white-box* approach, and (2)

the *identification* of a model. The white-box approach uses first principles to build models based on the physical laws and relationships (e.g., Newton equations) that are supposed to govern the behavior of the real system; in these models, the structure reflects all physical insights about the process, and variables and parameters have direct physical interpretations (e.g., heat transfer coefficients). Often, such a direct modeling may not be possible. The reason may be that (a) the knowledge of the system's mechanisms is incomplete, (b) the properties exhibited by the system may change in an unpredictable manner, or (c) the system is too complex to model. In such cases, variables, characterizing the behavior of the considered system, can be measured from the real system and used to construct a model. This is where the *identification* approach takes place. Essentially, the identification approach can be performed either by gray-box or black-box modeling. The *gray-box* modeling approach utilizes physical insights about the underlying system (i.e., by using first principles), but parameter values are not know in advance and need to be derived from data observed from the real system. The *black-box* modeling approach uses measured data to build a model which is representative of the inputs and outputs observed from the real systems. Due to the difficulty of constructing first-principles models of computing systems, the black-box modelling approach is usually the preferred way to model such systems since it requires a less detailed knowledge of the relationships between system inputs and outputs [84]. For this reason, in this thesis, we limit the discussion to only black-box models.

In the system identification framework, in addition to system inputs and outputs, it is useful to consider additional type of signals, called *disturbances*, which are external stimuli that affect the behavior of the system (see Fig. 4.1). Disturbances cannot be manipulated since they are not generally under control (e.g., they are part of the inherent nature of system dynamics or derive from the measurement process). They are usually divided into *noise* (or measurement noise) $\mathbf{v}(\cdot)$, which is any effect that changes the observed output (e.g., the noise introduced by the sensor that measures the output), and *disturbance input* (or uncontrollable input) $\mathbf{w}(\cdot)$, which is any change that affects the way in which the input influences the measured output (e.g., wind guts and turbulence that affect the movement of an aeroplane). The explicit modeling of disturbances leads to a *stochastic* system model, since $\mathbf{w}(\cdot)$ and $\mathbf{v}(\cdot)$ are generally modeled as stochastic processes; conversely, when they

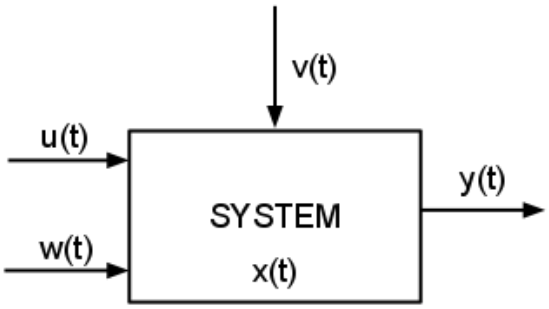


Figure 4.1: Representation of a system with input $\mathbf{u}(t)$, output $\mathbf{y}(t)$, state $\mathbf{x}(t)$, noise $\mathbf{v}(t)$ and input disturbance $\mathbf{w}(t)$, each of which varying with the independent variable t (e.g., the time).

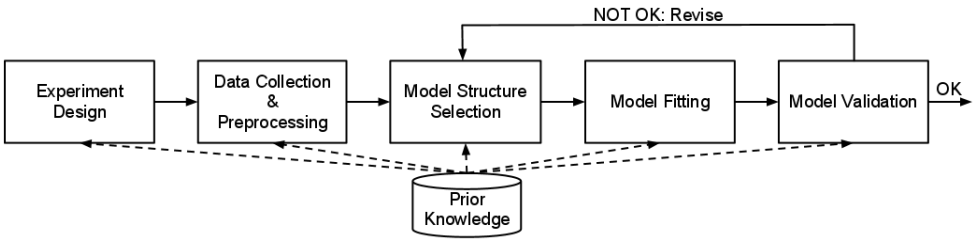


Figure 4.2: The system identification workflow.

are not explicitly modeled, the model is called *deterministic*. In addition to the different type of signals, it is also useful to take into consideration the delay that takes a given input to have effect on the system output. This is usually called *input delay* or *dead time*.

As shown in Fig. 4.2, system identification is an iterative process which comprises the following steps: (1) *experiment design*, (2) *data collection and preprocessing*, (3) *model structure selection*, (4) *model estimation*, and (5) *model validation*. Every step is possibly driven by *prior knowledge* of the modeled system in order to reduce uncertainty in the model estimation. In the rest of this section we provide an overview of each step.

4.1.1 Experiment Design

In this step, an identification experiment is designed so that the real system is suitably instrumented and stressed in order to excite all of the system dynamics

(even the ones that may be unknown) and to capture sufficient variability.

The design is primarily concerned with the selection of design variables and their corresponding values. The selection of design variables comprise the choice of which system variables to measure and when to measure them (i.e., system inputs), which system variables to manipulate and how to manipulate them (i.e., system outputs), and finally which system variables affect the system behavior but that are not manipulable (i.e., disturbance inputs). For what regards the choice of input values, they should be enough "rich" in order to provide a dense and uniform coverage of the range of possible input values, and thus to excite and force the system to show all of its dynamics. Input values are usually represented as input signals which probe the system under test. These signals determine the operating points of the system. While the system under test often limits the choice of signals, the input signal must exhibit certain characteristics. These characteristics must produce a response that provides the information one needs for developing an accurate model. The following list summarizes some of these characteristics.

- To obtain meaningful dynamic behavior, the system must be tested under conditions similar to the actual operating conditions. This criterion is extremely important for nonlinear systems.

- The input signal must sufficiently excite the system with an input frequency similar to the frequency at which such inputs change during normal operations.

- The amplitude of the step input must cover a wide range of variations, so that the normal operation range system inputs is covered as well.

- The input signal must stay within the limits of the physical system.

The system response data is dependent on the physics of the system under study. Some systems tend to respond faster than others. Other systems have large time constants and delays. For these reasons, defining an input signal that provides enough excitation to the system is important. The system response must capture the important features of the system dynamics. Common choice of input signals include [152]:

- *Filtered Gaussian white noise*, which is a signal randomly generated according to a Normal probability distribution (with mean and variance given) filtered by a linear filter.

- *Random binary signal*, which is a random process that can assume one of two possible values at any time. A simple method of generating a random binary signal is to take Gaussian white noise, filter it for the desired spectra and then convert it to a binary signal by taking the sign of the filtered signal.

- *Pseudo-random binary signal* (PRBS): which is a periodic, deterministic signal with white-noise-like properties, that can assume only two values.

- *Sinusoidal signal*, which is a periodic, deterministic signal generated according to a sinusoidal law.

- *Random uniform signal*, which is a signal randomly generated according to a Uniform probability distribution.

For verification and validation reasons, one should acquire two sets of input-output data samples or split the data into two sets. The first set of samples is used to estimate the mathematical model of the system, while the second one is used to validate the resulting model. If the resulting model does not meet the desired level of fit (e.g., measured by means of mean squared error), the structure and parameters of the model should be revised, and then repeating the verification and validation process.

4.1.2 Data Collection and Preprocessing

Data from the experiment are collected and possibly preprocessed in order to remove possible external noise, scaling problems, outliers, corruptions, missing or partial data, and other type of disturbances.

There are different techniques to dealing with such deficiencies:

- *Visually inspecting data*. The graphical exploration of the data is one of the best way to detect disturbances that however is applicable when the size of data is small and preprocessing can be performed offline. Traditionally, data

samples are examined either in the time domain or the frequency domain. An effective approach is to display the data in the joint time-frequency domain, which provides a better understanding about the measured signals. To this end, one can use, for instance, a *spectrogram*, which is a time-varying spectral representation [83], that shows how the spectral density of a signal varies with time.

- *Explicitly removing disturbances*. Another approach to remove disturbances, especially when they represents offsets and trends, is by explicit pretreatment of the data [112]. Most of the available techniques can be considered as variants the *linearization around an equilibrium* method, which consists in subtracting some operating point from data samples, thus transforming the original model in a model which relates deviations from a physical equilibrium. It is possible to use either a single steady-state equilibrium point, or, more generally, time-varying equilibria.

- *Using statistical predictors to estimate missing data*. In case some of the input, or of the output, or both are missing it is possible to use some statistical prediction framework (like regression) to estimate them.

- *Filtering*. In the case one is interested in only a specific frequency range of the frequency response for a model, it is possible to filter and enhance the data in the frequency range to improve the fit in the regions of interest. For instance, high-pass filters are used to eliminate offsets and load disturbances, while low-pass filters are employed to eliminate irrelevant high frequency components, including noise and system response.

- *Data scaling*. Often inputs and outputs of a physical system have different amplitude ranges and hence different magnitude. This diversity can result in an ill-conditioned model estimation, which reduces the accuracy of the model. A commonly used approach to this problem is to *normalize* both input and output signals to ensure that they have a zero mean and unit variance over the sample data range used for model estimation.

- *Estimating the noise explicitly*. Unlike the previously described approaches, which directly aim at removing disturbances from the data, this approach

take care of disturbances by explicitly adding a noise term to the input-output model. This is done by employing specific models, like the *AutoRegressive Moving Average with eXogenous variables* (ARMAX) model which is used in the *Box-Jenkins* methodology [35].

4.1.3 Model Structure Selection

A *model structure* is a mathematical relationship between input and output variables that contains unknown parameters. Examples of model structures are transfer functions with adjustable poles and zeros, state space equations with unknown system matrices, and nonlinear parameterized functions.

The system identification process requires that one chooses a model structure and apply the estimation methods to determine the numerical values of the model parameters. There are two basic approaches to select a model structure, namely black-box modeling and grey-box modeling approaches.

In the *black-box modeling* approach, the system behavior is inferred by simply relating system inputs and outputs from the data, without the need to know anything about the internal structure of the system. Black-box modeling is usually a trial-and-error process, where one estimates the parameters of various structures and compare the results. Typically, one starts with the simple linear model structure and progress to more complex structures. One can configure a model structure using the *model order*. The definition of model order varies depending on the type of model one selects. For example, in case of a transfer function representation, the model order is related to the number of poles and zeros. For state-space representation, the model order corresponds to the number of states. If the simple model structures do not produce good models, one can select more complex model structures by:

- Specifying a higher model order for the same linear model structure. Higher model order increases the model flexibility for capturing complex phenomena. However, unnecessarily high orders can make the model less reliable.

- Explicitly modeling the noise, by using, for instance, an additive disturbance model which treats the disturbance as the output of a linear system driven by a white noise source. Using a model structure that explicitly models

the additive disturbance can help to improve the accuracy of the measured system component. Furthermore, such a model structure is useful when the main interest is using the model for predicting future response values.

- Using a different linear model structure.
- Using a nonlinear model structure. Nonlinear models have more flexibility in capturing complex phenomena than linear models of similar orders.

Ultimately, one chooses the simplest model structure that provides the best fit to the measured data.

For what concerns the *gray-box modeling* approach, first principles that describes the behavior of the system are known (e.g., in the form of a set of difference equations) but not their parameters, which thus need to be estimated from data. In general, the process for building grey-box models involves three steps:

1. Creating a template model structure.
2. Configuring the model parameters with initial values and constraints (if any).
3. Applying an estimation method to the model structure and computing the model parameter values.

In this thesis, the focus is on the black-box modeling approach.

Black-box modeling can be performed both in a nonparametric and in parametric way. In a *nonparametric* model, the system is described by using the impulse response or frequency response. The *impulse response* reveals the time-domain properties of the system, such as time delay and damping. The impulse response of a dynamic system can be estimated by means of *least-squares* and *correlation analysis* methods [112]. The *frequency response* reveals the complete frequency-domain characteristics of the system, including properties like the natural frequency of the system. The frequency response of a dynamic system can be estimated with the *Fourier analysis* or with the *spectral analysis* methods [32, 112]. New approaches to nonparametric system identification also include theories of modern *nonparametric regression*, *approximation*, and *orthogonal expansions* [76].

The other type of black-box models are *parametric models*, which describe systems in terms of difference or differential equations, depending on whether a system is represented by a discrete or continuous model, respectively. As the name implies, such models has a specific analytic form and depend on a certain number of parameters, whose values need to be estimated from data. Parametric model structures for linear systems include transfer-function and state-space model structures.

The *transfer function* (or *polynomial*) model structure is a family of linear model that can be represented by the following general time-varying polynomial equation:

$$\mathbf{A}_k(q)\mathbf{y}(k) = \frac{\mathbf{B}_k(q)}{\mathbf{F}_k(q)}\mathbf{u}(k-n_k) + \frac{\mathbf{C}_k(q)}{\mathbf{D}_k(q)}\mathbf{e}(k) \tag{4.1}$$

where $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$, $\mathbf{D}_k$ and $\mathbf{F}_k$ are time-varying polynomial matrices expressed in terms of the *time-shift operator* q, [1] n_k is the time delay associated to the input, [2] and $\mathbf{e}(k)$ is a zero-mean stochastic process with a given finite variance, modeling the *white noise*.

A similar formulation is available for LTI systems, where polynomial matrices are now independent from the time:

$$\mathbf{A}(q)\mathbf{y}(k) = \frac{\mathbf{B}(q)}{\mathbf{F}(q)}\mathbf{u}(k-n_k) + \frac{\mathbf{C}(q)}{\mathbf{D}_k(q)}\mathbf{e}(k) \tag{4.2}$$

To estimate polynomial models, it is necessary to specify the *model order* as a set of integers that represent the number of coefficients for each polynomial of the selected structure. For the general model of Eq. (4.1), this means to estimate the numbers n_a, n_b, n_c, n_d, and n_f corresponding to the number of matrices $\mathbf{A}_k$, $\mathbf{B}_k$. $\mathbf{C}_k$. $\mathbf{D}_k$, and $\mathbf{F}_k$, respectively. It is also necessary to specify the number of samples n_k, corresponding to the input delay, given by the number of samples before the

[1] The *time-shift operator* q is such that when applied to an operand $x(k)$, that is $q^\tau x(k)$, it shifts the time k of its operand $x(\cdot)$ by τ steps forward or backward, according to the sign of τ; for instance, $q^{-1}x(k) \triangleq x(k-1)$. The notation $A(q)$ represents the time-shift operator applied to a polynomial such that $A(q)x(k) \triangleq \left(1+\sum_{i=1}^{n_a} a_i q^{-i}\right) y(k) \triangleq y(k) + a_1 y(k-1) + \cdots + a_{n_a} y(k-n_a)$. The notation $\mathbf{A}(q)$ represents the time-shift operator applied to a polynomial matrix such that $\mathbf{A}(q)\mathbf{x}(k) \triangleq \left(\mathbf{I}+\sum_{i=1}^{n_a} \mathbf{A}_i q^{-i}\right) \mathbf{y}(k) \triangleq \mathbf{y}(k) + \mathbf{A}_1\mathbf{y}(k-1) + \cdots + \mathbf{A}_{n_a}\mathbf{y}(k-n_a)$.

[2] It is worth noting that, in general, there may be a different input delay n_{k_j} for each component $u_j(\cdot)$ of the input $\mathbf{u}(\cdot)$. However, to keep the notation simple, we assume the same input delay for each input component.

output responds to the input. Typically, one begins modeling using simpler forms of this generalized structure and, if necessary, increases the model complexity.

There are different specialization of the general structure of Eq. (4.1). Among them, two widely adopted model structures are:

- *AutoRegressive with eXogenous variables* (ARX):
  - LTV formulation:

$$\mathbf{A}_k(q)\mathbf{y}(k) = \mathbf{B}_k(q)\mathbf{u}(k-i-n_k) + \mathbf{e}(k) \tag{4.3}$$

  - LTI formulation:

$$\mathbf{A}(q)\mathbf{y}(k) = \mathbf{B}(q)\mathbf{u}(k-i-n_k) + \mathbf{e}(k) \tag{4.4}$$

  In this model structure, the model order is given by n_a and n_b. Usually, the notation ARX(n_a, n_b, n_k) is employed to indicate an ARX model with order n_a and n_b, and with input delay n_k. This is one of the more simplest model structure. The major drawback of this model is that, because of the noise term is coupled to the model dynamics, ARX does not allow to model noise and dynamics independently.

- *AutoRegressive Moving Average with eXogenous variables* (ARMAX)
  - LTV formulation:

$$\mathbf{A}_k(q)\mathbf{y}(k) = \mathbf{B}_k(q)\mathbf{u}(k-i-n_k) + \mathbf{C}_k(q)\mathbf{e}(k) \tag{4.5}$$

  - LTI formulation:

$$\mathbf{A}(q)\mathbf{y}(k) = \mathbf{B}(q)\mathbf{u}(k-i-n_k) + \mathbf{C}(q)\mathbf{e}(k) \tag{4.6}$$

  In this model structure, the model order is given by n_a, n_b, and n_c. Usually, the notation ARMAX(n_a, n_b, n_c, n_k) is employed to indicate an ARX model with order n_a, n_b and n_c, and with input delay n_k. This model structure extends the ARX structure by providing more flexibility for modeling noise

(using a moving average of white-noise). For such reason, it is particularly indicated to model systems where dominating disturbances enter at the input.

Several techniques have been proposed in the literature to estimate both the model order and the input delay (e.g., see [91, 112]). For instance, in [112], a model structure selection based on statistical hypothesis testing is proposed, whereby each model structure is evaluated according to a specific goodness-of-fit criterion like the *Akaike's Information Criterion* (AIC) [124] or the *Minimum Description Length* (MDL) criterion [141].

The other type of parametric linear model structure is the *state-space model*. In this model, the relationship between input, output and noise signals is represented as a system of first-order difference equations, instead of specifying one or more nth-order difference equations, by using auxiliary state variables $\mathbf{x}(\cdot)$:

$$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u}(k) + \mathbf{w}(k), \tag{4.7a}$$

$$\mathbf{y}(k) = \mathbf{C}(k)\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k) + \mathbf{v}(k), \tag{4.7b}$$

with the noise covariance matrix:

$$\mathrm{E}\left[\begin{pmatrix}\mathbf{w}(p)\\ \mathbf{v}(p)\end{pmatrix}\begin{pmatrix}\mathbf{w}^T(q) & \mathbf{v}^T(q)\end{pmatrix}\right] = \begin{pmatrix}\mathbf{Q} & \mathbf{S}\\ \mathbf{S}^T & \mathbf{R}\end{pmatrix}\delta_{pq} \geq 0 \tag{4.8}$$

where $\mathbf{w}(\cdot) \in \mathbb{R}^{n_w}$ is the *disturbance input vector*, $\mathbf{v}(\cdot) \in \mathbb{R}^{n_v}$ is the *noise vector*, and δ_{ij} is the Kronecker's delta. Usually, $\mathbf{w}(\cdot)$ and $\mathbf{v}(\cdot)$ are assumed to be zero-mean stationary white noise stochastic processes with a given variance.

An alternative and frequently used form for state-space model structures is the one where disturbances $\mathbf{w}(\cdot)$ and $\mathbf{v}(\cdot)$ are related each other by the *Kalman matrix*, such that $\mathbf{w}(k) = \mathbf{K}\mathbf{v}(k)$. In this case, the disturbance vector $\mathbf{w}(k)$ is usually denoted by $\mathbf{e}(k)$, and the state-space model structure form is called the *innovation form*:

$$\mathbf{x}(k+1) = \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u}(k) + \mathbf{K}(k)\mathbf{e}(k), \tag{4.9a}$$

$$\mathbf{y}(k) = \mathbf{C}(k)\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k) + \mathbf{e}(k) \tag{4.9b}$$

with the noise covariance matrix:

$$\mathrm{E}\left[\mathbf{e}(p)\mathbf{e}^T(q)\right] = \mathbf{S}\delta_{pq} \geq 0 \tag{4.10}$$

Similar formulations are available for LTI systems, where matrices are now independent from the time. Thus, for Eq. (4.7), the following LTI form is obtained:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) + \mathbf{w}(k), \tag{4.11a}$$
$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k) + \mathbf{v}(k), \tag{4.11b}$$

while the LTI innovation form is given by:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) + \mathbf{K}\mathbf{e}(k), \tag{4.12a}$$
$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k) + \mathbf{e}(k) \tag{4.12b}$$

Similarly to the transfer function model structure, also in the state-space model structure it is possible to include time delays. However, unlike the transfer function representation, there are different type of time delays:

- *input delays*, which model delays at the input;
- *output delays*, which model delays at the output;
- *I/O delays*, which model independent transport delays from a given input to a given output of a MIMO transfer function model;
- *internal delays*, which model interconnection of systems with input, output, or I/O delays, including feedback loops with delays. Internal delays can arise, for instance, from:
  - concatenating state-space models with input and output delays,
  - feeding back a delayed signal, and
  - converting MIMO transfer function with I/O delays to state-space model.

The state-space model structure is a good choice for quick estimation because it requires only the estimation of the model order (which is an integer number equal to the dimension of the state vector and relates to the number of delayed input and outputs used in the corresponding linear difference equation) and, possibly, one or more delays.

Compared to nonparametric models, parametric models might provide a more accurate estimation if one has prior knowledge about the system dynamics to determine parameters like model orders and time delays. Nonparametric model estimation is more efficient, but often less accurate, than parametric estimation. One possible use of nonparametric models, is as an estimation method to obtain useful information about a system before applying parametric model estimation. For example, one can use nonparametric model estimation to determine whether the system requires preconditioning, what the time delay of the system is, what model order to select, and so on. Another possible use of nonparametric model estimation is for parametric model verification. For instance, one can compare the *Bode plot* [3] of a parametric model with the frequency response of the nonparametric model.

In this thesis, the focus is on parametric model structures.

4.1.4 Model Estimation

Once the model structure has been chosen, its parameters have to estimated from the collected data according to a "best-fit" criterion. System identification techniques for model parameters estimation can roughly be classified in two groups: offline techniques and online techniques In the rest of this section, we describe some of the most widely adopted algorithms both for offline and online system identification.

In this thesis, we concentrate on the online system identification of ARX model structures and, specifically, on algorithms based on the least-squares

[3] The *Bode plot* is a graph of the transfer function of an LTI system versus frequency, plotted with a log-frequency axis, to show the system's frequency response

criterion.

Offline Algorithms

Offline system identification (or *batch system identification*) techniques are generally based on iterative methods that exploit the advantage of having a complete set of data available for processing. Scientific literature provides many different techniques for the offline estimation of model structure parameters (e.g., see [112, 160, 152]).

For the identification of transfer function model structures, a widely adopted technique is the *prediction-error minimization* (PEM) method. This approach, has a close connection with the maximum likelihood method, originating from [66] and introduced into the estimation of dynamical models by [138]. The basic idea behind the prediction-error minimization method is very simple. Let $\mathscr{M}$ be a particular model structure, $\mathscr{Z}^N = \left\{ \left(\mathbf{y}^T(1), \mathbf{u}^T(1)\right)^T, \ldots, \left(\mathbf{y}^T(N), \mathbf{u}^T(N)\right)^T \right\}$ be the set of collected input-output pairs up to time N, and $f(\cdot)$ be an arbitrary function of past, observed data.

1. Describe the model as a predictor of the next output:

$$\hat{\mathbf{y}}_m(k|k-1) = f\left(\mathscr{Z}^{k-1}\right) \tag{4.13}$$

   where $\hat{y}_m(k|k-1)$ denotes the one-step ahead prediction of the output.

2. Parameterize the predictor in terms of a finite-dimensional parameter vector θ:

$$\hat{\mathbf{y}}(k|\theta) = f\left(\mathscr{Z}^{k-1}, \theta\right) \tag{4.14}$$

3. Determine the estimate $\hat{\theta}_N$ of θ from the model parametrization of the observed data set $\mathscr{Z}^N$, so that the distance between $\hat{\mathbf{y}}(1|\theta), \ldots, \hat{\mathbf{y}}(N|\theta)$ and

$\mathbf{y}(1),\ldots,\mathbf{y}(N)$ is minimized in a suitable norm $V_N(\cdot)$, that is:

$$V_N\big(\theta,\mathscr{Z}^N\big)=\frac{1}{N}\sum_{k=1}^{N}\ell\big(\varepsilon(k,\theta)\big), \tag{4.15}$$

$$\hat{\theta}_N=\arg\min_{\theta}V_N\big(\theta,\mathscr{Z}^N\big) \tag{4.16}$$

where $\varepsilon(k,\theta)=\mathbf{y}(k)-\hat{\mathbf{y}}(k|\theta)$ is the *prediction error* (i.e., the difference between the observed and the predicted output value), and $\ell(\cdot)$ is a scalar-valued (typically positive) function.

The PEM identification method can be considered a family of approaches each of which varies according to the choice of the model structure $\mathscr{M}$, the choice of $\ell(\cdot)$, and, in some cases, the choice of the method by which the minimization is realized. Under this family there are well-known procedures, such as the least-squares (LS) method and the maximum likelihood (ML) method, and it is at the same time closely related to the Bayesian maximum a posteriori (MAP) estimation and Akaike's information criterion (AIC) [112].

For instance, the *least-squares* (LS) method can be obtained by choosing:

$$\mathscr{M}=\mathrm{ARX}(n_a,n_b,n_k), \tag{4.17}$$

$$f(\mathscr{Z}^k,\theta)=\theta^T\varphi(k)+\mu(k), \tag{4.18}$$

$$\ell(\varepsilon)=\frac{1}{2}\varepsilon^T\Lambda^{-1}\varepsilon \tag{4.19}$$

where $\varphi(k)$ is the *regression vector* at time k, that is an $(n_an_y+n_bn_u)$-dimensional vector (with n_y and n_u being the size of $\mathbf{y}(\cdot)$ and $\mathbf{u}$, respectively):

$$\varphi(k)=\begin{pmatrix}-\mathbf{y}(k-1)\\ \cdots\\ -\mathbf{y}(k-n_a)\\ \mathbf{u}(t-1)\\ \cdots\\ \mathbf{u}(t-n_b)\end{pmatrix}, \tag{4.20}$$

$\mu(k)$ is a sequence of independent and identically distributed random variables

with zero mean (e.g., describing disturbances or non-modeled dynamics), and Λ is a positive semidefinite $n_y \times n_y$ matrix, that weights together the relative importance of the components ε. For ease of notation, in the rest of this section, we assume that the matrix $\Lambda = \mathbf{I}$, which means that all the components of ε are equally weighted. In this case, the parameter θ is given by the following $(n_a n_y + n_b n_u) \times n_y$ matrix:

$$\theta = \begin{pmatrix} \mathbf{A}_1 & \dots & \mathbf{A}_{n_y} & \mathbf{B}_1 & \dots & \mathbf{B}_{n_u} \end{pmatrix}^T \tag{4.21}$$

The norm $V_N(\cdot)$ becomes:

$$V_N\big(\theta, \mathscr{Z}^N\big) = \frac{1}{N} \sum_{k=1}^{N} \big\| \mathbf{y}(t) - \theta^T \varphi(k) \big\|^2 \tag{4.22}$$

while the estimate $\hat{\theta}_N$ is obtained as the *least-squares estimate* $\hat{\theta}_N^{\mathrm{LS}}$ given by:

$$\hat{\theta}_N^{\mathrm{LS}} = \Big(\frac{1}{N} \sum_{k=1}^{N} \varphi(k) \varphi^T(k) \Big)^{-1} \frac{1}{N} \sum_{k=1}^{N} \varphi(k) \mathbf{y}^T(k) \tag{4.23}$$

This approach is sometimes referred to as the *output-error* (OE) method since, in the estimated model, the uncertainties due to noises acting on the system are assumed to be lumped together as an additive perturbation at the output [164].

For the state-space model structure, there are essentially two approaches, namely the optimization-based approach and the subspace-based approach. The optimization-based approach is based on the PEM method, which in general requires iterative methods to solve a multidimensional, nonlinear optimization problem for the minimization of the prediction error. The *subspace identification method* [160] offers numerically reliable algorithms for computing state-space descriptions directly from data. The method is competitive with respect to traditional PEM techniques, in particular for the high-order multi-input multi-output case. The computations are based on QR-factorization and singular-value decomposition (SVD), for which numerically reliable algorithms are available [75]. No numerical search is necessary, nor is a potentially ill-conditioned canonical system description used. This approach is based on deriving a certain subspace that contains information about the system, from structured matrices constructed from the input

and output data. To estimate this subspace, the SVD is used. The singular values obtained from this decomposition can be used to estimate the order of the system.

The subspace identification problem for an LTI system can be stated as follows. Given measurements of the inputs $\mathbf{u}(k)$ and output $\mathbf{y}(k)$ of an unknown LTI system of the state-space innovation form Eq. (4.12):

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) + \mathbf{K}\mathbf{e}(k), \tag{4.24a}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k) + \mathbf{e}(k) \tag{4.24b}$$

the problem is to determine an estimate of the system matrices **A**, **B**, **C**, and **D** and the noise related matrices **S** and **K**.

The usual steps in a subspace identification algorithm are the following:

1. Construct a data block Hankel matrix from the measured input and output data.

2. Perform a QR-decomposition with this matrix.

3. Perform an SVD on the low dimensional rank-deficient *R*-component of the QR-decomposition to find the extended observability matrix and/or the Kalman filter state sequence of the system.

Although these three steps are common to most of the subspace identification algorithms, the way they are finally implemented in practice can differ considerably. The next steps in the subspace identification algorithms, i.e. that of finding the actual system matrices **A**, **B**, **C**, **D**, **K**, and **S**, can be completely different according to the method. For instance certain methods use the observability matrix of the system to find the system matrices whereas other methods use an estimate of the state sequence (see [160, 164] for further details). Currently, the three most commonly used subspace identification algorithms are the *Canonical Variate Analysis* (CVA) [108, 131], the *Multivariable Output-Error State-sPace* (MOESP) [162, 163], and the *Numerical algorithms for Subspace State-Space System IDentification* (N4SID) [159, 160].

Online Algorithms

Algorithms for *online system identification* (or *recursive system identification*) [113] use input-output data recursively (i.e., sequentially) as they become available. This can be done by adding the new input-output observation $\left(\mathbf{y}^T(k), \mathbf{u}^T(k)\right)^T$ at time k to the previous set of observations and recomputing the estimate $\hat{\theta}(k)$ of model parameters θ. However, instead of recomputing the estimates with all available data, the previous parameter estimates are updated with the new data sample. In order to do this, the estimation formula is written in the form of a recursive algorithm:

$$\begin{aligned}\text{new estimate} = \text{old estimate} \\ + \text{correction factor}\,(\text{new observation} - \text{prediction with old estimate})\end{aligned}$$

Online system identification is a valuable tool in the design of adaptive control and adaptive prediction since it allows to develop a model that adjusts based on real-time data coming from the system, thus adapting itself to current operating conditions. By comparison, the offline model estimation methods identify a model for a system based on input-output data gathered at a time prior to the current time. The reason for the need of online system identification might be that knowledge about the properties of a system is not available offline and instead consolidates as more observations become available. An even more common situation is that the properties in question are varying over time, so that a model that continuously adapts to a changing operating conditions is required.

In contrast to offline identification, there are two disadvantages in using online identification. The first is that the decision of what model structure to use has to be made a priori, before starting the (online) identification procedure; instead, in the offline identification, different type of models can be evaluated. The second disadvantages is that, with a few exceptions, recursive methods do not give as good accuracy of the models as offline methods, even if for long datasets, this difference should not be significant.

Like offline algorithms, scientific literature provides many different techniques (e.g., see [113, 112]), and many of them have an offline counterpart. For instance, for the identification of transfer function model structures, two widely adopted

methods are the recursive prediction-error minimization method and the recursive least-square algorithm. In the following, we limit the description to the recursive least-squares method since it is the one we used in this thesis.

The *recursive least-squares* (RLS) algorithm is the online version of the *weighted least-squares* offline algorithm, whereby, according to the notation used for the description of the PEM method, the norm $V_N(\cdot)$ is defined as:

$$V_N\big(\theta, \mathscr{Z}^N\big) = \frac{1}{N}\sum_{k=1}^{N} \beta(N,k)\big\|\mathbf{y}(t) - \theta^T\varphi(k)\big\|^2 \tag{4.25}$$

where $\beta(N,k)$ are specific weights which may depend on the given time k.

In order to derive a recursive formulation, suppose that the weighting sequence has the following property:

$$\begin{aligned}\beta(n,k) &= \begin{cases}\lambda(n)\beta(n-1,k), & 0 \le k \le n-1,\\ 1, & k = n\end{cases}\\ &= \prod_{i=k+1}^{n}\lambda(i)\end{aligned} \tag{4.26}$$

Thus, after some algebraic manipulation, it is possible to obtain the following general form for the RLS algorithm, where the weighting factor $\lambda(k)$ is left unspecified:

$$\varepsilon(k) = \mathbf{y}(k) - \hat{\theta}^T(k-1)\varphi(k), \tag{4.27a}$$

$$\mathbf{R}(k) = \lambda(k)\mathbf{R}(k-1) + \varphi(k)\varphi^T(k), \tag{4.27b}$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + \mathbf{R}^{-1}(k)\varphi(k)\varepsilon^T(k) \tag{4.27c}$$

In Eq. (4.27), the *information matrix* $\mathbf{R}^{-1}(k)$ (also referred to as *covariance matrix* [4]) needs to be inverted at each time step k; in order to avoid this, the matrix $\mathbf{P}(k)$ is introduced in place of matrix $\mathbf{R}^{-1}(k)$, making the recursive step on $\mathbf{R}(k)$ to

[4]Matrix $\mathbf{R}^{-1}(k)$ is referred to as the *covariance matrix* of the parameters $\theta(k)$ since $\mathbf{R}_2(k)\mathbf{R}^{-1}(k)/2$ is approximately equal to the covariance matrix of the estimated parameters (and $\mathbf{R}_2(k)$ is the variance of the innovations, that is the true prediction errors $\mathbf{e}(k)$).

become:

$$\mathbf{P}^{-1}(k) = \mathbf{P}^{-1}(k-1) + \varphi(k)\varphi^T(k) \tag{4.28}$$

From the *matrix inversion lemma* [75, 154]:

$$(\mathbf{A}+\mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{C}^{-1}+\mathbf{DA}^{-1}\mathbf{B}\right)^{-1}\mathbf{DA}^{-1} \tag{4.29}$$

we can make the following substitution:

$$\begin{aligned}
&\mathbf{A} := \lambda(k)\mathbf{R}^{-1}(k-1) = \lambda(k)\mathbf{P}(k-1),\\
&\mathbf{B} := \varphi(k),\\
&\mathbf{C} := 1,\\
&\mathbf{D} := \varphi^T(k)
\end{aligned}$$

and rewrite the recursive algorithm as follows:

$$\varepsilon(k) = \mathbf{y}(k) - \hat{\theta}^T(k-1)\varphi(k), \tag{4.30a}$$

$$\mathbf{P}(k) = \frac{1}{\lambda(k)}\left[\mathbf{P}(k-1) - \frac{\mathbf{P}(k-1)\varphi(k)\varphi^T(k)\mathbf{P}(k-1)}{\lambda(k)+\phi^T(k)\mathbf{P}(k-1)\varphi(k)}\right], \tag{4.30b}$$

$$\mathbf{L}(k) = \frac{\mathbf{P}(k-1)\varphi(k)}{\lambda(k)+\varphi^T(k)\mathbf{P}(k-1)\varphi(k)}, \tag{4.30c}$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + \mathbf{L}(k)\varepsilon^T(k) \tag{4.30d}$$

Initialization Step. The recursive algorithm needs some initial conditions from which to start from. The regression vector is usually initialized to zero, that is:

$$\varphi(k_0) = \mathbf{0} \tag{4.31}$$

For the parameters vector $\hat{\theta}$ and the information matrix $\mathbf{P}(k)$, when no prior knowledge is available, a good choice would be to apply the ordinary least-squares

method to the first $k_0 > (n+m)$ samples (if available):

$$\mathbf{P}(k_0) = \left[\sum_{i=1}^{k_0} \varphi(i)\varphi^T(i)\right]^{-1}, \tag{4.32}$$

$$\hat{\theta}(k_0) = \mathbf{P}(k_0)\sum_{i=1}^{k_0} \varphi(i)\mathbf{y}^T(i) \tag{4.33}$$

However, it is not a simple matter to select the length of data required for ensuring that $\mathbf{P}$ be invertible. Another common choice for $\hat{\theta}(k_0)$ and $\mathbf{P}(k_0)$ is to set:

$$\hat{\theta}(k_0) = \xi\mathbf{1}, \tag{4.34}$$

$$\mathbf{P}(k_0) = \delta\mathbf{I} \tag{4.35}$$

where ξ and δ are some small and large positive constants, respectively. A large value for δ corresponds to large uncertainty about the initial values of the parameters $\hat{\theta}(k)$ (i.e., prior with high variance), thus ensuring a high degree of correction (i.e., adaptation).

Variants of RLS. Several variants have been proposed in literature, each of which essentially differs either for the definition of the weight $\lambda(\cdot)$ or for the recursive step of the information matrix. In the following, we present the ones that we used in this thesis.

RLS with Exponential Forgetting (RLS-EF). This variant is obtained by setting $\lambda(k) = \lambda$ for each k. This approach discounts old measurements exponentially such that an observation that is τ samples old carries a weight that is equal to λ^τ times the weight of the most recent observation. The value $\tau = \frac{1}{1-\lambda}$ represents the *memory horizon* of the algorithm. The parameter $\lambda \in (0,1)$ is called the *forgetting factor* and typically has a positive value between 0.97 and 0.995. The main drawback of this strategy is the so-called *estimator wind-up*, which occurs when the system input is not persistently excited (as shown in [140]).

RLS with Directional Forgetting (RLS-DF). In this variant, first proposed by [81, 104, 103], the discount effect is not uniformly distributed in the parame-

ter space, so that old data is forgotten only in the direction where the new ones are coming from. This is motivated by the fact that, when the input is not persistently excited, no information about system dynamics is available in some direction. Therefore, the forgetting operation should be applied only to the part of the information matrix, where new information is available from input and output data. The RLS-DF algorithm is given by the following steps:

$$\varepsilon(k) = \mathbf{y}(k) - \hat{\theta}^T(k-1)\varphi(k), \tag{4.36a}$$

$$r(k) = \varphi^T(k)\mathbf{P}(k-1)\varphi(k), \tag{4.36b}$$

$$\lambda(k) = \begin{cases} \mu - \frac{1-\mu}{r(k)}, & r(k) > 0, \\ 1, & r(k) = 0 \end{cases}, \tag{4.36c}$$

$$\mathbf{P}(k) = \mathbf{P}(k-1) - \lambda(k)\frac{\mathbf{P}(k-1)\varphi(k)\varphi^T(k)\mathbf{P}(k-1)}{1+\lambda(k)r(k)}, \tag{4.36d}$$

$$\mathbf{L}(k) = \frac{\mathbf{P}(k-1)\varphi(k)}{1+r(k)}, \tag{4.36e}$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + \mathbf{L}(k)\varepsilon^T(k) \tag{4.36f}$$

where the parameter $\mu \in (0,1)$ is called *forgetting factor*. If μ is close to one, the algorithm is sluggish; conversely, for values of μ near to zero, the algorithm is faster but is also more sensible to disturbances.

RLS with Bittanti's Correction (RLS-DF*). This algorithm, proposed by [31], is a variation of the RLS-DF algorithm Eq. (4.38), such that the recursive equation Eq. (4.36d) for the information matrix $\mathbf{P}(k)$ is replaced by the following:

$$\mathbf{P}(k) = \mathbf{P}(k-1) - \lambda(k)\frac{\mathbf{P}(k-1)\varphi(k)\varphi^T(k)\mathbf{P}(k-1)}{1+\lambda(k)r(k)} + \delta\mathbf{I} \tag{4.37}$$

where the parameter $\delta > 0$ is called *correcting factor*. Such a variant enforces an increment of the covariance matrix aiming at improving the alertness of the RLS algorithm, and thus at achieving exponential convergence.

Exponential Weighting RLS (EW-RLS) This variant, proposed by [129], uses a variable forgetting factor, in order to compensate for the inability of the RLS-EF algorithm to track parameter changes in nonstationary (i.e., time-varying) environments. The basic idea is to adequately vary the forgetting factor in order to give a good tracking adaptability and low parameter error variance. If the forgetting factor is kept small when the parameters are changed abruptly, and is increased to unity appropriately so that the estimated parameter vector converges to the true value, then the algorithm has good tracking capabilities during the transient stage and fewer misadjustment errors of parameters in the steady state. The EW-RLS algorithm is given by the following steps:

$$\varepsilon(k) = \mathbf{y}(k) - \hat{\theta}^T(k-1)\varphi(k), \tag{4.38a}$$

$$\lambda(k) = \lambda_0 + (1-\lambda_0)2^{\mu(k)}, \tag{4.38b}$$

$$\mu(k) = -\text{round}(\rho\|\varepsilon\|^2), \tag{4.38c}$$

$$\mathbf{P}(k) = \frac{1}{\lambda(k)}\Big[\mathbf{P}(k-1) - \mathbf{L}(k)\varphi^T(k)\mathbf{P}(k-1)\Big], \tag{4.38d}$$

$$\mathbf{L}(k) = \frac{\mathbf{P}(k-1)\varphi(k)}{\lambda(k) + \varphi^T(k)\mathbf{P}(k-1)\varphi(k)}, \tag{4.38e}$$

$$\hat{\theta}(k) = \hat{\theta}(k-1) + \mathbf{L}(k)\varepsilon^T(k) \tag{4.38f}$$

where round$(\cdot)$ is the nearest integer function, and parameters λ_0 and ρ are called *minimum forgetting factor* and *sensitivity gain*, respectively. The sensitivity gain parameter ρ is a design parameter which controls the width of a unity zone. Specifically, when the error $\varepsilon(k)$ goes to infinity, the value forgetting factor $\lambda(k)$ decreases to the minimum value λ_0. Conversely, when the error $\varepsilon(k)$ goes to zero, the value of the forgetting factor $\lambda(k)$ increases to unity, at an exponential rate. This rate is controlled by the sensitivity gain ρ.

4.1.5 Model Validation

The fitted model is validated against the observed data in order to test if it is "good-enough" to describe the observed system behavior; it the result is unsatisfactory

(with respect to any prior knowledge and to the purpose for which the model is used), model parameters need to be revised and the identification process is reiterated. Informally, a "good model" is the simplest model that best describes the dynamics and successfully simulates or predicts the output for different inputs. The importance of model validation is motivated by the fact that an under-parameterized model is inaccurate and not enough flexible, while an over-parameterized model is not parsimonious and leads to unnecessary complicated computations.

The goodness of a model is usually determined by two factors, namely the model quality and the model price. The *model quality* is a scalar measure of the goodness of a model. One common choice for it is the *mean-squared error* (MSE), which represents the expected value of the quadratic prediction error. The MSE is equal to the sum of the variance and the squared bias of the model estimator, and thus it assesses the quality of the model in terms of its variation and unbiasedness. Usually, when considering the bias-variance reduction problem, one has to seek a good trade-off between flexibility (i.e., if the model structure is "large" enough to cover the true system) and parsimony (i.e., if the model structure does not use unnecessary parameters to model the system), since the bias reduction leads to more flexible model structures but increases model parameters, and variance reduction decreases the number of estimated parameters but leads to less flexible model structures. For what concerns the *model price*, it takes into consideration the algorithm complexity, and hence the associated computational time needed to estimate the model.

The first step in the model validation procedure is the choice of the *validation set* (or *test set*), which is the set that has to be used for the evaluation of the model quality. There are essentially two approaches to choose the validation test. The first approach consists in using the same dataset used for model estimation. The other approach, called *cross-validation test*, uses a dataset that is different from the one used for model validation. This can be made possible by, for instance, dividing the initial dataset into two parts. A common choice is to select the first $2/3$th of the total number of samples for the parameter estimation, and using the remaining $1/3$th for evaluating the quality of the model by computing the value of the prediction-error cost function and performing at least one of the above correlation tests. Usually the cross-validation test is the preferred approach for

model validation since it overcomes the chance of "over-fitting". However, there are cases where the cross-validation test cannot be applied, for instance when there is a small number of observation in the original dataset.

Once, the validation set is chosen, one can use it to evaluate the flexibility of the model. There are basically two ways to tackle this problem: a graphical approach and an approach based on statistical tests on the prediction error [152]. The graphical approach is based on plots and common sense. In this approach, the observed and the predicted outputs are plotted with respect to the same observed input sequence. In a good model, the predicted output should resemble the observed output; this evaluation is done visually according to prior knowledge and to the purpose for which the model has been created (e.g., for simulation or for prediction). For what regards the approach based on statistical tests, it consists in using the *residual analysis* on the prediction error $\varepsilon(k)$ (i.e., the residual). If the model is capable of describing the observed data, the residual is zero-mean white noise and independent of the input signal. To verify this hypothesis, different statistical tests can be used. For instance, with the *auto-correlation test*, it is possible to test if the residual is zero-mean white noise. In this test, the auto-correlation function is estimated from the dataset and the model quality is evaluated by checking whether the estimated correlation function represents a unit pulse (within a certain confidence interval). Another commonly used test is the *cross-correlation test*, which can be used to test if the residual is independent of the input signal. In this test, the cross-correlation function is estimated from the dataset and the model quality is evaluated by checking whether the estimated correlation function equals zero (within a certain confidence interval).

Chapter 5

Linear Control Theory

Control theory is an interdisciplinary science, originating in engineering and mathematics, that deals with the behavior of dynamical systems.

> In this thesis, we focus on the adaptive feedback control structure with self-tuning regulation scheme, and we use the linear quadratic control design to synthesize controller parameters.

In the rest of this chapter, we provide an overview of control theory for linear systems, with a particular focus on the techniques used in this thesis. First, in Section 5.1, we introduce basic notions of linear control theory. Then, in Section 5.2, we present common control structures. In Section 5.3, we describe the response of linear systems under the closed-loop control structure. Finally, in Section 5.4, we present common control design techniques.

5.1 Basic Definitions

A *control system* is an arrangement of components connected or related in such a manner as to command, direct, or regulate itself or another system. The essential elements of a control system are:

- *Target system*, which is the system to be controlled. It is also called *plant*, *process*, or *controlled system*.

- *Control input*, which is the stimulus (or excitation, or command) applied to the target system thus affecting its behavior and that can be adjusted dynamically. It is also referred to as *control signal* or *manipulated variable*.
- *Measured output*, which is a measurable characteristic of the target system and represents the actual response obtained from the target system. It is also known as *controlled output*.
- *Reference input*, which is the desired value of the measured outputs (or transformations of them). This is also referred to as *desired output* or *set-point*.
- *Controller*, which determines the setting of the control input needed to achieve a specified response from the target system (i.e., the reference input). The controller computes values of the control input according to a given mathematical law, usually referred to as *control law* (or *control algorithm*).
- *Disturbance input*, which is any uncontrollable change that affects the way in which the control input influences the measured output.
- *Noise input*, which is any uncontrollable effect that changes the measured output produced by the target system. This is also called *sensor noise* or *measurement noise*.

A control system may be part of a larger system, in which case it is called a *control subsystem* (or simply *subsystem*), and its inputs and outputs may then be internal variables of the larger system.

A typical example of control system is the cruise control of a car, which is a device designed to maintain the vehicle speed at a constant desired speed provided by the driver. The controller is the cruise control, the target system is the car, and the control system is the car and the cruise control. The system output is the speed of the car, and the control input itself is the engine's throttle position which determines how much power the engine generates. Possible disturbances include the wind direction and speed.

Controllers are designed for some intended purpose, commonly referred to as *control objective* (or *control goal*). Some common control objectives are:

- *Regulation* (or *disturbance rejection*), to ensure that the measured output is equal to (or near) the reference input. The goal is to keep the measured output near to the reference input in spite of disturbances.

- *Tracking*, to follow changes in the reference input. Tracking often comes with disturbance rejection requirements as well.

- *Optimization*, rather than regulating a variable to a specified level, sometimes there is a need to optimize a variable. This can be seen as regulating the derivative of the variable to zero.

A central task in control theory is the design of control systems. The *control design* is the process of design of control systems in order to obtain the configuration, specifications, and identification of the key parameters of a given system to meet an actual need. The ultimate goal of the controller design is, given a model of the system to be controlled (including its sensors and actuators) and a set of control objectives, to find a suitable controller, or determine that none exists.

The control system design process, shown in Fig. 5.1 (from [57]), consists of seven main building blocks, which can be arranged into three groups:

1. Establishment of goals and variables to be controlled, and definition of specifications (metrics) against which to measure performance.

2. System definition and modeling.

3. Control system design and integrated system simulation and analysis.

The first step in the design process consists of establishing the control goals. For example, we may state that our goal is to control the response time of a Web server accurately.

The second step is to identify the variables that we desire to control (for example, the response time of the Web server).

The third step is to write the specifications in terms of the accuracy we must attain. This required accuracy of control will then lead to the identification of a sensor to measure the controlled variable. The performance specifications will describe how the control system should perform and will include (1) good regulation

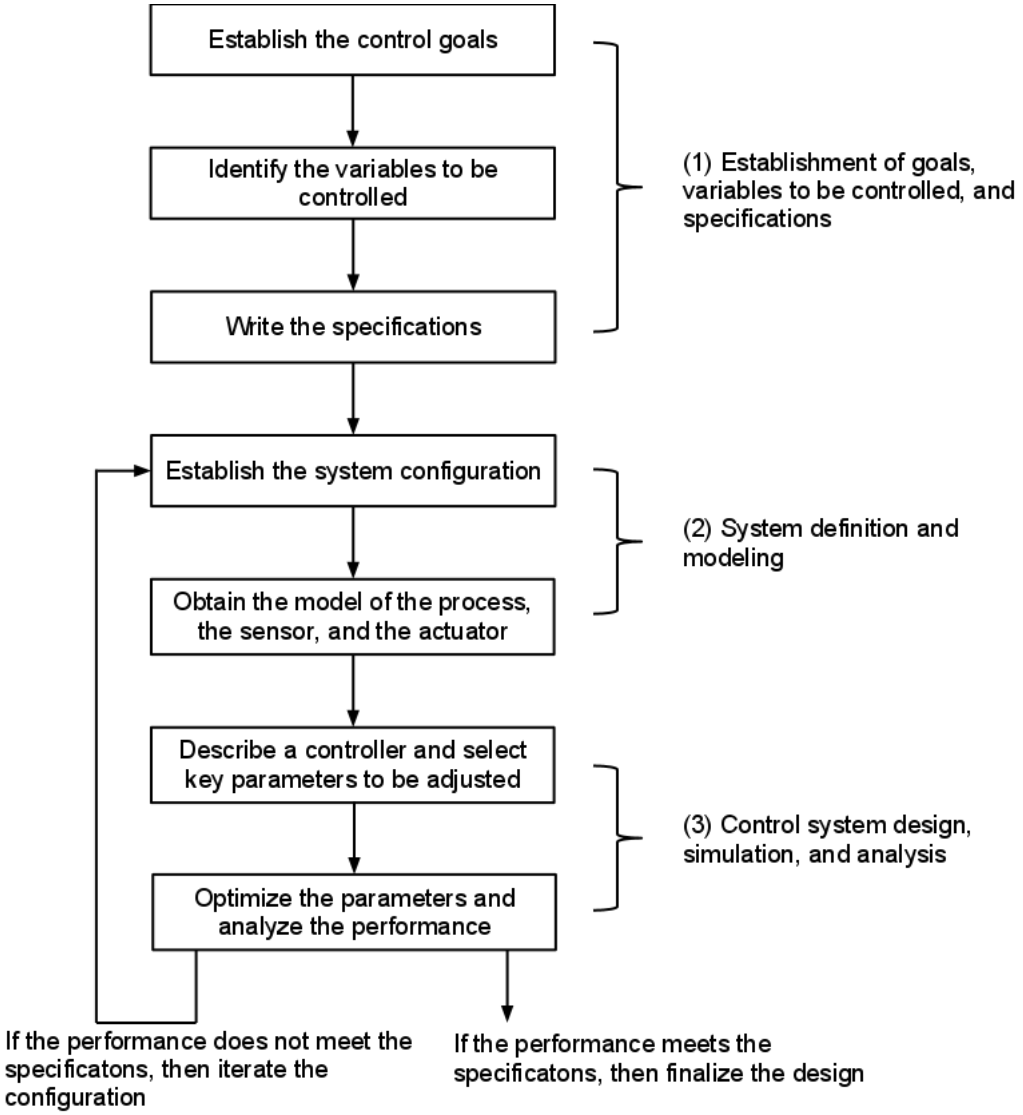


Figure 5.1: The control system design process (from [57]).

against disturbances, (2) desirable responses to commands, (3) realistic actuator signals, (4) low sensitivities, and (5) robustness.

The fourth step is concerned with the configuration of a system that will result in the desired control performance. This system configuration will normally consist, at least, of a sensor, the process under control, an actuator, and a controller.

The fifth step consists of identifying a model for each component of the control system, such as the sensor and the actuator. This will, of course, depend on the process, but the actuation chosen must be capable of effectively adjusting the performance of the process. For example, in the control of the response time of a Web server, we can select the buffer length as the actuator. The sensor, in this case, must be capable of accurately measuring the buffer size.

The sixth step is the selection of a controller, which often include a feedback signal used to compare the desired response and the actual response.

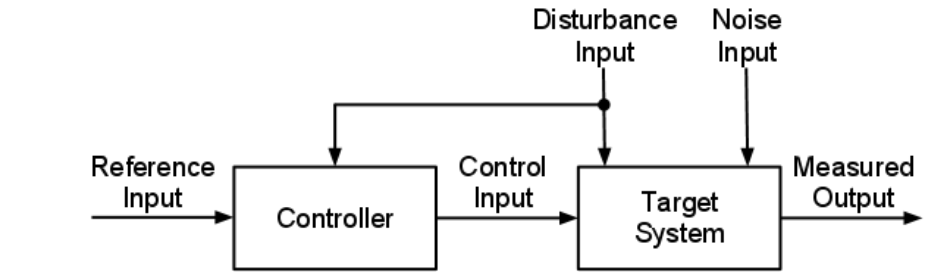
The final step is the adjustment of the parameters of the system to achieve the desired performance. If we can achieve the desired performance by adjusting the parameters, we will finalize the design. If not, we will need to establish an improved system configuration and perhaps select an enhanced actuator and sensor. Then we will repeat the design steps until we are able to meet the specifications, or until we decide the specifications are too demanding and should be relaxed.

5.2 Control Structures

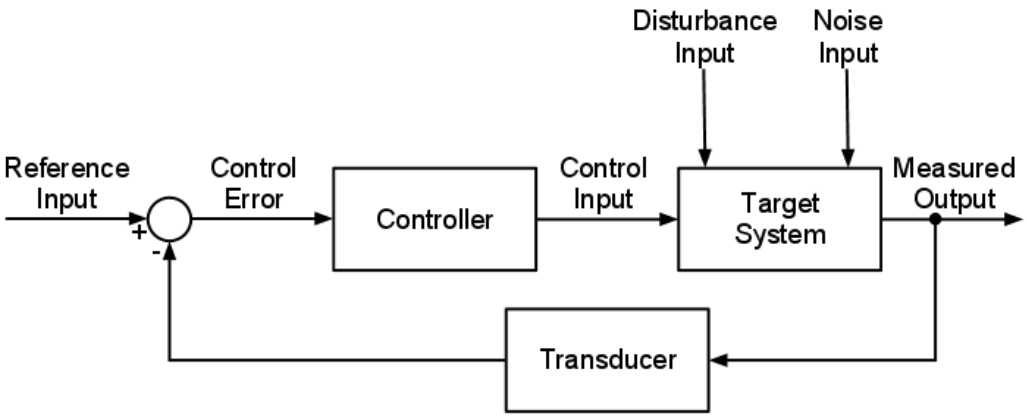
In this section we describe basic structure of control systems, that can be used to build more complex structures. First, we introduce the two simplest and most common control structures, namely open-loop and closed-loop control structures. Then, we present some other commonly used control structures.

5.2.1 Open-loop and Closed-loop Control Structure

As shown in Fig. 5.2 (from [84]), control systems are classified into two general categories: open-loop and closed-loop systems. The distinction is determined by the *control action*, which is responsible for activating the system to produce the output.



(a) Open-loop control system.



(b) Closed-loop control system.

Figure 5.2: Control system categories (from [84]).

Table 5.1: Comparison of open- and closed-loop control systems.

Feature	Open-loop	Closed-loop
Avoid using measured outputs	Yes	No
Cannot make stable systems unstable	Yes	No
Simple (i.e., not accurate) system model	No	Yes
Adapts/compensate to disturbances or noise	No	Yes

An *open-loop* (or *feedforward*) control system (shown in Fig. 5.2a) is one in which the control action is independent of the output. [1] In such a system, the controller computes values of the control input based on only the current value of the reference input.

In contrast to an open-loop control system, a *closed-loop* (or *feedback*) control system (shown in Fig. 5.2b) is one in which the control action is somehow dependent on the output. In such a control system, the controller component determines the setting of the control input, needed to achieve the reference input, according to current and past values of *control error*, which is the difference between the reference input and the measured output. The measure of the output used to compute the control error is called *feedback signal*. The transmission path from the summing point to the measured output is called *forward path*, while the transmission path from the measured output back to the summing point is called *feedback path*. Components of the control systems located on the feedforward path are called *feedforward (control) elements*, and typically include controller(s), compensator(s) and amplifier(s). Instead, components located located on the feedback path are referred to as *feedback (control) elements*, and typically include sensor of the measured output, compensator(s), and other controller elements.

The system shown in Fig. 5.2b is a *negative feedback* control system, because the output is subtracted from the input and the difference is used as the input signal to the controller.

Table 5.1 summarizes the comparison of open- and closed-loop systems in

[1] It is important to note that, according to some author(e.g., [88]), open-loop and feedforward control denote two different control structure. Specifically, an open-loop control system is a control system that is not able to deal with any disturbance, while feedforward control system is a control system which is able to take into account of disturbances to the system by measuring the disturbances and altering the control action accordingly.

terms of few important features. On the one hand, two outstanding features of open-loop control systems are:

- their ability to perform accurately is determined by their *calibration*, that is by their ability to establish (or reestablish) the input-output relation to obtain a desired system accuracy, and
- they do not usually suffer of instability problems.

On the other hand, closed-loop controllers have a number of advantages over open-loop controllers:

- unstable processes can be stabilized,
- guaranteed performance even with model uncertainties, when the model structure does not match perfectly the real process and the model parameters are not exact,
- reduced sensitivity to parameter variations,
- improved reference tracking performance, and
- disturbance rejection.

Although open-loop systems have are appealing for their ability of reducing design complexity (e.g., avoiding the use of measured outputs) and ensuring stability, they are rarely used in practice because they cannot adapt to change and it is almost impossible to obtain an accurate system model (at least for what concerns the modeling of computing systems).

For these reasons, in this thesis we focus on closed-loop systems.

It is worth noting that, in some systems, open- and closed-loop controls are used simultaneously. In such systems, the open-loop control is termed *feedforward* and serves to further improve reference tracking performance.

5.2.2 Other Control Structures

Besides open- and closed-control, many other control structures are used in practice, but they can be seen as a combination, or perhaps a repeated combination of these two basic concepts.

Two Degrees-of-Freedom (2DoF) Control

The stability properties of a controlled system are determined by the control loop(s). When stability is of concern, and there are also disturbances acting on the system, and/or there are tracking requirements, a single loop control strategy does not provide enough design freedom to achieve all objectives. Because signals can be shaped by systems, the tracking performance clearly depends not only on the loop but also on any system in cascade. This reasoning leads to the very common control structure with two degrees of freedom, called *two degrees-of-freedom* (2DoF) *control* [89]. This is a standard technique in linear control theory that separates a controller into a feedforward compensator and a feedback compensator. The feedforward compensator generates the nominal input required to track a given reference trajectory. The feedback compensator corrects for errors between the desired and actual trajectories.

Cascade Control

Full system state information may be unavailable, or difficult to use at once. In *cascade control*, successive control loops are used, each using a single measurement and a single actuated variable. The output of the primary controller is an input to the secondary and so on. A judicious choice of how to pair variables and the ordering of the multiple loops can lead to a very efficient control implementation without the need for a full state feedback control.

Adaptive Control

In an *adaptive control* structure [140], the controller must adapt to a controlled system where parameters may vary over time, or are initially uncertain. The way the parameter estimation law, that is the *adaptation law*), is combined with the

control law gives rise to different approaches to adaptive control. In general one should distinguish between two approaches:

- *Direct methods*, where the estimated parameters are those directly used in the adaptive controller. In this approach, the plant model is parameterized in terms of the desired controller parameters, which are then estimated directly without intermediate calculations involving plant parameter estimates. This approach has also been referred to as *implicit adaptive control* because the design is based on the estimation of an implicit plant model.

- *Indirect methods*, where the estimated parameters are used to calculated required controller parameters. In this approach, the plant parameters are estimated online and used to calculate the controller parameters. In other words, at each control time, the estimated plant is formed and treated as if it is the true plant in calculating the controller parameters (i.e., the so called *certainty equivalence principle*). This approach has also been referred to as *explicit adaptive control*, because the controller design is based on an explicit plant model.

In literature, several adaptation schemes have been proposed. The most widely used adaption schemes are: gain scheduling, self-tuning regulation, and model reference adaptive control.

Gain scheduling. In some situation it is known how system dynamics change with the operating conditions. It is then possible to change the parameters of the controller by monitoring the operating conditions of the system. This idea is called *gain scheduling*. Essentially, the gain scheduler consists of a lookup table and the appropriate logic for detecting the operating point and choosing the corresponding value of other controller gain from the lookup table. With this approach, plant parameter variations can be compensated by changing the controller gains as functions of the input, output, and auxiliary measurements. The advantage of gain scheduling is that the controller gains can be changed as quickly as the auxiliary measurements respond to parameter changes. Frequent and rapid changes of the controller gains, however, may lead to instability [153]; therefore, there is a limit to how often and how fast the controller gains can be changed. One

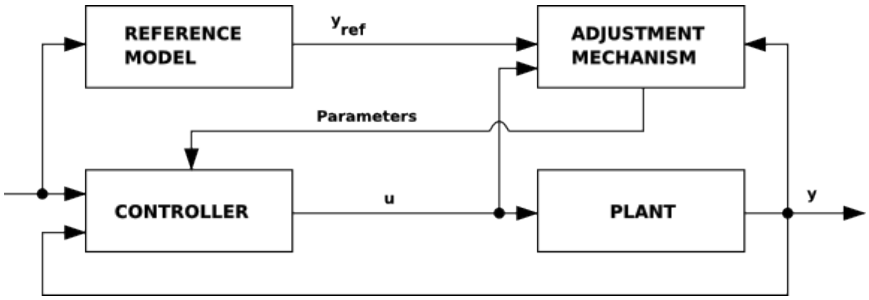


Figure 5.3: Adaptive control – The MIAC structure.

of the disadvantages of gain scheduling is that the adjustment mechanism of the controller gains is precomputed offline and, therefore, provides no feedback to compensate for incorrect schedules.

Self-Tuning Regulation (STR). The *Self-Tuning Regulation* (STR) adaptation scheme, also known as *Model Identification Adaptive Control* (MIAC), is based on the idea of separating the estimation of unknown parameters from the design of the controller (as shown in Fig. 5.4). Specifically, target system parameters are estimated on-line and controller parameters are obtained from the solution of a control design problem using such estimated parameters as if they were correct (as stated by certainty equivalence principle) The STR scheme is composed of two control loops: (1) the inner loop, which contains the target system (i.e., the "target system" box) and an ordinary feedback controller (i.e., the "controller" box), acts on the controlled system in order to track the reference signal, while (2) the outer loop, which is composed by a recursive parameter estimator (i.e., the "estimation" box) and design calculations (i.e., the "controller design" box), adjusts the parameters of the inner controller.

Model Reference Adaptive Control (MRAC). The *Model Reference Adaptive Control* (MRAC) adaptation scheme, also known as *Model Reference Adaptive System* (MRAS), is an adaptive control technique where the performance specifications are given in terms of a model, which represents the ideal response of the process to a reference signal (see Fig. 5.4). The basic idea of MRAC, is to create a closed-loop controller with parameters that can be updated to change the response

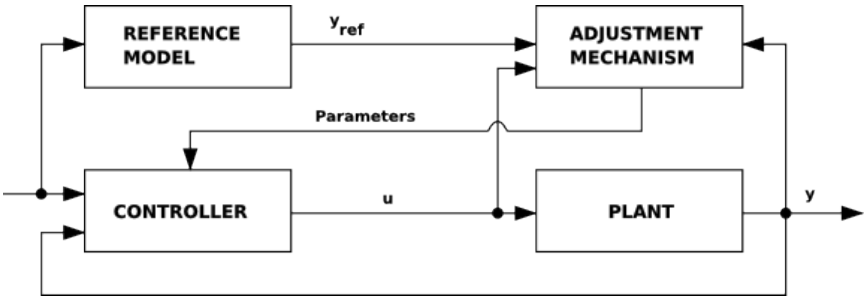


Figure 5.4: Adaptive control – The MRAC structure.

of the system. First, the output of the system is compared to a desired response from a reference model, and then control parameters are update based on this error. The goal is for the parameters to converge to ideal values that cause the system response to match the response of the reference model.

Optimal Control

In an *optimal control* structure [122], the objective of the controller is to "determine the control signals that will cause a process to satisfy the physical constraints and, at the same time, minimize (or maximize) some performance criterion" [99]. Put in another way, optimal control deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved.

If the information which the control system must use is uncertain or if the dynamic system is subjected to random disturbances, it may not be possible to satisfy this criterion with certainty; in this case, the best one can hope is to minimize (or maximize) its expected value. This leads to the concept of *stochastic optimal control*.

An optimal control problem includes (1) a mathematical representation of the controlled system, (2) a cost functional (or performance index) which is a function of state and control variables, and (3) a statement of boundary conditions and physical constraints on states and/or controls.

A special case of optimal control structure is the *Linear Quadratic Regulator* (LQR) optimal control [107], a feedback controller where the optimal control problem (in this case, called the LQ problem) comprises a set of linear difference equations and the performance index is described by a quadratic functional. More

details about the LQR, will be provided below, in Section 5.4.2.

Decentralized and Hierarchical Control

Decentralized control structures [150] present a practical and efficient way for designing control algorithms that utilize just the state of each subsystem, possibly without any information from other subsystems, to achieve a specific control objective (e.g., the regulation of each subsystem state to zero)..

A special case of decentralized control structure is the *hierarchical control* structure, where a coordinating (centralized) control is introduced to ensure that the local controls are properly modified according to a common global objective. Specifically, a hierarchical control system is a control system in which a set of devices and governing software is arranged in a hierarchical tree. Each element of the hierarchy is a *linked node* in the tree. Commands, tasks and goals to be achieved flow down the tree from superior nodes to subordinate nodes, whereas command results flow up the tree from subordinate to superior nodes. Nodes may also exchange messages with their siblings. The two distinguishing features of a hierarchical control system are [64]:

- each higher layer of the tree operates with a longer interval of planning and execution time than its immediately lower layer;

- the lower layers have local tasks and goals, and their activities are planned and coordinated by higher layers which do not generally override their decisions.

5.3 Response of Closed-loop Control Systems

Usually, closed-loop control systems are used for various purposes and must meet certain performance requirements. These requirements not only affect such things as speed of response and accuracy, but also the manner in which the system responds in carrying out its control function. All systems contain certain errors. The problem is to keep them within allowable limits.

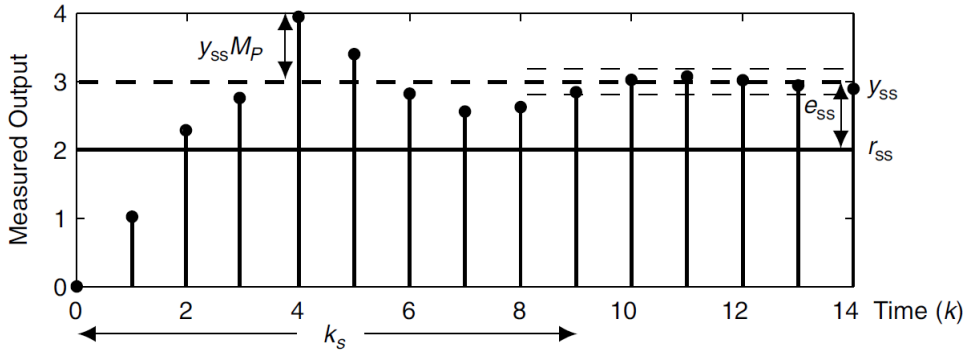


Figure 5.5: Closed-loop control properties (from [84]). Depicted are the reference input r_{ss}, the steady-state output y_{ss}, the steady-state error e_{ss}, the settling time k_s, and the maximum overshoot M_p.

The most important properties to study when considering the response of a closed-loop system are related to the concepts of stability, accuracy, settling time, and overshoot:

- *Stability*. A system is said to be *stable* if for any bounded input, the output is also bounded (see definition Def. 3.5.1 of BIBO stability in Section 3.5). Stability is typically the first property considered in designing control systems since unstable systems cannot be controlled.

- *Accuracy*. The control system is accurate if the measured output converges (or becomes sufficiently close) to the reference input. Accurate systems are essential to ensuring that control objectives are met. Typically, we do not quantify accuracy. Rather, we measure *inaccuracy* (or *steady-state error*), which is the steady-state value of the control error.

- *Settling time*. The time required for the response curve to reach and stay within a range of certain percentage (usually 5% or 2%) of the final, steady-state value. The system has short settling times if it converges quickly to its steady-state value. Short settling times are particularly important for disturbance rejection in the presence of time-varying workloads so that convergence is obtained before the workload changes.

- *Overshoot*. The term *overshoot* is used to refer to an output signal exceeding

its final, steady-state value. Under stability condition, the system should achieve its objectives in a manner to reduce the overshoot. Thus, we are interested to quantity the *maximum overshoot* of the system response, that is the maximum peak value of the response curve measured from the desired response of the system. Overshoot is usually followed by the *ringing* (or *hunting*) phenomenon, representing unwanted oscillations around the final, steady-state value.

These four properties are usually referred to as *SASO properties* [84]. In Fig. 5.5 (from [84]), the meaning of each SASO properties is graphically shown for the response of a stable system to a step change in the reference input r_{ss}. At time 0, the reference input r_{ss} changes from the value 0 to 2. The system reaches its steady-state output value y_{ss} when its measured output always lies between the lightweight dashed lines.

5.4 Closed-loop Control Design

In this section we present common approaches used to design closed-loop controls.

5.4.1 Proportional-Integral-Derivative Control

In the *proportional-integral-derivative* (PID) control [139], the control action is the sum of three separate terms related to the current error, the integral of the past error, and a simple prediction of the future error.

The *proportional* (P) action is proportional to the observed error. It provides an immediate response to a current error. The control law for the proportional action is given by:

$$\mathbf{u}_P(k) = \mathbf{K}_P \mathbf{e}(k) \tag{5.1}$$

where $\mathbf{u}_P(k)$ is the contribution of the proportional part to the control action $\mathbf{u}(k)$ at time k, $\mathbf{K}_P \in \mathbb{R}^{n_y \times n_y}$ is the *proportional gain matrix*, and $\mathbf{e}(k)$ is the control error at time k.

The *integral* (I) action integrates the past error to provide an enduring response. It aims to remove steady-state error in a regulation environment. The control law

Table 5.2: Effect of independent PID parameters tuning (from [23]).

Parameter	Overshoot	Settling time	Steady-state error	Stability
$\mathbf{K}_P$	Increase	Small increase	Decrease	Degrade
$\mathbf{K}_I$	Increase	Increase	Large decrease	Degrade
$\mathbf{K}_D$	Decrease	Decrease	Minor change	Improve (if $\mathbf{K}_D$ is small)

for the integral action is given by:

$$\mathbf{u}_I(k) = \mathbf{K}_I \sum_{i=0}^{k} \mathbf{e}(i) \tag{5.2}$$

where $\mathbf{u}_I(k)$ is the contribution of the integral part to the control action at time k, $\mathbf{K}_I \in \mathbb{R}^{n_y \times n_y}$ is the *integral gain matrix*, and $\mathbf{e}(k)$ is the control error at time k.

The *derivative* (D) action anticipates the process response by taking action proportional to the derivative of the error. It aims to provide better damping, and faster response.

$$\mathbf{u}_D(k) = \mathbf{K}_D\big(\mathbf{e}(k) - \mathbf{e}(k-1)\big) \tag{5.3}$$

where $\mathbf{u}_I(k)$ is the contribution of the integral part to the control action at time k, $\mathbf{K}_D \in \mathbb{R}^{n_y \times n_y}$ is the *derivative gain matrix*, and $\mathbf{e}(k)$ is the control error at time k.

Different combination of the above control terms can lead to different control laws. For instance, with a PI controller, the control law is given by the sum of the proportional control law $\mathbf{u}_P(\cdot)$ with the integral control law $\mathbf{u}_I(\cdot)$. It is worth noting that a D controller is never used by itself since, if the error remains constant, the output of the derivative controller would be zero.

The resulting control law for the PID controller is thus given by the sum of the above three terms:

$$\begin{aligned} \mathbf{u}(k) &= \mathbf{u}_P(k) + \mathbf{u}_I(k) + \mathbf{u}_D(k) \\ &= \mathbf{K}_P\mathbf{e}(k) + \mathbf{K}_I \sum_{i=0}^{k} \mathbf{e}(i) + \mathbf{K}_D\big(\mathbf{e}(k) - \mathbf{e}(k-1)\big) \end{aligned} \tag{5.4}$$

In general, the design and the tuning of a PID controller is a difficult problem.

The design of a PID controller involves the selection of three gains, while the tuning of a PID controller is the task of adjusting its control parameters to the optimum values for the desired control response. Stability is usually a basic requirement, but beyond that, different systems have different behavior, different applications have different requirements, and requirements may conflict with one another. The primary difficulty in designing and tuning a PID controller, is that the three gain parameters depends from each other. This can also be observed in Table 5.2 (from [23]), which summarizes the individual effects of the three terms of a PID control on the closed-loop performance. There is a vast scientific literature dedicated to the design and the tuning of PID controllers (e.g., see [139, 93, 125]).

5.4.2 Linear Quadratic Control

The *Linear Quadratic* (LQ) control design is a widely adopted technique for designing optimal controls [107, 122]. Basically, an LQ controller tries to compute the best control inputs in order to follow a zero trajectory.

The LQ design is used to design state-feedback controllers, and thus works with system models in the state-space representation. In the rest of this section, we assume the following discrete-time state-space system model:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k), \tag{5.5a}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k) \tag{5.5b}$$

Linear Quadratic Regulator

Given the system Eq. (5.5), the infinite-horizon discrete-time *Linear Quadratic Regulator* (LQR) is an LQ controller whereby the following quadratic cost function is minimized:

$$\mathbf{J}(\mathbf{u}) = \sum_{k=0}^{\infty} \left(\mathbf{x}(k)^T \mathbf{Q}\mathbf{x}(k) + \mathbf{u}(k)^T \mathbf{R}\mathbf{u}(k) + 2\mathbf{x}(k)^T \mathbf{N}\mathbf{u}(k)\right) \tag{5.6}$$

where $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{N}$ are the *output weighting matrix*, the *control weighting matrix* and the *cross-coupling matrix* (also known as *transmission matrix*), respectively.

It is possible to show that under the following condition [86]: [2]

$$(\mathbf{A},\mathbf{B}) \text{ is } \textit{stabilizable}, \tag{5.7a}$$

$$\left(\mathbf{A}-\mathbf{B}\mathbf{R}^{-1}\mathbf{N}^T, \mathbf{Q}-\mathbf{N}\mathbf{R}^{-1}\mathbf{N}^T\right) \text{ is } \textit{detectable}, \tag{5.7b}$$

$$\mathbf{R} > 0, \tag{5.7c}$$

$$\mathbf{Q}-\mathbf{N}\mathbf{R}^{-1}\mathbf{N}^T \geq 0 \tag{5.7d}$$

the optimal control sequence $\mathbf{u}^*(k)$ that minimizes Eq. (5.6), at control interval k, is

$$\mathbf{u}^*(k) = -\mathbf{L}\mathbf{x}(k) \tag{5.8}$$

where $\mathbf{L}$, given by:

$$\mathbf{L} = \left(\mathbf{B}^T\mathbf{S}\mathbf{B}+\mathbf{R}\right)^{-1}\left(\mathbf{B}^T\mathbf{S}\mathbf{A}+\mathbf{N}^T\right) \tag{5.9}$$

is the feedback gain matrix obtained from the solution $\mathbf{S}$ of the associated infinite-horizon *Discrete-time Algebraic Riccati Equation* (DARE):

$$\mathbf{A}^T\mathbf{S}\mathbf{A}-\mathbf{S}-\left(\mathbf{A}^T\mathbf{S}\mathbf{B}+\mathbf{N}\right)\left(\mathbf{B}^T\mathbf{S}\mathbf{B}+\mathbf{R}\right)^{-1}\left(\mathbf{A}^T\mathbf{S}\mathbf{B}+\mathbf{N}\right)^T+\mathbf{Q}=0 \tag{5.10}$$

If conditions shown in Eq. (5.7) hold, then the DARE Eq. (5.10) has a unique, symmetric, positive-semidefinite and stabilizing solution $\mathbf{S}$.

It is important to note that, for a tracking control objective (i.e., when the trajectory to follow is different from zero), we can still use an LQR controller by simply using as state $\mathbf{x}(k)$ the difference between the desired (reference) value $\mathbf{x}_r(k)$ and the actual value $\mathbf{x}_a(k)$:

$$\mathbf{x}(k) = \mathbf{x}_r(k) - \mathbf{x}_a(k) \tag{5.11}$$

In this way, the tracking problem reduces to a regulation problem where now the LQR controller try to minimize the error between the desired and the actual value.

[2] A matrix pair $(\mathbf{A},\mathbf{B})$ is said *stabilizable* if all of its eigenvalues outside the unit disk of the complex plane are controllable. A matrix pair $(\mathbf{A},\mathbf{B})$ is said *detectable* if it has no unobservable mode on the unit circle of the complex plane.

Linear Quadratic Regulator with Output Weighting

The *Linear Quadratic Regulator with Output Weighting* (LQRY) is a state-feedback regulator where the quadratic cost function to be minimized contains the output variable in place of the state variable; that is:

$$\mathbf{J}(\mathbf{u}) = \sum_{k=0}^{\infty} \left(\mathbf{y}(k)^T \mathbf{Q} \mathbf{y}(k) + \mathbf{u}(k)^T \mathbf{R} \mathbf{u}(k) + 2\mathbf{y}(k)^T \mathbf{N} \mathbf{u}(k) \right) \tag{5.12}$$

Basically, an LQRY can we considered as an LQR with weighting matrices $\bar{\mathbf{Q}}$, $\bar{\mathbf{R}}$, and $\bar{\mathbf{N}}$, such that:

$$\begin{pmatrix} \bar{\mathbf{Q}} & \bar{\mathbf{N}} \\ \bar{\mathbf{N}}^T & \bar{\mathbf{R}} \end{pmatrix} = \begin{pmatrix} \mathbf{C}^T & \mathbf{0} \\ \mathbf{D}^T & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{Q} & \mathbf{N} \\ \mathbf{N}^T & \mathbf{R} \end{pmatrix} \begin{pmatrix} \mathbf{C} & \mathbf{D} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \tag{5.13}$$

Part II

Methodology

Chapter 6

The Resource Management Framework

The problem of reducing the TCO of a cloud infrastructure provider and, at the same time, achieving SLAs of hosted services, is a difficult problem due to the conflicting nature of these two objectives. Intuitively, on the one hand, resource over-provisioning helps to achieve SLAs, but increases the TCO. On the other hand, resource under-provisioning helps to reduce the TCO, but increases the possibilities of incurring in SLA violations.

In this chapter, we present a framework able to automatically manage physical and virtual resources of a cloud infrastructure in such a way to maximize the profit of the IaaS provider by minimizing SLA violations while, at the same time, reducing the energy consumed by the physical infrastructure.

Basically, we accomplish this goal (1) by providing each application with the minimum amount of physical resource capacity needed to meet its SLAs, (2) by dynamically adjusting it according to various parameters, that include the intensity of its workload, the number of competing VMs allocated on the same physical resource, and their time-varying behavior induced by variations in the respective workloads, and (3) by consolidating as many VMs as possible onto few physical machines, thus powering off those that are unused.

The rationale underlying this approach is that, in order to balance energy consumption and SLAs satisfaction, each application needs exactly the fraction of

physical resource capacity dictated by current operating conditions of the cloud infrastructure (e.g., workload characteristics and physical resource utilization, just to name a few). As a matter of fact, on the one hand, a greater amount of physical resource capacity would imply an increase of energy consumption without any benefit to the profit (i.e., to stay away from the performance target is essentially identical – in terms of positive income – to stay very close to such objective). On the other hand, a smaller fraction of physical resource capacity would increase the chance of incurring in a SLA violation.

The rest of this chapter is organized as follows. First, in Section 6.1, we provide a high-level description of the architecture of the resource management framework. Then, we focus on the design of the Resource Manager, the core of our framework, which comprises three types of components: the Application Manager, the Physical Machine Manager, and the Migration Manager. Specifically, in Section 6.2, we present the design of the Application Manager component. Then, in Section 6.3, we describe the design of the Physical Machine Manager component. Finally, in Section 6.4, we present the design of the Migration Manager component.

6.1 System Architecture

The goal of our framework is three-fold: (1) to provide automated resource management mechanisms and policies, (2) to monitor and maintain application performance targets and (3) to reduce energy consumption in cloud computing systems.

A high-level architecture of our framework is depicted in Fig. 6.1, where a certain number of user applications (on the left side) have to be deployed as virtualized services on a cloud infrastructure (on the right side). The applications we considered are multi-tier; this choice does not limit the applicability of our framework since other type of applications (like high-performance computing applications) could always be modeled as single-tier applications. As shown in the figure, every application tier is deployed in a separate VM, which in turn is placed on one of the available physical machines. At the center of the figure, the *Resource Manager* continuously monitors the performance of each deployed application and suitably acts on the system in order to maintain application performance goals and, at the same time, to minimize the energy consumption of computing resources.

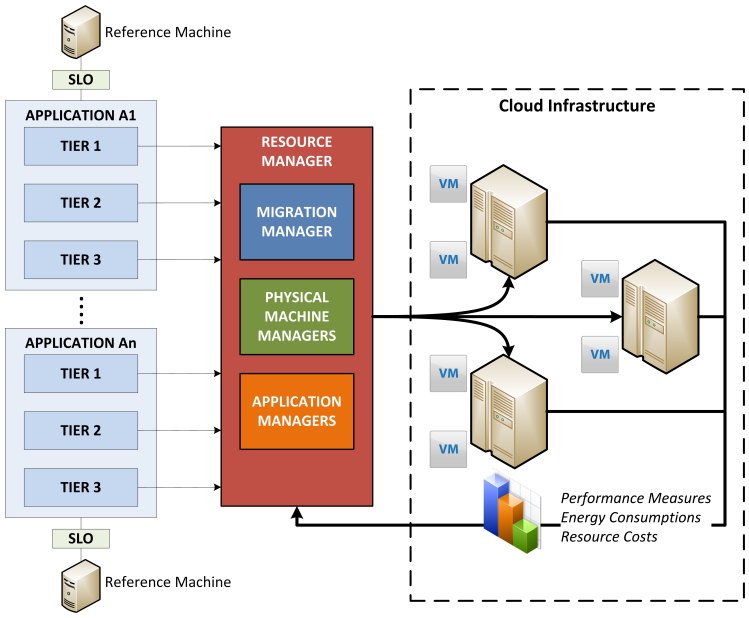


Figure 6.1: Architecture of the proposed framework.

We suppose that each application comes with its SLA specifications. An SLA is a formal description of the level of a service (i.e., of the guarantees offered for hosting a certain application), upon which two negotiating parties (the provider and the recipient of the service) agree. It is commonly described in terms of *Service Level Objectives* (SLOs), which in turn define temporal and performance metrics for measuring the quality of service (QoS). To illustrate, an SLA may specify the level of availability, reliability and performance that must be guaranteed by the infrastructure provider along with economical penalties the provider must pay in case of QoS violation; a possible SLO may specify that 99% of served user requests has a response time no greater than a particular amount of seconds.

Various techniques to determine the SLOs corresponding to a given SLA have been published in the literature (e.g., [105, 169]). For instance, in [105] a state-space modeling approach is integrated with Bayesian probabilistic techniques in order to create several micro-models, each of which representing different behaviour of the modeled application, while in [169] a combination of micro-benchmarks and statistical regression is presented.

We assume that the SLO constraints of each application are known, and are

expressed in terms of a specific physical machine that we called *reference machine* (e.g., in the Amazon EC2 terminology, this could be the equivalent of the "standard instance" [1]). This choice appears to be natural, as (1) application performance generally vary according to the capacity of physical resources assigned to that application, and (2) physical resources inside cloud computing systems are usually heterogeneous. It is responsibility of the Resource Manager to appropriately scale SLO constraints according to the capacity of physical resources belonging to the physical machines where each application tier is actually run. To do so, we assume that the relative computing power of two physical resources of the same category (i.e., the measure of how much a physical resource is more powerful than another one) can be expressed by a simple proportional relationship between the capacity of the two resources. This means that if a resource has capacity equals to 10, it will be able to serve requests at a service rate double than a resource with capacity equals to 5.

In order to reduce energy consumption and achieve application performance targets, the Resource Manager combines virtualization technologies and control-theoretic techniques. On the one hand, by deploying each application tier inside a separate VM, virtualization provides both a runtime isolated environment and a mean for dynamically provisioning physical resources to virtualized applications so that an effective use of physical resources can be achieved. On the other hand, control theory provides a way for enabling computing systems to automatically manage performance and power consumption, without human intervention. Thus, the Resource Manager accomplishes its goal by dynamically adjusting the fraction of the capacity of physical resources assigned to each VM (hosting a particular application tier), and, if needed, by *migrating* one or more VMs into other and more appropriated physical machines (possibly, by turning on or off some of them). As shown in Fig. 6.1, the Resource Manager consists in a set of independent components that we called *Application Manager*, *Physical Machine Manager*, and *Migration Manager*.

In the subsequent sections, we present the design of each of these component. It is worth noting that, although out framework is general enough to deal with any type of physical resource and performance metric, for the sake of simplicity, in this thesis we restrict our focus to the CPU as the type of physical resource, and on the

application-level response time, as SLO performance metric.

6.2 Application Manager

The purpose of the Application Manager is to provide the controlled application with the needed amount of resource capacity in order to satisfy its SLO constraints. There is one Application Manager component for which hosted application, and each application is controlled by a single Application Manager. Moreover, each Application Manager component works independently from each other.

The Application Manager accomplishes its task by periodically performing the following actions: (1) it monitors the interested performance metrics, (2) it compares them with the related counterparts defined by the SLOs (associated to the controlled application), (3) it computes the amount of resource capacity each tier should obtain to meet its SLO constraints, and (4) it forwards these resource capacity demands to Physical Machine Managers associated to physical machines where each tier of the controlled application is running. Moreover, in order to cope with physical machine heterogeneity, at the beginning and the end of its periodic activity it converts actual resource demands (related to physical machines where tiers are currently running) to/from "reference" ones (stated in terms of the reference machine), respectively.

We designed the Application Manager by means of control-theoretic techniques in order to automatically computes optimal CPU shares for each tier of the controlled application, under time-varying workloads. Specifically we relied on:

- Feedback control theory (see Section 5.2.1), to achieve a sort of a self-managing behavior by means of mechanisms (provided by this theory) for creating a kind of "control loop" which continuously monitors the behavior of the controlled system through "sensors", analyzes collected measurements, plans the next action according to a specific "reference value", and then executes that action through "actuators".

- Adaptive control theory with the STR adaptation scheme (see Section 5.2.2), in order to adapt to time-varying workloads and operating conditions.

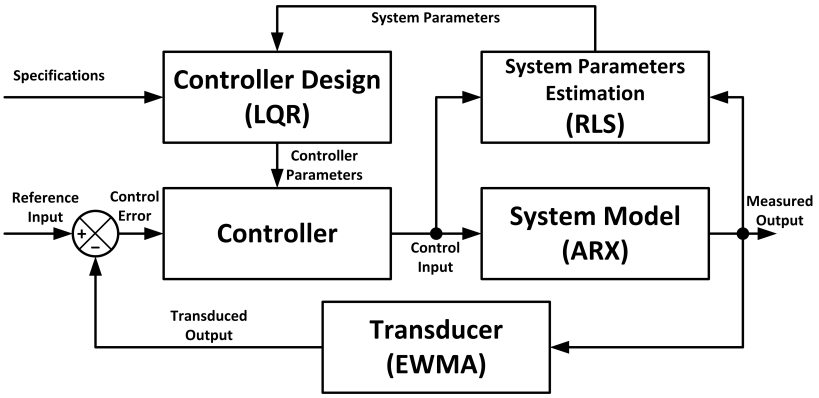


Figure 6.2: Internal structure of the Application Manager.

- Optimal control theory (see Section 5.2.2), to compute the optimal control sequence which minimizes the control error.

As shown in Fig. 6.2, we used the STR adaptation scheme, whereby system parameters are estimated on-line and controller parameters are obtained from the solution of a control design problem using such estimated parameters as if they were correct (according to the certainty equivalence principle).

In the rest of this section, we present some detail about the design of each of its component. For the sake of clarity, we provide in Table 6.1 a summary of the main mathematical symbols used along this section. Furthermore, to keep notation simple, we fixed the number of tiers to be the same for every application and denoted it as m.

For what regards the design of the "target system" box (which in our case represents the behavior of the application controlled by the Application Manager), we modeled it by means of a black-box model, whereby only inputs and outputs of the target system need to be known and the system behavior is inferred by experimentally measuring input-output relationships. This choice is motivated by two important reasons. First, unlike many traditional physical systems, the dynamics of computing systems can seldom be described by first-principle models (which use known physical laws to describe the behavior of the target system), thus making classical control theory not applicable [84]. Moreover, relying on just input-output relationships, free us to make any assumption on the internal structure of the modeled system, thus making our framework general enough to be

Table 6.1: Application Manager – Mathematical Notation.

Symbol	Meaning
m	The number of tiers of a multi-tier application
$s_i(k)$	Mean CPU share of VM hosting tier i, at control interval k
$p_i(k)$	Mean residence time of tier i, at control interval k
$\bar{s}_i(k)$	Operating value for mean CPU share for tier i, at control interval k
$\bar{p}_i(k)$	Operating value for mean residence time of tier i, at control interval k
$\Delta\tilde{s}_i(k)$	Normalized deviation for mean CPU share of tier i, at control interval k
$\Delta\tilde{p}_i(k)$	Normalized deviation for mean residence time of tier i, at control interval k
$\Delta\tilde{\mathbf{s}}(k)$	The $m \times 1$ column vector of mean CPU shares, at control interval k
$\Delta\tilde{\mathbf{p}}(k)$	The $m \times 1$ column vector of mean residence times, at control interval k

adapted to any type of application. In our case, we modeled such relationship by a discrete-time MIMO linear system model (see Chapter 3), where in our case, for a given sampling interval k, system inputs $s_i(k)$ are the fractions of CPU capacity (or, simply, the CPU shares) assigned to VMs running each application tier i and scaled with respect to the reference machine, while system outputs $p_i(k)$ are the mean residence times observed from each application tier i (i.e., the average residence time of requests departed from tier i during the sampling interval k), for $i = 1, \ldots, m$. It is worth noting that, such input-output relationships is usually best described by nonlinear models, mainly due to the complexity of computing systems and to saturation phenomena (e.g., caused by the constrained nature of CPU shares which are bounded in the $[0, 1]$ real interval) [84]. However, nonlinear models generally make complex the system design. A standard way to cope with this situation is the use of linearization. Specifically, we employed local linearization around an equilibrium point (see Section 3.2), whereby the original input-output relationship is restated in terms of perturbations with respect to such nominal point. This operating point is dynamically recomputed at each control interval, by taking the average of measures (representing both response times, residence times, and CPU shares) observed in the last n_e control intervals. In addition to the linearization technique, we applied normalization with respect to the same operating point to

make sure that inputs and outputs were of the same order of magnitude and thus to hopefully reduce numerical instability issues (as described in Section 4.1.2 and suggested in [96]). After these transformations, for each tier $i = 1, \ldots, m$, original inputs $s_i(k)$ and outputs $p_i(k)$ were replaced by normalized deviations $\Delta\tilde{s}_i(k)$ and $\Delta\tilde{p}_i(k)$ from their equilibrium point $(\bar{s}_i(k), \bar{p}_i(k))$, respectively, such that:

$$\begin{aligned} \Delta\tilde{s}_i(k) &= \frac{s_i(k)}{\bar{s}_i(k)} - 1, \\ \Delta\tilde{p}_i(k) &= \frac{p_i(k)}{\bar{p}_i(k)} - 1 \end{aligned} \tag{6.1}$$

To model this new input-output relationship we employed the following ARX model (see Eq. (4.4) in Section 4.1.3), shown in matrix form:

$$\Delta\tilde{\mathbf{p}}(k) + \sum_{j=1}^{n_a} \mathbf{A}_j(k)\Delta\tilde{\mathbf{p}}(k-j) = \sum_{j=1}^{n_b} \mathbf{B}_j(k)\Delta\tilde{\mathbf{s}}(k-j-n_k) \tag{6.2}$$

where:

- $\Delta\tilde{\mathbf{p}}(k) = \left(\Delta\tilde{p}_1(k) \ \ldots \ \Delta\tilde{p}_m(k)\right)^T$ is the vector of system outputs at control interval k,
- $\Delta\tilde{\mathbf{s}}(k) = \left(\Delta\tilde{s}_1(k) \ \ldots \ \Delta\tilde{s}_m(k)\right)^T$ is the vector of system inputs at control interval k,
- $\mathbf{A}_1(k), \ldots, \mathbf{A}_{n_a}(k)$ and $\mathbf{B}_1(k), \ldots, \mathbf{B}_{n_b}(k)$ are the matrices of system parameters with dimension $\mathbb{R}^{m\times m}$ and $\mathbb{R}^{m\times m}$, respectively, and
- n_a, n_b and n_k are the ARX model structure parameters: the number of poles, the number of zeros plus one, and the input delay (i.e., the number of input observations that occur before the input affects the output), respectively. [1]

System parameters $\mathbf{A}_1(k), \ldots, \mathbf{A}_{n_a}(k)$ and $\mathbf{B}_1(k), \ldots, \mathbf{B}_{n_b}(k)$ of the ARX model can be estimated either offline or online (see Section 4.1.4). In general, models estimated by offline system identification are unable to follow the dynamics that

[1] To simplify the notation, we assumed that each input variable has the same number of zeros and the same delay; however, in general, this is not true and thus n_b and n_k, instead of scalar values, should be represented as vectors whose dimension is equal to the number of system inputs.

may occur in a computing system (e.g., see [97, 128]). For such reason, we chose the online approach in order to adapt them to dynamic workload changes, and, in particular, we employed the RLS algorithm (see Section 4.1.4). With respect to Fig. 6.2, the online identification algorithm represents the design of the "estimation" box.

For what concerns the "transducer" box of Fig. 6.2, we designed it as an *Exponentially Weighted Moving Average* (EWMA) filter [143] for the following reasons:

- to take into account the recent past behavior of the system,
- to obtain smoothed increments of the system output in case of short peaks in the observed system output (and hence to prevent the controller to be too reactive), and
- to obtain smoothed decrements of the system output during idle or low-intensity control periods (i.e., when no or too few requests leaving the application are observed).

The peculiarity of this filter is that, by means of its *smoothing factor* parameter $\alpha \in [0,1]$, the influence of past observations decays exponentially with time, according to the following law:

$$p_i(\tau) = \alpha \hat{p}_i(\tau) + (1-\alpha) p_i(\tau), \quad \tau \in (k-1,k] \tag{6.3}$$

where k is the current control interval, and $\hat{p}_i(\tau)$ is the observed system output of tier i at time τ.

For what concerns the "controller design" box, we chose, among the many alternatives available in the literature, the discrete-time LQ control design (see Section 5.4.2). We used this type of LQ controller since our ultimate goal is to follow the trajectory described by the application-level response time (which, in our case, is the sum of the residence time of each tier). As a matter of fact, the physical meaning of the output variable $y(k)$ (which in our case is a scalar) is that of representing the normalized deviation from the expected mean response time, at control interval k. In order to use such type of controller, we needed to convert our

original system model of Eq. (6.2) into an equivalent state-space representation. To do so, we used the following (non-minimal) state-space realization (see Section 3.4):

$$\mathbf{x}(k) = \begin{pmatrix} \Delta\tilde{\mathbf{p}}(k-n_a+1) \\ \vdots \\ \Delta\tilde{\mathbf{p}}(k) \end{pmatrix}, \tag{6.4a}$$

$$\mathbf{u}(k) = \begin{pmatrix} \Delta\tilde{\mathbf{s}}(k-n_b-n_k+1) \\ \vdots \\ \Delta\tilde{\mathbf{s}}(k-n_k) \end{pmatrix}, \tag{6.4b}$$

$$\mathbf{A} = \begin{pmatrix} \mathbf{Z} & \mathbf{I} & \mathbf{Z} & \dots & \mathbf{Z} \\ \mathbf{Z} & \mathbf{Z} & \mathbf{I} & \dots & \mathbf{Z} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathbf{Z} & \mathbf{Z} & \mathbf{Z} & \dots & \mathbf{I} \\ -\mathbf{A}_{n_a}(k) & -\mathbf{A}_{n_a-1}(k) & -\mathbf{A}_{n_a-2}(k) & \dots & -\mathbf{A}_1(k) \end{pmatrix}, \tag{6.4c}$$

$$\mathbf{B} = \begin{pmatrix} \mathbf{Z} & \dots & \mathbf{Z} \\ \vdots & & \vdots \\ \mathbf{Z} & \dots & \mathbf{Z} \\ \mathbf{B}_{n_b}(k) & \dots & \mathbf{B}_1(k) \end{pmatrix}, \tag{6.4d}$$

$$\mathbf{C} = \begin{pmatrix} \mathbf{0}^T & \dots & \mathbf{0}^T & \mathbf{1}^T \end{pmatrix}, \tag{6.4e}$$

$$\mathbf{D} = \mathbf{0}^T \tag{6.4f}$$

It is important to note that, in the above equations, the value of the matrix $\mathbf{C}$ implies that the normalized deviation from the mean application-level response time is computed as the sum of normalized deviations from the mean tier-level residence times, that is:

$$\begin{aligned} y(k) &= \mathbf{C}\mathbf{x}(k) + \mathbf{D}\mathbf{u}(k) \\ &= \sum_{i=1}^{m} \Delta\tilde{p}_i(k) \end{aligned} \tag{6.5}$$

This assumption is valid since the mean is a linear function, for which the superposition principle holds.

Once the vector $\mathbf{u}^*(k)$ of optimal normalized CPU share deviations is computed by the LQ controller (according to Eq. (5.8)), the optimal CPU shares $s_i^*(k)$, for $i = 1,\ldots,m$, are obtained by applying back the transformation shown in Eq. (6.1), such that:

$$s_i^*(k) = \bar{s}_i(k)(1+\Delta\tilde{s}_i^*(k)) \tag{6.6}$$

Finally, each of such shares (which is stated in terms of the reference machine) is rescaled back with respect to the CPU capacity of the physical machine where the associated tier is running, and then is forwarded to the Physical Machine Manager controlling that machine.

6.3 Physical Machine Manager

The purpose of the Physical Machine Manager is to satisfy CPU share demands coming from those Application Managers which have tiers running on the controlled physical machine. There is one Physical Machine Manager for each physical machine of the cloud infrastructure.

The reason for the need of such a component is that, since (1) the same physical machine may hosts VMs running tiers belonging to different applications, and (2) Application Managers work independently from each others, CPU share demands arriving at the Physical Machine Manager are generally uncorrelated, so that the aggregated CPU share demand may exceed the maximum CPU capacity of the controlled physical machine.

Thus, the Physical Machine Manager has to arbitrate among all incoming CPU share demands by adjusting them according to a given *policy*. In our current implementation, we designed the Physical Machine Manager according to a *proportional policy*, whereby adjusted CPU share demands are computed proportionally to the original ones. Specifically, assuming that a particular physical machine hosts n VMs, CPU shares are bounded in the $(0,D]$ real interval (with $0 < D \leq 1$), and denoting with $d_1,\ldots,d_n$ the incoming CPU share demands for the n VMs, the adjusted CPU share demands $\hat{d}_1,\ldots,\hat{d}_n$ will be computed according to the following formula:

$$\hat{d}_i = \frac{d_i}{\sum_{j=1}^{n} d_j} D \tag{6.7}$$

6.4 Migration Manager

The purpose of the Migration Manager is to find an optimal allocation of currently running VMs on a group of physical machines (where the CPU is the only shared physical resource), in order to preserve the SLO of hosted applications (as much as possible) and, at the same time, to reduce the energy consumption of the cloud infrastructure.

This objective is achieved with the combination of two tasks:

- the monitoring of application performance targets and of the energy consumption of the cloud infrastructure, and
- the computation, at middle- or long-time scale, of a new optimal VMs allocation in order to achieve applications SLOs and reduce energy consumption.

Once the optimal allocation is computed, the following (non-mutually exclusive) actions can be triggered:

- One or more physical machines need to be powered on since the actual capacity of the cloud infrastructure is unable to satisfy the aggregated resource demand of current VMs.
- One or more VMs are migrated to more suitable physical machines.
- One or more physical machines can be powered off since the actual capacity of the cloud infrastructure exceeds the aggregated resource demand of current VMs.

The optimal allocation can be computed by means of *mathematical optimization*. Thus, in the rest of this section, we focus on the formulation of the optimization problem.

6.4.1 Optimization Problem

Before presenting the mathematical formulation of the optimization problem, we introduce some notation.

To keep the notation simple, we assume that every application has associated the same reference machine, and we denote with $\bar{C}$ the capacity of its CPU. We use the symbol M to denote the set of all physical machines in the cloud infrastructure (including the one that are currently powered off), the symbol V to denote the set of all powered on VMs hosted by the cloud infrastructure, and the symbol T to denote the length of each control interval (i.e., the time elapsed between two consecutive decisions taken by the Migration Manager). Furthermore, whenever we use the proposition "control interval k" we refer to the kth execution of the Migration Manager, with $k \in \mathbb{N}^+$; thus, if T denotes the length of each kth control interval, the time at which the Migration Manager executes is given by kT.

For the sake of clarity, we provide in Table 6.2 a summary of all the mathematical symbols used along this section (which complements the table of mathematical notation presented at the beginning of this thesis).

Objective Function

As a first step in the formulation of the optimization problem, we need to identify the decision variables and to define the objective function $J(\cdot)$. The decision variables for the optimization problem are the following:

- The binary decision variables $x_i(k)$, which indicate if the physical machine i is selected to host one or more VMs (at control interval k); that is:

$$x_i(k) = \begin{cases} 1, & \text{physical machine } i \text{ hosts at least one VM,} \\ 0, & \text{otherwise} \end{cases}, \quad i \in M \quad (6.8)$$

- The binary decision variables $y_{ij}(k)$, which indicate if the physical machine i is selected to host the VM j (at control interval k); that is:

$$y_{ij}(k) = \begin{cases} 1, & \text{physical machine } i \text{ hosts the VM } j, \\ 0, & \text{otherwise} \end{cases}, \quad i \in M, j \in V \quad (6.9)$$

- The decision variables $s_{ij}(k) \in [0,1]$, which indicate the CPU share (i.e., the fraction of CPU capacity) of physical machine i assigned to VM j (at control

Table 6.2: Migration Manager – Mathematical Notation.

Symbol	Meaning
$\bar{C}$	Capacity of the reference machine
C_i	Capacity of physical machine i
g_e	Normalizing constant for the power consumption cost $J_e(\cdot)$
g_m	Normalizing constant for the migration cost $J_m(\cdot)$
g_p	Normalizing constant for the performance cost $J_p(\cdot)$
$J(k)$	The objective function to be minimized, at control interval k
$J_e(k)$	The power consumption cost in the objective function $J(k)$, at control interval k
$J_m(k)$	The VM migrations cost in the objective function $J(k)$, at control interval k
$J_p(k)$	The performance cost in the objective function $J(k)$, at control interval k
M	The set of all physical machines
$s_i^{\max}$	Maximum aggregated CPU share demand for physical machine i
$s_{ij}(k)$	Decision variable representing the CPU share assigned to VM j on physical machine i, at control interval k
$\tilde{s}_{ij\tau}(k)$	Observed CPU share demand of VM j on physical machine i at time τ, in control interval k
$\hat{s}_{ij}(k)$	Predicted mean CPU share demand of VM j on physical machine i, at control interval k
$\hat{s}_j(k)$	Predicted mean CPU share demand of VM j on the reference machine, at control interval k
$\bar{s}_t$	Mean CPU share of tier t on the reference machine
$\bar{s}_t^{\min}$	Minimum CPU share assignable to tier t on the reference machine
t_j	Tier hosted by VM j
$u_i^{\max}$	CPU utilization threshold for physical machine i
$\hat{u}_i(k)$	Predicted mean CPU utilization of physical machine i, at control interval k
$\bar{u}_t$	Mean CPU utilization of tier t with respect to the reference machine
T	Length of each control interval of the Migration Manager
V	The set of all powered on VMs
$W_i(k)$	The (instantaneous) power consumption (in Watt) of physical machine i, at control interval k
w_e	Weight for the power consumption cost J_e in the objective function J
w_m	Weight for the VM migrations cost J_m in the objective function J
w_p	Weight for the performance cost J_p in the objective function J
$x_i(k)$	Binary decision variable; set to 1 if physical machine i is powered on, at control interval k
$y_{ij}(k)$	Binary decision variable; set to 1 if physical machine i hosts VM j, at control interval k
β	Smoothing factor for the EWMA filter used to compute $\hat{u}_{ij}(\cdot)$
γ	Smoothing factor for the EWMA filter used to compute $\hat{s}_{ij}(\cdot)$
π_{jih}	Cost for migrating VM j from physical machine i to the physical machine h
$\tilde{v}_{ij\tau}(k)$	Observed contribution to the mean CPU utilization on physical machine i by VM j at time τ, in control interval k
$\hat{v}_j(k)$	Predicted contribution to the mean CPU utilization on the reference machine by VM j, at control interval k
$\hat{v}_{ij}(k)$	Predicted contribution to the mean CPU utilization on physical machine i by VM j, at control interval k
$\omega_{i0}, \omega_{i1}, \omega_{i2}, \rho_i$	Power model parameters for physical machine i

interval k).

The objective function $J(k)$ of the optimization problem, at control interval k, can be stated as a cost function that need to be minimized, which, in turn, can be expressed as the weighted sum of three separate costs, namely: the *energy consumption cost* $J_e(\cdot)$, the *VM migrations cost* $J_m(\cdot)$, and the *performance cost* $J_p(\cdot)$; that is:

$$J(k) = w_e J_e(k) + w_m J_m(k) + w_p J_p(k) \tag{6.10}$$

where w_e, w_m, and w_p are nonnegative real values (provided as configuration parameters) that can be used to assign a different weight to each cost $J_e(\cdot)$, $J_m(\cdot)$, and $J_p(\cdot)$, respectively.

The cost $J_e(k)$ represents the cost due to the energy consumption induced by whole cloud infrastructure (at control interval k), which is computed as the sum of the energy consumptions due to each powered on physical machine. To provide an analytical formulation of such cost, we introduce the following quantities:

- The contribution $\tilde{\upsilon}_{ij\tau}(k)$ to the CPU utilization of the physical machine i brought by VM j as observed at time $\tau \in \big((k-1)T, kT\big]$.

- The expected contribution $\hat{\upsilon}_j(k)$ to the mean CPU utilization of a physical machine equivalent to the reference machine, that will be brought by VM j (at control interval k). In order to estimate such quantity, we apply an EWMA filter to each observed contribution $\tilde{\upsilon}_{ij\tau}(k)$, at time $\tau \in \big((k-1)T, kT\big]$ (where i is the physical machine where VM j runs at time τ). This filter provides a way to take into account the past behavior of a VM, and to obtain smoothed predicted values in case of sudden and short changes in the working conditions (e.g., short bursts). Since, in general, the resource characteristics of the reference machine may be different from the ones of the physical machines of the cloud infrastructure, we have to formulate each $\hat{\upsilon}_j(k)$ in a way that is independent of the physical machine where VM j actually runs. To do so, we express such quantity in terms of the characteristics of the reference machine, by scaling each observed $\tilde{\upsilon}_{ij\tau}(k)$ with respect to the capacity C_i of the physical machine i and the capacity $\bar{C}$ of the reference machine; that is,

for $\tau \in \big((k-1)T, kT\big]$:

$$\hat{v}_j(k) = \beta\, \tilde{v}_{ij\tau}(k)\frac{C_i}{\bar{C}} + (1-\beta)\hat{v}_j(k-1), \quad i \in M, j \in V, k \in \mathbb{N}^+ \tag{6.11}$$

where $\beta \in [0,1]$ is the smoothing factor of the EWMA filter (that must be provided as a configuration parameter), $\frac{C_i}{\bar{C}}$ is the scaling factor (which assumes a proportional relationship between both the CPU capacity and utilization of two different physical machines), and $\hat{v}_j(k-1)$ is the mean contribution to the utilization, in terms of the reference machine, brought by VM j estimated at the previous control interval $k-1$. The initial value $\hat{v}_j(0)$ is computed from the reference utilization $\bar{u}_{t_j}$ (i.e., the mean utilization that the tier t_j – running inside the VM j – would generate on the reference machine); that is:

$$\hat{v}_j(0) = \bar{u}_{t_j}, \quad i \in M, j \in V \tag{6.12}$$

- The expected contribution $\hat{v}_{ij}(k)$ to the mean CPU utilization of physical machine i, that will be brought by VM j (at control interval k). This quantity is computed from $\hat{v}_j(k)$, by performing a scaling operation similar to the one described for $\hat{v}_j(k)$; that is:

$$\hat{v}_{ij}(k) = \hat{v}_j(k)\frac{\bar{C}}{C_i} \tag{6.13}$$

- The expected mean CPU utilization $\hat{u}_i(k)$ of physical machine i (at control interval k). This quantity is computed as the sum of the expected contributions $\hat{v}_{ij}(k)$ to the mean CPU utilization that will be brought by each powered on VM j hosted by that machine (at control interval k); that is:

$$\hat{u}_i(k) = \sum_{j \in V} \hat{v}_{ij}(k) y_{ij}(k), \quad i \in M \tag{6.14}$$

where the decision variables $y_{ij}(k)$ are used to select only those VMs hosted by physical machine i (at control interval k).

- The (instantaneous) power $W_i(\cdot)$ (in Watt) consumed by a physical machine i.

To quantify this information, a power model for each physical machine needs to be provided. Among the various power models available in literature (e.g., [142]), we adopted the one described in [62], whereby the power $W_i(\cdot)$ is related to the CPU utilization $\hat{u}_i(\cdot)$ by the following nonlinear model:

$$W_i(k) = \omega_{i0} + \omega_{i1}\hat{u}_i(k) + \omega_{i2}\hat{u}_i(k)^{\rho_i}, \quad \omega_{i0}, \omega_{i1}, \omega_{i2}, \rho_i \in \mathbb{R} \tag{6.15}$$

where ω_{i0}, ω_{i1}, ω_{i2}, and ρ_i are model parameters that vary according to the characteristics of the physical machine i.

Thus, the energy consumption cost $J_e(\cdot)$ is given by the cost (in Joule) due to the energy consumed by all of the selected physical machines at control interval k; that is:

$$J_e(k) = T \sum_{i \in M} W_i(k) x_i(k) \tag{6.16}$$

where the decision variables $x_i(k)$ are used to select the powered on physical machines (at control interval k), and T is the length of control interval k.

The other cost in the objective function $J(k)$ that needs to be presented is $J_m(k)$, which represents the cost due to VM migrations, that is the cost for moving one or more VMs to different physical machines (at control interval k). We introduce the variable π_{jhi}, that represents the cost for migrating VM j from physical machine h to physical machine i. To quantify this cost, we suppose that the impact of VM migration on the networking layer is negligible; this assumption can be considered realistic if the following conditions hold:

1. the cloud infrastructure is confined to a single data center,

2. the VM migration is performed by means of *live migration* [50], and

3. VM images are stored on storage system shared by all physical machines of the cloud infrastructure.

According to this assumption, we can quantify π_{jhi} by simply assigning a unit cost to the migration of VM j from physical machine h to physical machine i, with

$h \neq i$; that is:

$$\pi_{jhi} = \begin{cases} 1, & h \neq i, \\ 0, & h = i \end{cases}, \quad h,i \in M, j \in V \tag{6.17}$$

Thus, the VM migrations cost $J_m(k)$ is given by the cost due to all VM migrations involved during control interval k; that is:

$$J_m(k) = T \sum_{i \in M} \sum_{h \in M} \sum_{j \in V} \pi_{jhi} y_{hj}(k-1) y_{ij}(k) \tag{6.18}$$

where $y_{hj}(k-1)$ denotes the value of the decision variable $y_{hj}(\cdot)$ at the previous control interval $k-1$, the decision variables $y_{ij}(k)$ are used to select only those VMs hosted by physical machine i (at control interval k), and T is the length of control interval k.

The last cost in the objective function $J(k)$ that is to be presented is $J_p(k)$, which represents the cost due to the performance preservation of hosted applications. To quantify the contribution to the performance of a given application brought by each associated VM (i.e., by those VMs that run the application's tiers), we need to introduce the following quantities:

- The CPU share $\tilde{s}_{ij\tau}(k)$ requested by VM j to physical machine i, as observed at time $\tau \in \big((k-1)T, kT\big]$.

- The expected mean CPU share $\hat{s}_j(k)$ that is assumed VM j will demand to a physical machine equivalent to the reference machine (at control interval k). In order to estimate such quantity, we apply an EWMA filter to each observed CPU share demand $\tilde{s}_{ij\tau}(k)$, at time $\tau \in \big((k-1)T, kT\big]$. Such filter allows to take into account the past behavior of a VM, and to obtain smoothed estimates in case of abrupt and short changes in the working conditions (e.g., short bursts). Since, in general, the resource characteristics of the reference machine may be different from the ones of the physical machines of the cloud infrastructure, we have to express each $\hat{s}_j(k)$ in a way that is independent of the physical machine where VM j actually runs. To do so, we formulate such quantity in terms of the characteristics of the reference machine, by scaling each observed $\tilde{s}_{ij\tau}(k)$ with respect to the capacity C_i of the physical

machine i and the capacity $\bar{C}$ of the reference machine; that is, for each time $\tau \in \big((k-1)T, kT\big]$:

$$\hat{s}_j(k) = \gamma \tilde{s}_{ij\tau}(k)\frac{C_i}{\bar{C}} + (1-\gamma)\hat{s}_j(k-1), \quad i \in M, j \in V, k \in \mathbb{N}^+ \tag{6.19}$$

where $\gamma \in [0,1]$ is the smoothing factor of the EWMA filter (that must be provided as a configuration parameter), $\frac{C_i}{\bar{C}}$ is the scaling factor (which assumes a proportional relationship for both the CPU capacity and CPU share of two different physical machines), and $\hat{s}_j(k-1)$ is the mean CPU share demand of VM j, in terms of the reference machine, estimated at the previous control interval $k-1$. The initial value $\hat{s}_j(0)$ is computed from the reference CPU share demand $\bar{s}_{t_j}$ (i.e., the CPU share that the tier t_j – running inside the VM j – would require on the reference machine); that is:

$$\hat{s}_j(0) = \bar{s}_{t_j}, j \in V \tag{6.20}$$

- The expected mean CPU share $\hat{s}_{ij}(k)$ that is assumed that VM j will demand to the physical machine i (at control interval k). This quantity is computed from $\hat{s}_j(k)$, by performing a scaling operation similar to the one described for $\hat{s}_j(k)$; that is:

$$\hat{s}_{ij}(k) = \hat{s}_j(k)\frac{\bar{C}}{C_i} \tag{6.21}$$

- The performance of an application, in principle, is determined by the performance metric of interest (which, in our case, is the application-level response time), and thus it can be assessed by comparing the observed performance metric (i.e., the one measured from the system) with the one defined by the SLO. Since the solution of the optimization problem is an optimal allocation defined in terms of CPU shares, we need a way to relate such metric with the CPU shares $s_{ij}(\cdot)$ computed by the optimization problem. Unfortunately, to do so, one should assume a specific performance model for the application (e.g., a queueing model), thus limiting the applicability of our resource manager to the range of applications whose dynamics can be described by this model. Since we want our framework to be as more general as possible,

we avoid to make this assumption and replace the use of such metric with a "performance indicator" computed as the sum of the distances between the CPU shares received and the ones expected by each tier. The rationale here is to limit, as much as possible, cases of either over- or under-provisioning of the CPU to each VM, because, on the one hand, CPU under-provisioning can reduce the energy consumption to the expense of a higher risk to incur in SLO violations, while, on the other hand, CPU over-provisioning can lower the chance to incur in SLO violations to the expense of a greater energy consumption. The use of a distance metric let us to summarize this idea with a unique measure, since the larger is the distance of the obtained CPU share from the one that is expected, the higher is the performance cost. We compute such quantity as the sum of the contributions to the performance cost brought by each application's tier. Specifically, we define the contribution of tier t_j to the performance cost of its application as the squared difference between the CPU share $s_{ij}(k)$ that the associated VM j would obtain if it is hosted by physical machine i, and the CPU share $\hat{s}_{ij}(k)$ that VM j is expected to demand to physical machine i (at control interval k); that is:

$$\Big(s_{ij}(k) - \hat{s}_{ij}(k)\Big)^2 \tag{6.22}$$

Therefore, if an application has n tiers each of which running into a separate VM j_h hosted by a physical machine i_h, for $h = 1, 2, \ldots, n$, we estimate the performance cost of this application (at control interval k) as:

$$\sum_{h=1}^{n} \Big(s_{i_h j_h}(k) - \hat{s}_{i_h j_h}(k)\Big)^2 \tag{6.23}$$

Thus, the performance cost $J_p(k)$ is given by the cost due to all performance costs of applications running at control interval k; that is:

$$J_p(k) = T \sum_{i \in M} \sum_{j \in V} y_{ij}(k) \Big(s_{ij}(k) - \hat{s}_{ij}(k)\Big)^2 \tag{6.24}$$

where the decision variables $y_{ij}(k)$ are used to select only those VMs hosted by physical machine i (at control interval k), and T is the length of control interval k.

The objective function $J(\cdot)$ cannot still be used in the optimization problem since the three costs $J_e(\cdot)$, $J_m(\cdot)$, and $J_p(\cdot)$ can have a different order of magnitude and hence need to be normalized. To this end, we define the following quantities:

- The normalizing constant g_e for the cost $J_e(\cdot)$, which is computed by taking the product between the number of physical machines $|M|$ and the maximum power consumption at full CPU capacity among the physical machines inside the cloud infrastructure; that is:

$$g_e = |M| \max_{i \in M} \left\{ \omega_{0i} + \omega_{1i} + \omega_{2i} \right\} \tag{6.25}$$

- The normalizing constant g_m for the cost $J_m(\cdot)$, which is computed by taking the product between the number of VMs $|V|$ and maximum VM migration cost; that is:

$$\begin{aligned} g_m &= |V| \max_{i \in M} \max_{h \in M} \max_{j \in V} \left\{ \pi_{jih} \right\} \\ &= |V| \end{aligned} \tag{6.26}$$

  where the last equality derive from how π_{jih} has been defined (see Eq. (6.17)).

- The normalizing constant g_p for the cost $J_p(\cdot)$, which is computed by taking the product between the number of VMs $|V|$ and the maximum distance that can result between the obtained and expected CPU share. Since both $s_{ij}(k)$ and $\hat{s}_{ij}$ (i.e., the terms involved in the computation of the distance) are bounded in the $[0,1]$ interval, we have:

$$g_p = |V| \tag{6.27}$$

Thus, the resulting objective function $J(k)$ (at control interval k) is given by:

$$\begin{aligned} J(k) &= \frac{w_e}{g_e} J_e(k) + \frac{w_m}{g_m} J_m(k) + \frac{w_p}{g_p} J_p(k) \\ &= T \left[\frac{w_e}{g_e} \sum_{i \in M} W_i(k) x_i(k) \right. \\ &\quad + \frac{w_m}{g_m} \sum_{i \in M} \sum_{h \in M} \sum_{j \in V} \pi_{jhi} y_{hj}(k-1) y_{ij}(k) \\ &\quad \left. + \frac{w_p}{g_p} \sum_{i \in M} \sum_{j \in V} y_{ij}(k) \left(s_{ij}(k) - \hat{s}_{ij}(k) \right)^2 \right] \\ &= \frac{w_e}{g_e} \sum_{i \in M} W_i(k) x_i(k) \\ &\quad + \frac{w_m}{g_m} \sum_{i \in M} \sum_{h \in M} \sum_{j \in V} \pi_{jhi} y_{hj}(k-1) y_{ij}(k) \\ &\quad + \frac{w_p}{g_p} \sum_{i \in M} \sum_{j \in V} y_{ij}(k) \left(s_{ij}(k) - \hat{s}_{ij}(k) \right)^2 \end{aligned} \tag{6.28}$$

where, in the last equality, the time-horizon T has been removed since it is a common constant term.

For what concerns the *energy consumption weight* w_e, *migration weight* w_m, and *performance weight* w_p, they are positive real numbers representing user-configurable design parameters that can be used to assign different importance to each cost of the objective function. According to the value assigned to these weights, the user can design Migration Managers that can behave differently from each other. For instance, in order to design an energy-conserving Migration Manager (i.e., a Migration Manager which favors reduction of energy consumption over SLO achievement), the user can choose a value for w_e which is much greater than the other two. A typical choice is to use the same value for all the weights (e.g. $w_e = w_m = w_p = 1$), thus resulting in a Migration Manager that will try to find a good trade-off between all the three costs.

Constraints

The optimization problem is characterized by a number of constraints. In the rest of this section, we refer to a "selected physical machine" as a physical machine i

such that $x_i(k) = 1$ (at control interval k).

In the following we present each of the problem constraints: [2]

(C01) Decision variables $x_i(\cdot)$ are binary (integer) variables:

$$x_i(k) \in \{0,1\}, \quad i \in M \tag{6.29}$$

(C02) Decision variables $y_{ij}(k)$ are binary (integer) variables:

$$y_{ij}(k) \in \{0,1\}, \quad i \in M, j \in V \tag{6.30}$$

(C03) Decision variables $s_{ij}(\cdot)$ are real variables bounded in $[0,1]$:

$$s_{ij}(k) \in \left[0,1\right], \quad i \in M, j \in V \tag{6.31}$$

(C04) In the optimal allocation , each VM must be hosted by one and only one physical machine:

$$\sum_{i \in M} y_{ij}(k) = 1, \quad j \in V \tag{6.32}$$

(C05) In the optimal allocation, each VM must be hosted only by a selected physical machine:

$$y_{ij}(k) \leq x_i(k), \quad i \in M, j \in V \tag{6.33}$$

The purpose of this constraint is to avoid that a VM is allocated to a physical machine that will be powered off.

(C06) In the optimal allocation, each selected physical machine must host at least one VM:

$$x_i(k) \leq \sum_{j \in V} y_{ij}(k), \quad i \in M \tag{6.34}$$

The purpose of this constraint is to avoid to keep powered on a physical machine that does not host any VM.

[2] For the sake of readability, we omit to specify the validity range of control interval k, which is $k \in \mathbb{N}$.

(C07) In the optimal allocation, a VM can obtain a CPU share greater than 0 only on the physical machine that hosts it:

$$s_{ij}(k) \leq y_{ij}(k), \quad i \in M, j \in V \tag{6.35}$$

The purpose of this constraint is to avoid allocation inconsistencies in the optimal solution.

(C08) In the optimal allocation, the CPU share assigned to a VM must be no less than the minimum CPU share on the reference machine. To formulate this constraint, we need to introduce the new quantity $\bar{s}_{t_j}^{\min}$, representing the minimum CPU share that tier t_j (running inside VM j) can obtain on the reference machine. Furthermore, we have to scale the CPU share $s_{ij}(k)$, assigned to a VM j on a given physical machine i (at control interval k), in terms of the characteristics of the reference machine. Thus, the wanted constraint is given by:

$$s_{ij}(k)\frac{C_i}{\bar{C}} \geq y_{ij}(k)\bar{s}_{t_j}^{\min}, \quad i \in M, j \in V \tag{6.36}$$

where the decision variable $y_{ij}(k)$ is used to trigger the constraint only if the physical machine i has been selected to host VM j (i.e., when $y_{ij}(k) = 1$), and $\frac{C_i}{\bar{C}}$ is the scaling factor. The purpose of this constraint is to avoid to assign to a VM a too small CPU share that makes unable the VM to react to a change in the operating condition. For instance, in case of a change in the working condition from an idle to a busy period, a too small CPU share can lead to system instability for the combination of queueing phenomena and the inability of the system to satisfy already queued requests.

(C09) In the optimal allocation, the aggregated CPU share demand for a physical machine must not exceed the maximum allowable share. To formulate this constraint, we need to introduce the new quantity $s_i^{\max}$, representing the maximum CPU share that can be demanded to the physical machine i. Thus,

the wanted constraint is given by:

$$\sum_{j \in V} s_{ij}(k) \leq s_i^{\max}, \quad i \in M \tag{6.37}$$

The purpose of this constraint is to avoid that hosted VMs occupy the whole resource capacity; for instance, it is common practice to leave some fraction of CPU unshared in order to run system or administration processes.

(C10) In the optimal allocation, the utilization of a physical machine must not exceed a predefined threshold. To formulate this constraint, we need to introduce the new quantity $u_i^{\max}$, representing the CPU utilization threshold for the physical machine i. Thus, the wanted constraint is given by:

$$\hat{u}_i(k) \leq u_i^{\max}, \quad i \in M \tag{6.38}$$

The purpose of this constraint is to avoid to overload a physical machine.

The resulting mathematical program (shown on Alg. 1) is a *Mixed-Integer Nonlinear Program* (MINLP) [67, 110] which is known to be NP-hard [71, 92] (more details are provided in Appendix B). The nonlinearity arises in the objective function, both for the quadratic terms in the performance cost $J_p(\cdot)$ and the non-linear terms in the energy consumption cost $J_e(\cdot)$. Moreover, both the objective function and some of the constraint inequalities lack of the *convexity* property, thus limiting the range of applicable and efficient solution techniques [36].

For these reasons, a global optimum cannot be found in a reasonable amount of time, and thus we need to look for an approximated solution, which is the subject of the subsequent subsection.

6.4.2 Approximated Algorithms

In this thesis we consider two approximated solutions: one based on a greedy algorithm, and the other based on a local optimization technique.

Algorithm 1 Optimization Problem.

$$
\begin{aligned}
\text{minimize} \qquad J(k) = & \left[\frac{w_e}{g_e} \sum_{i \in M} W_i(k) x_i(k) \right. \\
& + \frac{w_m}{g_m} \sum_{i \in M} \sum_{h \in M} \sum_{j \in V} \pi_{jhi} y_{hj}(k-1) y_{ij}(k) \\
& \left. + \frac{w_p}{g_p} \sum_{i \in M} \sum_{j \in V} y_{ij}(k) \left(s_{ij}(k) - \hat{s}_{ij}(k) \right)^2 \right]
\end{aligned}
$$

subject to

$$
\begin{aligned}
x_i(k) &\in \{0,1\}, && i \in M, \\
y_{ij}(k) &\in \{0,1\}, && i \in M, j \in V, \\
s_{ij}(k) &\in [0,1], && i \in M, j \in V, \\
\sum_{i \in M} y_{ij}(k) &= 1, && j \in V, \\
y_{ij}(k) &\leq x_i(k), && i \in M, j \in V, \\
x_i(k) &\leq \sum_{j \in V} y_{ij}(k), && i \in M, \\
s_{ij}(k) &\leq y_{ij}(k), && i \in M, j \in V, \\
s_{ij}(k) \frac{C_i}{\bar{C}} &\geq y_{ij}(k) \bar{s}_{t_j}^{\min}, && i \in M, j \in V, \\
\sum_{j \in V} s_{ij}(k) &\leq s_i^{\max}, && i \in M, \\
\hat{u}_i(k) &\leq u_i^{\max}, && i \in M.
\end{aligned}
$$

Greedy Algorithm

A possible way to obtain an approximated solution to the optimization problem shown in Alg. 1 is by using a greedy algorithm which implements a *Best-Fit Decreasing* (BFD) strategy.

The basic idea of this strategy is to place the largest number of VMs on the fewest number of physical machines, while still preserving SLO constraints. To do this, we order the set of all physical machines by three criteria: (1) by their power status, by first taking the ones already powered on and then those still powered off, (2) for each power status, by decreasing values of the shareable CPU capacity, and (3) in case of same CPU capacity, by increasing values of idle power consumption. Then, we order the set of powered on VMs by decreasing CPU share demand. Finally, we sequentially iterate over the set of ordered VMs and place each of them on the first physical machine that appears in the ordered set of physical machines that has enough space to host it and for which the utilization threshold is not exceeded.

The pseudocode of this strategy is shown in Alg. 2. The notation used for the input and output arguments follows the one presented in Table 6.2.

A special consideration is to be given to the notation used for the sorting operation. In Alg. 2, we denote this operation with a statement like the following:

$$S' \leftarrow \textsc{Sort}\big(S, \{\textsc{Property}1 : \textsc{Criterion}1, \ldots, \textsc{Property}N : \textsc{Criterion}N\}\big)$$

In this case, we mean that the sequence S, which is the first argument to the sorting function, is ordered according to properties and criteria specified in the second argument, and the result is assigned back to the new sequence S'. Each element in the property-criterion list (i.e., the second argument to the sorting function) is a pair that associates a property (of each element of S) to a sorting criterion to apply to this property. The sorting operation is done in the order of the property-criterion list. Specifically, the sequence S is first sorted by values of the property PROPERTY1 according to CRITERION1. In case there are elements of S with the same value of PROPERTY1, the ordering is done by the value of the property PROPERTY2 according to CRITERION2, and so on. We use the special symbols $\uparrow$, and $\downarrow$ to denote the increasing and decreasing order criterion, respectively.

Algorithm 2 BESTFITDECREASING$(M, V, \mathbf{C}, \bar{C}, \mathbf{s}^{\max}, \hat{\mathbf{s}}, \mathbf{u}^{\max}, \hat{\upsilon}(k), \omega_0)$

Input: M the set of all physical machines,
V the set of powered on VMs,
$\mathbf{C}$ the vector of CPU capacities,
$\bar{C}$ the CPU capacity of the reference machine,
$\mathbf{s}^{\max}$ the vector of maximum aggregated CPU share demands,
$\hat{\mathbf{s}}(k)$ the vector of predicted CPU share demand for VMs,
$\mathbf{u}^{\max}$ the vector of maximum CPU utilizations,
$\hat{\upsilon}(k)$ the vector of predicted contributions to CPU utilization,
ω_0 the vector of idle power consumptions.

Output: $\mathbf{x}$ the vector of selected physical machines,
$\mathbf{Y}$ the matrix of VM allocations (each element y_{ij} represent the allocation of VM j on physical machine i),
$\mathbf{S}$ the matrix of CPU shares for all VMs (each element s_{ij} represent the CPU share assigned to VM j on physical machine i).

```
//Initializations
x ← 0
Y ← Z
S ← Z
r ← pmin{1, C ◇ s^max}
u ← pmin{1, u^max}

//Sort physical machines and VMs
M' ← SORT(M, {POWERSTATUS : (ON, OFF), r : ↓, ω_0 : ↑})
V' ← SORT(V, {ŝ : ↓})

//Find the "best" VMs placement
for all v ∈ V' do
  for all m ∈ M' do
    ŝ ← ŝ_v(k) C̄/C_m
    υ̂ ← υ̂_v(k) C̄/C_m
    if ŝ ≤ r_m and υ̂ ≤ u_m then
      x_m ← 1
      y_mv ← 1
      s_mv ← ŝ
      r_m ← r_m − ŝ
      u_m ← u_m − υ̂
      break
    end if
  end for
end for
return ⟨x, Y, S⟩
```

The rationale of this solution is that, since the power consumed by a physical machine is largely determined by its idle (fixed) power consumption, it is better to place the largest number of VMs first on those physical machines that exhibit the highest idle power consumption.

Assuming that the two sorting operations take a *loglinear* time in the dimension of their input, [3] the computational complexity of Alg. 2 is $\mathcal{O}(|V||M|)$.

Unfortunately this solution may suffer of some issue. Firstly, there is no guarantee to find a global optimum since the problem does not exhibit the greedy optimal substructure [4] [54]. Another possible issue of this strategy is that it cannot explore the whole space of possible CPU shares, since (1) each CPU share is a real number and hence there are potentially infinite candidates to evaluate, and (2) all VMs must be selected, so we cannot adopt a solution similar to the one used for the fractional knapsack problem [54, 116].

Local Optimization

Local optimization is a method for solving computationally hard optimization problems. A local optimization algorithm starts from a candidate solution and then iteratively moves to a neighbour solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

Typically, such algorithms only build up a local model of the problem. Furthermore, to ensure convergence of the iterative process, many such algorithms insist on a certain decrease of the *target function* (i.e., of the objective function or of a merit function that is a combination of the objective and constraints). Such algorithms will, if convergent, only find the local optimum; thus the name of local optimization algorithms. Conversely, *global optimization* algorithms attempt to find the global optimum, typically by allowing decrease as well as increase of the target function. Such algorithms are usually computationally more expensive.

In this thesis, we use the local optimization algorithm provided by the *Simple*

[3] An algorithm take a *loglinear* time if its computational complexity is $\mathcal{O}(n \log n)$, where n is the size of its input.

[4] A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.

Branch & Bound (SBB) solver [14, 37], which is a *nonlinear programming* (NLP) based branch-and-bound solver that is available through the *General Algebraic Modeling System* (GAMS) modeling language [5]. The NLP subproblems can be solved by CONOPT [58], MINOS [120, 121] and SNOPT [73]. As a node selection strategy, depth-first search, best-bound, best-estimate or combination these three can be employed.

A local optimization algorithm needs to start the (local) optimum search from a feasible (candidate) solution. Usually, when no initial solution is specified, the solver computes such candidate in a random way. Unfortunately, due to the presence of constraints in our optimization problem, a randomly generated solution can result in an unfeasible candidate. In order to overcome to this problem, we explicitly pass an initial solution, generated by using the BFD algorithm presented in Alg. 2.

In general, local optimization algorithms should perform better than greedy algorithms like the one in Alg. 2. However there are several factors that can impact on the quality of the solution (e.g., see [147]). Firstly, the starting condition may lead to a local optimum, and different starting conditions can lead to worst or better local optima. Moreover, the algorithm can get stuck into a *plateau*, which is a (sufficiently) flat region of the search space that makes the value returned by the target function indistinguishable from the value returned for nearby regions (mostly due to the precision used by the machine to represent its value). Thus, a plateau makes the solver unable to determine the direction that leads to an improvement of the target function. A possible mitigation to such issues is the use of the *random-restart* technique, whereby the algorithm is run several times with randomly generated initial conditions. Unfortunately, in our case, we cannot generate random initial conditions due to the presence of constraints.

Part III

Experimental Evaluation

Chapter 7

Experimental Settings

In order to assess the capability of the resource management techniques described in this thesis, we perform an extensive experimental evaluation using discrete-event simulation (DES). To this end, we developed a C++ ad-hoc discrete-event simulator, that is used to run our experiments.

In this chapter, we present the experimental setup of the experiments we perform for evaluating the performance of our resource management framework. Specifically, in Section 7.1, we provide an overview of the DESEC simulator, that is the simulator we use to perform the experiments. In Section 7.2, we describe settings that are common to all of our experiments. Finally, in Section 7.3, we describe how each experiment is designed.

7.1 The DESEC Simulator

In this section, we present an overview of the *Discrete-Event Simulator for Energy-aware Computing* (DESEC) project, the simulator we developed to evaluate our resource management framework.

The core of the DESEC simulator is a generic and extensible event-based engine which provides two types of simulation (with regard to output analysis), namely terminating and nonterminating simulation [28]. A *terminating* (or *transient*, or *finite-horizon*) simulation is one that runs for some duration of time or until a given terminating event occurs. The simulated system is not expected to achieve any

steady-state behavior and any parameter estimated from the output data will be transient in the sense that its value will depend upon the *initial conditions*. Conversely, a *nonterminating* (or *steady-state*) simulation is one whose objective is to study the long-run (steady-state) behavior of the system of interest. A performance measure of a system is called a steady-state parameter if it is a characteristic of the equilibrium distribution of an output stochastic process.

Currently, DESEC provides two output analysis techniques, namely batch means and independent replications. For the *independent replication* method, the duration of each replication and the total number of replications can be chosen according to different criteria. Specifically, for the duration (length) of each replication, it is possible to set it on the base of either the time duration, the number of observed output samples, or by the occurrence of a given event. For what concerns the total number of replications, DESEC offers two methods. The first method consists in to manually set it, by explicitly specifying the number of replications. The second method allows an automatic computation through a statistical analysis based on confidence intervals, such that the number of independent replicas needed to achieve, for each statistic of interest, the required confidence level is computed online by means of the *relative precision of the confidence interval* method [49].

The other output analysis technique provided by DESEC is the *batch means* method, whereby DESEC offers two ways to specify the length of the initial transient period and that of each batch. The first method consists in to manually set the duration of the initial transient and of each batch (or, alternately, in to specify the events that determine their termination). The second method is an implementation of the automatic procedure described in [130].

In Fig. 7.1, is depicted a high-level block overview of the main components composing the DESEC simulator architecture. As can be noted from the bottom of the figure, DESEC is implemented in ISO C++. Also, from the figure, we can observe that DESEC comprises of several components. In addition to the event-based engine (that we have just described), we developed other components incorporating different functionalities. Some of them are targeted at a particular purpose, like algorithms for the control design (e.g., LQR) and system identification (e.g., RLS-EF). Others are general-purpose functionalities, like container data structures, random variate generators, and logging facilities, just to name a few.

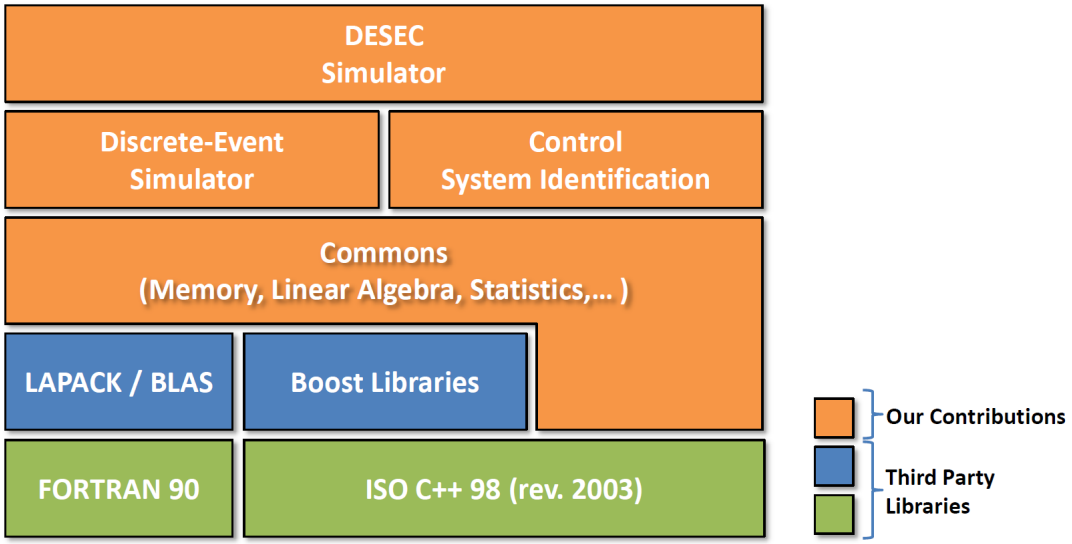


Figure 7.1: The DESEC simulator – High-level block diagram.

Besides the code developed by use, we also rely upon some third-party library. In particular, we extensively use the *Boost* project [4], a set of free peer-reviewed portable C++ source libraries, that (in part) has been selected to be included in the *C++0x* [9], that is the next-coming ISO/IEC C++ standard, also formerly known as C++11.

For matrix computations, we mostly rely on FORTRAN algorithms provided by the *Basic Linear Algebra Subprograms* (BLAS) and the *Linear Algebra PACKage* (LAPACK) [10, 21], which are software libraries for numerical linear algebra. They provides routines for solving systems of linear equations and linear least squares, eigenvalue problems, and singular value decomposition. They also includes routines to perform several matrix factorizations such as LU, QR, Cholesky and Schur decomposition.

7.2 Experimental Setup

7.2.1 Physical Infrastructure Configuration

For our experiments we consider a physical infrastructure consisting of a certain number of physical machines (the exact number depends on the experimental case under study), each one characterized by its computing power and its energy consumption. Computing power is modeled as an integer number directly proportional to the computing speed of the processor. The actual number of physical machines and the associated values of computing power chosen for our experiments are reported in chapters related to each experimental case study (see Chapter 8 and Chapter 9). The reference machine is assumed to have a computing power set to 1000.

For all the physical machines of the cloud infrastructure, the consumption of electrical energy is modeled by means of the power model described in [62, 142], whereby the power W is related to the CPU utilization u by means of the formula:

$$W = \omega_0 + \omega_1 u + \omega_2 u^{\rho} \tag{7.1}$$

We estimate the values of the parameters ω_0, ω_1, ω_2 and ρ through a statistical regression analysis over data collected by the *SPECpower_ssj2008* benchmark [15]. The actual values of the above coefficients are reported in chapters related to each experimental case study (see Chapter 8 and Chapter 9).

It is worth noting that we do not take into consideration any possible overhead introduced by the virtualization layer. As a matter of fact, we ignore possible virtualization overhead since the availability of native (hardware-assisted) virtualization solution has greatly reduced its impact on the performance of the hosted application. Furthermore, we do not model any overhead of the networking layer, since we assume a data-center based cloud infrastructure, where communication delays are generally negligible.

7.2.2 Application Configuration

We consider a set of 3 distinct multi-tier applications, that we name A_1,A_2, and A_3, respectively. All these applications have 3 tiers: each incoming service request arrives at the first tier that, after processing it, forwards it to the second tier where this process is repeated. Finally, after the processing in the third tier has taken place, the result of the service request is sent back to the respective client. The applications considered in our experiments differ from each other in the amount of processing time requested by each tier. In particular:

- A_1 is an application where all tiers have the same processing demand, that has been set to 0.06 req/sec;
- A_2 is an application with a bottleneck in the second tier; more specifically, the first and third tiers have the same service rate set to 0.03 req/sec, while the second tier has a service rate set to 0.06 req/sec;
- A_3 is an application where tiers have decreasing processing capacity, and has been obtained by setting the service rates of the first, second, and third tiers to 0.015, 0.03, and 0.06 req/sec, respectively.

All the above service rates are expressed in terms of the computing power of the reference machine.

Each application is characterized by its workload, that consists in a stream of service requests, continuously generated by a population of users of unknown size, arriving according to a specific arrival process. In order to reproduce various operating conditions that may occur for real-world hosted services, we consider three different arrival processes, namely:

- *Deterministic Modulated Poisson Process* (DMPP), to generate workloads exhibiting user behavioral patterns like daily-cycles of activity. In particular, we consider a three-state DMPP, henceforth denoted as

$$\text{DMPP}(\lambda_1, \lambda_2, \lambda_3, \tau) \tag{7.2}$$

  where λ_i, for $i = 1, \ldots, 3$, is the arrival rate of the Poisson process in state i, and τ is the deterministic state-residence time;

- *Pareto Modulated Poisson Process* (PMPP) [109] to generate self-similar workloads. In particular, we consider a two-states PMPP, henceforth denoted as

$$\text{PMPP}(\lambda_1, \lambda_2, x_\text{m}, \alpha) \tag{7.3}$$

  where λ_i, for $i = 1, 2$, is the arrival rate of the Poisson process in state i, and x_m and α are the minimum value and shape parameters of the Pareto distribution, respectively;

- *Markov Modulated Poisson Process* (MMPP) [65] to generate arrival processes exhibiting temporal burstiness [118]. In particular, we consider a two-states MMPP, henceforth denoted as

$$\text{MMPP}(\lambda_1, \lambda_2, \mu_1, \mu_2) \tag{7.4}$$

  where λ_i, for $i = 1, 2$, is the arrival rate of the Poisson process in state i, and μ_i, for $i = 1, 2$, is the state-transition rate when the process is in state i.

7.2.3 Application Manager Configuration

The configuration of the Application Manager includes the parameters related to the RLS algorithm, the ARX model structure, the smoothing factor of the EWMA filter, the parameters for the LQ controller, the parameter for computing the operating point, and the control sampling time. In the following, we provide some detail about the choice of each of them.

RLS Algorithm

As described in Chapter 4, in this thesis we consider four variants of the RLS algorithm, namely RLS-EF, RLS-DF, RLS-DF*, and EW-RLS. For each variant we provide the following parameters.

- *RLS-EF.* The RLS-EF algorithm is characterized by only one parameter, that is the forgetting factor λ, which we set to the commonly used value of 0.98, whereby the weight of past observations decays very rapidly with time.

- *RLS-DF.* The RLS-DF algorithm is characterized by only one parameter, that is the forgetting factor μ, which we set to the commonly used value of 0.98, whereby the weight of past observations decays very rapidly with time.

- *RLS-DF**. The RLS-DF* algorithm is characterized by two parameters, that is the forgetting factor μ and correction factor δ, which we set to 0.98 and 0.001 (where the choice of δ is based on [31]), respectively.

- *EW-RLS.* The EW-RLS algorithm is characterized by two parameters, that is the minimum forgetting factor λ_0 and the sensitivity gain ρ, which we set to 0.70 and 0.05 (as proposed by [129]), respectively

Furthermore, for all variants of RLS, we set the following initial conditions:

$$\theta(0) = \varepsilon, \tag{7.5a}$$

$$\varphi(0) = \mathbf{0}, \tag{7.5b}$$

$$\mathbf{P}(0) = 10^4\mathbf{I} \tag{7.5c}$$

where ε is the floating-point relatively accuracy [134] (i.e., the distance from 1.0 to the next largest double-precision number).

ARX Model Structure

The ARX model structure is parameterized by the system order (n_a, n_b) and the input delay n_k (see Chapter 4).

To find such parameters, we performed, for each type of application, several offline system identification experiments, by evaluating several combinations of triplets (n_a, n_b, n_k) (specifically, from a $(1,1,1)$ to a $(10,10,10)$) and selecting the best one by means of the AIC criterion.

From these experiments, we found that, for each type of application, the model structure with $n_a = 2$, $n_b = 2$, and $n_k = 1$ (i.e., with 2 poles, 1 zero and a single delay per input) was good enough to approximate the behavior of our simulated applications.

EWMA Filter

The EWMA filter has one parameter, namely the smoothing factor α (see Eq. (6.3) in Chapter 6). We experimentally evaluated different candidates, from $\alpha = 0.5$ to $\alpha = 0.9$ with a step increment of 0.1, and we found that the value of 0.70 provided a good trade-off between memory and smoothness, such that the influence of past observation does not vanish too fast and, at the same time, the impact of possible outlier values is slightly delayed.

LQ Control Design

The LQ control design is parameterized by the weighting matrices **Q**, **R**, and **N** (see Chapter 5). For our experiments, we set **N** to the zero matrix and used the *Bryson's rule* [27] for **Q** and **R**, so that the most effective way to decrease the LQ cost function is to obtain a very small control output (or control state), even if this is achieved at the expense of a large control input. Specifically:

$$\mathbf{Q} = \frac{1}{0.05^2}\mathbf{I}, \tag{7.6a}$$

$$\mathbf{R} = \mathbf{I}, \tag{7.6b}$$

$$\mathbf{N} = \mathbf{Z} \tag{7.6c}$$

Operating Point

The operating point used to linearize the application system model, is recomputed online by averaging observations of the last $n_e = 5$ control intervals. Such value has been found by means of trial-and-error experiments.

Control Sampling Time

The value of control sampling time is derived by means of offline trial-and-error experiments, from which we found that a reasonable value (with respect to control responsiveness and control overhead) was to set it to 8 times the value of the request arrival rate. We used such trial-and-error approach since, to the best of our knowledge, there is no systematic method that can be directly applied to computing systems.

The rationale under the choice of using the request arrival rate as a "reference" value for control sampling time is that the arrival rate can be seen as an indicator of how much effort is needed by the controller (i.e., the higher is the arrival rate, the higher is the number of requests the application has to serve, and then the faster the controller needs to operate, and vice versa).

7.2.4 Performance Metrics

We assess the performance of our resource management framework by measuring the application-level response time and the number of violations with respect to the SLOs constraints of the various applications, and the amount of electrical energy (expressed in Joule) consumed to serve each submitted service request.

In our experiments, we use as SLO specification of each application the 99^{th} percentile of its response time. This means that the IaaS provider will pay penalties only when the percentage of SLO violations (i.e., the percentage of the number of times the observed response time is larger than the SLO value) is greater than 1% during a pre-determined time interval.

To compute the SLO value for each application, we use a benchmark-like approach similar to the one described in [169] for application profiling, that consists in running a series of simulations for each application and for each type of arrival process (assigning to each tier exactly the amount of CPU capacity as defined by the reference machine specifications) and measuring the 99^{th} percentile of the response time empirical distribution.

A similar approach is adopted to compute the set-point used by the LQ control design, such that the operating value for the response time is set to the mean response time observed in each simulation, and the one for the CPU share is set to the ratio between the CPU capacity of the reference machine and the one of the machine used in the simulated system.

7.3 Resource Management Approaches

To show the effectiveness of our solution, hereinafter referred to as OUR-APPROACH, we compare it with three other static approaches, traditionally applied in data center

resource management, named in the following as STATIC-SLO, STATIC-ENERGY, and STATIC-TRADEOFF.

In the STATIC-SLO approach, the IaaS provider statically assigns to each VM (running a particular application tier) the amount of CPU capacity defined by the reference machine. This is a SLO-conserving approach since, by assigning exactly the amount of capacity needed to satisfy SLOs, the provider favors SLOs satisfaction in place of energy consumption reduction.

Conversely, in the STATIC-ENERGY approach, the IaaS provider statically assigns to each VM a fixed amount of capacity that is 25% lower than the one defined by the reference machine. This is an energy-conserving approach since, by assigning less capacity than required, the provider favors energy consumption reduction instead of SLOs satisfaction (in the hope to still get a low number of SLO violations).

Finally, in the STATIC-TRADEOFF approach, the IaaS provider statistically assigns to each VM a fixed amount of capacity that is 10% lower than the one defined by the reference machine. Thus, this approach tries to find a reasonable trade-off between energy consumption reduction and SLOs preservation.

For the sake of readability, we have grouped the results in four different scenarios, namely S-DMPP, S-PMPP, S-MMPP and S-MIX, which differ from each other in the type of arrival process, whose settings are summarized in Table 7.1 (where we used the same notation described in Section 7.2). It is important to point out that in each scenario, except for S-MIX, we fixed the type of arrival process and varied the type of application to A_1, A_2, and A_3. Instead, in the S-MIX scenario, we fixed the type of application to A_3 and varied, for each application instance, the type of arrival process. The purpose of this scenario is to show that under different type of time-varying workloads our solution is still effective.

Table 7.1: Experimental Evaluation – Scenarios.

Scenario	Application #1			Application #2			Application #3	
	Type	Arrival Process	SLO	Type	Arrival Process	SLO	Type	Arrival Process	SLO
S-DMPP	A_1	DMPP$(1,5,10,3600)$	1.176	A_2	DMPP$(10,5,1,3600)$	0.612	A_3	DMPP$(5,10,1,3600)$	0.608
S-PMPP	A_1	PMPP$(5,10,1,1.5)$	1.245	A_2	Same as Application #1	0.655	A_3	Same as Application #1	0.624
S-MMPP	A_1	MMPP$(5,15,0.0002,0.002)$	4.001	A_2	Same as Application #1	1.962	A_3	Same as Application #1	1.935
S-MIX	A_3	DMPP$(5,10,1,3600)$	0.608	A_3	MMPP$(5,15,0.0002,0.0020)$	1.935	A_3	PMPP$(5,10,1,1.5)$	0.624

Chapter 8

Performance Evaluation without VM Migration

In this chapter, we evaluate the performance of our resource management framework for the case where:

- only the Application Manager and Physical Machine Manager components are used, and
- the lifetime of each hosted application spans for the length of each replication.

Put in another way, in this case study we do not use VM migration and we do not allow an application to start of finish in the middle of the simulation.

The chapter is organized as follows. First, in Section 8.1, we describe specific settings used to setup this case study. Then, in Section 8.2, we present experimental results.

8.1 Experimental Setup

In addition to settings presented in Chapter 7, we provide the following configuration setup.

8.1.1 Physical Infrastructure Configuration

We consider a set of five homogeneous physical machines whose computing power, set to 2000, is twice that of the reference machine.

For all physical machines, we consider the power consumption model as defined by Eq. (7.1). We estimate the values of the parameters ω_0, ω_1, ω_2 and ρ through a statistical regression analysis over data collected by the *SPECpower_ssj2008* benchmark [15], and we set them to $\omega_0 = 143$, $\omega_1 = 258.2$, $\omega_2 = 117.2$, and $\rho = 0.355$, thus leading to the following model instantiation:

$$W = 143 + 258.2u + 117.2u^{0.355} \tag{8.1}$$

The placement of each VM is performed at the beginning of the simulation with a static approach; each VM is randomly assigned to 1 of 5 physical machines, thus resulting in at most 2 VMs per physical machine.

8.1.2 Performance Metrics

The performance of our resource management framework is assessed by means of simulation, by using the independent replication method (see Section 7.1), where:

- the terminating condition of each replication is set to be the achievement of at least 1000000 served requests per application, and
- the number of independent replicas needed to achieve, for each simulation experiment, the required 95% confidence level is computed online by means of the relative precision of the confidence interval method set to 4%.

8.2 Results and Discussion

The results of the various scenarios are presented in four separate tables: Table 8.1 for S-DMPP, Table 8.2 for S-PMPP, Table 8.3 for S-MMPP, and finally Table 8.4 for S-MIX. For the sake of readability, we limit to report only those results deriving from the best combination of RLS algorithm and LQ control design, among all of their variants we considered in this thesis.

Table 8.1: Experimental Evaluation – Results for the S-DMPP Scenario.

			Approach			
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH
A_1	Response Time	(s)	1.16 (0.02)	3.08 (0.07)	1.54 (0.03)	1.15 (0.02)
	% SLO Violations	(%)	0.66	19.40	2.77	0.66
A_2	Response Time	(s)	0.61 (0.01)	1.58 (0.03)	0.81 (0.01)	0.61 (0.01)
	% SLO Violations	(%)	0.75	14.05	2.68	0.75
A_3	Response Time	(s)	0.61 (0.00)	1.69 (0.02)	0.82 (0.01)	0.61 (0.00)
	% SLO Violations	(%)	0.78	19.41	3.14	0.78
Uptime		(Ms)	0.96	0.96	0.96	0.96
Energy Consumption	Total Joules	(MJ)	201.26	195.35	199.60	200.47
	Wasted Joules	(MJ)	1.57	34.68	5.75	1.47

Table 8.2: Experimental Evaluation – Results for the S-PMPP Scenario.

			Approach			
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH
A_1	Response Time	(s)	1.24 (0.03)	3.18 (0.18)	1.65 (0.04)	1.24 (0.03)
	% SLO Violations	(%)	0.66	21.52	3.00	0.67
A_2	Response Time	(s)	0.71 (0.15)	1.68 (0.06)	0.87 (0.03)	0.89 (0.43)
	% SLO Violations	(%)	0.70	16.88	2.72	0.70
A_3	Response Time	(s)	0.59 (0.00)	1.60 (0.11)	0.79 (0.01)	0.60 (0.01)
	% SLO Violations	(%)	0.47	15.77	2.03	0.47
Uptime		(Ms)	0.74	0.78	0.74	0.74
Energy Consumption	Total Joules	(MJ)	167.38	168.54	165.22	166.10
	Wasted Joules	(MJ)	1.03	30.35	4.30	1.02

Table 8.3: Experimental Evaluation – Results for the S-MMPP Scenario.

			Approach			
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH
A_1	Response Time	(s)	4.11 (0.59)	n/a	n/a	4.30 (1.00)
	% SLO Violations	(%)	0.68	n/a	n/a	0.81
A_2	Response Time	(s)	1.98 (0.22)	n/a	n/a	1.97 (0.21)
	% SLO Violations	(%)	0.76	n/a	n/a	0.77
A_3	Response Time	(s)	1.98 (0.21)	n/a	n/a	1.83 (0.12)
	% SLO Violations	(%)	0.77	n/a	n/a	0.66
Uptime		(Ms)	0.87	n/a	n/a	0.87
Energy Consumption	Total Joules	(MJ)	185.81	n/a	n/a	185.49
	Wasted Joules	(MJ)	1.37	n/a	n/a	1.38

Table 8.4: Experimental Evaluation – Results for the S-MIX Scenario.

			Approach			
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH
A_1	Response Time	(s)	0.61 (0.00)	n/a	0.82 (0.00)	0.61 (0.00)
	% SLO Violations	(%)	0.77	n/a	3.15	0.79
A_2	Response Time	(s)	2.10 (0.12)	n/a	19.92 (5.71)	2.08 (0.35)
	% SLO Violations	(%)	1.00	n/a	18.67	0.77
A_3	Response Time	(s)	0.61 (0.02)	n/a	0.81 (0.03)	0.70 (0.20)
	% SLO Violations	(%)	0.65	n/a	2.59	0.65
Uptime		(Ms)	0.82	n/a	0.85	0.82
Energy Consumption	Total Joules	(MJ)	171.31	n/a	173.57	170.76
	Wasted Joules	(MJ)	1.36	n/a	12.94	1.24

In each table, every column reports the results obtained by the various applications, under a specific resource management approach (i.e., STATIC-SLO, STATIC-ENERGY, STATIC-TRADEOFF, and OUR-APPROACH). A column filled with the symbol "n/a" (which stands for "result not available") means that the use of the corresponding resource management approach made the simulation unable to converge. Numbers inside parenthesis (when present) represent the standard deviations of the related measures, while letters inside parenthesis represent unit of measures (e.g., "(s)" means seconds). The rows of each table have instead the following meaning. Tows labeled by A_i, for $i = 1, \ldots, 3$, report the 99^{th} percentile of the response time (label "Response Time"), expressed in seconds (s), and the mean percentage of SLO violations (label "% SLO Violations") for each application instance; lower values correspond to better results. The row labeled by "Uptime" reports the sum of the mean uptime of all the physical machines, where the uptime of a physical machine is defined as the total time (from the beginning of each simulation replica) that the physical machine has been powered on. This metric quantifies the efficiency of a given approach in using the physical machines of the cloud infrastructure, since lower "Uptime" values indicate the usage of a lower amount of physical resource capacity to serve a given workload. The "Uptime" metric is expressed in million of seconds (Ms). The row labeled by "Energy Consumption" reports two energy-related metrics: the total energy consumed, on average, by the cloud infrastructure (label "Total Joules") and an estimate of the total consumed energy that the cloud infrastructure spend, on average, to serve out-of-SLO requests (label "Wasted Joules"). In particular, the "Wasted Joules" metric provides an indication of how efficiently a given approach used physical resources of the cloud infrastructure in order to lower the number of SLO violations and, at the same time, to reduce energy consumption; thus, the lower is its value, the better is the result. Both metrics are expressed in MegaJoules (MJ).

By looking at the results reported in the tables, we can observe that our approach (column OUR-APPROACH) always achieves a lower number of SLO violations (with respect to the 1% threshold defined by SLO specifications), and always outperforms the STATIC-ENERGY and STATIC-TRADEOFF approaches. As a matter of fact, for all scenarios, OUR-APPROACH, with respect to the STATIC-ENERGY and STATIC-TRADEOFF ones, is able to satisfy SLOs for a greater

number of requests with a lower energy consumption and, more importantly, without resulting in any penalty to be paid by the provider (as instead is for the STATIC-ENERGY and STATIC-TRADEOFF approaches).

The advantage of OUR-APPROACH is more evident for both S-MMPP (see Table 8.3) and S-MIX (see Table 8.4), where the STATIC-ENERGY approach has not been able to converge to the prescribed accuracy because of the aggregation of too many queueing phenomena (caused by the inability to dynamically adjust CPU shares during high-intensity arrival periods), which resulted in an unstable (simulated) system. A similar behavior can be observed for the STATIC-TRADEOFF approach, whereby in S-MIX (see Table 8.4) the simulation fails to converge.

The comparison between OUR-APPROACH and STATIC-SLO approaches needs more attention. First, the energy consumption implied by OUR-APPROACH is always lower (or, at least, comparable) than the one obtained with the STATIC-SLO one, thus resulting in a lower (or comparable) waste of Joules. Second, both approaches always keep the percentage of SLO violations under the 1% threshold (as defined by SLO specifications). For all scenarios, but for S-MMPP, the percentage of such violations yield by OUR-APPROACH is always lower than or comparable to the STATIC-SLO one. For what regards the S-MMPP scenario, there are mixed situations. Indeed, as can be observed in Table 8.3, OUR-APPROACH, with respect to STATIC-SLO, leads to a greater percentage of SLO violations for application A_1; however, this is somewhat compensated by the lower percentage of SLO violations obtained for application A_3. This can be ascribed to the following reason. OUR-APPROACH, unlike the STATIC-SLO approach, is able to dynamically react to high-intensity workload periods by increasing the fraction of resource capacity for the interested application; thus it generally should exhibit a lower percentage of SLO violations than the STATIC-SLO approach. However, there might be cases where such periods overlap in a way that some tier get less resource capacity than needed, thus increasing the probability to violate SLO constraints; indeed, this is the main reason motivating the cases where the percentage of SLO violations obtained with OUR-APPROACH is greater than the one resulted with the STATIC-SLO one.

Finally, it is important to note that the STATIC-SLO approach requires an overcommitment of resources, whereby a larger fraction of CPU capacity is assigned

to each VM *regardless of the fact that this fraction will be actually used by the VM*. As a consequence, this approach implies that the number of VMs that can be consolidated on the same physical machine is lower than those attained by OUR-APPROACH (that, instead, allocates to each VM the fraction of CPU capacity it needs). Therefore, the STATIC-SLO approach potentially requires – for a given number of VMs – a larger number of physical resources than the OUR-APPROACH one, thus yielding a larger energy consumption.

In conclusion, OUR-APPROACH is able to achieve better results compared to other approaches both in terms of efficiency and energy consumption.

Chapter 9

Performance Evaluation with VM Migration

In this chapter, we evaluate the performance of our resource management framework for the case where:

- all components of the resource management framework are used, and
- the lifetime of each hosted application can potentially be shorter than the length of a replication.

Put in another way, in this case study, VMs can migrate to different physical machines and applications can start and finish in the middle of the simulation.

The chapter is organized as follows. First, in Section 9.1, we describe specific settings used to setup this case study. Then, in Section 9.2, we report and discuss experimental results.

9.1 Experimental Setup

In addition to settings presented in Chapter 7, we provide the following configuration setup.

Table 9.1: Experimental setup – Physical machines characteristics.

# Instances	CPU Capacity	Reference Machine Multiplier	Power Model			
			ω_0	ω_1	ω_2	ρ
3	1000	x1	86.7	119.1	69.06	0.400
2	2000	x2	143.0	258.2	117.2	0.355
2	3000	x3	178.0	310.6	160.4	0.311
2	4000	x4	284.0	490.1	343.7	0.462

9.1.1 Physical Infrastructure Configuration

Unlike the case study presented in Chapter 8, the evaluation of our resource management framework is done in a heterogeneous environment in order to better take advantage of the potential offered by VM migration.

Specifically, we consider a set of nine heterogeneous physical machines whose characteristics are reported in Table 9.1. In this table, there is one row for each type of physical machines. In particular, the first column (named "# Instances") shows the number of instances (of a particular type of physical machines) used in the experiments. The second column (labeled "CPU Capacity") reports the CPU capacity. The third column (called "Reference Machine Multiplier") indicates the relative CPU capacity (in terms of a multiplicative factor) with respect to the reference machine; for instance, a value of x2 means that the capacity of that type of physical machines is twice the one of the reference machine. The fourth to last columns (grouped under the label "Power Model") reports the coefficients of the power consumption model, as defined by Eq. (7.1).

For the power consumption model, we estimate the values of the parameters ω_0, ω_1, ω_2 and ρ through a statistical regression analysis over data collected by the *SPECpower_ssj2008* benchmark [15], and the resulting fit is shown in the last columns of Table 9.1. A graphical comparison of these power models is shown in Fig. 9.1. Interestingly, from the figure we can observe that, assuming a proportional relationship between capacities and utilizations of different physical machines (as we do in this thesis), it is not always effective (in terms of power consumption) to aggregate the largest number of VMs on the smallest number of physical machines, mostly because of idle power consumption. For instance, suppose that there are 3 identical VMs, each of which using the 100% of a physical

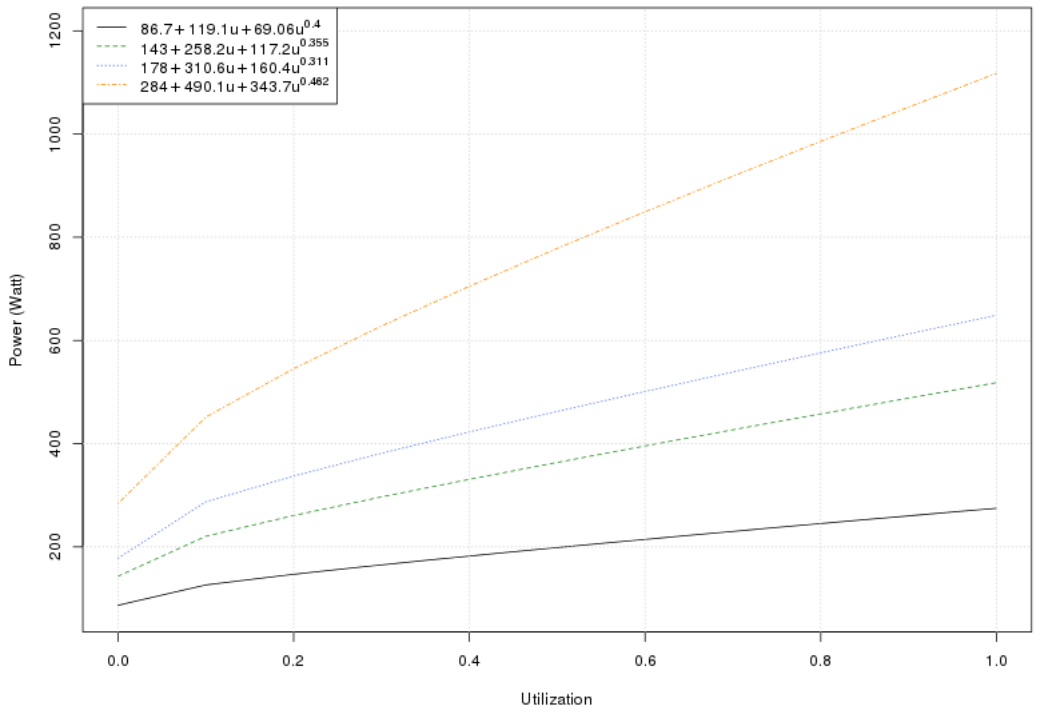


Figure 9.1: Experiments setup – Utilization vs power consumption of physical machines.

machine of CPU capacity 1000; this means, that (according to our assumptions on proportional relationship between CPU capacities) if each VM is deployed on a physical machine of capacity 2000, 3000, or 4000, it will lead to a utilization of 50%, 33%, or 25%, respectively. Thus the resulting aggregated power consumption will amount to:

- 824.58 Watt, if 3 physical machines of capacity 1000 are used;
- 882.14 Watt, if 2 physical machines of capacity 2000 are used;
- 649.00 Watt, if 1 physical machine of capacity 3000 is used;
- 952.50 Watt, if 1 physical machine of capacity 4000 is used.

In this case, it is much more effective to consolidate the 3 VMs on the physical machine of capacity 3000 rather than put them on two or more physical machines with less capacity. However, this is not true if a physical machine of capacity 4000 is used in place of the one with capacity 3000.

9.1.2 Application Configuration

We use the same three types of applications, namely A_1, A_2, and A_3, as defined in Chapter 7. However, we allow each application to start and finish in the middle of each replica of the simulation experiment. Specifically, for each type of application, we create two instances, one with a lifetime that spans for the entire length of each replication, and the other with a shorter duration, in order to study system configurations in which the number of applications that are simultaneously present dynamically varies over time.

In Table 9.2, we report the lifetime specifications for each application instance, inside each simulation replica. In the table, there is a row for each instance of application. Specifically, the first column (named "Type") shows the type of application. The second column (called "ID") indicates the identifier associated to a particular application instance, that will be used later in the discussion of experiment results. The third to last columns (grouped under the label "Lifetime") reports the start time and the duration of the execution of a particular application

Table 9.2: Experiments setup – Lifetime of each application instance.

Application Type	Application ID	Lifetime Start Time	Lifetime Duration
A_1	A_{11}	0	–
A_1	A_{12}	10000	70000
A_2	A_{21}	0	–
A_2	A_{22}	70000	70000
A_3	A_{31}	0	–
A_3	A_{32}	130000	70000

instance inside each simulation replica; the symbol "–", used for the duration time, means that the application instance stops at the end of each replication.

From the above table, it can be noted that the maximum number of simultaneously running application instances is 5, which comprises the 3 "always running" instances, and 2 instances with shorter lifetime. This happens during the time intervals $[70000, 80000]$, when both A_{12} and A_{22} are executing, and $[130000, 140000]$, when both A_{22} and A_{32} are running.

9.1.3 Migration Manager Configuration

The configuration of the Migration Manager includes the smoothing factor of the EWMA filters, the parameters specific to the optimization problem, and the sampling time. In the following, we provide some detail about the choice of each of them.

EWMA Filters

The Migration Manager uses two EWMA filters: one for computing $\hat{v}_j(k)$ (i.e., the expected contribution to the mean CPU utilization of a physical machine with a capacity equivalent to the one of the reference machine, that will be brought by VM j at control interval k – see Table 6.2 and Eq. (6.11)), and the other for computing $\hat{s}_j(k)$ (i.e., the expected mean CPU share that is assumed VM j will demand to a physical machine with a capacity equivalent to the one of the reference machine, at control interval k – see Table 6.2 and Eq. (6.19)).

For such filters, we need to specify their respective smoothing factors, namely β and γ. For both of them, we choose a value of 0.70 so that the influence of past observations does not vanish too fast.

Optimization Problem

For the parameters of the optimization problem, we choose the following values:

- Maximum aggregated CPU share demand $s_i^{\max}$ for physical machine i: we choose the value 1 for all the physical machines.

- Minimum CPU share $\bar{s}_t^{\min}$ assignable to tier t on the reference machine: we choose the value 0.2 for all the tiers of all the applications. We derive it by means of offline system identification experiments. Specifically, for every experiment, we excite each application with three randomly generated and uniformly distributed in $[\bar{s}^{\min}, 1]$ signals (each of which representing the CPU share assigned to each tier), by varying $\bar{s}^{\min}$ in the range $(0, 0.5]$ with a step increment of 0.05. From these experiments, we find that, for values of $\bar{s}^{\min}$ below 0.2, the behavior of an application becomes unstable due to too many queueing phenomena, with the result that the response time diverges (theoretically) to the infinity.

- CPU utilization threshold $u_i^{\max}$ for physical machine i: we set it to 1 for every physical machine.

- Weights w_e, w_m, and w_p for the objective function J: we choose the value 1 for all the weights, so that the three costs J_e, J_m and J_p, of the objective function, are equally weighted.

For what concerns the value of the mean CPU share $\bar{s}_t$ of each tier t (of every application) on the reference machine, we use the same benchmark-like approach (similar to the one described in [169] for application profiling), already employed for computing the set-point for the LQ control design of the Application Manager (see Chapter 7).

Sampling Time

The value of the sampling time T is derived by means of offline trial-and-error experiments, from which we find that a reasonable value is to set it to 1800 ticks of simulated times (e.g., if one tick of the simulated time represents 1 second, the sampling time amounts to half hour).

9.1.4 Performance Metrics

The performance of our resource management framework is assessed by means of simulation, by using the independent replications output analysis method, where the length of each replication is fixed to 2100000 ticks of simulated time, and the number of total replicas is fixed to 5.

We use these fixed values since the simulated system never reaches the steady-state due to the presence of applications that can start and stop in the middle of the simulation.

9.1.5 Experimental Scenarios and Resource Management Approaches

Experiments performed for this case study, use the same four scenarios described in Chapter 7, that is S-DMPP, S-PMPP, S-MMPP, and S-MIX.

Also, as discussed in Chapter 7, for each scenario, we evaluate our approach with three commonly used techniques, that is STATIC-SLO, STATIC-ENERGY, and STATIC-TRADEOFF. However, in order to evaluate the efficacy of VM migration, we consider two different variants of our approach (instead of only one), namely OUR-APPROACH-NM and OUR-APPROACH-M. Specifically, in OUR-APPROACH-NM, we do not use the Migration Manager component, while in the OUR-APPROACH-M variant, this component is employed.

For what concerns the implementation of the Migration Manager, as discussed in Section 6.4, we propose two possible implementations based on approximated algorithms, that is a greedy algorithm and a strategy based on local optimization.

In order to evaluate both implementations, we divide the experiments in two different groups, namely MM-GREEDY and MM-LOCOPT. In the MM-GREEDY

group, the Migration Manager is implemented by means of the greedy algorithm, while in the MM-LOCOPT group, the Migration Manager is driven by local optimization (see Section 6.4.2).

9.2 Results and Discussion

This section is organized as follows. In Section 9.2.1 and Section 9.2.1, we present and discuss the results obtained by each individual experiments groups, that is MM-GREEDY and MM-LOCOPT, respectively. Finally, in Section 9.2.3, we present some concluding remark about the convenience of using VM migration and the behavior two different implementations of the Migration Manager.

9.2.1 Results for the MM-GREEDY Experiments Group

The results of the various scenarios, in the MM-GREEDY group, are presented in four separate tables: Table 9.3 for S-DMPP, Table 9.4 for S-PMPP, Table 9.5 for S-MMPP, and finally Table 9.6 for S-MIX. For the sake of readability, we limit to report only those results deriving from the best combination of RLS algorithm and LQ control design, among all of their variants we considered in this thesis.

In each table, every column reports the results obtained by the various applications, under a specific resource management approach (i.e., STATIC-SLO, STATIC-ENERGY, STATIC-TRADEOFF, OUR-APPROACH-NM, and OUR-APPROACH-M). A column filled with the symbol "n/a" (which stands for "result not available") means that the use of the corresponding resource management approach made the simulation unable to converge. Numbers inside parenthesis (when present) represent the standard deviations of the related measures, while letters inside parenthesis represent unit of measures (e.g., "(s)" means seconds). The rows of each table have instead the following meaning. Rows labeled by A_{ij}, for $i = 1,\ldots,3$ and $j = 1,2$, report the 99^{th} percentile of the response time (label "Response Time"), expressed in seconds (s), and the mean percentage of SLO violations (label "% SLO Violations") for each application instance; lower values correspond to better results. The row labeled by "Uptime" reports the sum of the mean uptime of all the physical machines, where the uptime of a physical machine is defined as the total

Table 9.3: Experimental evaluation – Results for the S-DMPP scenario in the MM-GREEDY group.

			Approach				
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH-NM	OUR-APPROACH-M
A_{11}	Response Time	(s)	1.17 (0.02)	3.21 (0.23)	1.53 (0.02)	1.16 (0.01)	1.17 (0.03)
	% SLO Violations	(%)	0.62	19.19	2.72	0.63	0.63
A_{12}	Response Time	(s)	1.17 (0.00)	3.08 (0.04)	1.57 (0.01)	1.17 (0.00)	1.17 (0.00)
	% SLO Violations	(%)	0.63	19.73	2.84	0.63	0.63
A_{21}	Response Time	(s)	0.62 (0.01)	1.59 (0.03)	0.81 (0.01)	0.62 (0.01)	0.62 (0.01)
	% SLO Violations	(%)	0.76	14.35	2.72	0.76	0.76
A_{22}	Response Time	(s)	0.63 (0.02)	1.66 (0.01)	0.84 (0.02)	0.63 (0.02)	0.63 (0.02)
	% SLO Violations	(%)	0.76	14.65	2.75	0.77	0.76
A_{31}	Response Time	(s)	0.61 (0.01)	1.70 (0.01)	0.82 (0.00)	0.61 (0.01)	0.61 (0.01)
	% SLO Violations	(%)	0.77	19.40	3.19	0.77	0.77
A_{32}	Response Time	(s)	0.61 (0.00)	1.69 (0.04)	0.83 (0.00)	0.61 (0.00)	0.61 (0.00)
	% SLO Violations	(%)	0.80	19.29	3.20	0.80	0.80
Uptime		(Ms)	0.97	1.25	1.03	0.97	0.85
Energy Consumption	Total Joules	(MJ)	319.91	344.98	325.68	318.42	299.18
	Wasted Joules	(MJ)	2.32	61.47	9.48	2.32	2.18

Table 9.4: Experimental evaluation – Results for the S-PMPP scenario in the MM-GREEDY group.

			Approach				
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH-NM	OUR-APPROACH-M
A_{11}	Response Time	(s)	1.19 (0.02)	3.05 (0.04)	1.59 (0.02)	1.19 (0.02)	1.19 (0.02)
	% SLO Violations	(%)	0.58	19.60	2.57	0.58	0.59
A_{12}	Response Time	(s)	1.29 (0.00)	3.41 (0.10)	1.73 (0.01)	1.29 (0.01)	1.29 (0.00)
	% SLO Violations	(%)	0.89	33.89	4.17	0.90	0.90
A_{21}	Response Time	(s)	0.81 (0.23)	1.64 (0.01)	0.86 (0.00)	0.81 (0.23)	0.81 (0.23)
	% SLO Violations	(%)	0.68	16.04	2.68	0.68	0.68
A_{22}	Response Time	(s)	0.68 (0.02)	1.36 (1.29)	0.90 (0.02)	0.69 (0.00)	0.68 (0.01)
	% SLO Violations	(%)	0.82	9.33	3.24	0.83	0.82
A_{31}	Response Time	(s)	0.59 (0.00)	1.56 (0.01)	0.79 (0.00)	0.59 (0.00)	0.59 (0.00)
	% SLO Violations	(%)	0.53	15.62	2.37	0.53	0.53
A_{32}	Response Time	(s)	0.59 (0.01)	1.55 (0.05)	0.82 (0.03)	0.59 (0.01)	0.59 (0.01)
	% SLO Violations	(%)	0.47	11.57	1.84	0.47	0.47
Uptime		(Ms)	1.35	1.83	1.79	1.32	1.04
Energy Consumption	Total Energy	(MJ)	393.42	419.24	427.40	386.33	353.78
	Wasted Joules	(MJ)	2.51	74.33	11.64	2.47	2.26

Table 9.5: Experimental evaluation – Results for the S-MMPP scenario in the MM-GREEDY group.

			Approach				
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH-NM	OUR-APPROACH-M
A_{11}	Response Time	(s)	4.29 (0.17)	n/a	43.63 (0.34)	4.44 (0.12)	4.43 (0.17)
	% SLO Violations	(%)	1.01	n/a	28.31	1.12	1.14
A_{12}	Response Time	(s)	4.45 (0.45)	n/a	24.03 (1.55)	4.08 (0.10)	4.39 (0.32)
	% SLO Violations	(%)	0.86	n/a	19.61	0.88	0.88
A_{21}	Response Time	(s)	2.04 (0.18)	n/a	35.18 (3.83)	2.04 (0.17)	2.04 (0.19)
	% SLO Violations	(%)	0.89	n/a	21.83	0.90	0.90
A_{22}	Response Time	(s)	1.73 (0.35)	n/a	10.93 (0.16)	1.76 (0.39)	1.73 (0.34)
	% SLO Violations	(%)	0.36	n/a	9.90	0.37	0.36
A_{31}	Response Time	(s)	1.82 (0.06)	n/a	21.57 (0.36)	1.82 (0.06)	1.82 (0.06)
	% SLO Violations	(%)	0.67	n/a	15.68	0.67	0.66
A_{32}	Response Time	(s)	1.95 (0.58)	n/a	15.92 (2.81)	2.02 (0.71)	2.00 (0.66)
	% SLO Violations	(%)	0.62	n/a	15.68	0.62	0.63
Uptime		(Ms)	0.97	n/a	1.17	0.97	0.89
Energy Consumption	Total Joules	(MJ)	330.81	n/a	329.94	330.13	310.11
	Wasted Joules	(MJ)	2.66	n/a	67.31	2.76	2.61

Table 9.6: Experimental evaluation – Results for the S-MIX scenario in the MM-GREEDY group.

			Approach				
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH-NM	OUR-APPROACH-M
A_{11}	Response Time	(s)	0.61 (0.00)	n/a	0.83 (0.00)	0.61 (0.00)	0.61 (0.00)
	% SLO Violations	(%)	0.81	n/a	3.23	0.81	0.81
A_{12}	Response Time	(s)	0.62 (0.00)	n/a	0.83 (0.00)	0.62 (0.00)	0.89 (0.38)
	% SLO Violations	(%)	0.79	n/a	3.14	0.79	0.79
A_{21}	Response Time	(s)	1.98 (0.03)	n/a	35.14 (19.65)	2.00 (0.05)	1.96 (0.06)
	% SLO Violations	(%)	0.87	n/a	20.86	0.84	0.82
A_{22}	Response Time	(s)	1.71 (0.19)	n/a	16.19 (2.95)	1.90 (0.36)	1.79 (0.28)
	% SLO Violations	(%)	0.44	n/a	15.09	0.51	0.52
A_{31}	Response Time	(s)	0.59 (0.03)	n/a	0.79 (0.04)	0.59 (0.03)	0.59 (0.03)
	% SLO Violations	(%)	0.55	n/a	2.36	0.55	0.56
A_{32}	Response Time	(s)	0.61 (0.02)	n/a	0.83 (0.02)	0.61 (0.02)	0.61 (0.02)
	% SLO Violations	(%)	0.65	n/a	2.75	0.65	0.66
Uptime		(Ms)	0.97	n/a	1.08	0.97	0.86
Energy Consumption	Total Joules	(MJ)	316.75	n/a	329..06	315.27	297.53
	Wasted Joules	(MJ)	2.25	n/a	25.63	2.23	2.10

time (from the beginning of each simulation replica) that the physical machine has been powered on. This metric quantifies the efficiency of a given approach in using the physical machines of the cloud infrastructure, since lower "Uptime" values indicate the usage of a lower amount of physical resource capacity to serve a given workload. The "Uptime" metric is expressed in million of seconds (Ms). The row labeled by "Energy Consumption" reports two energy-related metrics: the total energy consumed, on average, by the cloud infrastructure (label "Total Joules") and an estimate of the total consumed energy that the cloud infrastructure spend, on average, to serve out-of-SLO requests (label "Wasted Joules"). In particular, the "Wasted Joules" metric provides an indication of how efficiently a given approach used physical resources of the cloud infrastructure in order to lower the number of SLO violations and, at the same time, to reduce energy consumption; thus, the lower is its value, the better is the result. Both metrics are expressed in MegaJoules (MJ).

By looking at the results reported in these tables, we can observe that both variants of our approach (columns OUR-APPROACH-NM and OUR-APPROACH-M) always achieve a lower number of SLO violations (with respect to the 1% threshold defined by SLO specifications), but the S-MMPP scenario, where our approach exceeds the 1% threshold. Moreover, we can note that while both OUR-APPROACH-M and OUR-APPROACH-NM practically result in identical values of SLO violations for all the scenarios, the former always leads to a more efficient usage of physical resources. As a matter of fact, OUR-APPROACH-M always results in lower values of both "Total Energy" and "Wasted Energy" metrics: this means that the exploitation of VM migration allows to both save energy (lower "Total Energy" values) and to better use the energy that is consumed (lower "Wasted Energy" values). Furthermore, the lower value of "Uptime" exhibited by OUR-APPROACH-M means that a smaller amount of physical resource capacity is required to meet SLOs: this means that, potentially, more VMs can be consolidated on the same number of physical machines without increasing the percentage of violated SLOs.

With respect to the other approaches, both variants of our approach always outperform the STATIC-ENERGY and STATIC-TRADEOFF approaches. Furthermore, STATIC-ENERGY can be considered worse than STATIC-TRADEOFF since

it results in a moderate improvement of (unit) energy consumption at the price of much higher values of SLO violations. Therefore, we can conclude that, with respect to these two approaches, our approach is able to satisfy SLOs for a greater number of requests with a lower energy consumption, in a lower amount of time, and, more importantly, without resulting in any penalty to be paid by the provider.

The comparison with the STATIC-SLO approach needs more attention. First of all, we can observe that, for all scenarios, both our approaches practically exhibit the same values of SLO violations as STATIC-SLO. Second, both approaches keep the percentage of SLO violations under the 1% threshold (as defined by SLO specifications) for all scenarios, except for the S-MMPP one, where, however, also the STATIC-SLO approach exceeds this threshold. Specifically, in the S-MMPP scenario, our approach results in a higher number of SLO violations only for application A_{11}, while, for the remaining applications, our approach and STATIC-SLO practically show the same performance for this metric.

If, however, we look at the efficiency-related metrics, we can observe that our approach, and in particular OUR-APPROACH-M, always results in lower values of "Uptime", "Total Energy" and "Wasted Energy" than STATIC-SLO, indicating that the former is able to consume less energy and to use physical resources more effectively. For instance, in the S-DMPP scenario, the STATIC-SLO approach leads to an energy consumption which exceeds by nearly 7% (i.e., by approximately 20.73 MJ) the one obtained with OUR-APPROACH-M.

Moreover, it is important to observe that STATIC-SLO requires an overcommitment of resources, whereby a larger fraction of CPU capacity is assigned to each VM regardless of the fact that this fraction will be actually used by the VM. As a consequence, this implies that the number of VMs that can be consolidated on the same physical machine is lower than those attained by our approach (that, instead, allocates to each VM just the fraction of CPU capacity it needs). Therefore, STATIC-SLO potentially requires, for a given number of VMs, a larger number of physical resources than the our approach one, thus yielding a larger energy consumption.

Finally, it is worth noting that a possible reason to the inability of our approach to always keep the percentage of SLO violations under the 1% threshold in the S-MMPP scenario (e.g., see application A_{11}), is the combination of the temporal

burstiness (caused by the MMPP arrival process) with the dynamic execution of application instances. Together, these two factors contribute to the increase of the amount of physical resource demanded, so that if a VM has received a low or medium CPU share (with respect to the reference machine) during a low-intensity workload period, it is possible that it is unable to react in time to such bursts (even if a greater share has been assigned to it), due to the aggregation of too many queueing phenomena and to delays in the reaction time. This is also demonstrated by the behavior of the STATIC-TRADEOFF approach, whereby neither a fixed share of 90% (with respect to the capacity of the reference machine) per VM is able to avoid these situations.

9.2.2 Results for the MM-LOCOPT Experiments Group

The results of the various scenarios, in the MM-LOCOPT group, are presented in four separate tables: Table 9.7 for S-DMPP, Table 9.8 for S-PMPP, Table 9.9 for S-MMPP, and finally Table 9.10 for S-MIX. As done for the MM-GREEDY group, for the sake of readability, we limit to report only those results deriving from the best combination of RLS algorithm and LQ control design, among all of their variants we considered in this thesis.

Results tables are structured as the one presented for the MM-GREEDY group. Specifically, in each table, every column reports the results obtained by the various applications, under a specific resource management approach (i.e., STATIC-SLO, STATIC-ENERGY, STATIC-TRADEOFF, OUR-APPROACH-NM, and OUR-APPROACH-M). A column filled with the symbol "n/a" (which stands for "result not available") means that the use of the corresponding resource management approach made the simulation unable to converge. Numbers inside parenthesis (when present) represent the standard deviations of the related measures, while letters inside parenthesis represent unit of measures (e.g., "(s)" means seconds). The rows of each table have instead the following meaning. Rows labeled by A_{ij}, for $i = 1,\ldots,3$ and $j = 1,2$, report the 99^{th} percentile of the response time (label "Response Time"), expressed in seconds (s), and the mean percentage of SLO violations (label "% SLO Violations") for each application instance; lower values correspond to better results. The row labeled by "Uptime" reports the sum of the

Table 9.7: Experimental evaluation – Results for the S-DMPP scenario in the MM-LocOpt group.

			Approach				
			Static-SLO	Static-Energy	Static-TradeOff	Our-Approach-NM	Our-Approach-M
A_{11}	Response Time	(s)	1.17 (0.02)	3.02 (0.02)	1.41 (0.04)	1.18 (0.02)	1.18 (0.02)
	% SLO Violations	(%)	0.62	19.87	1.86	0.62	0.64
A_{12}	Response Time	(s)	1.17 (0.00)	3.08 (0.04)	1.57 (0.01)	1.18 (0.03)	1.19 (0.03)
	% SLO Violations	(%)	0.63	19.73	2.84	0.63	0.64
A_{21}	Response Time	(s)	0.62 (0.01)	1.65 (0.03)	0.81 (0.01)	0.62 (0.01)	0.62 (0.01)
	% SLO Violations	(%)	0.76	14.69	2.72	0.76	0.77
A_{22}	Response Time	(s)	0.63 (0.02)	1.66 (0.01)	0.84 (0.02)	0.63 (0.02)	0.63 (0.02)
	% SLO Violations	(%)	0.76	14.65	2.75	0.76	0.77
A_{31}	Response Time	(s)	0.61 (0.01)	1.68 (0.00)	0.82 (0.00)	0.61 (0.01)	0.61 (0.01)
	% SLO Violations	(%)	0.77	19.30	3.19	0.77	0.77
A_{32}	Response Time	(s)	0.61 (0.00)	1.69 (0.40)	0.83 (0.00)	0.61 (0.00)	0.61 (0.00)
	% SLO Violations	(%)	0.80	19.29	3.20	0.80	0.80
Uptime		(Ms)	0.97	1.18	0.97	0.97	0.87
Energy Consumption	Total Joules	(MJ)	321.67	340.14	319.20	319.47	303.87
	Wasted Joules	(MJ)	2.33	54.79	8.66	2.32	2.23

Table 9.8: Experimental evaluation – Results for the S-PMPP scenario in the MM-LocOpt group.

			Approach				
			Static-SLO	Static-Energy	Static-TradeOff	Our-Approach-NM	Our-Approach-M
A_{11}	Response Time	(s)	1.19 (0.02)	3.26 (0.03)	1.46 (0.01)	1.19 (0.02)	1.19 (0.01)
	% SLO Violations	(%)	0.58	32.17	1.74	0.59	0.60
A_{12}	Response Time	(s)	1.29 (0.00)	3.41 (0.10)	1.73 (0.01)	1.29 (0.01)	1.29 (0.01)
	% SLO Violations	(%)	0.89	33.89	4.17	0.89	0.88
A_{21}	Response Time	(s)	0.81 (0.23)	1.89 (0.02)	0.86 (0.00)	0.81 (0.23)	0.81 (0.23)
	% SLO Violations	(%)	0.68	18.27	2.68	0.68	0.68
A_{22}	Response Time	(s)	0.68 (0.02)	2.04 (0.32)	0.90 (0.02)	0.68 (0.02)	0.68 (0.01)
	% SLO Violations	(%)	0.82	19.42	3.24	0.82	0.82
A_{31}	Response Time	(s)	0.59 (0.00)	1.46 (0.07)	0.79 (0.00)	0.59 (0.00)	0.59 (0.00)
	% SLO Violations	(%)	0.53	10.45	2.37	0.53	0.53
A_{32}	Response Time	(s)	0.59 (0.01)	1.55 (0.05)	0.82 (0.03)	0.59 (0.00)	0.59 (0.01)
	% SLO Violations	(%)	0.47	11.57	1.84	0.47	0.47
Uptime		(Ms)	0.97	1.24	0.97	0.97	0.90
Energy Consumption	Total Joules	(MJ)	349.47	387.45	345.10	346.23	337.17
	Wasted Joules	(MJ)	2.22	13.32	8.73	2.22	2.16

Table 9.9: Experimental evaluation – Results for the S-MMPP scenario in the MM-LocOpt group.

			Approach				
			Static-SLO	Static-Energy	Static-TradeOff	Our-Approach-NM	Our-Approach-M
A_{11}	Response Time	(s)	4.29 (0.17)	n/a	21.10 (1.04)	15.82 (16.00)	4.29 (0.19)
	% SLO Violations	(%)	1.01	n/a	20.11	1.83	1.00
A_{12}	Response Time	(s)	4.45 (0.45)	n/a	24.03 (1.55)	5.12 (1.10)	4.76 (0.90)
	% SLO Violations	(%)	0.86	n/a	19.61	1.21	0.94
A_{21}	Response Time	(s)	2.04 (0.18)	n/a	10.43 (0.16)	2.05 (0.18)	2.04 (0.18)
	% SLO Violations	(%)	0.89	n/a	8.19	0.91	0.89
A_{22}	Response Time	(s)	1.73 (0.35)	n/a	10.93 (0.16)	1.81 (0.45)	1.74 (0.36)
	% SLO Violations	(%)	0.36	n/a	9.90	0.36	0.36
A_{31}	Response Time	(s)	1.82 (0.06)	n/a	15.03 (1.38)	1.82 (0.06)	1.82 (0.06)
	% SLO Violations	(%)	0.67	n/a	13.59	0.67	0.67
A_{32}	Response Time	(s)	1.95 (0.58)	n/a	15.92 (2.81)	1.77 (0.31)	1.96 (0.61)
	% SLO Violations	(%)	0.62	n/a	15.07	0.74	0.64
Uptime		(Ms)	0.97	n/a	1.17	0.97	0.89
Energy Consumption	Total Joules	(MJ)	327.46	n/a	355.89	326.41	315.22
	Wasted Joules	(MJ)	2.63	n/a	17.73	3.48	2.55

Table 9.10: Experimental evaluation – Results for the S-MIX scenario in the MM-LOCOPT group.

			Approach				
			STATIC-SLO	STATIC-ENERGY	STATIC-TRADEOFF	OUR-APPROACH-NM	OUR-APPROACH-M
A_{11}	Response Time	(s)	1.25 (0.00)	n/a	1.54 (0.02)	1.25 (0.00)	1.25 (0.00)
	% SLO Violations	(%)	1.08	n/a	3.12	1.09	1.09
A_{12}	Response Time	(s)	1.25 (0.01)	n/a	1.69 (0.01)	1.26 (0.01)	1.26 (0.01)
	% SLO Violations	(%)	1.03	n/a	4.38	1.03	1.03
A_{21}	Response Time	(s)	2.02 (0.03)	n/a	35.28 (19.60)	2.29 (0.40)	2.29 (0.45)
	% SLO Violations	(%)	0.90	n/a	20.91	0.95	0.92
A_{22}	Response Time	(s)	1.76 (0.19)	n/a	16.65 (2.41)	1.74 (0.21)	1.71 (0.20)
	% SLO Violations	(%)	0.47	n/a	15.12	0.46	0.40
A_{31}	Response Time	(s)	0.59 (0.03)	n/a	0.79 (0.04)	0.59 (0.03)	0.59 (0.03)
	% SLO Violations	(%)	0.55	n/a	2.36	0.56	0.56
A_{32}	Response Time	(s)	0.61 (0.02)	n/a	0.83 (0.02)	0.61 (0.02)	0.61 (0.02)
	% SLO Violations	(%)	0.65	n/a	2.75	0.66	0.65
Uptime		(Ms)	0.97	n/a	0.89	0.97	0.89
Energy Consumption	Total Joules	(MJ)	334.05	n/a	318.72	333.46	320.01
	Wasted Joules	(MJ)	2.69	n/a	25.10	2.73	2.58

mean uptime of all the physical machines, where the uptime of a physical machine is defined as the total time (from the beginning of each simulation replica) that the physical machine has been powered on. This metric quantifies the efficiency of a given approach in using the physical machines of the cloud infrastructure, since lower "Uptime" values indicate the usage of a lower amount of physical resource capacity to serve a given workload. The "Uptime" metric is expressed in million of seconds (Ms). The row labeled by "Energy Consumption" reports two energy-related metrics: the total energy consumed, on average, by the cloud infrastructure (label "Total Joules") and an estimate of the total consumed energy that the cloud infrastructure spend, on average, to serve out-of-SLO requests (label "Wasted Joules"). In particular, the "Wasted Joules" metric provides an indication of how efficiently a given approach used physical resources of the cloud infrastructure in order to lower the number of SLO violations and, at the same time, to reduce energy consumption; thus, the lower is its value, the better is the result. Both metrics are expressed in MegaJoules (MJ).

By looking at the results reported in these tables, we can observe that only in the S-DMPP and S-PMPP scenarios (see Table 9.7 and Table 9.8, respectively), both variants of our approach always achieve a lower number of SLO violations (with respect to the 1% threshold defined by SLO specifications). For the other two scenarios, the OUR-APPROACH-M exhibits better performance than OUR-APPROACH-NM. As a matter of fact, in the S-MMPP scenario, the OUR-APPROACH-M achieves very good performance, while that of OUR-APPROACH-NM are very poor since (1) there is a very high variability in the distribution of response time (e.g., see application A_{11}), and (2) the number of SLO violations exceeds the prescribed threshold of 1% (e.g., see applications A_{11} and A_{12}). For what concerns the S-MIX scenario, the performance obtained by OUR-APPROACH-M is better than the one of OUR-APPROACH-NM, since, despite the two approaches nearly exhibit the same results in term of SLO violations, the former also leads to a better reduction of energy consumption.

With respect to the STATIC-ENERGY and STATIC-TRADEOFF approaches, both variants of our approach are always able to outperform them. As a matter of fact, with respect to these two approaches, our approach is able to satisfy SLOs for a greater number of requests with a lower energy consumption and, more

importantly, without resulting (most of the time) in any penalty to be paid by the provider.

The comparison with the STATIC-SLO approach needs more study. Firstly, with respect to our approach, the STATIC-SLO approach is always able to achieve similar (and sometimes better) performance in terms of percentage of SLO violations. However, for all scenarios, such better behavior is not useful since it does not provide any benefit to the IaaS cloud provider because of the greater amount of consumed energy. This can be observed both from the "Total Energy" and the "Wasted Energy" metrics. For instance, in the S-DMPP scenario, the STATIC-SLO approach leads to an energy consumption which exceeds by nearly 6% (i.e., by approximately 17.73 MJ) the one obtained with OUR-APPROACH-M. In this case, a behavior like the one exhibited by OUR-APPROACH-M is more desirable, since, on average, allows to save energy, while still keeping the number of SLO violations under the prescribed threshold.

Finally, it is important to note that between the two variants of our approach there is no a clear winner. In general, both of them are able to keep the number of SLO violations low, while limiting the energy consumption. However, as already observed for the MM-GREEDY experiments group (see Section 9.2.1), the OUR-APPROACH-M variant is more effective in reducing the energy consumption and in trying to achieve SLO constraints, especially in high-intensity workload scenarios like S-MMPP (see Table 9.4).

9.2.3 Concluding Remarks

In this section, we study the efficacy of using VM migration and the behavior of the two different implementations of the Migration Manager, with respect to the experiments presented in the above sections.

Effectiveness of VMs Migration. From the results of the experiments just described, we can conclude that the use of the Migration Manager (and thus of VM migration) helps to keep the percentage of SLO violations under (or very nearly to) the prescribed threshold of 1%, even under high-intensity bursty workloads (e.g., like in the S-MMPP scenario of the MM-GREEDY group – see Table 9.5).

Moreover, even when the resulting performance are comparable with the one obtained without the Migration Manager, the use of the Migration Manager leads to a better reduction of energy consumption (e.g., like in the S-PMPP scenario of the MM-GREEDY group – see Table 9.4).

Greedy vs Local Optimization Implementation. From the results of the above experiments, we can note that the behavior of both implementations is very similar. However, we can try to perform a thorough comparison by observing the following facts:

- S-DMPP scenario: for approximately the same percentage of SLO violations, the greedy implementation is able to lead to a slightly lower energy consumption, thus resulting in a (slightly) lower waste of Joules.
- S-PMPP scenario: the performance of both implementations are rather comparable, even if the local optimization implementation seems to be able to slightly waste less Joules.
- S-MMPP scenario: the advantages brought by one implementation over the other one are not so clear. On the hand, for the OUR-APPROACH-NM approach, the greedy implementation performs better than the local optimization one, since it is able to achieve a lower number of SLO violations and, at the same time, to lead to a lower waste of Joules. On the other hand, for the OUR-APPROACH-M approach, the local optimization implementation shows better performance than the greedy one, since it is able to keep the percentage of SLO violations under (or very nearly to) the prescribed threshold and, at the same time, to lead to a lower waste of Joules.
- S-MIX scenario: the greedy implementation performs better than the local optimization one, since it is able to always keep the percentage of SLO violations under the predefined threshold and, at the same time, to lead to a greater reduction of energy consumption.

From these facts, it results that, for the scenarios taken into consideration, the greedy implementation performs better than the one based on local optimization.

This can be ascribed to the way the objective function has been formulated. Indeed, interestingly, the local optimization algorithm starts the search for the (local) optimum just from the solution computed by the greedy algorithm. Since the solution obtained by the local optimization method is never worse than the one provided as initial starting point, this may mean that the objective function does not capture all the required dynamics that allow to strive the joint goal of power consumption minimization and performance optimization.

Part IV

Conclusion

Chapter 10

Related Works

In this chapter, we provide an overview of recent research works that are related to the contributions of this thesis. First, in Section 10.1, we present related works about resource management in cloud infrastructures. Then, in Section 10.2, we discuss works related to simulation of cloud systems.

10.1 Resource Management for Cloud Systems

In this section, we provide a brief state-of-the-art about dynamic performance- and power-aware resource management in cloud infrastructures. Over last years, many research works have arisen for dynamically managing physical resources of a Cloud infrastructure in order to take into consideration performance targets of hosted applications and power consumption of the infrastructure. However, the majority of them deals with these aspects separately (e.g., see [33, 44, 127, 157, 156] and [123, 135, 165] for disjoint performance-aware and power-aware resource management approaches, respectively). Only very recently, several works, combining dynamic power- and performance-aware resource management, have published in the literature (e.g., see [25, 70, 106, 158, 170, 167]).

The rest of this section is organized as follows. First, we provide an overview of separate performance- and power-aware resource management. Finally, we present recent works on joint performance- and power-aware resource management of cloud infrastructures.

10.1.1 Performance-aware Resource Management

In this section, we present recent works aimed at satisfying application performance constraints in cloud computing systems, without explicitly considering the reduction of power consumption.

In [157, 156], the authors present a dynamic resource provisioning framework that uses an analytical model for multi-tier Web applications to determine how much resource to allocate to each application tier. The analytical model consists in a closed queueing network model where each tier is represented by a G/G/1/PS queueing station. The model is designed to handle session-based workloads and can account for application characteristics such as replication at tiers, load imbalances across replicas, caching effects, and concurrency limits at each tier. Resource provisioning is implemented via a combination of predictive and reactive methods that determine when to provision resources over both long- and short-time scales. Predictive provisioning, employed for long-term decisions, is based on historical data and is done by means of a statistical workload predictor that predicts (from the tail of the arrival rate empirical distribution) the peak demand that will be seen over a specific time period. The predicted peak arrival rate (one for each application) is used together with the queueing network model to determine the number of servers that should be allocated to each application tier in current control period. Reactive provisioning, used to swiftly react to unexpected workload demands at the short-time scale, compares the currently observed arrival rate with the one predicted by the analytical model and, if the twos differ by more than a threshold, corrective actions are taken. Model validation, performed by means of an experimental prototype, consists in running two real applications and comparing observed response times with the ones predicted by the model. Results indicate that the proposed model is able to capture the performance of tested applications under a number of workloads and configurations. There two key points that make this work different from our solution. First, this work only handles average-based performance indices, since it relies on steady-state queueing model estimation. Instead, our solution can work with any performance index (like quantiles), and hence can offer a wider and possibly more realistic applicability (e.g., SLOs are usually expressed by means of threshold or quantile values). Second, reactive

provisioning in response to workload anomalies relies only on a user-defined threshold parameter; this value may be hard to know in advance. Instead, our solution is potentially able to work with any type of workload (like bursty workload) since model parameters are estimated online. Finally, this work can suffer of limited applicability in highly transient workload conditions since the model is accurate only for systems in steady-state condition. Instead, our solution can also work with system characterized by transient behavior.

In [127], the authors present a resource control system that automatically adapts to dynamic workload changes to achieve application-level performance constraints. The proposed solution is based on feedback control theory, and, specifically, on a combination of an online model estimator and a two-layer MIMO resource controller. The model estimator captures the complex relationships between application-level performance and resource allocations, by modeling them with a MISO ARX model whose parameters are estimated online with the RLS algorithm. The resource controller, based on the LQR control design, allocates the right amount of multiple virtualized resources to achieve application performance constraints. The first layer of the resource controller consists of a set of adaptive feedback controllers that, by using the information gathered by model estimators, automatically determine the amount of resources necessary to achieve individual application performance. The second layer is comprised of a set of controllers that detect resource bottlenecks on shared physical machines and properly allocate multiple types of resources to individual application tiers. The work is evaluated through experimental testbeds, and results show that it can detect and mitigate CPU and disk I/O bottlenecks that occur over time and across multiple nodes by allocating each resource accordingly. Conceptually, this work is very similar to ours for what concerns the Application and Physical Machine Manager components. However there is one important difference. Since this work models input-output relationships by a MISO model (which relates tier-level resource allocations to the single application-level performance index), one may loose the possibility to dynamically capture bottlenecks at tier-level; instead, in our solution this possibility is preserved since we model such relationships with a MIMO model, which relates tier-level resource allocations with tier-level performance index.

In [33], the authors provide a control framework for managing cloud infras-

tructure that uses statistical machine learning to predict system performance for future configurations and workloads. The resource controller uses a statistical performance model of the application to dynamically adjust the resource allocation in response to changes in the workload. Such model is built by means of nonlinear regression techniques based on splines (to avoid to select in advance the shape of the nonlinear function) and LOESS regression [52, 53] (to capture nonlinear relationships between workload and application-level performance). The optimal control policy for the resource controller is found via a control policy simulator, which uses the Pegasus policy search algorithm (to compare by simulation different control policies), historical workload data (to predict spikes or usage patterns in the workload), and a local search heuristic (to find those optimal parameter values that minimize the total cost of running the application). The performance model is dynamically adapted to changes via change-point detection techniques based on statistical hypothesis testing. The control framework is evaluated by means of an experimental prototype running a Web 2.0 benchmark application driven by workload traces on Amazon's EC2 cloud. Preliminary results show that the proposed solution is able to effectively control the number of servers, even in the face of performance anomalies. Due to lack of details in the way each component of the framework is implemented, it is hard to provide an accurate comparison of this work with ours. For instance, there is no mention on how the resource controller is implemented, and hence we are unable to know if it is able to support multi-tier applications and how it determines resource allocations for each application tier. Also, we do not know if this solution is able to support virtualized resources and, if it does, how conflicting demands for a shared virtualized resource are arbitrated.

Finally, in [44], an optimal resource provisioning algorithm for medium- and long-term resource reservation plan is proposed for minimizing resource provisioning cost in cloud computing environments. The work is motivated by the fact that, in general, from the point of view of cloud consumer (like service providers), the cost of utilizing computing resources provisioned by a reservation plan is cheaper than that provisioned by an on-demand plan. The resource provisioning algorithm is formulated as a stochastic integer programming model with multi-stage recourse which considers multiple provisioning stages with resource demand and price uncertainties. The solution methods to solve such optimization problem are based

on deterministic equivalent formulation [30], Benders decomposition [29] and sample-average approximation [101] algorithms. Performance evaluation, carried out by means of numerical studies, shows that, with the proposed algorithm, cloud consumer can successfully minimize the total cost of resource provisioning in cloud systems. Conceptually, this work is similar to our optimal initial placement strategy and optimal migration controller, since both aim to provide an optimal medium- to long-term placement of VMs. However, there are two key differences. The first difference is in the type of mathematical programming model, since we use a mixed-integer nonlinear programming model. The second difference is in the perspective from which the optimization problem has been formulated; indeed, this work focuses on the cloud consumer point-of-view (because the objective function includes resource provisioning costs and demands) while our approach focuses on both the service and infrastructure provider point-of-views (because the objective function includes both resource utilization and power consumption).

10.1.2 Power-aware Resource Management

In this section, we present recent works aimed at reducing power consumption in computing systems, but that are not necessarily subjected to application performance constraints. It is important to note that, in these works, the achievement of application performance (when considered) is not the primary goal, which instead consists in minimizing the energy consumption induced by the computing systems.

In [123], the authors propose VirtualPower, an online power management which integrates multiple independent power management mechanisms and policies with virtualization technologies, by exploiting both hardware power scaling and software-based methods for controlling the power consumption of underlying platforms. The objectives of VirtualPower are (1) to support the isolated and independent operation assumed by guest VMs running on virtualized platforms and (2) to make it possible to control and globally coordinate the effects of the diverse power management policies applied by these VMs to virtualized resources.

To attain these goals, VirtualPower extends to guest VMs virtualized (“soft”) versions of the hardware power states for which power management policies are designed; these “soft” states create an abstraction that allows guest VMs (1) to

run their own power management methods, regardless of the underlying hardware power management support (like processor frequency scaling), and (2) to have a consistent view of hardware management capabilities, regardless of the underlying set of physical resources. Such "soft" power states are interpreted as "hints" to be passed to localized power management and/or global policies (e.g., VM migration), thus enabling a sort of coordination between VM-level and hardware-level power management system. VirtualPower captures these "soft" states by intercepting (inside the VM hypervisor) power state change events, issued by each independent VM-level power management system via privileged actions (e.g., via the virtual ACPI interface).

VirtualPower uses state changes requested by VMs as inputs to virtualization-level management policies and maps actual power management actions to rules provided by hardware vendors and/or system administrators. These rules are based on a rich set of underlying virtual power management mechanisms that provide a uniform basis for implementing management methods across heterogeneous hardware platforms. Such methods include both local policies, to efficiently use specific platforms and their power management capabilities, and global policies, that consider goals derived from entire applications running across multiple machines and/or derived from global constraints, such as cluster-level power budget. Experimental evaluation is based on a real implementation of VirtualPower with the Xen hypervisor. Results show that VirtulPower is able (1) to considerably improve the active power consumption of underlying computing platforms, (2) to perform QoS-aware power throttling, and (3) to exploit the consolidation capabilities inherent in modern multi-core platforms.

With respect to this work, our solution takes a different approach on power management. Specifically, we ignore any power state change directly issued by a guest VM and concentrate on power management at the underlying hosting virtualized resource. This choice is primarily motivated by the fact that (1) we consider full system power consumption and (2) power consumption induced by a specific VM could be in principle inferred by the related resource utilization (e.g., as shown in [142]). Nevertheless, the integration of this work in our framework can potentially improve the reduction of power consumption since it can bring information that is not always possible to obtain from the hypervisor (such as

idleness detection of the virtual CPU in a specific guest VM).

In [135], the authors present an integrated solution for peak and average power management that is able to coordinate different individual software and hardware approaches. The work leverages a mechanism to federate multiple power management solutions and builds on a control-theoretic approach to unify solutions for tracking, capping, and optimization problems, with minimal interfaces across controllers. Experimental evaluation is carried out by means of trace-driven simulation, with traces coming from real-world enterprises. Results demonstrate that the proposed solution is "correct" (i.e., no excessive power budget violations), stable (i.e., no large oscillations), and efficient (i.e., it is able to obtain good trade-offs between power and performance). With respect to this work, our approach do not take into consideration neither power budget nor the coordination of different power management policies. Power budget can be easily integrated in our Resource Manager component (and, in particular, in the Migration Manager), while the coordination of different power management policies can be considered as future work since it can potentially allow a finer reduction of overall power consumption.

Finally, in [165], the authors present pMapper, a power-aware application placement framework for heterogeneous virtualized computing systems. The central intelligence of pMapper lies inside the Arbitrator component which ensures consistency among soft, hard, and consolidation actions. The Arbitrator consists of a power-aware application placement strategy and algorithm. The application placement strategy is formulated as a cost minimization problem under performance constraints and computes the best VM placement, while the application placement algorithm is implemented as a dynamic placement controller which applies the solution of the aforesaid strategy. The authors evaluates three strategies: (1) mPP which is designed to only minimize power cost and uses a first-fit decreasing heuristic (a local search algorithm traditionally used to solve bin packing problems) to relate VMs to physical servers, (2) mPPH which essentially is identical to mPP but applies the first-fit decreasing heuristic incrementally in order to reduce migration costs by migrating as few VMs as possible, and (3) pMaP which uses mPPH and aims to find an allocation that minimizes the aggregated power and migration cost. The authors also proposes a variant of pMaP, named pMaP+, which is particularly suitable for high load intensities. Performance evaluation is carried

out on a real prototype, by interfacing pMapper with existing commercial workload and power managers. Experiments compare the proposed application placement strategies with others that are traditionally adopted, namely Load Balanced (where the VM placement is done in a way that the load is balanced across all the physical servers) and Static (where the VM placement that minimizes the power is computed by taking into account long-term history). Results show that pMapper with pMaP and pMaP+ (and, partially, also with mPPH) strategies is always able to outperform both Load Balanced and Static strategies. The proposed solution can be used to complement our initial and incremental VM placement strategies. In particular, we can use either mPPH, pMaP or pMaP+ to provide our solution an alternative incremental VM placement strategy.

10.1.3 Integrated Power-aware and Performance-aware Resource Management

In order to achieve a good trade-off between energy consumption reduction and SLA achievement, the resource manager of a cloud system should take into consideration both the energy consumed by the system and the performance of hosted services. In the rest of this section, we present some of recent advances on this topic and we compare them with our proposed solution.

In [158], a two-level resource manager is proposed, which combines a utility-based approach to constraint programming for dynamic VM provisioning and placement. Specifically, the dynamic VM provisioning process aims at maximizing a global utility function which captures both application performance and energy consumption; the utility maximization problem is formulated as a constraint satisfaction problem (and, specifically, as a multi-choice knapsack problem), where constraints represent bounds over resource capacities. The dynamic VM placement process aims at consolidating VMs on the minimum number of physical servers through VM live migration so that idle servers can be turned off to save energy. It tries to achieve this goal by taking as input the solution computed by the VM provisioning problem and formulating a constraint satisfaction problem (with constraints on resource capacities) for maximizing the number of idle physical servers that can be turned off. Performance evaluation is done through experiments on a

real testbed. Results show the flexibility of the framework in reaching different trade-offs between application performance and energy consumption, and in arbitrating resource allocations in case of contention. This work differs from ours mainly in that (1) the architecture of the framework is not decentralized since both the VM provisioning and placement problems need a global view of the computing system state, (2) performance metrics are estimated offline by a queueing network model which, thus makes this work unable to adapt to time-varying workloads. In contrast, our approach (1) is based on a decentralized and time-hierarchical architecture, and (2) is able to adapt to time-varying workloads by using online model estimation.

In [25], a time-hierarchical and decentralized resource allocation framework for multi-tier applications is presented, which is able to dynamically place VMs and to turn on or off physical machines when needed, while preserving application SLAs. The proposed solution addresses the resource management problem as a unifying framework, by exploiting as actuation mechanisms both the allocation of VMs to servers, load balancing, capacity allocation, server power state tuning, and DVFS. The resource management problem is formulated as a mixed integer nonlinear programming problem, where the objective function includes revenues and penalties incurred on the achieved level of performance and the energy costs associated with the use of physical servers; since the problem is NP-hard, it is solved by means of a local search heuristic procedure. To validate its effectiveness, the proposed model is compared to state-of-the-art resource management techniques. The evaluation is based both on simulation and on real experiments performed in a prototype environment. Synthetic as well as realistic workloads and a number of different scenarios of interest are considered. Results show that the proposed solution is able to yield significant revenue gains for the provider when compared to alternative methods. Moreover, it is robust to service time and workload variations. Besides of the conceptual similarity with our framework, this work mainly differs from ours for the approach taken to minimize power consumption and SLO violations, which, in this case, consists in a utility-based approach combined to mixed integer nonlinear programming (while ours combines mixed-integer nonlinear programming with control theoretic techniques).

In [70], a combined predictive and reactive provisioning technique is proposed,

where the predictive component estimates the needed fraction of capacity according to historical workload traces analysis, while the reactive component is used to handle workload excess with respect to the analyzed one. The proposed approach works at different time scales; in particular, predictive control provisions the estimated "base" (regular) workload at coarse time scales (e.g., hours), while reactive control handles any excess demand at finer time scales (e.g., minutes). Workload forecasting is done by means of periodicity analysis (to find the periodogram of historical data) and workload discretization based on dynamic programming (to find disjoint time intervals, representing specific behavioral patterns, that minimizes the deviation of represented workload demand from the actual one). Predictive control is driven by a queueing performance model which is used to determine how much capacity has to be allocated (with respect to the forecast demand) to ensure that SLA requirements are met, and, at the same time, to avoid to consume too much power. Reactive control is based on a feedback approach with fixed monitoring interval. Performance evaluation is based on both trace-based analysis (with traces coming from real systems) and on experiments performed on a real testbed. Results show that the proposed approach is able to meet SLA requirements, is more power efficient than existing approaches, and avoids unnecessary provisioning changes. Unlike this work, our solution, by means of online system identification, is potentially able to work with any type of workload without any prior knowledge. Furthermore, our solution is able to work with multi-tier applications and can leverage virtualization technologies for resource provisioning.

In [167], a two-levels control architecture for coordinating power and performance management in virtualized computing systems is proposed. The outermost layer includes cluster-level power control loops which change hardware power states of each server in the cluster (by means of DVFS) with no regard to the application-level performance. The design of power controllers is based on the MPC theory, a control design method which is able to work with MIMO control problems with constraints (but that, in general, provides suboptimal control decisions). The innermost layer consists of a series of performance control loops, one for every VM, each of which trying to achieve the desired application performance (in this case, the application-level response time), even when the system model varies significantly due to the impact of power control actions. The design of

performance controllers is based on offline system identification (by means of the least-squares method) and on PID control. Empirical results on a real testbed show that the proposed solution can simultaneously provide effective control on both application-level performance and underlying power consumption. Unlike our framework, the achievement of performance targets is always subjected to the reduction of power consumption; conversely, in our framework, the reduction of power consumption is integrated with the achievement of application performance targets. Moreover, our approach is able to work with multi-tier applications and to leverage virtualization technologies for driving server consolidation and reducing power consumption. Furthermore, our work uses online system identification, which can potentially cover a wider range of time-varying workloads.

In [106], an online resource provisioning framework is proposed for combined power and performance management in a virtualized computing environment serving session-based workloads. The resource management problem is formulated as a sequential multi-objective optimization problem under uncertainty and is solved using limited lookahead control, which is a form of MPC. The approach accounts for the switching and opportunity costs incurred when provisioning VMs or turning on or off physical servers, and explicitly encodes the risk of provisioning resources in an uncertain and dynamic operating environment. Performance evaluation is done through experiments on a real testbed. Results indicate that the proposed solution is able, on average, to significantly save power consumption costs over a given period, when compared to a system operating without dynamic control. The key difference between this work and our approach is in the type of the architectural design adopted. As a matter of fact, this work uses a centralized architecture, while our framework has a decentralized architecture which makes it potentially more scalable. It is important to note that authors also show, via trace-driven simulation, that the computational burden caused by the centralized architecture can be reduced by using a feed-forward neural network (trained to learn tendencies of the controller in terms of decision making) as a mean to obtain run-time approximate control decisions; with this adjustment, authors demonstrate that the proposed solution is potentially able to scale to large systems while still preserving good performance, similar to the one obtained with the baseline implementation (i.e., the one without the neural network). Unfortunately, the lack of details about this

modification makes us unable to perform any sort of comparison.

Finally, in [170], a two-layers hierarchical control approach is used for tackling the resource provisioning problem for multi-tier Web applications. Specifically, in the outermost layer, an application controller (one for each multi-tier application) determines the total budget of CPU resources that are required for the application to meet SLA requirements (expressed in terms of end-to-end response time threshold) as the workload varies. The application controller uses an ARX model to represent relationships between CPU shares and end-to-end application response time, with parameters estimated offline using a least-squares method; moreover, the controller is designed as a Proportional-Integral controller, with parameters estimated offline by means of the Root-Locus method. In the innermost layer, an optimal resource partition controller allocates the per application total CPU resource (provided by the outermost level) to the different application tiers, given a specific end-to-end application response time threshold. The optimal (minimal) CPU resource entitlement for each tier is computed by relating the end-to-end application response time to CPU utilization; such relationship is found by modeling the multi-tier application as a tandem queueing network, where each tier is represented by an M/G/1/PS queueing station. Performance evaluation is done through experiments, by running a real application subjected to three different workload models (open, closed, and semi-open). Results indicate that, with respect to previous works, (1) fewer resources are provisioned to the applications to achieve the same performance, and (2) the proposed approach is robust enough to address various types of workloads with time-varying resource demand without reconfiguration. Even though this work shares some architectural similarity with our framework, there are two important differences; the first difference regards the way ARX parameters are computed (i.e., through offline system identification, in this work, and by means of online identification, in our solution), while the second one is about the way CPU shares are assigned to application tiers (which is not well suited for situations where tiers of different applications are hosted on the same physical machine).

10.2 Cloud Computing System Simulators

In this section, we present some recently proposed simulation tool suitable to cloud computing systems, and compare them with DESEC, the simulator we developed for this thesis (see Chapter 7).

In [39, 41], the authors propose CloudSim, an extensible simulation toolkit written in Java, that enables modeling and simulation of cloud computing systems and application provisioning environments. The CloudSim toolkit supports modeling and simulation of different cloud computing system components, including data centers, VMs, service brokers, resource allocation and provisioning policies, and also network connections among the simulated system components. It implements generic application provisioning techniques that can be extended with ease and limited effort. It also supports modeling and simulation of cloud computing environments consisting of inter-networked (federated) clouds. From the point-of-view of users, CloudSim provide following advantages: (1) availability of a virtualization engine (that helps in the creation and management of multiple, independent, and co-hosted virtualized services on a data center node), and (2) flexibility (to switch between space-shared and time-shared allocation of processing cores to virtualized services). Instead, from the point-of-view of developers, CloudSim brings following benefits: (1) extensibility (obtained by exposing customer interfaces for implementing policies and provisioning techniques to allocate VMs in – possibly inter-networked – cloud computing environments), and (2) time effectiveness (since developers can model and test the performance of their application services in heterogeneous cloud environments with little programming and deployment effort). The usefulness of CloudSim is demonstrated by a case study involving dynamic provisioning of application services in a hybrid federated clouds environment. There are several differences between CloudSim and DESEC. For instance, on one hand, CloudSim does not provide built-in facilities for simulating cloud system at the service provider level, but only ones at the infrastructure provider level; conversely, DESEC provides several abstractions (like multi-tier applications) and facilities (like workload models, and application performance and simulation models). Moreover, CloudSim does not provide any output analysis method since the simulation consists in a single long run; instead,

DESEC provides different output analysis methods (e.g., independent replications and batch means), several type of statistical estimators (e.g., average and order statistics) and various simulation terminating condition strategies (e.g., relative precision of output statistic confidence intervals). Furthermore, CloudSim is an ad-hoc simulator for cloud computing systems; instead, the core of DESEC is a general-purpose discrete-event simulator which can be used and extended separately. On the other hand, DESEC still lacks of important features provided by CloudSim, like the possibility of modeling and simulating federated clouds.

In [102], the authors present GreenCloud, a simulation environment for energy-aware cloud computing data centers written in C++ and TCL. Along with the workload distribution, GreenCloud is designed to capture details of the energy consumed by data center components (e.g., servers, switches, and links) as well as packet-level communication patterns in realistic setups. GreenCloud is developed as an extension of a packet-level discrete-event network simulator ns-2. The simulation results obtained for two-tier, three-tier, and three-tier high-speed data center architectures demonstrate the effectiveness of the simulator in utilizing power management schema, such as voltage scaling, frequency scaling, and dynamic shutdown that are applied to the computing and networking components. GreenCloud and our DESEC simulators have a different target. Specifically, GreenCloud is particularly indicated to advanced low-level energy-aware studies of cloud computing systems in realistic setups, while DESEC is better suited to study resource management strategies for cloud computing systems.

In [80], the authors present the Green Data Center Simulator (GDCSim), a simulation tool that unifies existing techniques to green data center management and allows holistic physical data center design and analysis before deployment. Specifically, GDCSim allows to analyze data center energy efficiency by studying and testing: (1) different data center geometries, (2) workload characteristics, (3) platform power management schemes, (4) scheduling algorithms, and (5) data center configurations. Features of GDCSim include: (1) automated processing (which allows interfacing different modules together and does not require user intervention once the simulation has started), (2) online analysis capability (which allows real time simulation of management decisions based on changes in the physical environment in the data center), (3) iterative design analysis (which enables

design time testing and analysis of different configurations before deployment), (4) thermal analysis capability (which characterizes the thermal effects within the data center room), (5) workload and power management (which enables workload scheduling and controlling the power modes of servers for higher data center efficiencies), and (6) consideration of cyber-physical interdependency (which enables feedback of information on temperature and air flow patterns in the data center to the management algorithms and the closed loop operation of the servers and cooling units to achieve energy efficient operation). The functionality of GDCSim is demonstrated by two case studies with two different data center layout and workload types. GDCSim and our DESEC simulator have different targets. In particular, DESEC can be used to study different power- and performance-aware resource management schemes, while GDCSim is especially indicated to study the design of green data centers.

Finally, in [85], a large-scale computing infrastructure simulator is proposed, in order to evaluate the impact of "what-if" scenarios on performance, availability and reliability of the system. The main goal is to provide data center operators a tool that allows understanding and predicting the consequences of the deployment of new network topologies, hardware configurations or software applications in a global computing infrastructure, without affecting hosted services. The simulator is implemented as a multi-agent system, using a multi-layered approach formed by components and operations. Components are stateful autonomous agents that represent computing infrastructure elements at various granularities (e.g., servers and network device) and interact with each other through message exchange. Operations define interactions between components (e.g., login to a server). The simulator takes as input the workload specification of each application, the resources allocated by individual user requests, the network topology of the organization, the hardware configuration deployed in each data center and details on background processes. Using this information, the simulator produces (by means of queueing network models modeling the behaviour of low-level devices) estimates of the response time for each user request, along with measurements of the resource allocation and network utilization, so as to facilitate optimization goals for data center operators. The simulator is validated using data gathered from different applications running together on a downscaled version of a real enterprise

data center. Moreover, the usefulness of the simulator is demonstrated by analyzing a case study of the same enterprise aiming to reduce costs by cutting down the number of data centers, while keeping the same quality of service. The simulator proposed in this work has the advantage to potentially provide a very low-level of abstraction, thus allowing a very precise and realistic simulation of a real system (possibly at the expense of extensibility and usability). Conversely, our DESEC simulator works at a higher level of abstraction, by replacing many low-level details with few high-level models, thus enabling extensibility and usability.

Chapter 11

Conclusions and Future Work

In this chapter, we provide a summary of the work presented in this thesis and then we present possible future works.

Cloud computing is an emerging computing paradigm which is gaining popularity in IT industry for its appealing property of considering "Everything as a Service".

In this thesis, we tackled the problem of efficiency managing computing resources of a cloud infrastructure under service performance constraints.

The goal of a cloud infrastructure provider is to maximize its profit by minimizing the amount of violations of Quality-of-Service levels (SLOs) agreed with service providers, and, at the same time, by lowering infrastructure costs (TCO). Among these costs, the energy consumption induced by the cloud infrastructure, for running cloud services, plays a primary role.

Unfortunately, the minimization of SLO violations and, at the same time, the reduction of energy consumption is a conflicting and challenging problem since, in general:

- the higher is the amount of resources provisioned to hosted services, the better are their performance, but the higher is the energy consumption spent to run such services, and, conversely
- the lower is the amount of resources provisioned to hosted service, the lower is the energy consumed by the cloud infrastructure, but the higher is the chance to incur in SLO violations.

In this thesis, we provided two contributions in order to pursue this goal:

- we proposed a framework to automatically manage computing resources of cloud infrastructures in order to simultaneously preserve SLO constraints and to reduce as much as possible the amount of energy used for providing services, and evaluated its performance by means of an exhaustive simulation study, and

- we implemented an ad-hoc discrete-event simulator in C++, which we used to evaluate the proposed framework.

Our resource management framework consists of a decentralized and time-hierarchical architecture, where different types of components, namely the Application Manager, the Physical Machine Manager and the Migration Manager, operate independently from each other and at different time scales.

To design the resource management framework, we used techniques based on adaptive and optimal feedback control theory and mathematical optimization.

By means of simulation, we showed that, compared to traditional static approaches, our work is able to dynamically adapt to time-varying workloads (without any prior knowledge) and, at the same time, to reduce both SLO violations and energy consumption.

In particular, results showed that our approach was almost always able to outperform traditional approaches based on static resource allocation, because of the dynamic modulation of VMs resource shares according to current working conditions. The only static approach that sometimes obtained better performance than ours, is STATIC-SLO, which however is an unrealistic approach, based on resource over-provisioning, that can be used only when enough prior knowledge about the dynamics of the services (hosted by the cloud infrastructure) is known.

We also evaluated the effectiveness of using VM migration by means of the Migration Manager. From the results, we observed that the use of the Migration Manager component (and hence of VM migration) is really effective especially in the presence of high-intensity and bursty workloads, which may cause several resource demand conflicts among the various VMs deployed on the cloud infrastructure(e.g., see the S-MMPP scenario).

A possible limitation of our framework is related to the number of configuration parameters that needs to be manually specified. Indeed, for most of them, it is necessary to perform trial-and-error experiments in order to find those values that better suit to the system under study. It is worth noting that, in part, this problem has already been solved by means of specific techniques, like online system identification. However, still there are different components that needs to be tuned by means of offline experiments (e.g, for the Application Manager, the weighting matrices of the LQ control design have to be specified).

Regarding possible future works, there are some interesting activity that can be carried on.

First, we would like to improve the Migration Manager, for instance by studying a different objective function or evaluating different predictors, others than the EWMA filters, for estimating resource share demands and utilizations.

In addition, we want to investigate different types of control design to use for the Application Manager. In particular, we are interested in the evaluation of *minimum-variance controllers* [137, 51], in order to reduce possible oscillations around the control set-point, and *Model Predictive Control* design [42, 115], to explicitly constraints control inputs (i.e., resource shares) during the computation of the optimal control sequence.

Then, we want to simplify our framework in order to reduce the number of configuration parameters that needs to be manually specified.

Also, we plan to extend our framework to consider federation of cloud infrastructures. Cloud federations are useful to realize, for instance, the practice known as *cloud bursting*, whereby a service bursts into a different cloud infrastructure to use its physical resources (e.g., either because of economical motivation, the capacity of the original infrastructure is reached, or the computing capacity spikes). To this end, we plan to introduce a new Resource Negotiator component, which is responsible to negotiate physical resources among different cloud infrastructures.

Another possible future work is the integration of our framework with existing green (power-aware) networking strategies; as a matter of fact, the lack of coordination among the computing and networking "green" strategies could bring in practice to suboptimal results because of conflicting decisions.

Finally, we plan to integrate our framework into a real testbed.

Part V

Appendices

Appendix A

z-transform

The *z-transform* provides a way to represent a discrete-time signal as polynomial in z.

In this chapter, we provide an overview of basic notions and properties of the z-transform. For more information, the interested reader can refer to several books on systems and control theory, like [69, 84, 146].

In the rest of this chapter, we denote with $\{x(k)\}$, for $k \in S$, a real-valued sequence indexed by a countable set S (typically, $S = \mathbb{Z}$ or $S = \mathbb{N}$), while we denote with $x(\cdot)$ the function (signal) generating such sequence.

A.1 Basic Definitions

Definition A.1.1 (Bilateral z-transform)**.** The *bilateral z-transform* $X(z)$ of a real-valued sequence time-domain signal $x(\cdot)$ is defined as:

$$\begin{aligned} X(z) &\triangleq \mathscr{Z}\{x(k)\} \\ &\triangleq \sum_{k\in\mathbb{Z}} x(k) z^{-k}, \quad z \in \mathbb{C} \end{aligned} \tag{A.1}$$

Definition A.1.2 (Unilateral z-transform)**.** The *unilateral z-transform* $X(z)$ of a

generic discrete-time signal $x(\cdot)$ is defined as:

$$\begin{aligned} X(z) &\triangleq \mathscr{Z}\{x(k)\} \\ &\triangleq \sum_{k\in\mathbb{N}} x(k)z^{-k}, \quad z\in\mathbb{C} \end{aligned} \tag{A.2}$$

If the rate of increase in the terms of sequence $\{x(k)\}$ is no greater than that of some geometric series as k approaches infinity, then $\{x(k)\}$ is said to be of *exponential order*. In this case, there exists a real number r, called the *radius of convergence*, such that the z-transform $X(z)$ of $\{x(k)\}$ converges for $|z| > r$. If r is finite, the signal $x(\cdot)$ is called *z-transformable*.

Since the z-transform is an infinite power series, it exists only for those values of the variable z for which the series converges to a finite sum. The *region of convergence* (ROC) of $X(z)$ is the set of all the values of z for which $X(z)$ attains a finite computable value.

Definition A.1.3 (Region of Convergence)**.** The *region of convergence* (ROC) is the set of points in the complex plane for which the z-transform summation converges; that is:

$$\text{ROC} \triangleq \left\{ z : \left| \sum_{n=-\infty}^{\infty} x(k)z^{-k} \right| < \infty \right\} \tag{A.3}$$

The *inverse z-transform* represents a time-domain sequence of a z-transform. The formal definition of the inverse z-transform is stated in terms of a counter integral in the z-plane.

Definition A.1.4 (Inverse z-transform)**.** The *inverse z-transform* is defined as:

$$\begin{aligned} x(k) &\triangleq \mathscr{Z}^{-1}\{X(z)\} \\ &\triangleq \frac{1}{2\pi j} \oint_C X(z)z^{k-1}\,dz \end{aligned} \tag{A.4}$$

where C is a counterclockwise closed path encircling the origin and entirely in the ROC.

The contour (or path) C must encircle all of the poles of $X(z)$. A special case of this contour integral occurs when C is the unit circle. In this case, the inverse

z-transform simplifies to the inverse discrete-time *Fourier transform*:

$$x(k) = \frac{1}{2\pi}\int_{-\pi}^{+\pi} X(e^{j\omega})e^{j\omega k}d\omega \tag{A.5}$$

There are other more practical way to obtain the inverse z-transform. A common method, known as the *inspection method* corresponds to utilizing the fact that simple z-transforms and the sequences that generate them are recognizable by inspection. An extension of this method, known as the *partial expansion method*, consists of expanding a more complicated z-transform in a partial fraction expansion and then recognizing the sequences that correspond to the individual terms.

A.2 Basic Properties

Definition A.2.1 (Linearity)**.** Given two signals $x_1(\cdot)$ and $x_2(\cdot)$, the *linearity* property implies that for any linear combination of $x_1(\cdot)$ and $x_2(\cdot)$ we have:

$$\mathscr{Z}\{a_1x_1(k)+a_2x_2(k)\} \triangleq a_1X_1(z)+a_2X_2(z), \quad a_1,a_2 \in \mathbb{R} \tag{A.6}$$

where $X_i(Z) \triangleq \mathscr{Z}\{x_i(k)\}$, for $i = 1,2$.

The Eq. (A.6) is known as the *superposition principle*.

Definition A.2.2 (Time Shifting)**.** Given a signal $x(\cdot)$, the *time shifting* property implies that:

$$\mathscr{Z}\{x(k-m)\} \triangleq z^{-m}\mathscr{Z}\{x(k)\} \tag{A.7}$$

Definition A.2.3 (Convolution)**.** Given two signals $x_1(\cdot)$ and $x_2(\cdot)$, the *convolution* property states that:

$$\mathscr{Z}\Big\{\big(x_1(k)\star x_2(k)\big)\Big\} \triangleq X_1(z)X_2(z) \tag{A.8}$$

where $X_i(Z) \triangleq \mathscr{Z}\{x_i(k)\}$, for $i = 1,2$.

that is the convolution of two signals in the time domain is equivalent to multiplication of their z-transforms and vice versa.

Appendix B

Mixed-Integer Nonlinear Programming

Mixed-Integer NonLinear Programs (MINLPs) are optimization problems where some of the variables are constrained to take integer values and the objective and feasible region of the problem are described by nonlinear function [67, 110].

The general form of a MINLP is

$$\begin{aligned} \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g_j(x,y) \leq 0, \qquad j \in J \\ & x \in X, \qquad X \subseteq \mathbb{R}^n \\ & y \in Y, \qquad Y \subseteq \mathbb{Z}^m \end{aligned} \tag{B.1}$$

The function $f(x,y) : X \times Y \to \mathbb{R}$ is a nonlinear objective function and $g_j(x,y) : X \times Y \to \mathbb{R}$ a nonlinear constraint function. Both functions are sufficiently smooth. The variables x and y are the decision variables, where y is required to be integer valued. X and Y are bounding-box-type restrictions on the variables (e.g., $\alpha_i \leq x_i \leq \beta_i$). If both f and g_j, for every $j \in J$, are convex functions [1], the problem Eq. (B.1) is called a *convex* MINLP problem.

[1] A set C is said to be *convex* if, for all x and y in C and all t in the interval $[0,1]$, the point $(1-t)x+ty$ is in C. In other words, every point on the line segment connecting x and y is in C. A real-valued function $f : X \to \mathbb{R}$ defined on a convex set X in a vector space is called *convex* if, for every two points x and y in X and every $t \in [0,1]$, it results $f(tx+(1-t)y) \leq tf(x)+(1-t)f(y)$.

MINLP problems are known to be NP-hard, and some of them are even undecidable [71, 92]. The class of convex MINLP is NP-hard as well. Nevertheless, many efficient solution techniques for convex MINLP have been proposed in the scientific literature [36].

Several MINLP methods have been proposed over the past few decades (e.g., see [22, 56, 59, 72, 79, 100, 111, 148, 151]).

The basic concept underlying algorithms for solving MINLP problems is to generate and refine bounds on its optimal solution value. Lower bounds are generated by solving a relaxation of the MINLP problem (usually referred to as RMINLP), and upper bounds are provided by the value of a feasible solution to the MINLP problem. Algorithms differ in the manner in which these bounds are generated and the sequence of subproblems that are solved to generate these bounds.

Bibliography

[1] Amazon elastic compute cloud (EC2). Available: http://aws.amazon.com/ec2.

[2] Amazon simple storage service (S3). Available: http://aws.amazon.com/s3.

[3] Amazon Web Services. Available: http://aws.amazon.com.

[4] Boost C++ Libraries. Available: http:www.boost.org.

[5] General algebraic modeling system (GAMS). Available: http://www.gams.com.

[6] Google AppEngine: Run your web apps on Google's infrastructure. Available: http://code.google.com/appengine/.

[7] Google Apps. Available: http://www.google.com/apps.

[8] IBM Smart Cloud. Available: http://www.ibm.com/cloud-computing.

[9] JTC1/SC22/WG21 - The C++ Standards Committee. http://www.open-std.org/jtc1/sc22/wg21/.

[10] Linear algebra package (lapack). Available: http://www.netlib.org/lapack/.

[11] Microsoft connected service framework (CSF). Available: http://www.microsoft.com/serviceproviders/solutions/connectedservicesframework.mspx.

[12] Microsoft Windows Azure platform. Available: http://www.microsoft.com/windowsazure.

[13] Salesforce.com. Available: http://www.salesforce.com.

[14] Simple branch & bound (sbb). Available: http://www.gams.com/dd/docs/solvers/sbb.pdf. User Manual.

[15] SPECpower_ssj2008 benchmark. Available: http://www.spec.org/power_ssj2008.

[16] Hal Abelson, editor. *Architects of the Information Society, Thirty-Five Years of the Laboratory for Computer Science at MIT*. MIT Press, 1999.

[17] R. Adair, R. U. Bayles, L. W. Comcau, and R. J. Creasy. A virtual machine system for the 360/40. Technical Report C320-2007, IBM Cambridge Scietific Center, May 1966.

[18] Kathleen T. Alligood, Tim D. Sauer, and James A. Yorke. *CHAOS: An Introduction to Dynamical Systems*. Springer-Verlag, New York, 1997.

[19] Daniel Alpay, Joseph A. Ball, and Yossi Peretz. System theory, operator models and scattering: the time-varying case. *Journal of Operator Theory*, 47(2):245–286, 2002.

[20] AMD. AMD64 virtualization codenamed "Pacifica" technology: Secure virtual machine architecture reference manual. Manual 33047 – Rev. 3.01, AMD, May 2005.

[21] Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, 3rd edition.

[22] I.P. Androulakis, C.D. Maranas, and C.A. Floudas. *alpha*BB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995.

[23] Kiam Heong Ang, Gregory Chong, and Yun Li. PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005.

[24] Panos J. Antsaklis and Anthony N. Michel. *Linear Systems*. Birhaüser, Boston, 2006.

[25] Danilo Ardagna, Barbara Panicucci, Marco Trubian, and Li Zhang. Energy-aware autonomic resource allocation in multi-tier virtualized environments. *IEEE Transactions on Services Computing*, 99(PrePrints), 2010.

[26] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and MateiZaharia. Above the Clouds: A Berkeley view of Cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[27] Jr. Arthur E. Bryson and Yu-Chi Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Taylor & Francis, revised edition, 1975.

[28] Jerry Banks, John S. Carson, II, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, 5th edition, 2010.

[29] Jacques F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[30] John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer Science+Business Media, LLC, 2nd edition, 2011.

[31] S. Bittanti, P. Bolzern, and M. Campi. Exponential convergence of a modified directional forgetting identification algorithm. *System Control Letter*, 14:131–137, 1990.

[32] Peter Bloomfield. *Fourier analysis of time series: An introduction*. Wiley-Interscience, 2nd edition, 2000.

[33] Peter Bodík, Rean Griffith, Charles Sutton Armando Fox, Michael Jordan, and David Patterson. Statistical machine learning makes automatic control

practical for Internet datacenters. In *Proc. of the 2009 USENIX Conf. on Hot Topics in Cloud Computing (HotCloud'09)*, 2009.

[34] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. Working Group Note NOTE-ws-arch-20040211, W3C Web Services Activity, Feb 2004.

[35] George E.P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, 3rd edition, 1994.

[36] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[37] Micheal R. Bussiek and Arne S. Drud. SBB: A new solver for mixed integer nonlinear programming. In *Proc. of the 2001 Operation Research Conference (OR'01)*, 2001.

[38] Rajkumar Buyya, James Broberg, and Andrzej Goscinski, editors. *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2011.

[39] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *Proc. of the International Conference on High Performance Computing Simulation (HPCS'09)*, pages 1–11, Leipzig, Germany, 2009.

[40] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[41] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A.F. De Rose, and Rajkumar Buyya. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[42] Eduardo F. Camacho and Carlos Bordons Alba. *Model Predictive Control*. Springer, 2nd edition, 2004.

[43] Philip D. Cha, James J. Rosenberg, and Clive L. Dym. *Fundamentals of modeling and analyzing engineering systems*. Cambridge University Press, 2000.

[44] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. Optimization of resource provisioning cost in cloud computing. *IEEE Transactions On Services Computing*, 99(PrePrints), 2011.

[45] Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM Journal on Computing*, 33(4):837–851, 2004.

[46] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, 3rd edition, 1999.

[47] Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana. Web Services Description Language (WSDL) version 2.0 part 2: Adjuncts. Recommendation REC-wsdl20-adjuncts-20070626, W3C Web Services Activity, Jun 2007.

[48] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Services Description Language (WSDL) version 2.0 part 1: Core language. Recommendation REC-wsdl20-20070626, W3C Web Services Activity, Jun 2007.

[49] Y.S. Chow and Herbert Robbins. On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *Annuals of Mathematical Statistics*, 36(2):457–462, 1965.

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

[51] D.W. Clarke and R. Hastings-James. Design of digital controllers for randomly disturbed systems. In *Proc. of the Institution of Electrical Engineers*, pages 1503–1506, Oct 1971.

[52] William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

[53] William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[54] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[55] R. J. Creasy. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490, Sep 1981.

[56] R.J. Dakin. A tree search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.

[57] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Prentice Hall, 12th edition, 2010.

[58] Arne S. Drud. CONOPT: A large-scale GRG code. *Journal on Computing*, 6(2):207–216, 1994.

[59] Marco Duran and Ignacio Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339, 1986.

[60] ENERGY STAR Program. Report to congress on server and data center energy efficiency. Technical report, U.S. EPA, Aug 2007.

[61] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2005.

[62] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proc. of the* 34^{th} *International Symposium on Computer Architecture (ISCA'07)*, pages 13–23, 2007.

[63] R. Figueiredo, P. A. Dinda, and J. Fortes. Resource virtualization renaissance. In *Proc. of the IEEE Internet Computing*, volume 38, May 2005.

[64] Władysław Findeisen, F.N. Bailey, M. Bryds, K. Malinowski, P. Tatjewski, and A. Wozniak. *Control and Coordination in Hierarchical Systems*. John Wiley & Sons, Ltd, 1980.

[65] Wolfgang Fischer and Kathleen Meier-Hellstern. The Markov-modulated Poisson Process (MMPP) cookbook. *Performance Evaluation*, 18(2):149–171, 1993.

[66] R.A. Fisher. On an absolute criterion for fitting frequency curves. *Messenger of Mathematics*, 41:155–160, 1912.

[67] Christodoulos A. Floudas. *Nonlinear and mixed-integer optimization: Fundamentals and applications*. Oxford University Press, 1995.

[68] Ian Foster and Carl Kesselman. *The Grid: Blueprint for e New Computing Infrastructure*. Morgan Kaufmann, 1999.

[69] Gene F. Franklin, J. David Powell, and Michael Workman. *Digital Control of Dynamic Systems*. Addison-Wesley Longman, Inc., 3rd edition, 1998.

[70] Anshul Gandhi, Yuan Chen, Daniel Gmach, Martin Arlitt, and Manish Marwah. Minimizing data center SLA violations and power consumption via hybrid resource provisioning. In *Proc. of the* 2^{nd} *International Green Computing Conference (IGCC'10)*, 2010.

[71] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[72] A.M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

[73] Philipt E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal of Optimization*, 12(4):979–1006, 2002.

[74] Robert P. Goldberg. Survey of virtual machine research. *IEEE Computer Magazine*, 7(6), 1974.

[75] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[76] Włodzimierz Greblicki and Mirosław Pawlak. *Nonparametric System Identification*. Cambridge University Press, 2008.

[77] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP version 1.2 part 1: Messaging framework (second edition). Recommendation REC-soap12-part1-20070427, W3C Web Services Activity, Apr 2007.

[78] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP version 1.2 part 2: Adjuncts (second edition). Recommendation REC-soap12-part2-20070427, W3C Web Services Activity, Apr 2007.

[79] Omprakash K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[80] Sandeep K.S. Gupta, Rose Robin Gilbert, Ayan Banerjee, Zahra Abbasi, Tridib Mukherjee, and Georgios Varsamopoulos. GDCSim - an integrated tool chain for analyzing green data center physical design and resource management techniques. In *Proc. of the* 2^{nd} *International Green Computing Conference (IGCC'11)*, Orlando, FL, USA, Jul 2011.

[81] T. Hägglund. Recursive estimation of slowly time-varying parameters. In *Proc. of the Symposium on Identification and System Parameter Estimation*, pages 1137–1142, 1985.

[82] Brian Hayes. Cloud computing. *Communications of the ACM*, 51(7):9–11, 2008.

[83] Simon Haykin, editor. *Advances in Spectrum Analysis and Array Processing (vol. I)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[84] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[85] Sergio Herrero-Lopez, John R. Williams, and Abel Sanchez. Large-scale simulator for global data infrastructure optimization. In *In Proc. of the IEEE International Conference on Cluster Computing (CLUSTER'11)*, pages 54–64, Austin, Texas, USA, 2011. IEEE Computer Society.

[86] João P. Hesphana. *Linear Systems Theory*. Princeton Press, 2009.

[87] G. E. Hoernes and H. Hellerman. An experimental 360/40 for time-sharing. *Datamation*, 14(4):39–42, Apr 1958.

[88] Adrian A. Hopgood. *Intelligent Systems for Engineers and Scientists*. CRC Press, 2nd edition, 2001.

[89] Isaac M. Horowitz. *Synthesis of Feedback Systems*. Academic Press, 1963.

[90] IBM. An architectural blueprint for autonomic computing (fourth edition). White paper, IBM, Jun 2006.

[91] Enso Ikonen and Kaddour Najim. *Advanced Process Identification and Control*. Marcel Dekker, Inc., 2002.

[92] R.G. Jeroslow. There cannot be any algorithm for integer programming with quadratic constraints. *Operations Research*, 21(1):221–224, 1973.

[93] Michael A. Johnson and Mohammad H. Moradi, editors. *PID Control: New Identification and Design Methods*. Springer-Verlag, 2005.

[94] Rudolf E. Kalman. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial and Applied Mathematics*, 1(2):152–192, 1963.

[95] E.W. Kamen, P.P. Khargonekar, and K.R. Poolla. A transfer-function approach to linear time-varying discrete-time systems. *SIAM Journal of Control and Optimization*, 23(4):550–565, 1985.

[96] Christos Karamanolis, Magnus Karlsson, and Xiaoyun Zhu. Designing controllable computer systems. In *Proc. of the* 10^{th} *USENIX Conference on Hot Topics in Operating Systems (HotOS'05)*, pages 1–6, 2005.

[97] Magnus Karlsson, Christos Karamanolis, and Xiaoyun Zhu. Triage: Performance differentiation for storage systems using adaptive control. *Transactions on Storage*, 1(4):457–480, 2005.

[98] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[99] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Prentice Hall, 1970.

[100] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[101] Anton J. Kleywegt, Alexander Shapiro, and Tito Homem de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

[102] Dzmitry Kliazovich, Pascal Bouvry, Yury Audzevich, and Samee Ullah Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *Proc. of the 2010 IEEE Global Telecommunications Conference (GLOBECOM'10)*, pages 1–5, Miami, FL, USA, Dec 2010. IEEE Communications Society.

[103] R. Kulhavý. Restricted exponential forgetting in real-time identification. In *Proc. of the Symposium on Identification and System Parameter Estimation*, pages 1143–1148, 1985.

[104] R. Kulhavý and M. Karny. Tracking of slowly varying parameters by directional forgetting. In *Proc. of the* 9^{th} *World Congress of IFAC*, pages 79–83, Budapest, Hungary, 1984.

[105] Vibhore Kumar, Karsten Schwan, Subu Iyer, Yuan Chen, and Akhil Sahai. A state-space approach to SLA based management. In *Proc. of the* 11^{th} *IEEE/IFIP Network Operations and Management Symposium (NOMS'08)*, pages 192–199, 2008.

[106] Dara Kusic, Nagarajan Kandasamy, and Guofei Jiang. Combined power and performance management of virtualized computing environments serving session-based workloads. *IEEE Transactions on Network and Service Management*, 8(3):245–258, 2011.

[107] Huibert Kwakernaak and Raphel Sivan. *Linear Optimal Control Systems*. Wiley-Interscience, 1972.

[108] Wallace E. Larimore. Canonical variate analysis in identification, filtering, and adaptive control. In *Proc. of the* 29^{th} *IEEE Conference on Decision and Control*, pages 596–604, Honolulu, HI, USA, Dec 1990.

[109] T. Le-Ngoc and S.N. Subramanian. A Pareto-modulated Poisson process (PMPP) model for long-range dependent traffic. *Computer Communications*, 23(2):123–132, 2000.

[110] Jon Lee and Sven Leyffer, editors. *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*. Springer Science+Business Media, LLC, 2012.

[111] Sven Leyffer. Integrating sqp and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, 18:295–309, 1998.

[112] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall, 2nd edition, 1999.

[113] Lennart Ljung and Torsten Söderström. *Theory and Practice of Recursive Identification*. The MIT Press, 1983.

[114] Alexander M. Lyapunov. *The General Problem of the Stability of Motion*. PhD thesis, University of Kharkov, 1892.

[115] Jan M. Maciejowski. *Predictive Control with Constraints*. Pearson Education Limited, 2002.

[116] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[117] Peter Mell and Timothy Grance. The NIST definition of Cloud computing: Recommendations of the national institute of standards and technology. Special Publication 800-145, NIST, Sep 2011.

[118] Ningfang Mi, Giuliano Casale, Ludmila Cherkasova, and Evgenia Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *Proc. of the* 6^{th} *IEEE International Conference on Autonomic Computing (ICAC'09)*, pages 149–158, 2009.

[119] Lars Mieritz and Bill Kirwin. Defining Gartner total cost of ownership. Research Report G00131837, Gartner, Inc., Dec 2005.

[120] Bruce A. Murtagh and Michael A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[121] Bruce A. Murtagh and Michael A. Saunders. MINOS 5.0 User's Guide. report SOL 83-20, Department of Operations Research, Stanford University, 1983.

[122] Desineni Subbaram Naidu. *Optimal Control Systems*. CRC Press, 2003.

[123] Ripal Nathuji and Karsten Schwan. VirtualPower: Coordinated power management in virtualized enterprise systems. In *Proc. of* 21^{st} *ACM SIGOPS Symposium on Operating Systems Principles (SOSP'07)*, pages 265–278, 2007.

[124] A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[125] Aidan O'Dwyer. *Handbook of PI and PID Controller Tuning Rules*. Imperial College Press, 3rd edition, 2009.

[126] Przemysław Orłowski. Discrete-time, linear periodic time-varying system norm estimation using finite time horizon transfer operators. *Automatika*, 51(4):325–332, 2010.

[127] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. Automated control of multiple virtualized resources. In *Proc. of the* 4^{th} *ACM European Conference on Computer Systems (EuroSys'09)*, pages 13–26, 2009.

[128] Sujay Parekh, Kevin Rose, Joseph Hellerstein, Sam Lightstone, Matthew Huras, and VictorChang. Managing the performance impact of administrative utilities. In *Proc. of the* 14^{th} *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'03)*, volume 2867, pages 130–142, Heidelberg, Germany, October 2003.

[129] D.J. Park, B.E. Jun, and J.H. Kim. Fast tracking RLS algorithm using novel variable forgetting factor with unity zone. *Electronics Letters*, 27(23):2150–2151, Nov 1991.

[130] Krzysztof Pawlikowski. Steady-state simulation of queueing processes: survey of problems and solutions. *ACM Computing Surveys*, 22(2):123–170, 1990.

[131] Klaus Peternell, Wolfgang Scherrer, and Manfred Deistler. Statistical analysis of novel subspace identification methods. *Signal Processing*, 52(2):161–177, 1996.

[132] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[133] Ben Pring, , Robert H. Brown, Andrew Frank, Simon Hayward, and Lydia Leong. Forecast: Sizing the Cloud; understanding the opportunities in cloud services. Research Report G00166525, Gartner, Inc., Mar 2009.

[134] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Springer, 2nd edition, 2007.

[135] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles: Coordinated multi-level power management for the data center. In *Proc. of the* 13^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIII)*, pages 48–59, 2008.

[136] Michael A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[137] Karl J. Åström. *Introduction to Stochastic Control Theory*. Academic Press, 1970.

[138] Karl J. Åström and Torsten Bohlin. Numerical identification of linear dynamic systems from normal operating records. In *Proc. of the* 2^{nd} *IFAC Symposium on Self-Adaptive Systems*, Teddington, UK, Sep 1965.

[139] Karl J. Åström and Tor Hägglund. *PID Controllers: Theory, Design, and Tuning*. Instrument Society of America, 2nd edition, 1995.

[140] Karl J. Åström and Björn Wittenmark. *Adaptive Control*. Addison-Wesley, 2nd edition, 1994.

[141] Jorma Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.

[142] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *Proc. of the 2008 USENIX Conference on Power Aware Computing and Systems (HotPower'08)*, pages 1–5, 2008.

[143] S.W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, 1959.

[144] Mendel Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. In *Proc. of the IEEE Internet Computing*, volume 38, May 2005.

[145] Jeanne W. Ross and George Westerman. Preparing for utility computing: The role of IT architecture and relationship management. *IBM Systems Journal*, 43(1):5–19, 2004.

[146] Wilson J. Rugh. *Linear System Theory*. Prentice Hall, Upple Saddle River, NJ, USA, 1996.

[147] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Inc., 2010.

[148] Hong S. Ryoo and Nikolaos V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996.

[149] D. Sayre. On virtual systems. Technical report, IBM T. J. Watson Research Laboratory, Apr 1966.

[150] Dragoslav D. Šiljak, editor. *Decentralized Control of Complex Systems*, volume 184 of *Mathematics in Science and Engineering*. Elsevier, 1991.

[151] Edward M.B. Smith and Constantinos C. Pantelides. Global optimisation of nonconvex MINLPs. *Computers & Chemical Engineering*, 21, Supplement(0):S791–S796, 1997.

[152] Torsten Söderström and Petre Stoica. *System identification*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[153] Kostas S. Tsakalis and Petros A. Ioannou. *Linear time-varying systems: control and adaptation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[154] Daniel J. Tylavsky and Guy R.L. Sohie. Generalization of the matrix inversion lemma. In *Proc. of the IEEE*, volume 74, pages 1050–1052, July 1986.

[155] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C.M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kägi, Felix H. Leung, and Larry Smith. Intel virtualization technology. *IEEE Computer*, 38(5):48–56, 2005.

[156] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. Analytic modeling of multitier internet applications. *ACM Transaction on the Web*, 1(1), 2007.

[157] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, and Pawan Goyal. Dynamic provisioning of multi-tier internet applications. In *Proc. of the 2nd International Conference on Autonomic Computing (ICAC'05)*, pages 217–228, Seattle, WA, USA, 2005. IEEE Computer Society.

[158] Hien Nguyen Van, Frédéric Dang Tran, and Jean-Marc Menaud. Performance and power management for cloud infrastructures. In *Proc. of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD'10)*, pages 329–336, 2010.

[159] Peter Van Overschee and Bart De Moor. N4SID: subspace algorithms for the identification of combined deterministic and stochastic systems. *Automatica*, 30(1):75–93, 1994.

[160] Peter Van Overschee and Bart De Moor. *Subspace Identification For Linear Systems: Theory, Implementation, Applications*. Kluwer Academic Pulishers, 1996.

[161] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the Clouds: towards a Cloud definition. *SIGCOMM Computer Communication Review*, 39(1):50–55, Dec 2009.

[162] Michel Verhaegen and Patrick Dewilde. Subspace model identification part 1: The output-error state-space model identification class of algorithms. *International Journal of Control*, 56(5):1187–1210, 1992.

[163] Michel Verhaegen and Patrick Dewilde. Subspace model identification part 2: Analysis of the elementary output-error state-space model identification algorithm. *International Journal of Control*, 56(5):1211–1241, 1992.

[164] Michel Verhaegen and Vincent Verdult. *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, 2007.

[165] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pMapper: Power and migration cost aware application placement in virtualized systems. In *Proc. of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08)*, pages 243–264, 2008.

[166] Werner Vogels. Beyond server consolidation. *ACM Queue*, 6(1):20–26, 2008.

[167] Xiaorui Wang and Yefu Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Transactions on Parallel and Distributed Systems*, 22(2):245–259, 2011.

[168] Aaron Weiss. Computing in the clouds. *netWorker*, 11(4):16–25, 2007.

[169] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proc. of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware'08)*, pages 366–387, 2008.

[170] Pengcheng Xiong, Zhikui Wang, Simon Malkowski, Qingyang Wang, Deepal Jayasinghe, and Calton Pu. Economical and robust provisioning of *n*-tier cloud workloads: A multi-level control approach. In *Proc. of the* 31st *International Conference on Distributed Computing Systems (ICDCS'11)*, pages 571–580, 2011.

[171] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *In Proc. of the 2008 Grid Computing Environments Workshop (GCE'08)*, pages 1–10, Nov 2008.

[172] L.A Zadeh. Frequency analysis of variable networks. In *Proc. of the Institute of Radio Engineers (IRE'1950)*, volume 38, pages 291–299, Mar 1950.

[173] Qi Zhang, Lu Cheng, and RaoufBoutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.