Universidade Federal
do Rio de Janeiro
Escola Politécnica

DEVELOPMENT AND VALIDATION OF AN OPTIMAL SELF-ADAPTIVE SLA-ORIENTED RESOURCE ALLOCATION ALGORITHM FOR CLOUD COMPUTING

Jonathan Ferreira Passoni

Projeto apresentado ao Curso de Engenharia Elétrica da Escola Politécnica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro Eletricista.

Orientadores: Professor Oumar Diene
Professor Amit Bhaya

Rio de Janeiro
Março de 2021

DEVELOPMENT AND VALIDATION OF AN OPTIMAL SELF-ADAPTIVE SLA-ORIENTED RESOURCE ALLOCATION ALGORITHM FOR CLOUD COMPUTING

Jonathan Ferreira Passoni

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELÉTRICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA.

Examinado por:

Prof. Oumar Diene, D.Sc.

Prof. Amit Bhaya, Ph.D.

Prof. Carmen Lucia Tancredo Borges, D.Sc.

Eng. Tiago Salviano Calmon, M.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2021

Jonathan Ferreira Passoni,

Desenvolvimento e validação de um algoritmo adaptativo de alocação ótima de recursos orientado à SLA para cloud computing/ Jonathan Ferreira Passoni. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2021.

XIX, 159 p. : il.; 29, 7cm.

Orientadores: Professor Oumar Diene

Professor Amit Bhaya

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Elétrica, 2021.

Bibliographie: p. 156 – 159.

1. Resource allocation. 2. Non-linear Optimal control. 3. Non-linear Adaptive control. 4. Recursive Least Squares. 5. Real-time system. 6. Self-adaptive software. I. Oumar Diene, Professor *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia Elétrica. III. Título.

"Il redonne des forces à celui qui en manque, il rend courage à celui qui est épuisé.
Les jeunes eux-mêmes deviennent faibles et se fatiguent.
Même les meilleurs tombent.
Mais ceux qui mettent leur espoir dans le Seigneur retrouvent des forces nouvelles.
Ils s'envolent comme des aigles,
ils courent sans se fatiguer,
ils avancent sans s'épuiser."
(Ésaïe 40 :29-31 - version PDV)

Acknowledgements

First and foremost, I give thanks to God for always reminding me that His ways are way too higher and better than the dreams that I have ever had and for allowing me to live extraordinary and turning point experiences in between my journey through life.

Aos meus pais, pelo esforço, dedicação e apoio em todas as situações. Por acima de tudo me proporcionarem o privilégio de ter uma educação de qualidade.

Ao Marcelo, Fernando e Aline pelas oportunidades que vocês me confiaram no IMF. Também agradeço à Adriana, Cássia e Alessandra pela amizade e companhia.

À UFRJ, por proporcionar um ambiente tão rico de oportunidades que não se resumem apenas à obtenção de um diploma.

À Carmen Lodi Maidantick, pela sua orientação durante minha primeira experiência em P&D. Ainda guardo e sigo muito dos seus ensinamentos.

À Telma Pará, pela orientação em diversos projetos e momentos, mas sobretudo pela grande amizade (pour toujours !). Por todo suporte desde meu retorno ao Brasil. Por compartilhar o sonho, a saudade de viver na França e todas as experiências gastronômicas que sempre rendem boas histórias.

À CAPES e aos idealizadores do programa BRAFITEC pela mais extraordinária oportunidade que me foi dada. Estudar na França não apenas contribuiu à minha formação profissional, foi uma experiência que mudou a minha vida.

À toute la famille Peyrard, pour l'incroyable accueil dans votre maison, vos vies et spécialement pour me faire adapter le mieux possible à la culture française !

À Loïc Queval, pour son tutorat et pour me faire apprendre comment conduire un projet de R&D à la française. Cela a été une formation incontournable pour moi.

Je remercie également Hervé Gueguén et Marie-Anne Lefebvre pour tous les conseils lors de mon dernière année à CentraleSupélec. Un grand merci aussi à l'administration de CentraleSupélec - campus de Rennes, pour toute l'aide apportée lors de mon accueil.

Je tiens vraiment à remercier aussi toutes les personnes auxquelles j'ai pu travailler pendant mes stages en France, chez Solystic et chez Safran Landing Systems. Je n'oublierai jamais les merveilleuses expériences et les moments de convivialité que j'ai passé chez vous. Plus particulièrement, je remercie Jean-Baptiste Lestage pour tous les échanges au cours de mon stage.

Ao Prof. José Luiz da Silva Neto, Katia e Luciana pela ajuda em todos os aspectos administrativos desde meu retorno ao Brasil.

À Prof. Carmen Lucia Tancredo Borges e ao Eng. Tiago Calmon por aceitarem o convite para compor a banca avaliadora deste trabalho. E finalmente, aos professores Oumar e Amit pela oportunidade de participar deste projeto e pelo auxílio na revisão deste manuscrito.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

DEVELOPMENT AND VALIDATION OF AN OPTIMAL SELF-ADAPTIVE SLA-ORIENTED RESOURCE ALLOCATION ALGORITHM FOR CLOUD COMPUTING

Jonathan Ferreira Passoni

March/2021

Advisors: Professor Oumar Diene
Professor Amit Bhaya

Course: Electrical Engineering

Cloud services providers try to reduce the operational cost of their computational infrastructure to make their services more profitable. In terms of resource consumption, applications using cloud services often do not use the full amount of computational resources allocated to them. This work provides an efficient dynamic resource allocation algorithm for applications that demand large computational effort. The algorithm is designed using control theory aspects, taking advantage of the self tuning regulator approach to integrate an adaptive mechanism for each running application. A dynamic model is designed to describe the behavior of each application by using Amdahl's Law, which establishes a relationship between the execution time and the number of allocated cores for a given application. Model parameters are estimated in real time, with a suitable mechanism to reduce transient effects, allowing resource allocation to be adaptive. As applications with varying computational requirements enter and exit a cloud data center, multi-processing aspects are considered in order to provide scalable performance. The global system, consisting of an optimal adaptive control strategy, is designed to determine a viable solution even when not all requirements can be satisfied. Validation is performed on small-scale cases, which nevertheless contemplates some scenarios which occur in the daily routine of cloud services resources management.

Résumé du Projet de Fin d'Etudes présenté à l'Ecole Polytechnique de l'UFRJ à titre d'exigence pour l'obtention du diplôme en Génie Eléctrique.

DÉVELOPPEMENT ET VALIDATION D'UN ALGORITHME ADAPTATIF D'ALLOCATION OPTIMALE DES RESSOURCES ORIENTÉ À SLA DANS LE CLOUD COMPUTING

Jonathan Ferreira Passoni

Mars/2021

Encadrants: Professeur Oumar Diene
Professeur Amit Bhaya

Formation: Génie Eléctrique

Les fournisseurs d'accès aux services en cloud computing essaient de réduire le coût d'exploitation de leur infrastructure informatique afin de rendre ces services plus rentables. Compte tenu de l'usage des ressources, les applications n'utilisent pas de la capacité totale qui leur est fournie. Ce projet offre un algorithme efficient d'allocation dynamique des ressources pour des applications connues par leur haut effort de calcul. Cet algorithme s'approprie des concepts de la théorie d'Automatique, notamment le contrôle adaptatif par contrôle autoréglable afin d'intégrer un processus adaptatif à chaque application en cours. Pour ces applications, un modèle dynamique est conçu de façon que leur comportement soit décrit en utilisant la loi d'Amdahl, qui établie une relation que associe le temps d'exécution avec le nombre des noyaux attribués à une certaine application. Les paramètres du modèle sont estimés en temps-réel, avec des processus complémentaires dans le but de réduire des effets transitoires, ce que permet l'allocation des ressources d'être adaptative. En raison du fort écoulement de différentes applications dans un service cloud, des aspects de parallélisme sont pris en compte de façon à assurer une performance appropriée et évolutive. Le système globale, une stratégie de contrôle optimale et adaptative, est développé dans le but d'approvisionner une solution viable même si les exigences ne peuvent pas être intégralement satisfaites. La validation se réalise en petite échelle, en prenant compte des différents scénarios qui peuvent se produire dans la routine d'une gestion de ressources aux services en cloud.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

DESENVOLVIMENTO E VALIDAÇÃO DE UM ALGORITMO ADAPTATIVO DE ALOCAÇÃO ÓTIMA DE RECURSOS ORIENTADO À SLA PARA CLOUD COMPUTING

Jonathan Ferreira Passoni

Março/2021

Orientadores: Professor Oumar Diene
Professor Amit Bhaya

Curso: Engenharia Elétrica

Os fornecedores de serviços em Cloud Computing tentam reduzir o custo operacional de sua infraestrutura computacional para tornar seus serviços mais rentáveis. Considerando o consumo de recursos, geralmente as aplicações não utilizam a capacidade total alocadas às mesmas. Este trabalho apresenta um algoritmo eficiente de alocação dinâmica de recursos capaz de lidar com aplicações conhecidas por seu alto esforço computacional. O algoritmo é desenvolvido usando a técnica de reguladores auto-ajustáveis para integrar um método adaptativo a nível de cada aplicação em execução. Um modelo dinâmico foi concebido para descrever o comportamento destas aplicações baseado na Lei de Amdahl, que estabelece uma relação entre Tempo de Execução e o Número de Cores (CPUs) para uma aplicação. Os parâmetros do modelo são obtidos através de uma estimação online, integrada com um mecanismo para reduzir efeitos transitórios, o que faz com que a alocação de recursos seja adaptativa. Devido ao grande fluxo em serviços cloud de aplicações com diferentes requisitos, aspectos ligados ao paralelismo foram considerados para assegurar um desempenho adequado em larga escala. O sistema global, compreendendo uma estratégia de controle ótimo e adaptativo, fornece uma solução viável mesmo em situações em que nem todos os requisitos podem ser atendidos. A validação é realizada em pequena escala, lidando com diferentes situações que podem ser verificadas na rotina diária da gestão de recursos de serviços cloud.

Contents

List of Algorithms **xiii**

List of Figures **xv**

List of Tables **xix**

Introduction **1**

Main Context 1
Objectives and Contributions of this work 3
Organization of the Manuscript 5

1 Current solutions applied in Software Systems and the prototype developed by DELL-EMC **8**

1.1 Related work 9
1.1.1 Addressing control theory in Software Systems 9
1.1.2 Literature overview of current solutions developed 10
1.2 Performance analysis of the control strategy developed by Dell-EMC for a single container execution 13
1.2.1 Top-level Overview 13
1.2.2 Control Strategy design 17
1.2.3 Deployment of a single container execution 19

I Adaptive control strategies considering a single container execution approach **28**

2 Elements of Adaptive Control **29**

2.1 Motivation 29
2.2 Adaptive Control System Approaches 30
2.2.1 Gain Scheduling 30
2.2.2 Model Reference Adaptive Controller (MRAC) 31

2.2.3 Self Tuning Regulator or Model Identification Adaptive Controller (MIAC) . . . 32
2.3 Adaptive Control System Design . . . 33
2.4 Stability Analysis: the Lyapunov Theorem . . . 35
2.4.1 The continuous-time system case . . . 35
2.4.2 Extending the Lyapunov Theorem to the discrete-time system case . . . 37
2.5 On-line parameter estimation . . . 37
2.5.1 Recursive Least-Squares (RLS) Method . . . 38

3 Adaptive Control Strategies 42
3.1 System Model Design . . . 43
3.1.1 The mathematical formulation of the input-output relationship . . . 43
3.1.2 The Curve Fitting Process . . . 44
3.1.3 Definition of a System Model . . . 48
3.2 Common Control System Block Diagram . . . 48
3.3 Stability analysis for a Self Tuning Regulator . . . 49
3.4 Adaptive Control Strategies . . . 53
3.4.1 Control strategies considering the workload characteristics described by a hyperbolic function . . . 53
3.4.2 Control strategies considering the workload characteristics described by an exponential function . . . 61
3.5 Synthesis of all control strategies . . . 68

4 Deployment of a single container using the Adaptive Control Strategies 70
4.1 Main goals for testing . . . 70
4.2 Design of test scenarios . . . 71
4.2.1 Tests environment . . . 72
4.3 Performance Analysis . . . 74
4.3.1 Control strategies based on a hyperbolic function for the workload characteristics . . . 75
4.3.2 Control strategies based on an exponential function for the workload characteristics . . . 76
4.4 Conclusion . . . 77

II Optimal Adaptive Control Strategy considering multiple

containers execution **79**

5 Optimal Adaptive Control Strategy **80**
5.1 System Structural Architecture 81
5.1.1 The new context of multiple containers execution 81
5.1.2 The Kafka issue 83
5.1.3 Overview of System Modules 85
5.2 Optimal Adaptive Control Strategy 86
5.2.1 Problem formulation update 86
5.2.2 Proposed Solution 87
5.2.3 Optimal Control Problems 88
5.2.4 The real-time operation 94
5.3 Synthesis of the global optimal adaptive control strategy algorithm 98

6 Deployment of multiple containers using the OACS **102**
6.1 Main objectives for the OACS validation 103
6.2 OACS validation for the deployment of a single container 103
6.2.1 Design of test scenarios 104
6.2.2 Performance Analysis 105
6.2.3 Conclusion 106
6.3 OACS validation for the deployment of multiple containers 107
6.3.1 Test case 1: Constant set-point with changes in environment 108
6.3.2 Test case 2: Varying set-point with no change in environment 113
6.3.3 Test case 3: Varying set-point with changes in environment . 118
6.3.4 Conclusion 123

Conclusion **124**

Appendix A Profile of applications performed on tests and their workload characteristics **126**
A.1 Deep learning training processes 126
A.1.1 Case using MNIST database 127
A.1.2 Case using Fashion-MNIST database 128
A.2 Video encoding process 129

Appendix B Detailed results from tests of the adaptive control strategies for single container execution **131**
B.1 Control strategies which used an hyperbolic model for the input-output relationship 131

B.1.1 Control Strategy 1: System Dynamics given by the Taylor expansion of an hyperbolic function 131
B.1.2 Control Strategy 2: System Dynamics given by a manipulation of an hyperbolic function 136
B.2 Control strategies which used an exponential model for the input-output relationship 140
B.2.1 Control Strategy 3: System Dynamics given by the Taylor expansion of an exponential function 140
B.2.2 Control Strategy 4: System Dynamics given by a manipulation of an exponential function 145

Appendix C Detailed results from tests of OACS in a single container execution environment 149
C.1 Test cases using MNIST ML application 150
C.2 Test cases using Fashion-MNIST ML application 152
C.3 Test cases using Video Encoding application 154

Bibliography 156

List of Algorithms

1 RLS algorithm 41

2 Curve Fitting Process Algorithm 44

3 Complementary logic aiming to deliver to the control module the most suitable values of $\hat{\alpha}$ and $\hat{\beta}$ for a given operating point 56

4 The initial resource allocation algorithm 83

5 The initial guess for $\hat{\beta}$ algorithm 95

6 Correcting the set-point value algorithm 97

7 The optimal adaptive control input evaluation algorithm 101

List of Figures

1 General workload characteristic understood by the Amdahl's Law 3
2 Manuscript dependency chart 7

1.1 Software architecture. 14
1.2 Interfaces of the Control Module. 15
1.3 Sequence Diagram (SysML representation) of the system operation. 16
1.4 Control system architecture for the process defined by a container . 17
1.5 Performance of the control strategy developed by DELL-EMC. Test case executed using set-point= $12.5s$. 22
1.6 Performance of the control strategy developed by DELL-EMC. Test case executed using set-point = $25.0s$. 23
1.7 Performance of the control strategy developed by DELL-EMC. Test case executed using set-point= $50.0s$. 24
1.8 Performance of the control strategy developed by DELL-EMC. Test case executed using set-point= $100.0s$. 25
1.9 All points recovered from the tests 26

2.1 A gain scheduling control system. (Extracted from [1]) 31
2.2 A model-reference adaptive control system. (Extracted from [1]) . . 32
2.3 A self-tuning controller. (Extracted from [2]) 33
2.4 Identification Process Scheme. (Translated from [3]) 34

3.1 Real characteristic and resultant curves from the Curve Fitting Processes performed considering a hyperbolic function for the plant . 45
3.2 Real characteristic and resultant curves from the Curve Fitting Processes performed considering an exponential function for the plant 47
3.3 Control system block diagram for all strategies to be presented in this chapter 48
3.4 Execution time in different approachs. Workload Characteristic from the container described in Appendix A.1.1. 54
3.5 Characteristic of β_c according to ΔTTF for $TTF(k+1)$ fixed 58

3.6 Linear characteristics given the regression model expressed in equation (3.45) . . . 62
3.7 Characteristic of β_c according to ΔTTF for $TTF(k+1)$ fixed 64
3.8 Delimited zones of stability in the sense of Lyapunov Theorem . . . 68

4.1 Jenkins's main user interface used in this work. . . . 73
4.2 Jenkins's set-up page in order to build a test case. . . . 74

5.1 Simple system architecture extended to multiple containers environment . . . 81
5.2 Sequence Diagram (SysML representation) for the deployment of a new container . . . 82
5.3 Example of the message flow from inside the container until its arrival at the Kafka Consumer method . . . 84
5.4 System architecture . . . 86
5.5 Test scenario . . . 96
5.6 Global system architecture, integrating the estimation process . . . 98
5.7 Sequence Diagram (SysML representation) for the global control strategy operation . . . 99
5.8 Sequence Diagram (SysML representation) for the operation of the estimator . . . 100

6.1 Test scenario . . . 104
6.2 Performance of the OACS in Test Case 1 . . . 109
6.3 Performance of each estimation process performed in Test Case 1 . 110
6.4 Performance of the OACS in Test Case 2 . . . 115
6.5 Performance of each estimation process performed in Test Case 2 . 116
6.6 Performance of the OACS in Test Case 3 . . . 120
6.7 Performance of each estimation process performed in Test Case 3 . 121

A.1 An example of a deep learning neural network . . . 127
A.2 A few samples from the MNIST test data set. . . . 127
A.3 Workload characteristics for a DNN process using a MNIST database. 128
A.4 A few samples from the Fashion-MNIST test data set. . . . 128
A.5 Workload characteristics for a DNN process using a Fashion-MNIST database. . . . 129
A.6 Workload characteristics of a video encoding process performed in a Youtube video . . . 130

B.1 Performance of Control Strategy 1 for a initial setpoint value of 12.5s 132
B.2 Performance of Control Strategy 1 for a initial setpoint value of 25.0s 133

B.3 Performance of Control Strategy 1 for a initial setpoint value of 50.0s134
B.4 Performance of Control Strategy 1 for a initial setpoint value of $100.0s$135
B.5 Performance of Control Strategy 2 for a initial setpoint value of 12.5s136
B.6 Performance of Control Strategy 2 for a initial setpoint value of 25.0s137
B.7 Performance of Control Strategy 2 for a initial setpoint value of 50.0s138
B.8 Performance of Control Strategy 2 for a initial setpoint value of 100.0s139
B.9 Performance of Control Strategy 3 for a initial setpoint value of 12.5s141
B.10 Performance of Control Strategy 3 for a initial setpoint value of 25.0s142
B.11 Performance of Control Strategy 3 for a initial setpoint value of 50.0s143
B.12 Performance of Control Strategy 3 for a initial setpoint value of 100.0s144
B.13 Performance of Control Strategy 4 for a initial setpoint value of 12.5s145
B.14 Performance of Control Strategy 4 for a initial setpoint value of 25.0s146
B.15 Performance of Control Strategy 4 for a initial setpoint value of 50.0s147
B.16 Performance of Control Strategy 4 for a initial setpoint value of 100.0s148

C.1 Guess for the initial value of $\hat{\beta}$ 149
C.2 Performance of the OACS. Single container environment with MNIST ML application. Initial set-point value: 12.5s 150
C.3 Performance of the OACS. Single container environment with MNIST ML application. Initial set-point value: 25.0s 150
C.4 Performance of the OACS. Single container environment with MNIST ML application. Initial set-point value: 50.0s 151
C.5 Performance of the OACS. Single container environment with MNIST ML application. Initial set-point value: 100.0s 151
C.6 Performance of the OACS. Single container environment with Fashion-MNIST ML application. Initial set-point value: 12.5s 152
C.7 Performance of the OACS. Single container environment with Fashion-MNIST ML application. Initial set-point value: 25.0s 152
C.8 Performance of the OACS. Single container environment with Fashion-MNIST ML application. Initial set-point value: 50.0s 153
C.9 Performance of the OACS. Single container environment with Fashion-MNIST ML application. Initial set-point value: 100.0s . . . 153
C.10 Performance of the OACS. Single container environment with Video Encoding application. Initial set-point value: 12.5s 154
C.11 Performance of the OACS. Single container environment with Video Encoding application. Initial set-point value: 25.0s 154
C.12 Performance of the OACS. Single container environment with Video Encoding application. Initial set-point value: 50.0s 155

C.13 Performance of the OACS. Single container environment with Video Encoding application. Initial set-point value: 100.0s 155

List of Tables

1.1 Characteristics of control systems solutions developed in some papers 12
1.2 Key aspects of the control strategy initialization 19
1.3 Configuration of the physical host 20
1.4 Performance aspects from the tests performed in the control strategy provided by DELL-EMC 26

3.1 Mean Squared Error (MSE) and Mean Relative Error (MRE) for the estimation considering a hyperbolic function model 46
3.2 MSE and MRE for the estimation considering an exponential function model 47
3.3 Key aspects of the control strategies based on a hyperbolic function 69
3.4 Key aspects of the control strategies based on an exponential function 69

4.1 Characteristic of each phase composing a test scenario 71
4.2 Best control strategy (CS) regarding two different approaches . . . 77

6.1 Configuration of each container designed to run at Test Case 1 . . . 108
6.2 Steps in which the total amount of available resources is changed in Test Case 1 and its related value to be applied 108
6.3 Configuration of each container designed to run at Test Case 2 . . . 113
6.4 Changes in set-point for containers designed to run at Test Case 2 . 114
6.5 Configuration of each container designed to run at Test Case 3 . . . 119
6.6 Changes in set-point for containers designed to run at Test Case 3 . 119
6.7 Steps in which the total amount of available resources is changed in Test Case 3 and its related value to be applied 119

Introduction

Main Context

The use of computational resources has changed for people and industry with the advance of cloud computing over the last years. Today, these resources can be provided as an *on-demand* service, where clients require them in a customized amount, rather than based only on the peak workload demand [4]. Thanks to virtualization, part of the infrastructure behind the "clouds" can be totally reserved to run any application.

Software applications are developed nowadays using a distributed approach. As a result, small applications, also known as *micro services*, are developed independently from each other and then integrated to the whole software. They also are characterized by independent execution, which implies a private usage of computational resources.

Virtualization can also be provided to these micro services using containers, which allow each one of them and their dependencies to be encapsulated on a single unit of software, leading to an independent execution. For a resource provider viewpoint, the use of containers increases the complexity of the resource allocation task.

Consumers are attracted to the *on-demand* resource provisioning since this reduces or even eliminates the cost of acquisition of physical components. On the other hand, service providers have to guarantee a certain quality of service (QoS), as the performance of an application is directly affected by the computational capabilities delivered to it. A Service Level Agreement (SLA) is then established between consumer and service provider to formalize the expected QoS required by the former and that should be provided by the latter, with penalties being applicable in case of violations in service delivery. Examples of QoS parameters, also known as SLA metrics, are maximum response time, average resource/service availability and minimum throughput [5].

Through a resource management system, cloud services providers deal with SLA metrics of different applications running in the same environment. The large number of resources which characterizes cloud data centers imposes an efficient

allocation to make it profitable. This can be accomplished by minimizing the resources usage while respecting SLAs requirements. However, this strategy is only achieved with the use of knowledge about the workload of each application, which can be difficult to obtain before and even at run-time. Resource allocation strategies which do not use such knowledge may produce large gaps between aggregate resource usage and aggregate allocation [6], leading to inefficiency.

Workloads are highly dynamic and vary over time, demanding specific scheduling decisions to avoid violations in SLA. The resource management system must cope with quickly changing requirements, unpredictable and uncertain operating conditions that occur at run-time. Self-adaptive mechanisms present some answers to these problems, allowing adjustments to deal with unexpected behavior or the lack of system information.

Control theory has gained a lot of attention in recent years concerning the integration of controllers into Self-Adaptive Software Systems [7], providing formal guarantees of effectiveness, under precise assumptions on the operating conditions [8]. However, one of the main obstacles experienced is the difficulty in accurately modelling software systems, since the design of controllers is a model-based technique.

Cloud computing involves diverse workload profiles that run in its servers. As a result, it is difficult to design a general resource allocation algorithm capable of satisfying requirements from all sorts of workloads. This work focuses on developing an algorithm which manages applications with a certain workload profile that permits the design of an optimal and self-adaptive control solution.

Iterative workloads consist of identical small jobs that run in succession. Examples of such workloads are training processes in most machine learning algorithms, images processing from a video in real-time and request processing in a RESTful API [9]. With an application in which these small jobs are considered as possessing identical workloads, one can take advantage of Amdahl's law [10, 11] to obtain a more accurate model of a plant representing the workload profile. Every workload with a fixed-length job follows the characteristic seen in Figure 1.

The behavior presented by the red curve in Figure 1 can be understood by considering the sequential and parallel parts of a given workload execution. As the number of cores allocated to an application grows, it gains in more parallel capacity, resulting in a faster execution on this specific part. After a certain amount of cores, the execution time converges to a constant value, since the parallel capacity for an application is fully reached, preventing any further speedup.

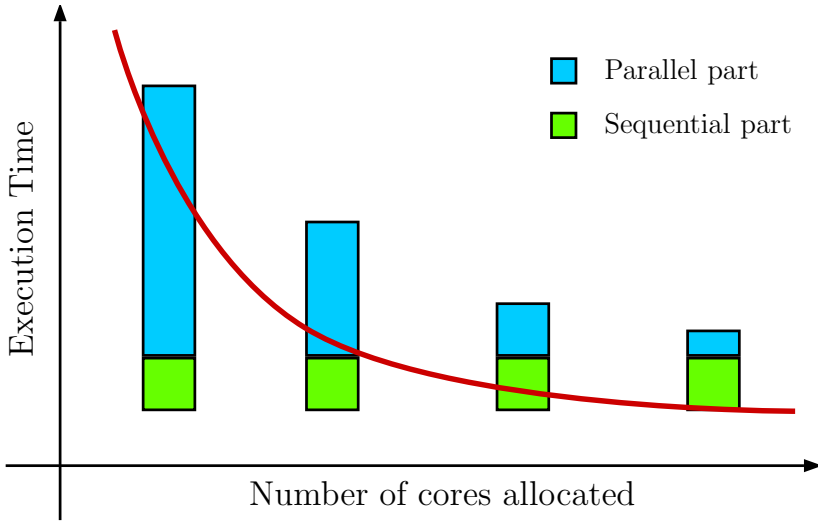


Figure 1 – General workload characteristic understood by the Amdahl's Law

Even though there is no specific mathematical formulation for the relationship between number of cores and execution time for a given workload, some approximate mathematical models for this red curve can be proposed and their accuracy analyzed. With a reasonable model available, control system design can be carried out.

Objectives and Contributions of this work

The present work arises from a collaboration between NACAD - the High Performance Computing Center (at the Federal University of Rio de Janeiro) and the R&D Center of the company Dell-EMC. The main objective is to conceive an algorithm that provides an optimal self-adaptive resource allocation for each container active in a cluster, based on a control theory approach. Any allocation must respect the SLA defined by an upper bound on response time for each unit of jobs, also referred to as *epochs*, performed by an application.

The following goals are defined:

1. **Identify the best model for the workload characteristics of an application**
   Every control system design starts from the definition of a mathematical model for the plant to be controlled. Dynamical system models use a mathematical formulation that describes evolution of states over time. The shape of the workload characteristics illustrated in Figure 1 provides information about the relationship input-output in a given moment of time. Two well-known functions were subjected to a Curve Fitting Process in order to study how accurately these mathematical relationships can represent the real data.
2. **Validate an adaptive control strategy for the deployment of only a single container**

The development of the resource allocation algorithm was carried out in two main phases. The initial one aimed to provide the resource allocation of only a single application running inside a container. This initial solution was obtained through the following steps:

2.a) Design of dynamic models based on the input-output relationship provided by Amdahl's Law

The design of control laws requires a dynamic model for the plant. With two different approaches, models are proposed based on the mathematical functions analyzed through curve fitting processes for the input-output relationship.

2.b) Integrate an online estimation process and implement adaptive control strategies

Non-linear dead-beat controllers are designed for a single container approach. They take advantage of the dynamics of a container process to drive the set-point tracking error to zero.

The control strategies do not use any *a priori* knowledge about the parameters of the workload characteristics. Consequently, the controller takes only their estimated values, given as a result of an estimation considering the validated mathematical model for the input-output relationship.

2.c) Study the stability of the adaptive control strategies

The integration of an online estimation process to the control action must be considered in the stability analysis, because of its own dynamics. This study focuses on establishing conditions to ensure that the overall system accomplishes its goals.

2.d) Study the performance of these control strategies under different operating conditions

With all design and theoretical analysis done, the control strategies operation are validated using the system architecture conceived by Dell-EMC. More complete test scenarios are created to analyze the control strategies in different conditions that may occur in real large-scale operation. These analyses are then taken into account to choose the best control strategy.

3. **Develop a real-time system that enables multiple containers to run in parallel**

The second and last phase of the resource allocation algorithm development consists of adapting the best control strategy found in the single container

approach to a new in which the algorithm manages multiple containers. The solution implemented for the single container approach was designed without considering some aspects that occur in multi-tasking systems. The performance of a resource allocation algorithm that handles requests from different containers may be compromised due to two main reasons.

First, without considering multiprocessing, the run-time operation could experience significant delay while handling requests for the resource allocation of each container. This can affect the data provided to the estimation process since this delay interferes in how precisely the measure represents the workload characteristics. Moreover, managing multiple applications running at the same environment compels some manual tasks of the former system to be developed as autonomous operations.

A real-time system must then be conceived to enable a proper deployment of multiple containers and their execution in parallel.

4. **Design and validate an optimal control strategy for the deployment of multiple containers**

   Looking at the control strategy chosen after the testing campaign for the single container approach, the dynamic model and the estimation process associated to it are adapted to formulate the optimal adaptive control strategy to handle multiple containers.

   An algorithm for the initial allocation of resources for each container that starts its execution is presented as well.

   The optimal adaptive control strategy design starts by examining the Model Predictive Control (MPC) technique. The prediction horizon significance for the control input evaluation is reviewed, aiming to develop a control strategy with a fast computation, spending the least computational resources as possible.

   A final algorithm is proposed to provide always a viable solution, in normal operation as well as in a constrained environment - with a limited amount of resources. The performance of the whole solution is studied by increasing the complexity of the system operation. This results in three different test cases dealing with multiple and distinct workload execution.

Organization of the Manuscript

This work is organized as follows: an introduction, six chapters, a final conclusion and three appendices.

Chapter 1 presents a literature overview of the control theory applied to the development of self-adaptive software systems, for different applications. The control strategy conceived by DELL-EMC at the beginning of this project is also described and has its performance studied.

Three chapters comprehends the first part dedicated to the development of control strategies considering the execution of only a single container. Chapter 2 introduces all theoretical aspects covering the development of adaptive control strategies (ACS), as well as enunciates the Lyapunov Theorem, applied for stability analysis. In addition, it presents the Recursive Least Squares (RLS) algorithm, integrated as the online estimation method in all control strategies in this work. Chapter 3 first presents the Curve Fitting Process performed to validate an accurate mathematical model for the input-output relationship. It also details all four control strategies for a single container execution. For the purpose of verification and validation (V&V) of these strategies, Chapter 4 provides a performance analysis of each one of them based on tests whose data is given in the Appendix B.

For the part dedicated to the global solution development, involving the deployment of multiple containers, there are two chapters. Chapter 5 starts by describing the different components of the software that handles the resource management system. It also specifies how the functionalities of this system are organized to provide an effective multi-tasking performance at run-time. After that, this chapter introduces the optimal adaptive control strategy (OACS), with a detailed design of the optimal control problems to handle situations with limited and insufficient amount of resources. Chapter 6 provides the performance analysis of this control strategy under a testing campaign using the new structural system architecture presented at the beginning of this part. Some results of these tests are given in the Appendix C.

The manuscript ends with some conclusions of this work and future directions about the pursuing of this project.

Appendix A describes the profile of all workloads contemplated in this work.

The manuscript dependency chart which shows the connection between the chapters briefly described above is depicted in Figure 2.

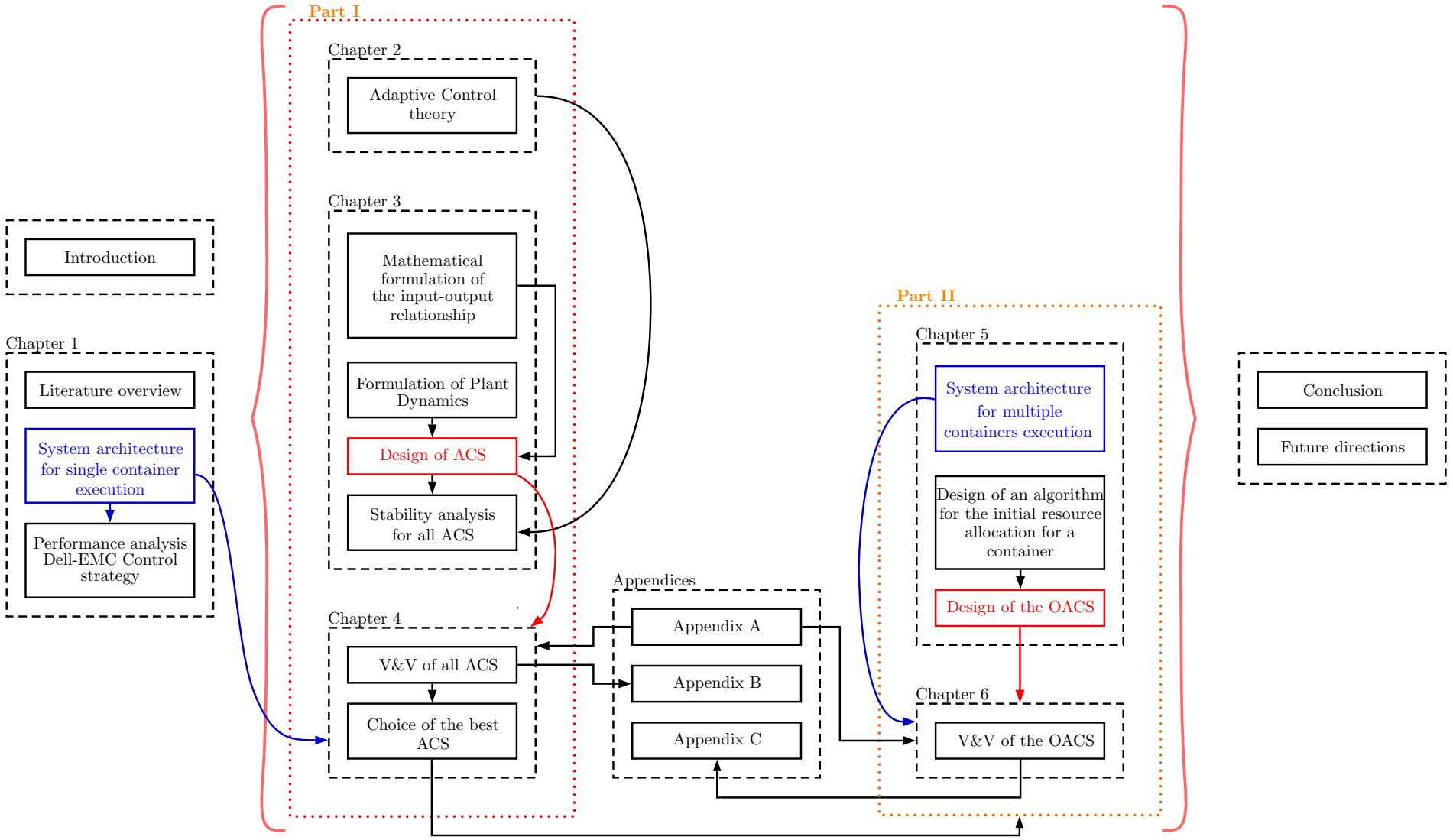


Figure 2 – Manuscript dependency chart

Chapter 1

Current solutions applied in Software Systems and the prototype developed by DELL-EMC

Contents

1.1 Related work 9
1.1.1 Addressing control theory in Software Systems 9
1.1.2 Literature overview of current solutions developed 10
1.2 Performance analysis of the control strategy developed by Dell-EMC for a single container execution 13
1.2.1 Top-level Overview 13
1.2.2 Control Strategy design 17
1.2.3 Deployment of a single container execution 19

Control theory has gained a lot of attention in the development of software systems due to the formal guarantees it can provide to their performance under some conditions. The first objective of this chapter is to present an overview of current solutions that applies this theory in Software Engineering, specially on cases in which the self-adaptive concept is considered. The second one is to introduce the prototype solution developed by DELL-EMC to the resource allocation problem for a single application, using control techniques. Through a testing campaign, the performance of this control strategy is examined, allowing the identification of key points to be considered at the design phase of the global solution, aimed to manage efficiently multiple applications.

1.1 Related work

A self-adaptive system does not have an unique and precise definition. In [12], it is referred as a system that "is able to adjust its behavior in response to its perception of the environment and the system itself". The authors in [13] focus on "the self prefix" to infer that the system must "decide autonomously (i.e., without or with minimal interference) how to adapt or organize to accommodate changes in its context and environment."

When dealing with software systems, architecture aspects have a strong influence on their performance. These systems are known by presenting unpredictable, uncertain and continuously changing execution environments [8]. Additionally, most of critical services require continuous availability to their applications. Considering also the high scalability of services provided by the cloud environment, the management of software systems requires new and innovative approaches in a way a certain QoS can still be satisfied in such conditions.

Motivated by these characteristics, self-adaptive systems are implemented aiming to automatically react to changes in both operating environment and application behavior [14].

1.1.1 Addressing control theory in Software Systems

Control theory techniques have allowed the design of controllers for industrial plants such that they can behave as expected. Through this technique, some properties can be verified, assuring an adequate performance. This is possible because the behavior of the physical components integrating these plants are easily represented by mathematical models. An advantage of using these techniques is that most of them implement feedback mechanisms, providing to the control input evaluation the current state of the plant. A more adaptive response can be applied, which drives the system to reach its goals more quickly.

For the case of software systems, models typically rely on architectural concepts, like components and connectors [7]. Mathematical models are usually established by associating some hardware components with *non-functional* aspects, such as reliability, performance, energy consumption, and cost [15]. In most of the cases, the relationship between these variables is not well defined by equations as for industrial plants, ruled by physical laws. Besides that, each change or update verified on the system architecture can result in a different behavior, making it difficult to establish mathematical models.

In a large number of cases, it is complicated to obtain reliable data before runtime operation, which implies a parameters identification to be performed by

an online process. The self-adaptive aspect comes to perform then a parametric adaptation, where the controller structure does not change but only its parameters values according to an adaptive mechanism [8]. The same aspect can also be extended to any additional functionality allowing the system itself to manage possible faults that may occur in its operation.

1.1.2 Literature overview of current solutions developed

The review led in [7] provides a broad profile of how software is modeled and which control techniques were implemented to conceive adaptive software systems. The most popular solutions were addressed to web applications (e-commerce) and video/image processing software.

Regarding software modelling, the most popular solutions consisted by linear gray-box models. The concept behind it comes from the use of *a priori* knowledge about the process to be modelled and for its unknown aspects, they are estimated from measured data [16]. Among the solutions developed by an analytical approach, non-linear models were the most frequent. A large majority of these solutions considered a discrete time-invariant model.

The parameters identification was mainly defined by an initial phase, in open loop form, with inputs and their corresponding output measures being provided to the estimator. After gathering enough knowledge about the parameters, the estimated results are considered in a subsequent phase with the controller activated. The number of steps reserved to this identification phase varies in each article, but with a clear understanding that it should be as fast as possible in order to accomplish the system requirements.

A general concern in the design of such solutions was dedicated to the model validation at run-time. Additional processes were introduced to deal with cases in which the identified model no longer represents the plant at the current operating point. The solution implemented in [15] and [17] covered a *change point detection mechanism*, allowing the system to identify critical situations that compel the re-initialization of the identification process.

Two main groups of controllers were identified. In the first, in which an adaptive concept is included, the large majority implemented PID controllers to address disturbance rejection and regulatory requirements. The other group of solutions, with a non-adaptive approach, had a significant proportion of controllers which implemented model predictive control (MPC) or limited lookahead control (LLC).

Table 1.1 highlights different solutions applying control theory to create self-adaptive software systems, based in some articles analyzed during the develop-

ment of this work. This table is structured in terms of control strategy aspects and allows to validate the same characteristics verified in [7], previously mentioned. It also provides a common component among most of the solutions, integrating a Kalman Filter to parameter/state estimation.

Although the current solutions implemented by most cloud provider companies does not consider the self-adaptation concept, they present some control techniques that aim to better manage resource usage. With AWS Auto Scaling [18], from Amazon, clients can choose whether they want to optimize the cost or the performance. However, the client must specify the configuration for possible scenarios before real-time execution. HP-UX Workload Manager [19], from Hewlett-Packard, uses a PI controller to establish CPU allocation. In this case, the tuning of controllers must be provided before the workload execution. With Workload Manager [20], from IBM, a client can define the boundary values for each resource type.

The absence of self-adaptation solutions in industry can be comprehended by the lack of generality that still remains on research community solutions. This work offers a different approach to address the optimal resource management problem. With a general concept given by Amdahl's Law, but applied and validated for a particular case, future works can extend the solution to remain feasible and performant for a large range of workload applications.

Table 1.1 – Characteristics of control systems solutions developed in some papers

Article	Type of software system	Model formulation	Estimation implemented	Type of controller designed
[17], [15]	unmanned underwater vehicle system / service-based system for health care / quality and energy control in Video compression / Service Dynamic Binding	set of first order linear model transfer functions.	Kalman Filter	Pole placement
[14]	Video compression / Service Dynamic Binding	linear state space	Kalman Filter	MPC
[21]	Mobile communication / Radar Signal Processing / Service Dynamic Binding	linear state space	*(not applied)* [1]	Multi-dimensional optimization; Deadbeat control
[22]	Dynamic resource provisioning on a server shared by multiple applications	First order auto-regressive model transfer function obtained through an input-output relationship - simple hyperbolic function.	RLS to estimate the two parameters from the AR model	PI controller
[23]	Meeting-Scheduling System	linear discrete-time state-space dynamic model. Model estimated before workload execution.	Kalman Filter	MPC
[4]	Dynamic resource allocation for virtualized computing environments	discrete-time state-space model.	Kalman Filter	LLC
[24]	Queueing Network	continuous linear parameter-varying (LPV) system	*(not applied)*[1]	fixed-parameters PI controller
[25]	*(theoretical paper)*	discrete LPV system	Kalman Filter	same of [17]

1. Adaptation mechanism not defined by parameter/state estimation

1.2 Performance analysis of the control strategy developed by Dell-EMC for a single container execution

The remainder of this chapter focuses on describing the main components of the software developed by Dell-EMC in order to empower the control system allowing the resource allocation for a single application. After the introduction of each component, the company's control strategy is presented as well as its performance analyzed considering different operating points.

The objective of studying this solution is to identify the aspects that must be retained and the ones that must be changed in order to extend its operation to other sorts of iterative workloads and mainly to the execution of multiple applications running in parallel.

1.2.1 Top-level Overview

In this work, the term software architecture defines how the components of a software system are organized and integrated with one another to execute a task.

The software architecture of the system described bellow was developed by the aforementioned company. It was deployed by the control strategy described later in this chapter as well as the ones to be presented in Chapter 3.

1.2.1.1 System requirements for a single container application

At the beginning of this project, the following requirements were identified:

(i) **Main requirement: Optimal resource allocation**
The Service Level Agreement (SLA) imposed to each application is that its allocated amount of resource must lead to an upper bound in execution time, whose reference value is previously provided as an input. Considering the project perspective, where multiple applications running at the same time have their SLA to be managed, the control action must lead each application to track its execution time set-point.

(ii) **Time to evaluate the control input**
Due to the asynchronism between controller action and the start of the execution of an application epoch, the control system must perform its evaluation as soon as possible.

(iii) **Use of data in a recurrent concept**
In order to design a control action even faster, the controller should make

use of the minimum amount of data as possible. Considering the case of this work, with iterative workload applications, the control logic must include only the data provided by the very last execution of an application epoch.

1.2.1.2 Structural Architecture of the System

Assuming a distributed architecture of software applications in which many instances can be built, the plant of the system corresponds to a single **Docker container**, where an application is encapsulated [26]. Inside a **Docker Engine**, the containers run-time, some infrastructure properties of each container can be customized. These includes, for instance, the allocation of number of cores and the maximum amount of memory provided to each container [27].

The main requirement presented above corresponds to a well-known control problem, for which the easiest method to overcome it corresponds to the introduction of a closed loop regarding the plant output.

The closed loop approach supposes the measurement of the system output is available. In the case of a process defined by the execution of a task inside a container, performance metrics cannot be obtained through the Docker run-time. They should be designed as part of the application. In order to analyze these metrics outside the container environment, but in a control system, **Kafka** makes possible data transfer from the inside of a Docker container to a control module. This communication is provided by messages sent from inside of applications and through Kafka architecture, allows another application to read them.

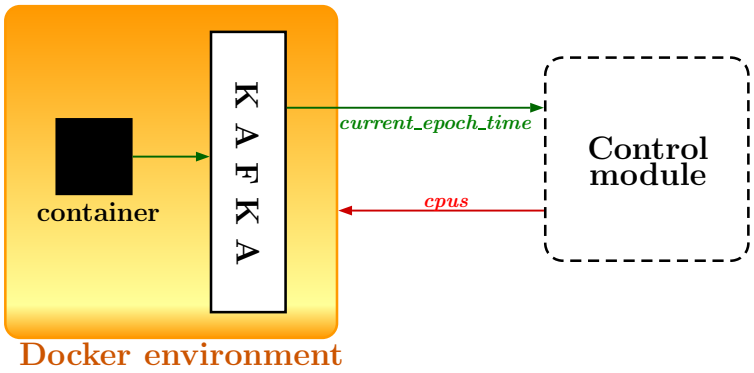


Figure 1.1 – Software architecture.

In all control strategies described in this work, a **Control Module** is designed to determine the number of cores to be allocated in order to satisfy a SLA. The software architecture of the system is displayed in Figure 1.1.

Considering this approach, a container sends a message through **Kafka** to the control module containing the execution time of a task. As a response to this

datum, the control module updates the number of cores of a container directly through a Docker command.

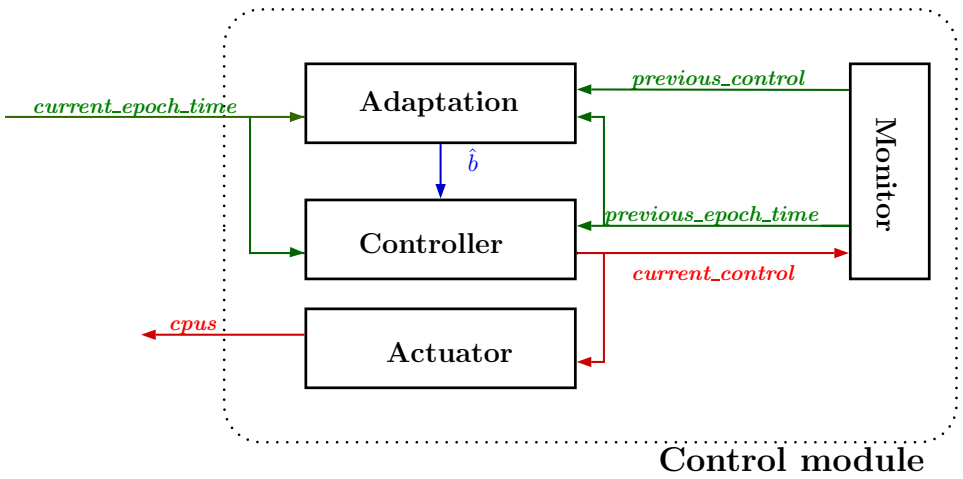


Figure 1.2 – Interfaces of the Control Module.

Without previous knowledge about the parameters of the process performed by a container, an adaptive control strategy was developed to allow the controller to adjust their parameters as long as more data is available. Four different interfaces in the control module are seen in Figure 1.2:

— **Monitor:** Responsible for providing to the Controller the data from the past executed epoch of the container in analysis;

— **Adaptation:** Responsible for updating the estimation of the parameters of the system and for providing to the Controller their estimated values;

— **Controller:** Responsible for evaluating the control input using the current parameters of the system;

— **Actuator:** Responsible for validating the control input determined by the Controller and then, execute the resource allocation update.

The Figure 1.3 shows how all the system components interact with one another to provide at the end the update on the allocated amount of resources.

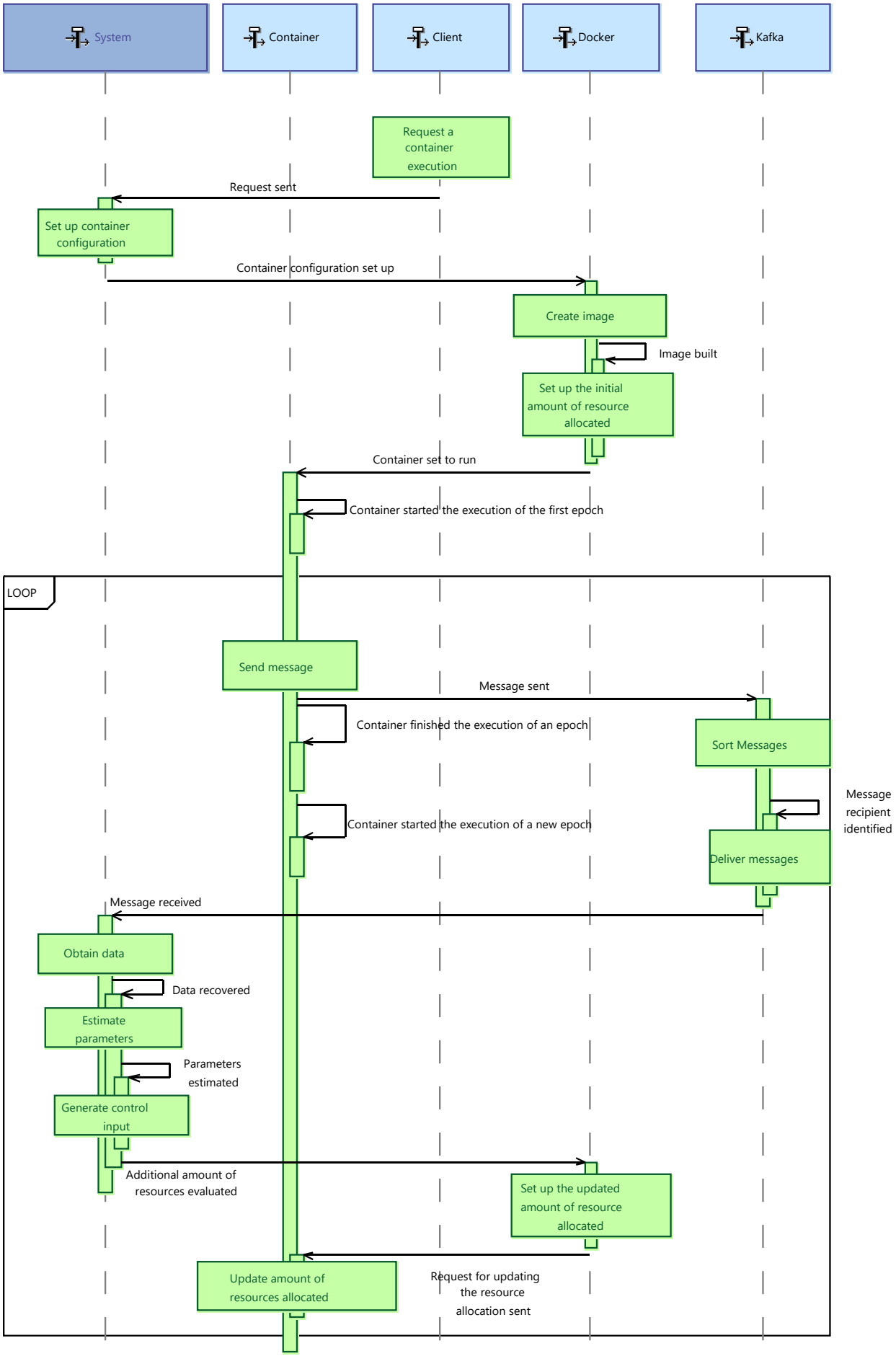


Figure 1.3 – Sequence Diagram (SysML representation) of the system operation.

1.2.2 Control Strategy design

The solution developed by the company is composed of a control strategy, characterized by a simple model describing the system dynamics and by the use of the RLS algorithm to estimate its parameters. After a detailed description of this control strategy, this section presents the tests performed and their outcomes in order to verify the performance of this strategy in relation to the system requirements.

1.2.2.1 Description

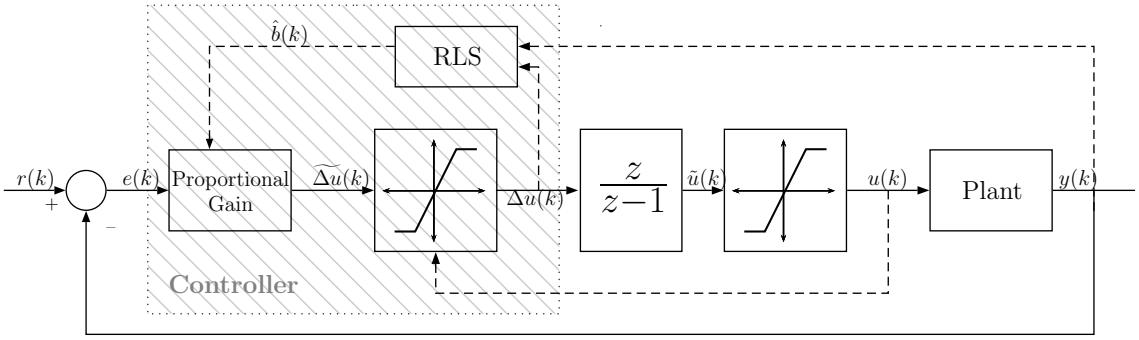


Figure 1.4 – Control system architecture for the process defined by a container

The Figure 1.4 introduces the control system architecture for a Single Input - Single Output (SISO) plant defined by a software system, represented, in this case, by a single container workload. The adaptive concept is due to the change in the controller gain by taking the most recent value of the estimated parameter $\hat{b}$ into account, whose evaluation is recursive.

The first saturation block placed between variables $\widetilde{\Delta u}(k)$ and $\Delta u(k)$ aims to limit the number of cores' increase or decrease in a given epoch. The second block, between variables $\tilde{u}(k)$ and $u(k)$ has the purpose of restricting the total number of cores allocated to a certain set of possible values.

The dynamics of the plant, in this strategy, is modelled in discrete-time domain. Its input is not defined by the number of cores allocated in a given step, but by the increment of number of cores between two subsequent steps - in Figure 1.4 defined by $\Delta u(k)$. Therefore, the state space representation of the system is given as follows:

$$x(k+1) = \alpha x(k) + \beta u(k) \tag{1.1a}$$

$$y(k) = x(k) \tag{1.1b}$$

where $u(k)$ represents the control input, the state variable $x(k)$ in the equation (1.1) corresponds to the execution time of the application in a given epoch k. Furthermore, α and β are the designed model parameters.

Despite the knowledge brought by the state equations (1.1), no *a priori* information about the parameters of the workload characteristics is supposed at run-time. This means that neither α nor β are known at the beginning of the container execution. The lack of this vital data for the control system performance is overcame by the introduction of an estimation mechanism. This classifies the system model developed as a gray-box linear model.

The estimation of the system parameters

In this strategy, the Recursive Least Squares Method is implemented to provide the estimated parameters of the model supposed for the dynamics of the plant, given by the equation (1.1). By choosing $\alpha = 1$, the estimator took the increment of number of cores and its corresponding increment in execution time as input and desirable value, respectively. The following dynamics is then identified by the estimator:

$$x(k+1) - x(k) = \hat{\beta} u(k) \tag{1.2}$$

The estimation process is implemented considering a standard application of the RLS algorithm with regard to the **forgetting factor**. Its value is set at 0.98, which keeps the importance of all previous data, giving to them different but high weights.

Control Law Design and Implementation

The gain introduced by the controller action is simply determined by the inversion of the estimated parameter $\hat{\beta}$ in equation (1.2) applied to the set-point tracking error. This results on the necessary increment in number of cores $\Delta cpus$ to bring the execution time of the process in next epoch to the set-point value, as follows:

$$\Delta cpus(k+1) = \frac{1}{\hat{\beta}(k)} e(k) \tag{1.3}$$

$$e(k) = set\_point - ttf(k) \tag{1.4}$$

The control action in this strategy also presents two different phases regarding the use of the estimated parameter $\hat{\beta}$ on the controller gain evaluation:

— **Identification Phase:** In this phase, the controller gain takes the initial value predefined for $\hat{\beta}$. The estimation is performed in each epoch, but its outcomes are not treated by the controller, resulting in a slow performance of the controller action. Depending on how distant from the steady state the initial allocation is, the estimator may consider a broader range of points while the whole system is still trying to reach its main goal.

— **Adaptation Phase:** The controller applies the current value of the estimated parameter evaluated.

1.2.3 Deployment of a single container execution

Tests were carried out to study the performance of this control strategy, using as workload profile executed inside a container the one described in Appendix A.1.1.

A single test case scenario was considered, in which the reference input for each test case corresponded to a constant value during all test case execution.

The table 1.2 describes the initial configuration considered for the following tests:

Table 1.2 – Key aspects of the control strategy initialization

Total number of cores	4.0
Initial allocated number of cores	4.0
Number of epochs assigned in an initial Identification Phase	First 5 epochs
Total number of epochs	50
Initial value of $\hat{b}$	-30.0
Forgetting Factor (μ)	0.98

1.2.3.1 Tests environment: Physical host Description

Table 1.3 describes the configuration of the machine in which the tests for the solution deployed for a single container were performed.

Table 1.3 – Configuration of the physical host

Operating System	Ubuntu 18.04.2 LTS
CPU	4 Intel(R) Core(TM) I5-3210m cpus @ 2.50GHz
Total Memory	4 GB

1.2.3.2 Test results

This section presents the results from the testing campaign performed using the control strategy mentioned in this chapter. Four test cases were considered with different execution time set-points in order to cover distinct zones of the workload characteristics, generally represented by Figure 1.

The data provided in each test case cover the evolution in time of three main aspects of the system:

— The execution time of the running application ;

— The performance of the estimation;

— The performance of the controller.

The estimation error due to the method implemented on this strategy was also evaluated, in which the real value of the execution time for a given epoch was compared to its estimated value.

In the following figures, the two aforementioned phases designed to associate the estimated parameters with the controller gain evaluation can be easily recognized. The Identification Phase corresponds to the part hatched in gray. The remaining part represents the Adaptation Phase.

Set-point: 12.5s

The delay in achieving a set point tracking error close to zero verified in Figure 1.5 is due to the estimated parameter value that remains distant to its value in steady state. This justifies the smallest control inputs verified between phases 6 and 15.

Set-point: 25.0s

Different from the previous test case, in this one the RLS method enables a faster convergence to the set-point value. At the beginning of the Adaptation Phase, the values of $\hat{\beta}$ are smaller than the one in steady state, leading to the initial

overshoots verified in Figure 1.6. The execution time behavior in the Adaptation Phase was similar to the one verified for the RLS Estimation Error.

Set-point: 50.0

The Figure 1.7 shows that the RLS configuration does not provide an adequate estimation because the RLS estimation error does not converge in steady state. As the estimator tries to fit all past data to the model previously defined, the value of $\hat{b}$ does not stabilize in a constant value after so many epochs performed. This leads to a conclusion that the model defined by equations (1.1) cannot describe properly the dynamics in this operating point.

Even with higher estimation errors, the execution time remains with a certain level of stability and close to the set-point value.

Set-point: 100.0s

Similar to the previous case tests, the Figure 1.8 presents an oscillatory response, but with higher estimation errors and higher amplitude of oscillation during all Adaptation Phase.

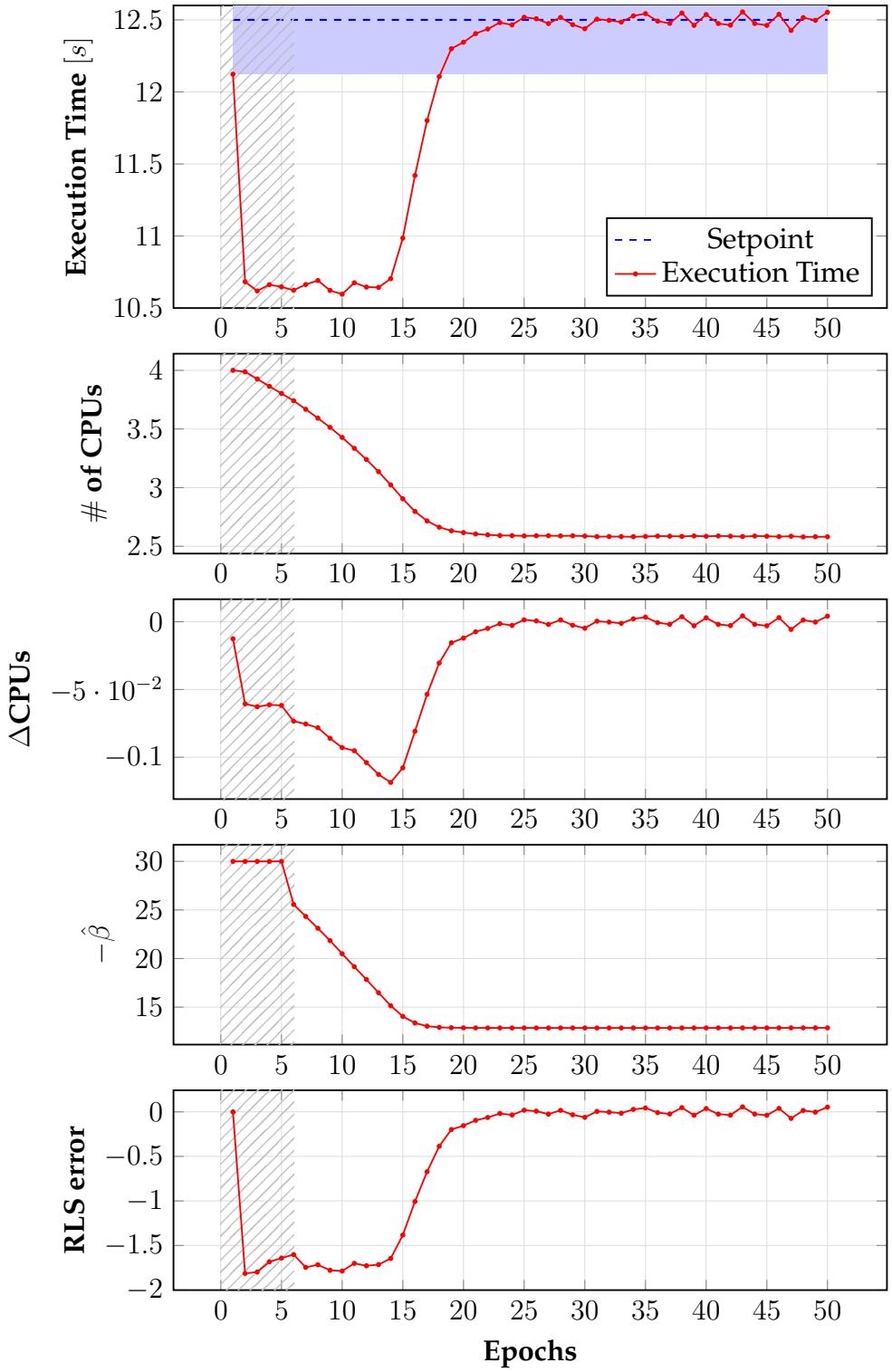


Figure 1.5 – Performance of the control strategy developed by DELL-EMC. Test case executed using set-point= $12.5s$.

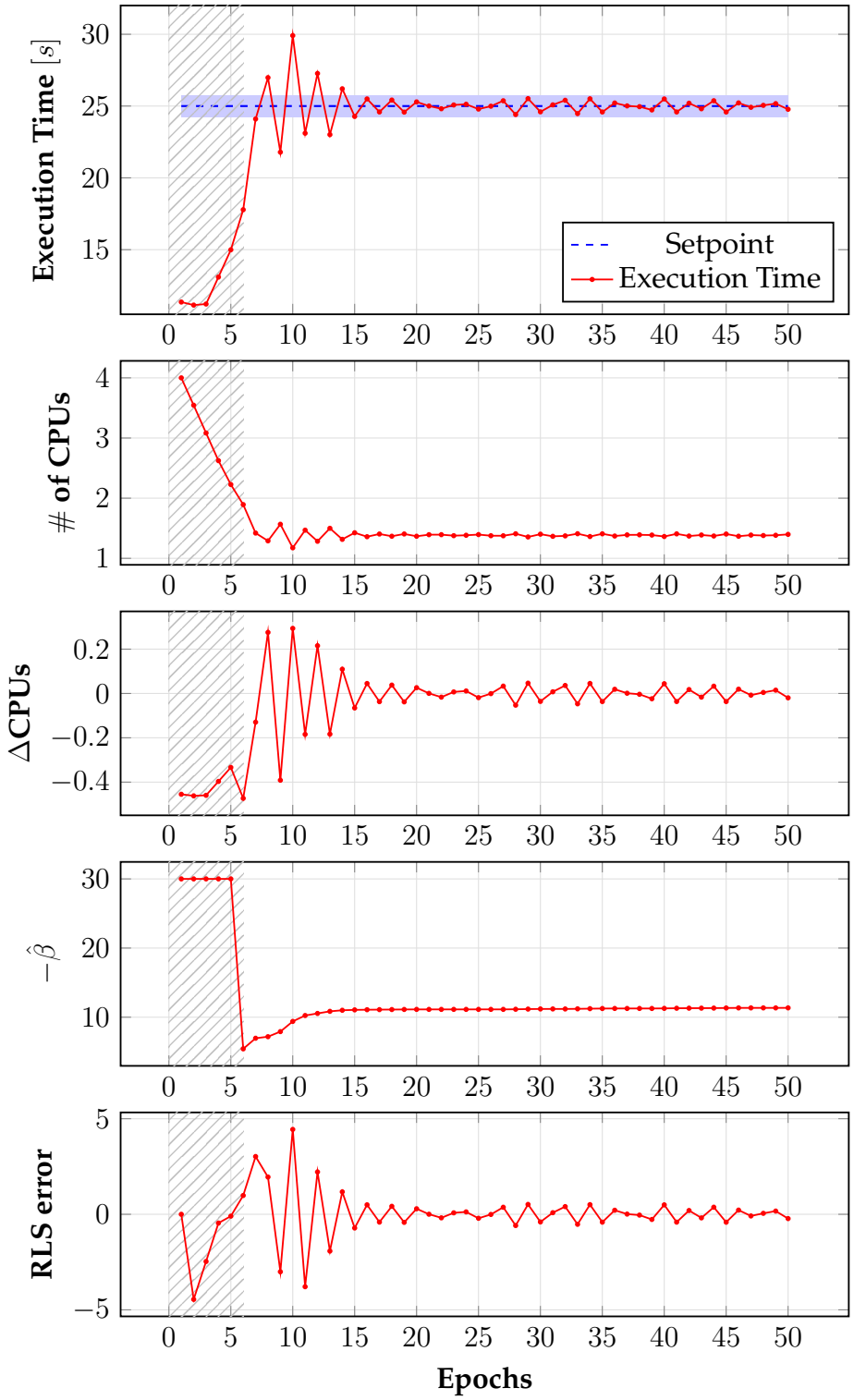


Figure 1.6 – Performance of the control strategy developed by DELL-EMC. Test case executed using set-point $= 25.0s$.

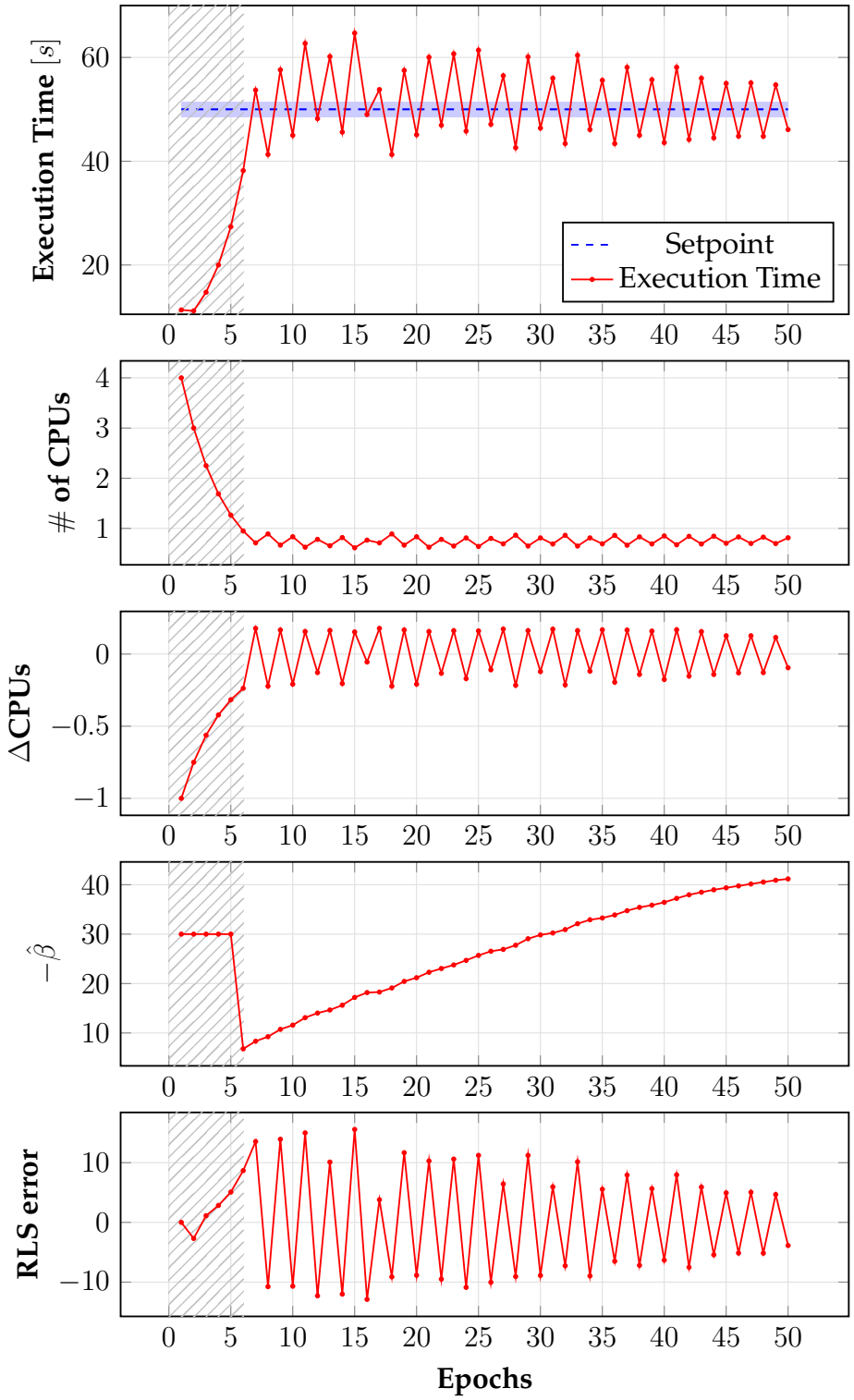


Figure 1.7 – Performance of the control strategy developed by DELL-EMC. Test case executed using set-point= $50.0s$.

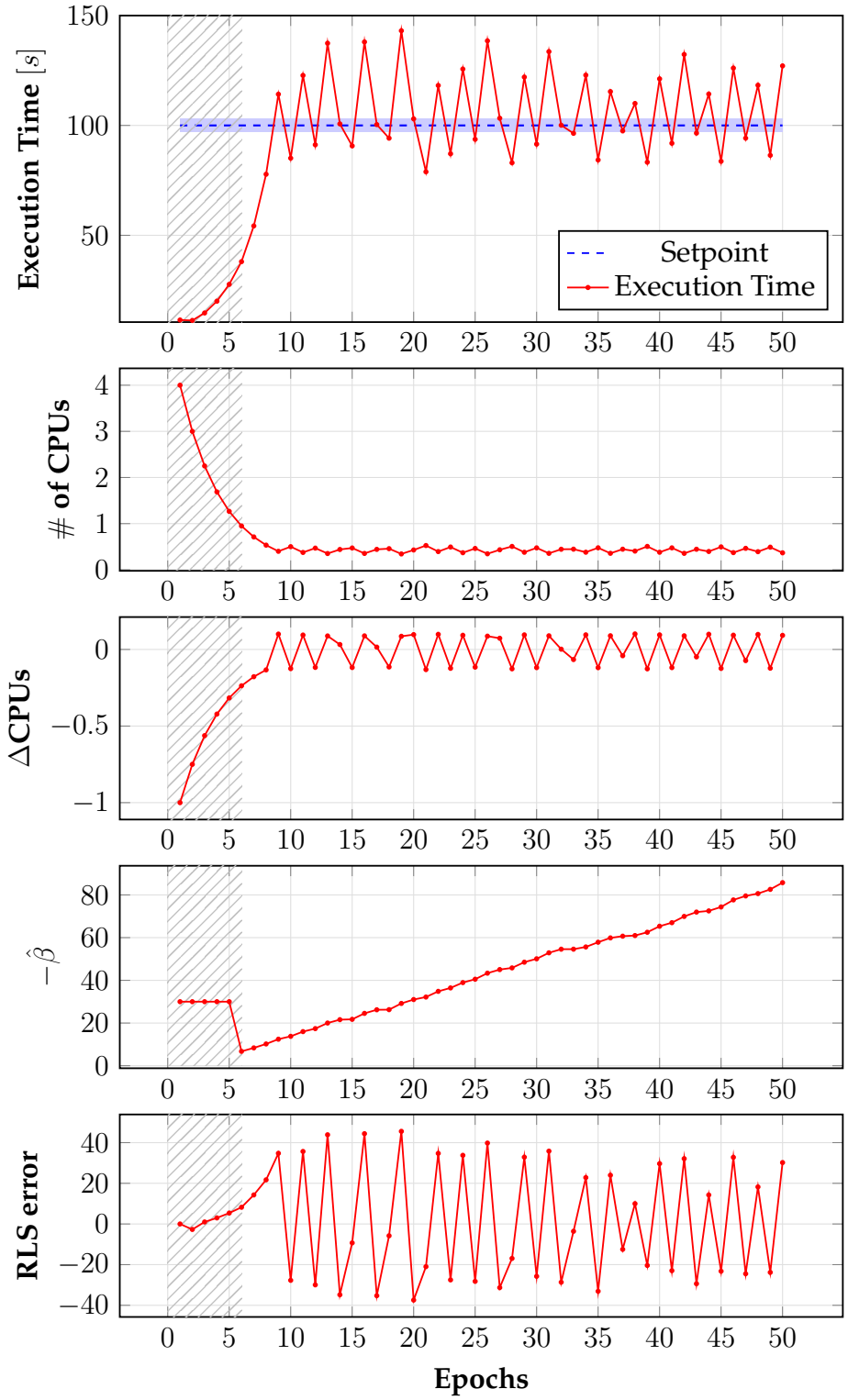


Figure 1.8 – Performance of the control strategy developed by DELL-EMC. Test case executed using set-point= $100.0s$.

1.2.3.3 All set-points considered

The Figure 1.9 reunites all operating points (number of cores, execution time) provided by the execution of the test cases previously presented. It shows a large

range of execution time values for cases with high set-points. This allows to infer that the characteristic can no longer be approximated by a linear behavior with constant parameters.

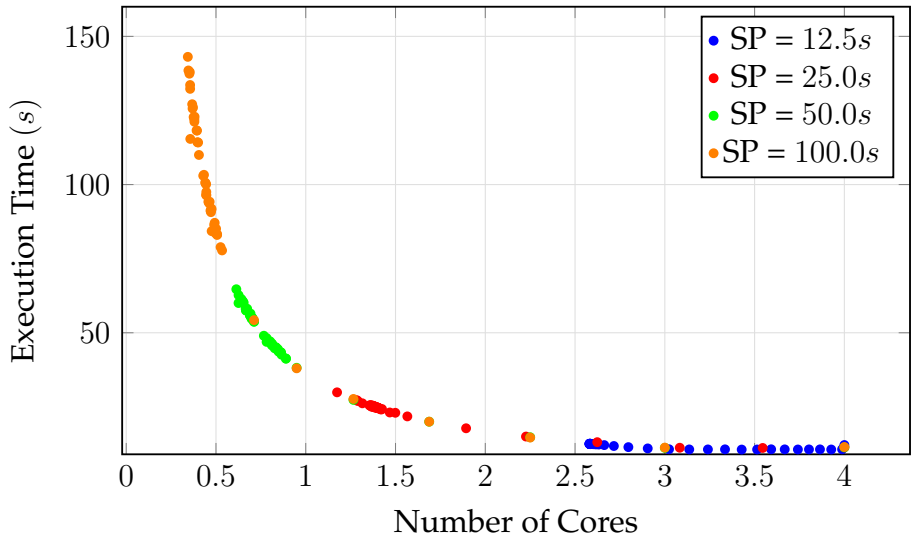


Figure 1.9 – All points recovered from the tests

1.2.3.4 Performance Analysis

Table 1.4 – Performance aspects from the tests performed in the control strategy provided by DELL-EMC

Performance aspects in steady state	**Set-point values**			
	12.5s	**25.0s**	**50.0s**	**100.s**
Set-point tracking error achieved?	YES	YES	NO	NO
Mean value [s]	12.42	25.06	51.39	106.41
Variance [s^2]	0.04	1.32	48.65	351.09
Maximum overshoot [%]	0.45	19.63	29.37	43.09
Minimum undershoot [%]	-8.64	-12.84	-17.43	- 22.21
Mean evaluation time - Estimation and control action [s]	0.22	0.21	0.27	0.27
Estimation characteristics in steady state				
$\hat{\beta}$	constant	constant	variant	variant
RLS estimation error	converges to zero	converges to zero	variant	variant

The adaptive control strategy performed in this chapter was designed to drive the set-point tracking error to zero. It was expected that by the time the execution

time comes to the set-point value, the estimated parameter $\hat{\beta}$ would become constant. This behavior is verified for small set-points, whose operating points stay in the zone whose characteristic Δcpu and $\Delta ExecutionTime$ can be approximated by a linear relationship, as seen in Figure A.3. As long as the desired set-point increases, the linear approximation of this characteristic in these new operating points drastically affects the performance of the control strategy. In Table 1.4, the difference on the system behavior for this particular situation is even more evident: the execution time variance in steady state is considerably excessive for high set-points. Although the mean value stays close to the set-point value, no convergence is verified.

Both estimation and control law use the linear approximation to describe the system dynamics. Moreover, the high value for the forgetting factor used on all these tests considers past data as important for the estimation as the one obtained in the most recent epoch. By doing this, the algorithm is compelled to find a value of β which fits all set of available data. With high set-points, the linear approximation could be used in a more local context than for smaller ones, but in this case, the forgetting factor would also be smaller, resulting in a varying β. Consequently, the control law design would change to consider now a linear and variant process.

Another option to reach the system requirement for all these set-point values remains on the redesign of the system model. Changing the estimator implementation and the controller design may provide a more accurate response to the resource allocation. The next part of this manuscript is dedicated to establish a new hypothesis for the workload characteristics behavior, as well as the definition of a new control strategy able to meet system requirements.

Part I

Adaptive control strategies considering a single container execution approach

Chapter 2

Elements of Adaptive Control

Contents

2.1 Motivation 29
2.2 Adaptive Control System Approaches 30
2.2.1 Gain Scheduling 30
2.2.2 Model Reference Adaptive Controller (MRAC) 31
2.2.3 Self Tuning Regulator or Model Identification Adaptive Controller (MIAC) 32
2.3 Adaptive Control System Design 33
2.4 Stability Analysis: the Lyapunov Theorem 35
2.4.1 The continuous-time system case 35
2.4.2 Extending the Lyapunov Theorem to the discrete-time system case 37
2.5 On-line parameter estimation 37
2.5.1 Recursive Least-Squares (RLS) Method 38

This chapter introduces the key aspects in the design of an adaptive control system. It also discusses how to ensure the formal guarantees well-known in general control techniques, specifically in the context of non-linear control laws associated with estimated parameters.

2.1 Motivation

Most general controllers are designed under the knowledge of the behavior of the system to be controlled. These control techniques are often developed using a preceding off-line identification process to obtain the system parameters. In other cases, they can only be provided instantaneously due to the peculiar changes in their dynamics or due to the lack of previous information. For instance, the dynamical behavior of an aircraft depends on some parameters with

strong variations that can occur during a flight. Rockets experience considerable mass change, since a significant part of it consists of the fuel, expelled during flight at a high velocity, providing its propulsive force. Robot manipulators can deal with large objects with unknown inertial parameters.

To do so, the control design must be associated with an identification technique aiming at progressively better understand the plant to be controlled. If the system identification process is recursive - whose plant model is periodically updated considering previous evaluations and new data - identification and control may be performed concurrently. [1]

Adaptive control is a technique with an on-line parameter estimation process. Its main objective is to maintain consistent performance of the system in the presence of uncertainty or unknown variation in its parameters. As the association between control and on-line estimation can turn the global control system nonlinear, its design and analysis are particularly connected with the Lyapunov Theorem. [2]

2.2 Adaptive Control System Approaches

The main difference of an adaptive controller from an ordinary one remains in its time-variant behavior due to the updating mechanism. The adaptive concept is based on an estimation process which considers the available signals of the system. This section presents some approaches, which are distinct from one another specially in terms of which purpose the control system must fulfill and also the available data to determine the control input.

2.2.1 Gain Scheduling

In this approach, auxiliary process variables which present strong correlation with the changes in the system dynamics are used to modify the parameters of the regulator. The controller parameters are then functions of these auxiliary variables, which must be available. The Figure 2.1 presents the architecture for this approach.

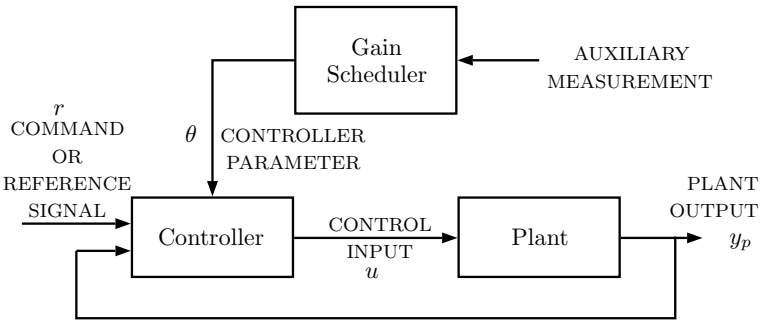


Figure 2.1 – A gain scheduling control system. (Extracted from [1])

As seen in Figure 2.1, the adaptation scheme corresponds to an open-loop process, which can lead the system to an unstable condition.

2.2.2 Model Reference Adaptive Controller (MRAC)

In this approach, different architectures are assembled in two main groups: the ones in which no explicit estimation or identification is performed, known as direct methods, and the ones in which an estimation process is required, known as indirect methods. In this work, only the most usual approach, the Parallel Scheme, an indirect method, is presented.

2.2.2.1 Parallel Scheme

The main innovation of this approach consists of including a reference model such that the plant output can match the reference model output asymptotically. As seen in Figure 2.2, its architecture is composed of two loops:

— an inner loop, which describes an ordinary control loop, including the plant and the controller;

— an outer one, assigned to the adaptation mechanism aimed to provide to the controller the parameters in order that the response of the plant under adaptive control behave exactly as the one of the reference model.

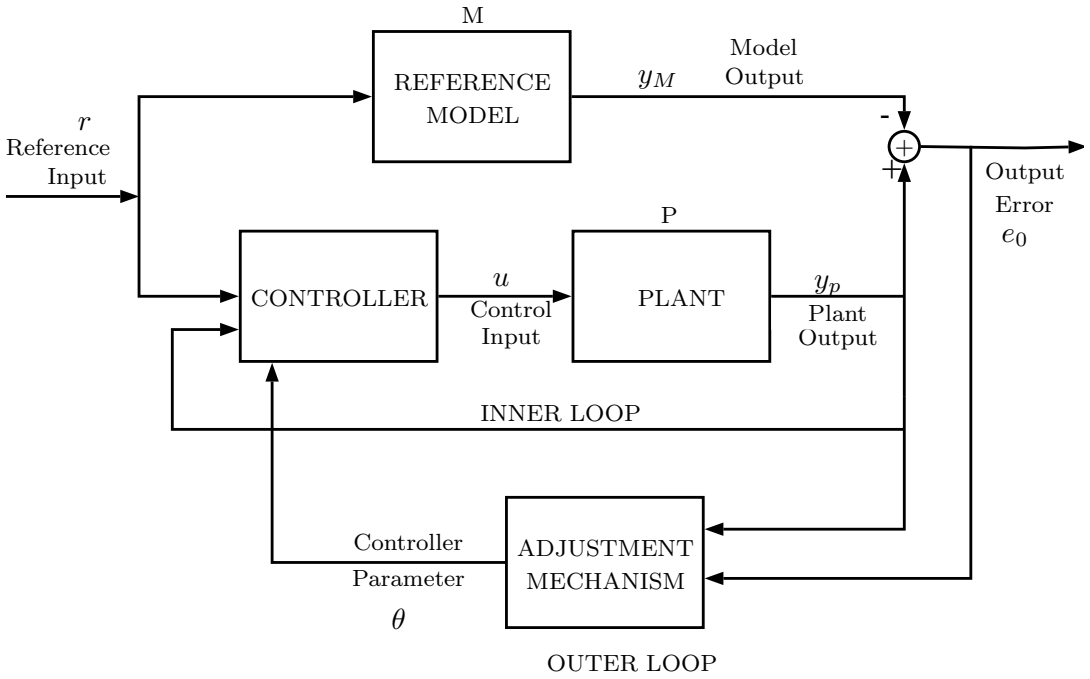


Figure 2.2 – A model-reference adaptive control system. (Extracted from [1])

The main issue in the design phase for this approach is to ensure that the error between the model reference output and the plant output will converge to zero. Different from standard controllers, this phase must also consider the adaptation mechanism to keep the whole system in stable conditions. For the case of this approach, most of the literature refers to the formalism found in the Lyapunov Theorem to establish some guarantees in terms of system stability. As in an example provided in [2], the stability analysis using this theorem includes the adaptation mechanism as an additional state of the system.

2.2.3 Self Tuning Regulator or Model Identification Adaptive Controller (MIAC)

For this approach, the main goal of the control system is to make the plant output behave as the reference input. Similar to the Parallel Scheme approach, this one also consists of two loops.

However, this sort of controller applies a different method in the outer loop. An estimation process is associated on it to estimate the plant's parameters, supposedly unknown, and, finally, apply them in the control input evaluation. With this estimation process integrated into the system architecture, an explicit separation between identification and control is verified.

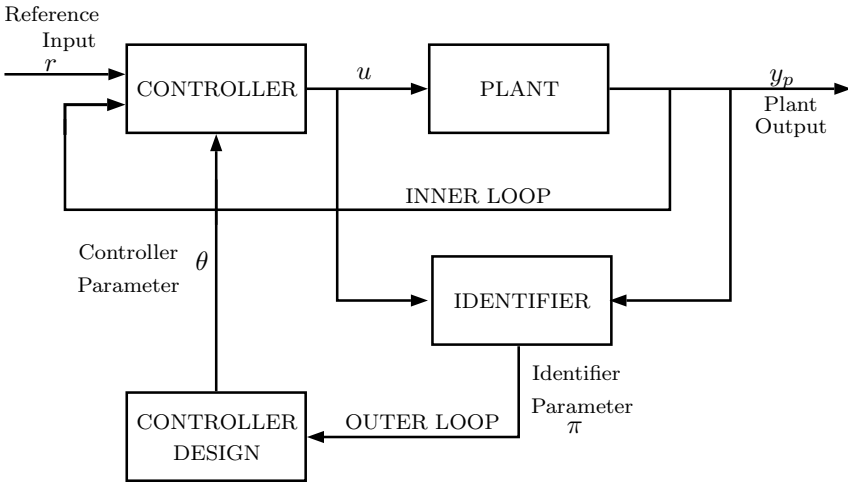


Figure 2.3 – A self-tuning controller. (Extracted from [2])

A disadvantage of this adaptive control approach lies in a more complex analysis of this system than for the model reference schemes, due to the possible nonlinear behavior present in the transformation from Identifier Parameters to Controller Parameters. [1]

2.3 Adaptive Control System Design

The objective of a control system is to determine the input u in order that the output response $y(t)$ of the plant achieves performance requirements. [28]

The design of a control system follows a well-known checklist of steps. When an adaptive control system is taken into consideration, new aspects must be considered on this checklist. The adaptation mechanism integrated into this particular approach has to be carefully designed.

In [28], the authors regroup the control system design in three steps:

Step 1: Modelling

In this step, the process defined by the plant, which generates an output $y(t)$ according to the provided input $u(t)$, is described in the form of mathematical equations. Those equations will constitute the mathematical model of the plant, which are intended to produce the same output response of the real plant when the same inputs and initial conditions are applied on the model plant.

Due to the complexity that this model can bring to the control design, some simplifications are used:

(i) Linearization around operating points, in which the plant is approximated by a linear model capable to describe the system behavior under a certain

operating point. This can be achieved by Taylor's series expansion, fitting of experimental data in a linear model, etc.;

(ii) Model order reduction techniques, in which some effects outside the frequency range are neglected, inducing a model with lower order and consequently, simpler.

According to [3], the task of modelling a system is partitioned in three steps:

(i) **the characterization**, where some hypothesis are considered to establish a mathematical formulation for the system.

(ii) **the identification**, where the parameters of the model defined on the previous step are identified aiming that its behavior and the one of the real system can be the closest as possible, according to a given criteria D. The most recurrent criteria explored for the parameters identification is the sum of the Square Estimation Error, which corresponds to $\sum(y_M - y_S)^2$ looking at the identification process represented in Figure 2.4.

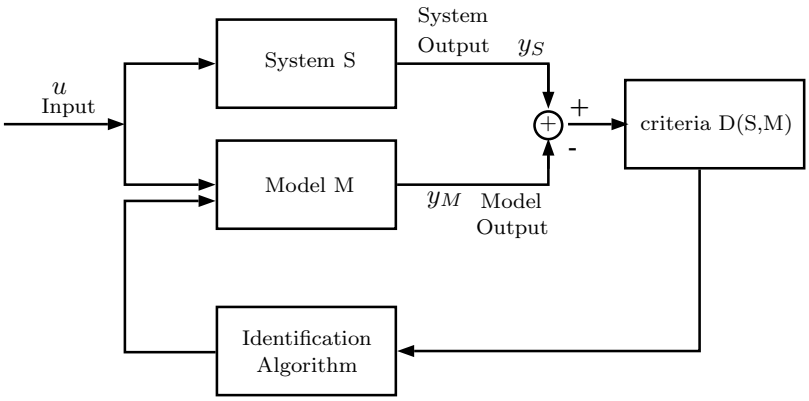


Figure 2.4 – Identification Process Scheme. (Translated from [3])

(iii) **the validation**, where some tests are performed to compare how similar to the real system is the behavior of the identified model. One of these tests consists of submitting the model to different conditions, in particular those not explored in the identification phase.

Step 2: Controller Design

Once the model of the plant is available, the controller can be designed. In control design, first a controller structure is chosen and then its parameters are computed based on the knowledge provided by the parameters of the plant.

Considering the self tuning regulators, in particular, the control input is evaluated given the parameters provided by an adaptation law. Therefore, the quality of this estimation has a huge impact in the controller action, affecting the ability of the system to meet its goals.

In [2], the authors gather in three main points the adaptive controller design:

— the choice of a control law containing variable parameters;

— the choice of an adaptation law for adjusting those parameters;

— the analysis of convergence properties of the resulting control system.

Regarding the self tuning regulators, the first two steps presented above are quite straightforward. However, the analysis, mainly executed by using Lyapunov Theorem, may not provide easy conditions to validate the system stability.

Step 3: Implementation

In this phase, the controller designed in Step 2, is applied to the real plant. It is also in this phase that final adjustments, often called tuning, are implemented to improve the performance of the whole system.

2.4 Stability Analysis: the Lyapunov Theorem

In the last section, a key point of the design of an adaptive control system is mentioned: the analysis of the convergence properties of the resulting control system, whose controller can be usually described as a non-linear function.

The most popular way to verify the system stability on such condition is through an analysis where the Lyapunov Theorem is applied. This section enunciates it for the continuous-time system case, based in [29], and then extends it for the discrete time system case, based in [30].

2.4.1 The continuous-time system case

Considering the autonomous system:

$$\dot{x} = f(x) \tag{2.1}$$

where $f : D \to R^n$ is a locally Lipschitz map from a domain $D \subset R^n$ into R. Supposing that $x = x_0$ is an equilibrium point, the main objective is to characterize and study the system stability at this point.

For simplicity, the following definitions and theorems consider the origin of R^n: $x_0 = 0$ as the equilibrium point. By changing variables, the following analysis can be extended to any point.

Definition 1 *The equilibrium point $x_0 = 0$ of (2.1) is:*

— **stable** *if, for each $\epsilon > 0$, there is $\delta = \delta(\epsilon) > 0$ such that:*

$$\| x(0) \| < \delta \Rightarrow \| x(t) \| < \epsilon \quad , \quad \forall t \geq 0 \tag{2.2}$$

— **unstable** *if it is not stable;*

— **asymptotically stable** *if it is stable and δ can be chosen such that:*

$$\| x(0) \| < \delta \Rightarrow \lim_{t \to \infty} x(t) = 0 \tag{2.3}$$

The Lyapunov Theorem takes advantage of a certain class of functions to determine the stability of an equilibrium point. Let $V : D \to R$ be a continuously differentiable function defined in a domain $D \subset R^n$ that contains the origin. The derivative of V along the trajectories of (2.1), denoted by $\dot{V}(x)$, is given by:

$$\dot{V}(x) = \sum_{1}^{n} \frac{\partial V}{\partial x_i} \dot{x_i} = \sum_{1}^{n} \frac{\partial V}{\partial x_i} f_i(x) \tag{2.4}$$

If $\dot{V}(x)$ is negative, V will decrease along the solution of (2.1).

Theorem 1 *Let $x = 0$ be an equilibrium point for (2.1) and $D \subset R^n$ be a domain containing $x = 0$. Let $V : D \to R$ be a continuously differentiable function such that*

$$V(0) = 0 \text{ and } V(x) > 0 \text{ in } D - \{0\} \tag{2.5}$$

$$\dot{V}(x) \leq 0 \text{ in } D \tag{2.6}$$

then $x = 0$ is stable. Moreover, if

$$\dot{V}(x) < 0 \text{ in } D - \{0\} \tag{2.7}$$

then $x = 0$ is asymptotically stable.

A Lyapunov function $V(x)$ which satisfies the conditions in (2.5) is said to be *positive definite*. The most popular Lyapunov functions considered has a quadratic form, as follows:

$$V(x) = x^T Q x \tag{2.8}$$

where Q is a symmetric positive definite matrix.

2.4.2 Extending the Lyapunov Theorem to the discrete-time system case

Considering a system of difference equations:

$$\begin{aligned} x(k+1) &= f(x(k)) \\ f(0) &= 0 \end{aligned} \tag{2.9}$$

the same conclusions about the stability of operating points can be established, but instead of analyzing the derivative of the Lyapunov function $V(x)$, the difference $\Delta V(x) = V(f(x)) - V(x)$ is taken into account. Hence, it can be enunciated as follows:

Theorem 2 *Let $x = 0$ be an equilibrium point for (2.9) and $D \subset R^n$ be a domain containing $x = 0$. Let $V : D \to R$ be a continuous function such that*

$$V(0) = 0 \text{ and } V(x) > 0 \text{ in } D - \{0\} \tag{2.10}$$

$$\Delta V(x) = V(f(x)) - V(x) \leq 0 \text{ in } D \tag{2.11}$$

then $x = 0$ is stable. Moreover, if

$$\Delta V(x) = V(f(x)) - V(x) < 0 \text{ in } D - \{0\} \tag{2.12}$$

then $x = 0$ is asymptotically stable.

2.5 On-line parameter estimation

Parameters estimation is useful in cases where the dynamical system presents uncertainties, or even lack of information. This process can be executed off-line, when there is enough time to perform the estimation before executing a specific task, or on-line, when these two processes are performed concurrently.

Giving the solution to be developed in this work, where the new control strategy implements a self tuning regulator, an on-line parameter estimation composes its architecture.

To be applied on the estimator algorithm, it is mandatory an estimation model that relates the available data to the unknown parameters. In addition, this model must also have a linear parameterization form, as follows:

$$\mathbf{y}(t) = \mathbf{W}(t)\mathbf{a} \tag{2.13}$$

where $\mathbf{y}(t)$ corresponds to the outputs of the system, $\mathbf{a}$ contains the unknown parameters to be estimated and $\mathbf{W}(t)$ is a signal matrix.

In on-line estimation, the evaluations are performed recursively, implying that the estimated value of $\hat{\mathbf{a}}$ is updated once a new set of data $\mathbf{y}$ and $\mathbf{W}$ is available. Due to its ability to deal with multiple parameters without loss of convergence properties, the Recursive Least Squares method, chosen to integrate the control strategy to be designed in this project, is introduced in the following subsection, based on [31].

2.5.1 Recursive Least-Squares (RLS) Method

When dealing with a recursive estimation, the length of observable data is variable. The cost function $\mathscr{E}(n)$, where n represents the variable length of observable data, introduces the weighting factor β, as follows:

$$\mathscr{E}(n) = \sum_{i=1}^{n} \beta(n,i)|e(i)|^2 \tag{2.14}$$

where $e(i)$ is the difference between the desired response $d(i)$ and the output $y(i)$ produced as a result of the estimated parameters provided by the algorithm. So, $e(i)$ can be expressed as:

$$\begin{aligned} e(i) &= d(i) - y(i) \\ &= d(i) - \mathbf{w}^T(n)\mathbf{u}(i) \end{aligned} \tag{2.15}$$

in which $\mathbf{u}(i)$ is the input vector at time i, defined by:

$$\mathbf{u}(i) = [u(i), u(i-1), ..., u(i-M+1)]^T \tag{2.16}$$

and $\mathbf{w}(n)$, the weight vector at time n, defined by:

$$\mathbf{w}(n) = [w_0(n), w_1(n), ..., w_{M-1}(n)]^T \tag{2.17}$$

The weighting factor $\beta(n,i)$ composes the cost function in order that the data

in a distant past can be "forgotten", which enables the estimator to follow the statistical variations of the observable data. The form of this factor is commonly expressed as:

$$\beta(n,i) = \lambda^{n-i}, \quad i = 1, 2, ..., n \tag{2.18}$$

where λ is a positive constant close to 1. If $\lambda = 1$, this configures the ordinary method of least squares. In this case, β is called as exponential weighting factor or forgetting factor. The cost function to be minimized can be expressed as:

$$\begin{aligned} \mathscr{E}(n) &= \sum_{i=1}^{n} \lambda^{n-i} |e(i)|^2 \\ &= \sum_{i=1}^{n} \lambda^{n-i} [d(i) - \mathbf{w}^T(n)\mathbf{u}(i)]^2 \end{aligned} \tag{2.19}$$

By changing variables to:

$$\mathbf{u}'(i) = \sqrt{\lambda^{n-i}}\mathbf{u}(i) \quad \text{and} \quad d'(i) = \sqrt{\lambda^{n-i}}d(i) \tag{2.20}$$

the cost function can be rewritten as:

$$\mathscr{E}(n) = \sum_{i=1}^{n} [d'(i) - \mathbf{w}^T(n)\mathbf{u}'(i)]^2 \tag{2.21}$$

corresponding to the standard Least-Squares criterion. Then, the solution can be obtained as:

$$\begin{aligned} \mathbf{w}^T(n) &= (\sum_{i=i_1}^{n} \mathbf{u}'(i)\mathbf{u}'(i)^T)^{-1} \sum_{i=i_1}^{n} \mathbf{u}'(i)d'(i) \\ &= [\Phi(n)]^{-1}\psi(n) \end{aligned} \tag{2.22}$$

in which

$$\Phi(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i)\mathbf{u}(i)^T \tag{2.23}$$

$$\psi(n) = \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i)d(i) \tag{2.24}$$

The recursive idea in this algorithm is applied by rewriting the variables $\Phi(n)$ in (2.23) and $\psi(n)$ in (2.24) as functions of $\Phi(n-1)$ and $\psi(n-1)$, respectively:

$$\begin{aligned} \Phi(n) &= \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i)\mathbf{u}(i)^T = \lambda \sum_{i=1}^{n} \lambda^{n-1-i} \mathbf{u}(i)\mathbf{u}(i)^T + \mathbf{u}(n)\mathbf{u}(n)^T \\ &= \lambda\Phi(n-1) + \mathbf{u}(n)\mathbf{u}(n)^T \end{aligned} \tag{2.25}$$

$$\begin{aligned} \psi(n) &= \sum_{i=1}^{n} \lambda^{n-i} \mathbf{u}(i)d(i)^T = \lambda \sum_{i=1}^{n} \lambda^{n-1-i} \mathbf{u}(i)d(i) + \mathbf{u}(n)\mathbf{d}(n) \\ &= \lambda\psi(n-1) + \mathbf{u}(n)\mathbf{d}(n) \end{aligned} \tag{2.26}$$

Lemma 1 (The matrix inversion formula) *Let **A** and **B** be two positive-definite M×M matrices related by:*

$$A = B^{-1} + CD^{-1}C^T \tag{2.27}$$

*where **D** is another positive-definite N×M matrix, and **C** is an M×N matrix. Then, the inverse of the matrix **A** is expressed as follows:*

$$A^{-1} = B - BC(D + C^T BC)^{-1} C^T B \tag{2.28}$$

Applying the matrix inversion formula (2.28) in (2.25), the following expression is obtained:

$$\Phi^{-1}(n) = \lambda^{-1}\Phi^{-1}(n-1) - \frac{\lambda^{-2}\Phi^{-1}(n-1)\mathbf{u}(n)\mathbf{u}^T(n)\Phi^{-1}(n-1)}{1 + \lambda^{-1}\mathbf{u}^T(n)\Phi^{-1}(n-1)\mathbf{u}(n)} \tag{2.29}$$

For convenience of computation, let

$$\mathbf{P}(n) = \Phi^{-1}(n) \tag{2.30}$$

and

$$\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)}{1+\lambda^{-1}\mathbf{u}^T(n)\mathbf{P}(n-1)\mathbf{u}(n)} \tag{2.31}$$

Using these two definitions, equation (2.29) can be rewritten as follows:

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^T(n)\mathbf{P}(n-1) \tag{2.32}$$

The time-update equation for the weight vector can be obtained by using equations (2.22), (2.24) and (2.30):

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}(n)\alpha(n) \tag{2.33}$$

where

$$\alpha(n) = d(n) - \mathbf{u}^T(n)\mathbf{w}(n-1) \tag{2.34}$$

The Algorithm 1 presents a summary of the RLS algorithm deployed to estimate model parameters.

Algorithm 1: RLS algorithm

Input : The input samples $u(i)$ to compose $\mathbf{u}(n)$;
the set of desired response $d(n)$;
the forgetting factor λ;
the small positive constant δ.

Output: The vector $\mathbf{w}(n)$

Initialize the algorithm by setting $\mathbf{P}(0) = \delta^{-1}\mathbf{I}$ and $\mathbf{w}(0) = \mathbf{0}$
for *each instant of time*, $n = 1, 2, ...$ **do**
  $\mathbf{k}(n) = \frac{\lambda^{-1}\mathbf{P}(n-1)\mathbf{u}(n)}{1+\lambda^{-1}\mathbf{u}^T(n)\mathbf{P}(n-1)\mathbf{u}(n)}$
  $\alpha(n) = d(n) - \mathbf{u}^T(n)\mathbf{w}(n-1)$
  $\mathbf{w}(n) = \mathbf{w}(n-1) + \mathbf{k}(n)\alpha(n)$
  $\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \lambda^{-1}\mathbf{k}(n)\mathbf{u}^T(n)\mathbf{P}(n-1)$

Chapter 3

Adaptive Control Strategies

Contents

3.1 System Model Design 43
3.1.1 The mathematical formulation of the input-output relationship 43
3.1.2 The Curve Fitting Process 44
3.1.3 Definition of a System Model 48
3.2 Common Control System Block Diagram 48
3.3 Stability analysis for a Self Tuning Regulator 49
3.4 Adaptive Control Strategies 53
3.4.1 Control strategies considering the workload characteristics described by a hyperbolic function 53
3.4.2 Control strategies considering the workload characteristics described by an exponential function 61
3.5 Synthesis of all control strategies 68

The control strategy analyzed in section 1.2 includes in its architecture an online parameters estimation of the dynamic model proposed for the plant. In this solution, important variation on the estimator error response is verified while the whole system tries to reach set-point tracking, specially in cases with high reference values. That performance led to the conclusion that the dynamic model proposed, for such cases, does not describe accurately the real behavior of the plant.

This chapter is devoted to design new control strategies. Different from the strategy presented in section 1.2, they are designed using new models for the process described by a container execution. Models for the input-output relationship are obtained through a Curve Fitting Process, described and analyzed at the beginning of this chapter. The validated models are then extended to determine new system dynamics. As a result, new control laws are obtained.

Two different approaches direct the design of control laws. First, an approximation method using the first order Taylor expansion is applied to obtain the dynamics of the plant around an operating point. In the second approach, a different dynamics is obtained by simple manipulation of the model equations.

The new designed control laws require model parameters to determine the control input value. The estimation process of these parameters is redesigned to consider the input-output relationship as regression functions. All these changes are proposed to make these strategies provide the right allocation for the largest set of operating points as possible.

3.1 System Model Design

This section starts by assuming two different mathematical equations for the input-output relationship of the system. This characteristic, allowing the execution time to be expressed as a function of the number of cores, is the representation of Amdahl's Law. This section verifies if this curve can be modelled, with a good level of accuracy, by these standard functions. To do so, curve fitting processes are analyzed by taking the data obtained through the execution of the container application described in section A.1.1.

Through these processes, a most suitable mathematical model can be obtained and generalized to a large range of workloads. The validation of a model which represents the plant allows the implementation of a parameter estimator and the design of new control laws.

3.1.1 The mathematical formulation of the input-output relationship

The shape of the curve presented in Figure 1 suggests the relationship between the execution time of a workload and its allocated resource to an exponential or a hyperbolic function.

Considering all necessary shifts applied on their primitive functions, the potential candidates which better suits the behavior verified in the aforementioned figure have the following forms:

— for the hyperbolic function:

$$TTF(k) = \alpha + \frac{\beta}{cpus(k) + \gamma} \tag{3.1}$$

— for the exponential function:

$$TTF(k) = \alpha e^{\beta cpus(k)} + \gamma \tag{3.2}$$

where α, β and γ are parameters to be identified. $TTF(k)$ represents the execution time at given epoch k and $cpus(k)$ is the related amount of allocated resources.

3.1.2 The Curve Fitting Process

Based on the model identification process, schematized in a general form in Figure 2.4, this process considers the two functions defined by equations (3.1) and (3.2). The validation of a model for the input-output relationship is established by finding parameters which provides the smallest estimation error as possible.

3.1.2.1 The optimization performed

To obtain the parameters which satisfy this goal, the optimization process considered as its cost function the sum of the square estimation error. The whole curve fitting process is described as follows:

Algorithm 2: Curve Fitting Process Algorithm

Input : vector **param** containing the set of initial values for the parameters of the model;
The input samples $cpus(k)$;
the set of desired response $ttf(k)$;

Output: The resultant vector **param**

1 **for** k *epochs* **do**
2 compute optimization considering as cost function $J = \sum_{i=1}^{k} [ttf(i) - \hat{ttf}(cpus(i), param)]^2$
3 **param** $\leftarrow$ resultant vector from optimisation

In the cost function presented in Algorithm 2, $ttf(i)$ corresponds to the execution time measured and $\hat{ttf}(i)$, its estimated value given by the model in analysis (described by equations (3.1) and (3.2)). In addition to $ttf(i)$, this algorithm requires as input the resource allocated $cpus(i)$ for the k epochs considered and the vector **param**.

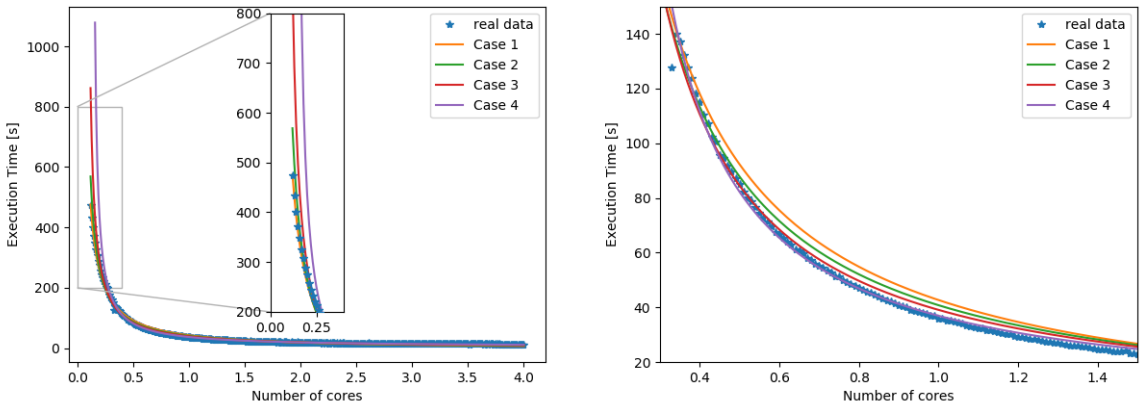
The optimization method applied in this process was the Nelder-Mead one, given that this algorithm does not demand much information for its performance which, in the case of this work, are not available. Examples of unavailable in-

formation for the plant in study are the first and second derivative of the cost function.

The next two sections provide the analysis of this curve fitting process carried out for both candidate functions.

3.1.2.2 Curve Fitting Performance considering a hyperbolic function

A high MSE value is verified when a hyperbolic function is imposed to the static characteristic form in the Curve Fitting Process using the whole set of input data. The optimal result tries to provide small estimation errors specially for the points with high execution time, responsible for the highest errors. Even though the optimization process first takes into account the points whose allocated amount of cores is the highest, the optimal solution neglects the points at the vicinity of the knee point of the curve. In this zone, a significant difference is verified when compared with the real characteristic.



(a) Overview of the whole set of points performed. Emphasis on points with small input values

(b) Difference verified in curves obtained through an optimization process compared to the one established by the real data at the vicinity of knee point

Figure 3.1 – Real characteristic and resultant curves from the Curve Fitting Processes performed considering a hyperbolic function for the plant

In order to analyze the influence of taking the smallest input value on the optimization performance, three additional processes were conducted differing from one another only by their smallest input values applied on each Curve Fitting Process. The Figure 3.1 allows to compare the execution time obtained by the curve fitting process considering the four cases with the real data obtained experimentally.

Table 3.1 – Mean Squared Error (MSE) and Mean Relative Error (MRE) for the estimation considering a hyperbolic function model

Case	1	2	3	4
minimum amount of cores considered	0.1	0.15	0.2	0.3
MSE in the last iteration of the optimization [s^2]	32.49	17.75	8.02	2.21
MRE in the last iteration of the optimization [$\%$]	17.72	12.95	9.24	5.74

Comparing the MSE value obtained in the last iteraction of the curve fitting process for all cases, a reduction of almost 75.32% is verified from case 1 to case 3. For the MRE value, the reduction verified was $47,86\%$. In table 3.1, the MSE and MRE values for all cases analyzed are displayed.

The model validation is strongly connected to the assurance of small errors regarding the experimental data. As it is desirable that the range of possible input values be as large as possible, the **Case 3** represents the best compromise regarding these two aspects.

The characteristics of the curve obtained in Case 3 are :

— Minimum value of cores allocated to the application: 0.2

— Parameters (regarding the function defined in equation (3.1)) obtained at the last iteraction of the Curve Fitting Process:

$$\alpha = 0.8923$$
$$\beta = 35.1970$$
$$\gamma = -0.0791$$

3.1.2.3 Curve Fitting Performance considering an exponential function

The same cases described for the hyperbolic function case were also performed for the exponential one. Different from the first, the case in which all set of data is used in the Curve Fitting process presents an underestimation of execution time for points under the knee point. The same is verified for the smallest value of resources applied as an input. However, this case provides the highest MSE verified for both functions.

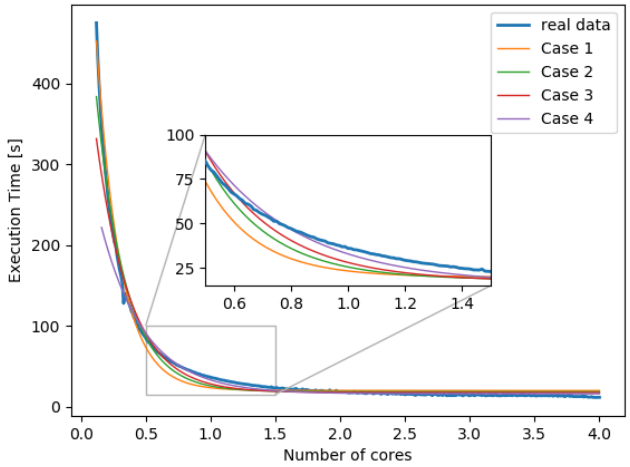


Figure 3.2 – Real characteristic and resultant curves from the Curve Fitting Processes performed considering an exponential function for the plant

Table 3.2 – MSE and MRE for the estimation considering an exponential function model

Case	1	2	3	4
minimum amount of cores considered	0.1	0.15	0.2	0.3
MSE in the last iteration of the optimization [s^2]	91.45	40.69	25.04	10.63
MRE in the last iteration of the optimization [%]	27.15	20.64	16.62	10.82

Comparing these four cases, the smallest MSE value achieved is found in **Case 4**, with the highest value of minimum resource to be allocated. Even in this case, the MSE value are close to the worst case verified for the hyperbolic candidate function.

The characteristics of the curve obtained in Case 4 are :

— Minimum value of cores allocated to the application: 0.3

— Parameters (regarding the function defined in equation (3.2)) obtained at the last interaction of the Curve Fitting Process:

$$\alpha = 330.6000$$
$$\beta = -2.9638$$
$$\gamma = 15.8435$$

3.1.3 Definition of a System Model

Considering the steps of a control system design, it is paramount to determine a mathematical model for the plant in order that all further development can take advantage of this knowledge. For the case of this work, a static relationship is intended to have its parameters identified as part of the resultant control system to be developed. Among all models provided as results of these processes, the best one corresponds to the case 3 supposing a hyperbolic function.

In the search for achieving smaller MSE values in curve fitting processes, a correlation was identified with the minimum amount of resources considered on each case. Regarding the design of the control system, this can be understood as a requirement to satisfy an adequate performance in parameters estimation. Another challenging aspect for the next phase of the control strategy design corresponds to the highest local MSE verified for small input values. In this zone, the estimated values may present a varying and undesirable behavior to decrease the estimation error .

3.2 Common Control System Block Diagram

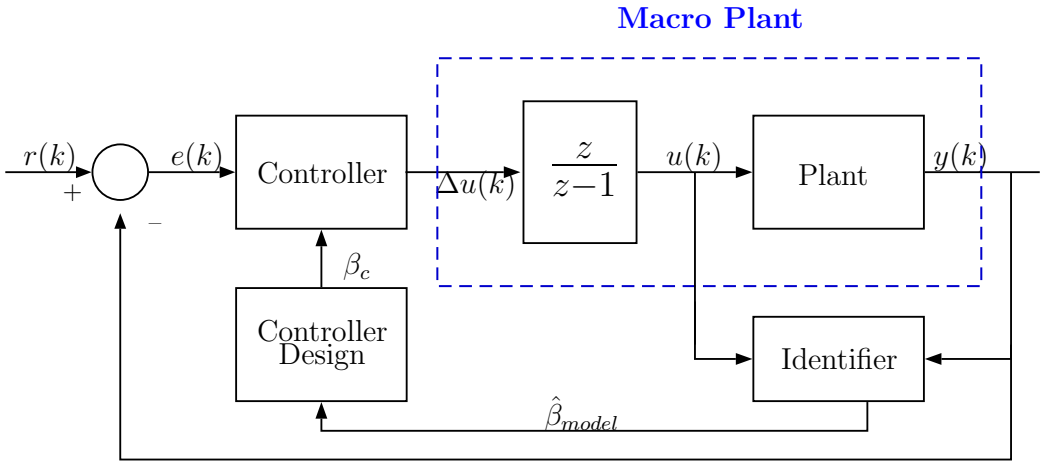


Figure 3.3 – Control system block diagram for all strategies to be presented in this chapter

Before presenting the characteristics of each new control strategy designed, this section focuses on their similar properties regarding the control system block diagram, presented in Figure 3.3.

From the block diagram exposed above, it is important to distinguish two terms that are usually mentioned from now on. *Plant* refers to the process described by the static characteristic between the number of cores $u(k)$ allocated and the execution time $y(k)$ for the same epoch in analysis. Moreover, *Macro*

Plant refers to the dynamic model of this process, considering now as input the increment of number of cores $\Delta u(k)$.

Except from the presence of saturations, hidden for sake of clarity in Figure 3.3, visible differences are identified when compared to the control system block diagram presented in Figure 1.4. The adaptation mechanism was reformulated with the following characteristics to contribute to the control input evaluation:

— **the regression model implemented on the RLS algorithm**
The RLS algorithm, represented in Figure 3.3 by the block *Identifier*, is used to estimate the parameters of the mathematical model describing the input-output relationship of the *Plant* for a given epoch (Equations (3.1) and (3.2)).

The great advantage of this new regression model is that it is considered as a linear model with constant parameters. The use of the RLS algorithm to identify the parameters of the *Macro Plant* dynamic model resulted on poor performances for the whole control system. This was due to strong variations verified in its parameters specially during transient response.

Furthermore, a new approach is considered for the integration between estimation and control action. Different from the previous strategy presented in section 1.2, the new control strategies do not implement an initial Identification Phase. The values provided by the estimator are applied on the controller design since the first epoch execution.

— **the parameter applied on the control law to determine the control input**
In the architecture defined in section 1.2, the estimated parameter is directly applied on the control input evaluation. Different from that approach, the new strategies make use of the parameter provided by the estimator, $\hat{\beta}_{model}$, to compute β_c. This new parameter is intended to adapt the estimated parameters of *Plant* to be integrated as a *Macro Plant* parameter.

3.3 Stability analysis for a Self Tuning Regulator

Another similarity among most of the control strategies to be presented in this chapter lies in the fact that the proposed dynamic models of the *Macro Plant* are expressed in two general forms:

$$x(k+1) = x(k) - \beta_c(u(k), u(k+1), \beta_{model}) f(u(k)) \Delta u(k) \tag{3.3}$$

$$y(k) = x(k) \tag{3.4}$$

or

$$x(k+1) = x(k) - \beta_c(u(k), u(k+1), \beta_{model}) f(u(k), \Delta u(k)) \tag{3.5}$$

$$y(k) = x(k) \tag{3.6}$$

where $x(k)$ is the execution time of a container epoch, β_c is a model parameter, expressed in terms of $u(k)$, $u(k+1)$ and β_{model}, which comes from the input-output relationship. Additionally, $f(\cdot)$ represents a non-linear function.

The control problem goal is to determine $\Delta u(k)$ that drives $x(k+1)$ to be equal to the set-point reference sp. In all strategies presented later on this chapter, a dead-beat control law is implemented. The design of this sort of controller considers that the control system requirement must be achieved in a finite number of control steps [32]. In this work, the set-point tracking requirement must be achieved as soon as possible, i.e. in the next control step.

Looking at the model expressed by equations (3.3) and (3.4), the resultant dead-beat control law is designed by making $x(k+1)$ be exactly sp. Given this, the control law are expressed as:

$$\Delta u(k) = -\frac{sp - x(k)}{\hat{\beta}_c(k) f(u(k))} \tag{3.7}$$

In this case, $\hat{\beta}_c$ represents the estimation of β_c determined using the estimated values of β_{model} and $u(k+1)$. Introducing equation (3.7) into (3.3), the following resultant control loop dynamics is obtained:

$$x(k+1) = \left(\frac{\hat{\beta}_c - \beta_c}{\hat{\beta}_c} \right) x(k) - \frac{\beta_c}{\hat{\beta}_c} sp \tag{3.8}$$

As the desired dynamics for the system is to make $x(k+1) = sp$, the equation (3.8) establishes an ideal matching condition assumption by making $\hat{\beta}_c = \beta_c$.

Once the control loop dynamics in equation (3.8) is defined, the stability of the whole system can be studied in the sense of Lyapunov Theorem. This study must be carried out in order to define the conditions in which the system can reach its requirements.

First, this theorem requires the definition of a Lyapunov function, which must contain all the states of the system. Including the execution time, $x(k)$, the parameter $\hat{\beta}_c$ is also seen as a state of the system due to its own dynamics in the estimation process.

The stability analysis assumes the operating point as the one around $x(k) = sp$

and $\hat{\beta}_c = \beta_c$. The following candidate Lyapunov function is considered:

$$V(k) = (x(k) - sp)^2 + (\hat{\beta}_c - \beta_c)^2 \tag{3.9}$$

Defining $\Delta\beta_c(k) = \hat{\beta}_c - \beta_c$, equation (3.9) can be rewritten as:

$$V(k) = (x(k) - sp)^2 + (\Delta\beta_c(k))^2 \tag{3.10}$$

This candidate function satisfies all impositions required in (2.10) to apply Theorem 2. Therefore, $V(k+1)$ can be expressed as:

$$V(k+1) = (x(k+1) - sp)^2 + (\Delta\beta_c(k+1))^2 \tag{3.11}$$

Replacing equation (3.8) into (3.11), it leads to:

$$V(k+1) = \left[\left(\frac{\hat{\beta}_c - \beta_c}{\hat{\beta}_c}\right)(x(k) - sp)\right]^2 + (\Delta\beta_c(k+1))^2 \tag{3.12}$$

According to the Lyapunov Theorem for discrete-time systems, the stability can be assured in an operating point when $V(k+1) - V(k) < 0$. For the case of the system in analysis, $V(k+1) - V(k)$ is expressed as:

$$\begin{aligned} V(k+1) - V(k) = &\left[\left(\frac{\hat{\beta}_c - \beta_c}{\hat{\beta}_c}\right)^2 - 1\right](x(k) - sp)^2 + \\ &+ (\Delta\beta_c(k+1))^2 - (\Delta\beta_c(k))^2 \end{aligned} \tag{3.13}$$

Due to the complexity in analyzing the terms altogether, the study of equation (3.13) only considers the following two conditions that make $V(k+1) - V(k)$ be negative:

(i) $\left[\left(\frac{\hat{\beta}_c - \beta_c}{\hat{\beta}_c}\right)^2 - 1\right](x(k) - sp)^2 < 0$

To satisfy this condition:

$$\left[\left(\frac{\hat{\beta}_c - \beta_c}{\hat{\beta}_c}\right)^2 - 1\right] < 0 \tag{3.14}$$

$$\left(\frac{\hat{\beta}_c - \beta_c}{\hat{\beta}_c} + 1\right)\left(\frac{\hat{\beta}_c - \beta_c}{\hat{\beta}_c} - 1\right) < 0 \tag{3.15}$$

Among the values that satisfy (3.15), the only relevant set for the stability of the whole system is when:

$$0 < \frac{\beta_c}{\hat{\beta}_c} < 2 \tag{3.16}$$

because it comprehends the case in which $\hat{\beta}_c = \beta_c$. As the adaptation mechanism acts to drive the estimation error to zero, a stable system must include the case which encloses this specific behavior. Considering the dynamics adopted, the integration of an estimation process as part of the control strategy does not lead the whole system to an unstable condition.

(ii) $(\Delta\beta_c(k+1))^2 - (\Delta\beta_c(k))^2 < 0$
For this case, it is necessary to study the dynamics of $\Delta\beta_c$. Supposing that the dynamic model of β_c and $\hat{\beta}_c$ can be expressed, respectively, as:

$$\beta_c(k+1) = \beta_c(k) + g(k) \tag{3.17}$$

$$\hat{\beta}_c(k+1) = \hat{\beta}_c(k) + f(k) \tag{3.18}$$

And defining $w(k) = f(k) - g(k)$, one can rewrite $\Delta\beta_c(k+1)$ as:

$$\Delta\beta_c(k+1) = \hat{\beta}_c(k+1) - \beta_c(k+1) \tag{3.19}$$

$$= \Delta\beta_c(k) + w(k) \tag{3.20}$$

Finally, $(\Delta\beta_c(k+1))^2 - (\Delta\beta_c(k))^2$ can be also rewritten as:

$$(\Delta\beta_c(k+1))^2 - (\Delta\beta_c(k))^2 = w(k)^2 + 2w(k)\Delta\beta_c(k) \tag{3.21}$$

which are only negative when

$$-2\Delta\beta_c(k) < w(k) < 0 \tag{3.22}$$

The important result of condition (3.22) remains in the fact that $w(k)$ must be negative. It means that the evolution of the estimated parameter must be slower than the one of the real parameter. Looking at the estimation process, this can be verified when the estimator does not focus on dealing only with local data. With the approach led in this work, this condition can be assumed validated during all system's operation.

Considering condition (ii) as always true, the system is considered stable when condition (i) is satisfied.

The application of the Lyapunov Theorem indicates a condition for the parameter $\hat{\beta}_c$ to meet the set-point tracking goal. It also reveals that the couple action between the controller and the estimator keeps the whole system in a stable configuration when that condition is satisfied.

With stability properties guaranteed, all control strategies can be fully described.

3.4 Adaptive Control Strategies

The remainder of this chapter is dedicated to describe all adaptive control strategies proposed in this work for the deployment of a single container.

The initial hypothesis considered in all these strategies concerns the mathematical formulation of the static relationship between *Plant* input and output for a given epoch. The first two strategies consider the same hyperbolic formulation, whereas the last two contemplate the exponential one. In each formulation, the *Macro Plant* dynamic models are developed using different approaches. This results in different control laws to respond to the same problem.

3.4.1 Control strategies considering the workload characteristics described by a hyperbolic function

Equation (3.1) corresponds to the hyperbolic function that better describes the characteristic proposed by the Amdahl's Law. Three parameters must be identified (α, β and γ) in order to fully characterize this curve.

However, a simpler model must be implemented considering the linear parameterization imposed by the RLS algorithm, restraining the estimation of β and γ at the same time. In order to apply a regression model on this algorithm, one

of the parameters must have its value fixed or even ignored, if it is possible. As γ represents an offset introduced on the input and can be identified easier than β, the first parameter is chosen to be considered a constant. From now on, γ is treated as γ_0, as follows:

$$TTF(k) = \alpha + \frac{\beta}{cpus(k) + \gamma_0} \tag{3.23}$$

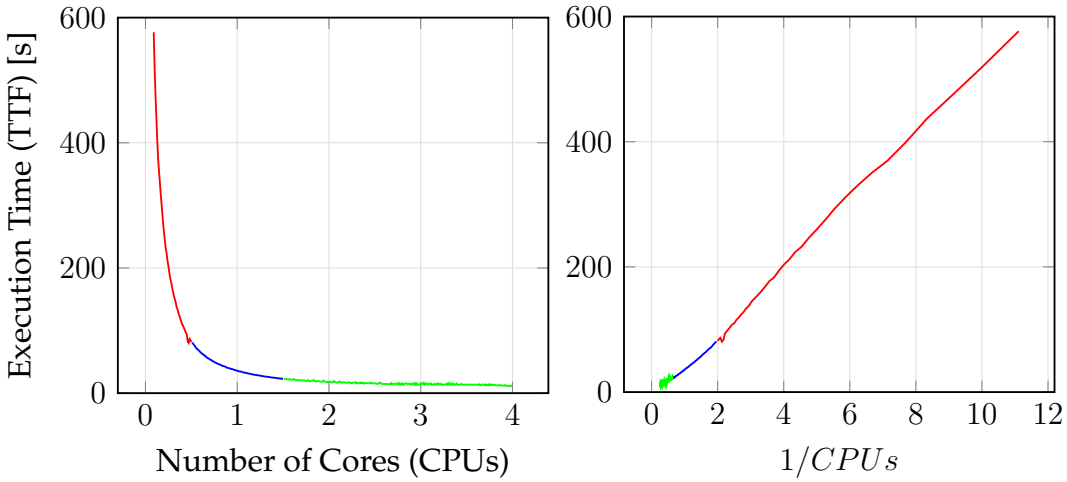
Integration of the estimation process: Regression model considered for both control strategies

As seen in Figure 3.3, the data provided to the estimation process (represented by the block *Identifier*) are only the *Plant* input and output for a given epoch, whose mathematical formulation corresponds to equation (3.23).

The regression model applied to the RLS algorithm are designed to provide the parameters α and β as a result of the estimation, considering the equivalence established by the linear parameterization in equation (3.25) :

$$\text{Hyperbolic function :} TTF(k) = \boldsymbol{\alpha} \quad + \boldsymbol{\beta}\frac{1}{cpus(k) + \gamma_0} \tag{3.24}$$

$$\text{Regression model :} \quad y \quad = \boldsymbol{\theta_1} x_1 + \boldsymbol{\theta_2} x_2 \tag{3.25}$$



(a) Characteristic execution time *versus* Number of Cores

(b) Same characteristic considering the linear parametrization

Figure 3.4 – Execution time in different approachs. Workload Characteristic from the container described in Appendix A.1.1.

In Figure 3.4(b), as the linear aspect of the characteristic is highlighted, it is possible to distinguish three distinct zones, indicated with different colors. A

change in the behavior of β, represented by the slope, is visible when the zone in red is compared with the one in blue. This is even more evident when compared to the zone in green. Depending of which zone a given point belongs to, the value of β can present a significant change.

With this regression model, an estimation not including an initial phase of identification should be more suitable and sufficient to achieve small mean square estimation errors for cases when only local estimation is performed. When this condition cannot be ensured, the absence of this initial phase may cause the system response to present a considerable transient behavior.

Dealing with the strong influence on the estimation process of points in the zone with small number of cores

As seen in section 3.1, the points which report high MSE are the ones with small amount of resources. To these points, the same hyperbolic function with constant parameters, which presents the lowest MSE for the remaining set of number of cores, no longer corresponds to the most accurate model of the *Plant*.

From the estimation process viewpoint, when these points are considered into the computation of the estimated parameters, an important change in their values is verified. This impacts the control input and consequently, the main objective of set-point tracking.

To mitigate this effect, a complementary mechanism is attached to the estimation process. Its main objective is to provide to the controller the available value of estimated parameters which present the smallest estimation error in a given operating point. This can then be understood as a mechanism that allows a better management of critical situations. Different from the change point detection mechanism verified in some solutions mentioned in Chapter 1, the new approach does not lead to a re-initialization of the estimator. But despite that, it uses the available data to decide what set of parameters is best suitable for a given case.

Keeping in mind the requirement of using the smallest amount of data as possible, this mechanism analyzes only two set of possibles values for $\hat{\alpha}$ and $\hat{\beta}$. Its operation, presented in Algorithm 3, takes advantage of a variable, called *change*, initially set as False, and which can be set as True as soon as high absolute estimation error is detected.

Now that all common aspects to both strategies were described, the peculiarities of each one of them can be presented.

Algorithm 3: Complementary logic aiming to deliver to the control module the most suitable values of $\hat{\alpha}$ and $\hat{\beta}$ for a given operating point

Input : the operating point $(cpus(k), ttf(k))$;
vector **param** with the estimated parameters from current iteration;

Output: parameters $\hat{\alpha}$ and $\hat{\beta}$ to be sent to the control module

Initialization *(At the first epoch execution)*
    change $\leftarrow$ False;
    param$_{last}$ $\leftarrow$ **param**;
Compute Absolute Estimation Error;
if *change == True* **then** `// when high estimation error has already been verified`
    Compute two estimated execution time for the same input $cpus(k)$, $\hat{ttf}_1(k)$ and $\hat{ttf}_2(k)$ using **param**$_{last}$ and **param** respectively ;
    if ***min***$(\frac{|ttf(k)-\hat{ttf}_1(k)|}{ttf(k)}, \frac{|ttf(k)-\hat{ttf}_2(k)|}{ttf(k)}) > 20\%$ **then** `// verify if the estimation error stills in a high thresold`
        if $\hat{\beta}_{min} < \hat{\beta}_{last}$ **then**
            `/* As` $\hat{\beta}_{last}$ `no longer provides the best local estimation, change is set to` *False* `to update the value of` param$_{last}$ `*/`
            change $\leftarrow$ False;
    Choose the vector of parameters whose estimation error is the smallest
else
    if *Absolute Estimation Error* $> 15\%$ **then**
        change $\leftarrow$ True;
    param$_{last}$ $\leftarrow$ **param**; `// Store values provided by the current epoch`
    Use the current vector of parameters given by the estimator

3.4.1.1 Control Strategy 1: System Dynamics given by the Taylor expansion of a hyperbolic function

The first method to establish the dynamics of the plant is the resultant approximation given by the first order Taylor expansion around an operating point $(cpus(k), ttf(k))$ at the epoch k. The small increment of resources is represented as $\Delta cpus$, producing the execution time for the next epoch $k+1$.

Proposition 1 *Let the behavior of the plant in an epoch k be defined by equation (3.23).*

The dynamic model of this plant can be defined by its first order Taylor expansion:

$$TTF(k+1) = TTF(k) - \frac{\beta_c \Delta cpus}{[cpus(k) + \gamma_0]^2} \tag{3.26}$$

Due to the approximation, β is replaced by β_c in equation (3.26) . Rewriting $TTF(k+1) - TTF(k)$ as ΔTTF , the increment of resource due to an increment in execution time is expressed as:

$$\Delta cpus = -\frac{[cpus(k) + \gamma_0]^2}{\beta_c} \Delta TTF \tag{3.27}$$

Control Law Design

Considering Proposition 1, a dead-beat controller can be designed by setting ΔTTF on equation 3.27 as the set-point tracking error $set\_point - TTF(k)$. This generates the following control law to drive the execution time of the next epoch $TTF(k+1)$ to the set-point value:

$$\Delta cpus = -\frac{[cpus(k) + \gamma_0]^2}{\beta_c} [set\_point - TTF(k)] \tag{3.28}$$

As it was previously introduced in equation (3.7), instead of using the real value of β_c, which is unknown, its estimated value, $\hat{\beta}_c$, is employed in the control input evaluation. With this change in equation (3.28), the following control law is obtained:

$$\Delta cpus = -\frac{[cpus(k) + \gamma_0]^2}{\hat{\beta}_c} [set\_point - TTF(k)] \tag{3.29}$$

Applying this control law into the dynamics of the plant (3.26), it leads to the dynamics in closed loop form:

$$TTF(k+1) = TTF(k) + \frac{\beta_c}{\hat{\beta}_c} [set\_point - TTF(k)] \tag{3.30}$$

which is exactly the one represented in equation (3.8). Since this control strategy matches all hypothesis considered in section 3.3 for the stability analysis, the same conclusions established can be applied to this case.

Relationship between the estimated value of $\hat{\beta}$ and $\hat{\beta}_c$

As equation (3.26) introduces the parameter β_c due to the approximation from the first-order Taylor expansion, it is important to define how this new parameter is determined.

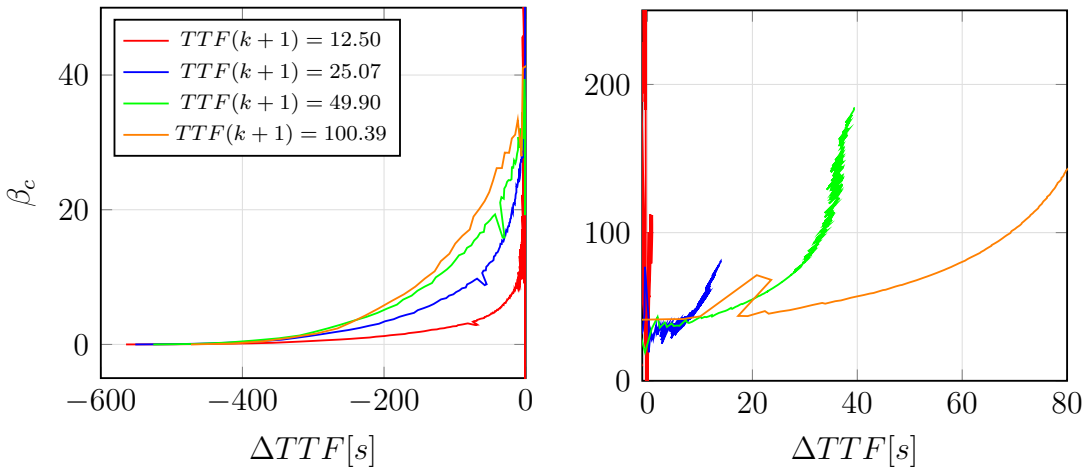


Figure 3.5 – Characteristic of β_c according to ΔTTF for $TTF(k+1)$ fixed

Using the data presented in Figure A.2 and the dynamic model proposed in equation (3.26), it is possible to characterize the behavior of β_c. Figure 3.5 shows the resultant curves for $TTF(k+1)$ chosen as the set-point values considered on testing campaigns. It is possible to observe a strong variation in β_c, whose characteristic changes according to the chosen set-point value. This confirms the initial inference that β_c cannot be directly replaced by the estimated parameter $\hat{\beta}$ to determine the control input. This would impact on how fast the system response achieve its requirements.

The relationship between the estimated parameters and $\hat{\beta}_c$ can be established replacing $TTF(k+1)$ and $TTF(k)$ in equation (3.26) by their estimated values:

$$T\hat{T}F(k+1) - T\hat{T}F(k) = -\frac{\hat{\beta}_c \Delta cpus}{[cpus(k) + \gamma_0]^2} \tag{3.31}$$

$$\hat{\alpha} + \frac{\hat{\beta}}{cpus(k+1) + \gamma_0} - \hat{\alpha} + \frac{\hat{\beta}}{cpus(k) + \gamma_0} = -\frac{\hat{\beta}_c * \Delta cpus}{[cpus(k) + \gamma_0]^2} \tag{3.32}$$

Considering that the value $cpus(k+1)$ is not available, its value is substituted by its estimated one, $\hat{cpus}(k+1)$, supposed to bring $TTF(k+1)$ to the set-point value:

$$\hat{\alpha} + \frac{\hat{\beta}}{\hat{cpus}(k+1) + \gamma_0} - \hat{\alpha} + \frac{\hat{\beta}}{cpus(k) + \gamma_0} = -\hat{\beta}_c \frac{[cpus(k+1) - cpus(k)]}{[cpus(k) + \gamma_0]^2} \quad (3.33)$$

which drives to the following relationship:

$$\hat{\beta}_c = \frac{[cpus(k) + \gamma_0]}{[\hat{cpus}(k+1) + \gamma_0]} \hat{\beta} \quad (3.34)$$

It can be easily inferred that as $\hat{cpus}(k+1)$ tends to $cpus(k)$, which is the expected behavior of the system in steady state, $\hat{\beta}_c$ will converge to the estimated value of β, as follows:

$$\lim_{\hat{cpus}(k+1) \to cpus(k)} \hat{\beta}_c = \hat{\beta} \quad (3.35)$$

3.4.1.2 Control Strategy 2: System Dynamics given by an algebraic manipulation of a hyperbolic function

The design of this control strategy first considers that the dynamic model of the plant can be obtained by a simple algebraic manipulation of equation (3.23) for two subsequent epochs.

This is possible by taking into account the execution time of a given epoch k and considering the value for the next one as a result of an additional amount of resources, $\Delta cpus$, provided to the plant in the operating point $(cpus(k), TTF(k))$. Therefore, $cpus(k+1)$ can be represented as $cpus(k) + \Delta cpus$.

Proposition 2 *Let the behavior of the plant in an epoch k be defined by equation (3.23). The dynamic model for this plant can be defined as:*

$$TTF(k+1) = \alpha + \frac{\beta}{(cpus(k) + \Delta cpus) + \gamma_0} \quad (3.36)$$

The system dynamics is obtained by expressing the equation (3.36) in terms of execution time values from the current and the next epochs. To do so, the term $\beta/[cpus(k)+\gamma_0]$ is included on both sides of that equation:

$$TTF(k+1) = \alpha + \frac{\beta}{(cpus(k) + \Delta cpus) + \gamma_0} + \frac{\beta}{cpus(k) + \gamma_0} - \frac{\beta}{cpus(k) + \gamma_0} \quad (3.37)$$

leading to a new equation for the dynamics of the plant:

$$TTF(k+1) = TTF(k) - \beta \frac{\Delta cpus(k)}{(cpus(k)+\gamma_0)^2 + \Delta cpus(k)[cpus(k)+\gamma_0]} \quad (3.38)$$

Then, the increment of resource due to an increment in execution time can be expressed as:

$$\Delta cpus(k) = -\frac{(cpus(k)+\gamma_0)^2 \Delta TTF(k)}{(cpus(k)+\gamma_0)\Delta TTF(k) + \hat{\beta}} \quad (3.39)$$

where $\Delta TTF(k) = TTF(k+1) - TTF(k)$.

Control Law Design

Applying the same approach utilized on the design of the control law in subsection 3.4.1.1, equation (3.39) leads to the following dead-beat control law:

$$\Delta cpus(k) = -\frac{(cpus(k)+\gamma_0)^2 (sp\_error)}{(cpus(k)+\gamma_0)(sp\_error) + \hat{\beta}} \quad (3.40)$$

where $sp\_error = set\_point - TTF(k)$.

Comparing the control law defined in equation (3.40) with the one in equation (3.29), the only difference is found in a term included in the denominator. This new control law is also nonlinear, but instead of representing an approximation, it corresponds to the real dynamics of the system considering a plant modelled by a hyperbolic function.

Remark 1 *The dynamics in closed loop form for this strategy, obtained when equations (3.38) and (3.40) are combined, are identical to the dynamics described in equation (3.30). The same conclusions identified by the application of the Lyapunov Theorem in section 3.3 are valid for the control strategy in study in this section.*

Relationship between the estimated value of $\hat{\beta}$ and $\hat{\beta}_c$

During the development of both dynamics and control law, represented by equations (3.38) and (3.40), respectively, the parameter of the *Plant* was considered instead of β_c, since these equations came directly from the relationship established by (3.23).

Dealing with inconsistency in control input computation

As mentioned before, the denominator in equation (3.38) presents two terms. Due to the characteristic of sp_error, which can be both positive and negative, it is necessary to investigate if the inverse relationship between the amount of resources and the execution time remains valid for any situation. This behavior is only satisfied when:

$$(cpus(k) + \gamma_0)(sp\_error) + \hat{\beta} > 0 \tag{3.41}$$

which leads to:

$$(sp\_error) > -\frac{\hat{\beta}}{(cpus(k) + \gamma_0)} \tag{3.42}$$

For cases in which this condition cannot be satisfied, the main goal is accomplished by decreasing the value of γ_0 and limitating the amount of resources to be allocated to an application. This allows the control strategy to reach a suitable performance regarding the resource allocation.

3.4.2 Control strategies considering the workload characteristics described by an exponential function

Using the same approach led on last section to define the dynamics of *Macro Plant* in order to obtain control laws able to meet system requirements, this section takes an exponential function into consideration.

Similar to the hyperbolic, the exponential function described in equation (3.2) cannot have all its three parameters (α, β and γ) identified by the RLS estimation method at the same time. The simplest parameter that can be identified is γ which, in this case, represents the offset in execution time. From now on, the following exponential function, with a constant value γ_0, is considered:

$$TTF(k) = \alpha e^{-\beta cpus(k)} + \gamma_0 \tag{3.43}$$

Integration of the estimation process: Regression model considered for both control strategies

The linear parameterization of an exponential equation must consider a little algebraic manipulation. The regression model applied to the RLS algorithm returns values which are expressed in terms of $\hat{\alpha}$ and $\hat{\beta}$:

$$\text{Exponential function} : ln[TTF(k) - \gamma_0] = \boldsymbol{ln}[\boldsymbol{\alpha}] - \boldsymbol{\beta} cpus(k) \tag{3.44}$$

$$\text{Regression model} : \quad y \qquad\qquad = \boldsymbol{\theta_1} x_1 \quad + \boldsymbol{\theta_2} x_2 \tag{3.45}$$

By equivalence of equations (3.44) and (3.45), $\hat{\alpha}$ and $\hat{\beta}$ can be obtained as follows:

$$\hat{\alpha} = e^{\theta_1} \tag{3.46}$$

$$\hat{\beta} = \theta_2 \tag{3.47}$$

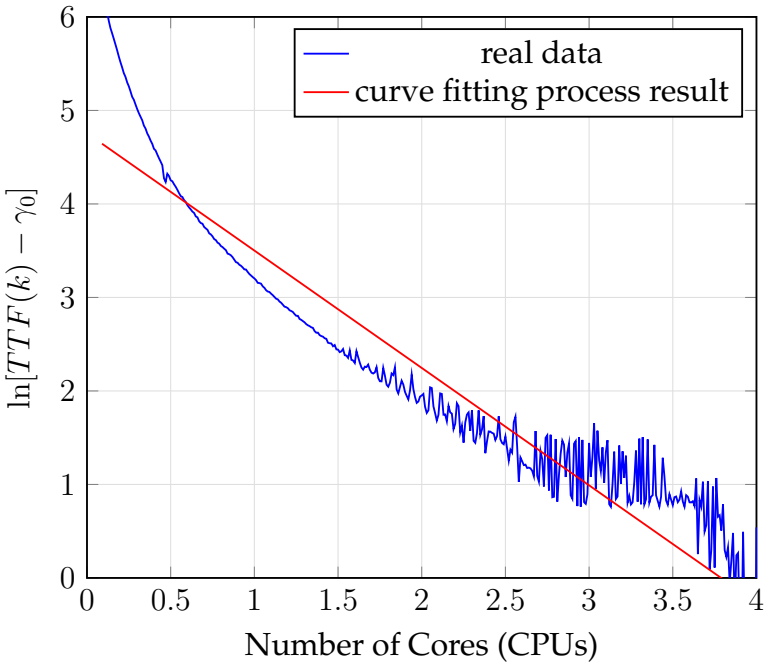


Figure 3.6 – Linear characteristics given the regression model expressed in equation (3.45)

The Figure 3.6 shows how the data provided by the execution of the container described in section A.1.1 fits the linear relationship established by equation (3.45), assuming an exponential behavior for the characteristics.

For both set boundaries, the real data does not follow this supposed behavior. Trying to minimize the estimation error, since the data in these zones are taken into account on an estimation process, its performance is affected. This can be illustrated by the red line in Figure 3.6, which corresponds to the result of a curve fitting process for a regression model supposed by equations (3.44) and (3.45). Although this curve cannot be considered as good as the linear relationship found in Figure 3.4b, a good estimation of the linear coefficient - which corresponds to $\hat{\beta}$ - is verified.

Regarding the integration of the data in a regression model applied to the RLS algorithm, the above discussion leads to consider that the estimation process must take all available data in a given moment, with a high forgetting factor.

Remark 2 *All control strategies to be described in the following consider Algorithm 3 to deal with high estimation errors for points with small amount of resources.*

3.4.2.1 Control Strategy 3: System Dynamics given by the Taylor expansion of an exponential function

The same approach using the first order Taylor expansion led in section 3.4.1.1 is, in this case, applied to the exponential function described in equation (3.43).

Proposition 3 *Let the behavior of the plant in an epoch k be defined by equation (3.43). The dynamic model for the plant can be defined by its first order Taylor expansion:*

$$TTF(k+1) = TTF(k) - \beta\alpha e^{-\beta cpus(k)}\Delta cpus \tag{3.48}$$

The same equation can be also written as:

$$TTF(k+1) = TTF(k) - \beta_c[TTF(k) - \gamma_0]\Delta cpus \tag{3.49}$$

Instead of including the parameter β, equation (3.49) replaces it by β_c, such that the influence of the neglected terms by the approximation can be taken into account. Rewriting $TTF(k+1) - TTF(k)$ as ΔTTF , the increment of resource due to an increment in execution time is:

$$\Delta cpus = -\frac{1}{\beta_c}\frac{\Delta TTF(k)}{TTF(k) - \gamma_0} \tag{3.50}$$

Control Law Design

Using the same approach to define a dead-beat controller applied on equation (3.50) and considering that the parameters of equation (3.43) are estimated, the following control law is obtained:

$$\Delta cpus = -\frac{1}{\hat{\beta}_c}\frac{set\_point - TTF(k)}{TTF(k) - \gamma_0} \tag{3.51}$$

Remark 3 *The dynamics in closed-loop form defined by the integration of equations (3.50) in (3.51) are identical to the dynamics analyzed in the past control strategies. All conclusions identified by the application of the Lyapunov Theorem in section 3.3 remain valid for the control strategy in study on this section.*

Relationship between the estimated value of $\hat{\beta}$ and $\hat{\beta}_c$

In equation (3.49), a new parameter β_c is introduced such that this approximation can better fit the real system response.

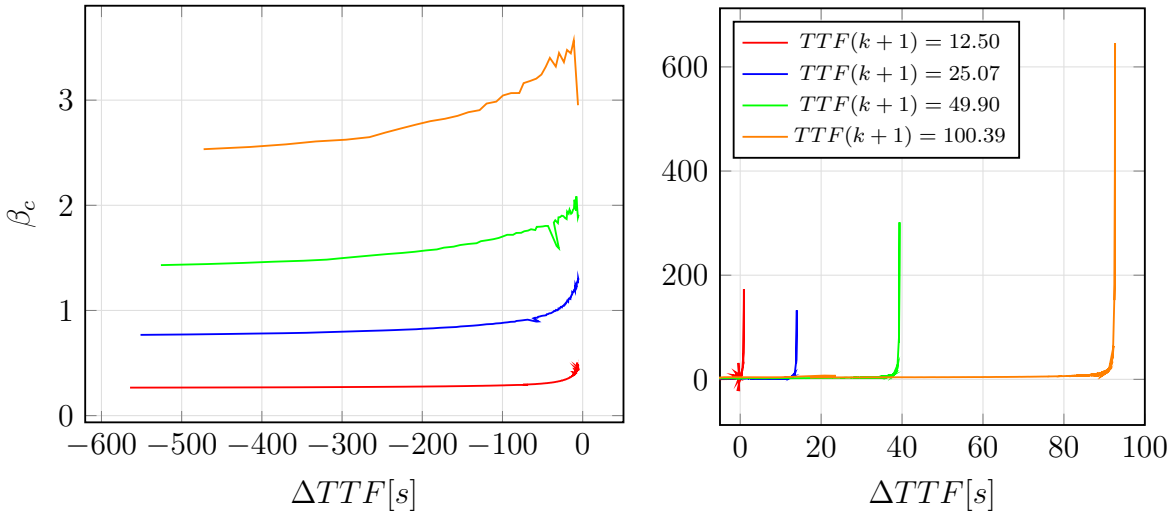


Figure 3.7 – Characteristic of β_c according to ΔTTF for $TTF(k+1)$ fixed

By taking the data from the characteristics of the workload exposed in Figure A.2, the behavior of β_c can be established in view of the dynamics described in equation (3.49). This is verified in Figure 3.7 for fixed set-point values, which reveals that β_c is a varying parameter, depending on the set-point reference as well as the current operating point.

Similar to the first control strategy presented in section 3.4.1.1, the relationship between estimated parameters and $\hat{\beta}_c$ is obtained by replacing $TTF(k+1)$ and $TTF(k)$ in equation (3.49) by their estimated values:

$$\hat{TTF}(k+1) - \hat{TTF}(k) = -\beta_c[\hat{TTF}(k) - \gamma_0]\Delta cpus \tag{3.52}$$

$$\begin{aligned} &\hat{\alpha}e^{-\hat{\beta}cpus(k+1)} + \gamma_0 - \hat{\alpha}e^{-\hat{\beta}cpus(k)} - \gamma_0 = \\ &= -\beta_c\hat{\alpha}e^{-\hat{\beta}cpus(k)}[cpus(k+1) - cpus(k)] \end{aligned} \tag{3.53}$$

Considering that the value $cpus(k+1)$ is not available, its value is replaced by its estimated one, $\hat{cpus}(k+1)$, supposed to bring $TTF(k+1)$ to the set-point value:

$$\begin{aligned}\hat{\alpha}e^{-\hat{\beta}c\hat{p}us(k+1)} + \gamma_0 - \hat{\alpha}e^{-\hat{\beta}cpus(k)} - \gamma_0 = \\ = -\beta_c\hat{\alpha}e^{-\hat{\beta}cpus(k)}[c\hat{p}us(k+1) - cpus(k)]\end{aligned} \tag{3.54}$$

leading to the following relationship:

$$\beta_c = \frac{e^{[\hat{\beta}(cpus(k) - c\hat{p}us(k+1))]} - 1}{cpus(k) - c\hat{p}us(k+1)} \tag{3.55}$$

It is also important to verify how β_c behaves in steady state:

$$\lim_{c\hat{p}us(k+1)\to cpus(k)} \beta_c = \frac{0}{0} \quad \text{(Indeterminate)} \tag{3.56}$$

$$\begin{aligned}\lim_{c\hat{p}us(k+1)\to cpus(k)} \beta_c &\overset{\text{(L'H)}}{=} \lim_{c\hat{p}us(k+1)\to cpus(k)} \hat{\beta}e^{[\hat{\beta}(cpus(k) - c\hat{p}us(k+1))]} \\ &= \hat{\beta}\end{aligned} \tag{3.57}$$

This indicates that in steady state, the value of β_c converges to $\hat{\beta}$.

3.4.2.2 Control Strategy 4: System Dynamics given by an algebraic manipulation of an exponential function

This control strategy also considers a new equation for the dynamics of the plant. The execution time of the epoch $k+1$ is expressed in terms of its value in the epoch k without invoking any approximation. The same initial approach led in section 3.4.1.2 is used: the number of cores to be allocated in a next epoch $k+1$ considers an additional amount of resources, $\Delta cpus$, applied on the value in epoch k. Therefore, $cpus(k+1)$ is represented as $cpus(k) + \Delta cpus$.

Proposition 4 *Let the behavior of the plant in an epoch k be defined by equation (3.43). It can be also expressed as:*

$$ln[TTF(k) - \gamma_0] = ln(\alpha) - \beta cpus(k) \tag{3.58}$$

The execution time of the epoch $k+1$ can be then defined as a increment of resources added to the value in epoch k:

$$ln[TTF(k+1) - \gamma_0] = ln(\alpha) - \beta cpus(k) - \beta\Delta cpus \tag{3.59}$$

Substituting the terms $ln(\alpha) - \beta cpus(k)$ by $ln[TTF(k) - \gamma_0]$, a new equation for the system dynamics is obtained:

$$ln[TTF(k+1) - \gamma_0] = ln[TTF(k) - \gamma_0] - \beta \Delta cpus \tag{3.60}$$

Then, the increment of resource due to a increment in execution time is:

$$\Delta cpus = -\frac{1}{\beta} ln \left[\frac{TTF(k+1) - \gamma_0}{TTF(k) - \gamma_0} \right] \tag{3.61}$$

Control Law Design

Using the same approach to obtain a dead-beat control, but applied in equation (3.61), it drives to:

$$\Delta cpus = -\frac{1}{\hat{\beta}} ln \left[\frac{set\_point - \gamma_0}{TTF(k) - \gamma_0} \right] \tag{3.62}$$

The evaluation of the control law is constrained to the definition of the logarithm function, which requires that:

$$\frac{TTF(k+1) - \gamma_0}{TTF(k) - \gamma_0} > 0 \tag{3.63}$$

which is always satisfied since $TTF(k+1) - \gamma_0$ and $TTF(k) - \gamma_0$ are positive values.

The closed-loop dynamics in this case gives:

$$ln \left[\frac{TTF(k+1) - \gamma_0}{TTF(k) - \gamma_0} \right] = \frac{\beta}{\hat{\beta}} ln \left[\frac{set\_point - \gamma_0}{TTF(k) - \gamma_0} \right] \tag{3.64}$$

which can also be expressed as:

$$ln[TTF(k+1) - \gamma_0] = ln \left[\frac{e(set\_point - \gamma_0)}{TTF(k) - \gamma_0} \right] \tag{3.65}$$

Different from the closed-loop dynamics of the former control strategies, equation (3.65) is not a function of the estimated parameters.

Stability analysis using Lyapunov Theorem

With the new closed-loop dynamics obtained in equation (3.65), the stability study analyzed in the beginning of this section cannot be extended to the case of this control strategy.

Keeping the same Lyapunov function presented in (3.9), but considering $x(k) = ln[TTF(k) - \gamma]$ and equation (3.65), $V(TTF(k+1)) - V(TTF(k))$ is expressed as:

$$\begin{aligned} V(TTF(k+1)) - V(TTF(k)) = &\left[ln\left(\frac{e(set\_point - \gamma_0)}{TTF(k) - \gamma_0}\right)\right]^2 + \\ &- [ln(TTF(k) - \gamma_0)]^2 + \\ &+ (\Delta\beta_c(k+1))^2 - (\Delta\beta_c(k))^2 \end{aligned} \tag{3.66}$$

which can be rewritten as:

$$\begin{aligned} V(TTF(k+1)) - V(TTF(k)) = &ln\left[\frac{e(set\_point - \gamma_0)}{(TTF(k) - \gamma_0)^2}\right] ln[e(set\_point - \gamma_0)] \\ &+ (\Delta\beta_c(k+1))^2 - (\Delta\beta_c(k))^2 \end{aligned} \tag{3.67}$$

Once again, due to the complexity in analyzing the terms altogether, the above equation is split in two parts. In order to verify the condition established in equation (3.9), these parts must satisfy the following requirements:

(i) $ln\left[\frac{e(set\_point-\gamma_0)}{(TTF(k)-\gamma_0)^2}\right] ln[e(set\_point - \gamma_0)] < 0$
This is only possible when

$$TTF(k) > \sqrt{e(set\_point - \gamma_0)} + \gamma_0 \tag{3.68}$$

In Figure 3.8, the condition established on equation (3.68) creates two different zones. The first one, in which starting from an execution time value $TTF(k)$, the system remains stable if the value for the next epoch is driven to the set-point value.

For the other zone, these actions are not stable. However, considering two subsequent epochs, it is possible to drive the response of the system to any set-point value considering more than two epochs. The system can be driven to a first value between the current execution time and the set-point value, but inside the stable zone. After that, it can be conducted to the right

set-point value, remaining always in the stable zone.

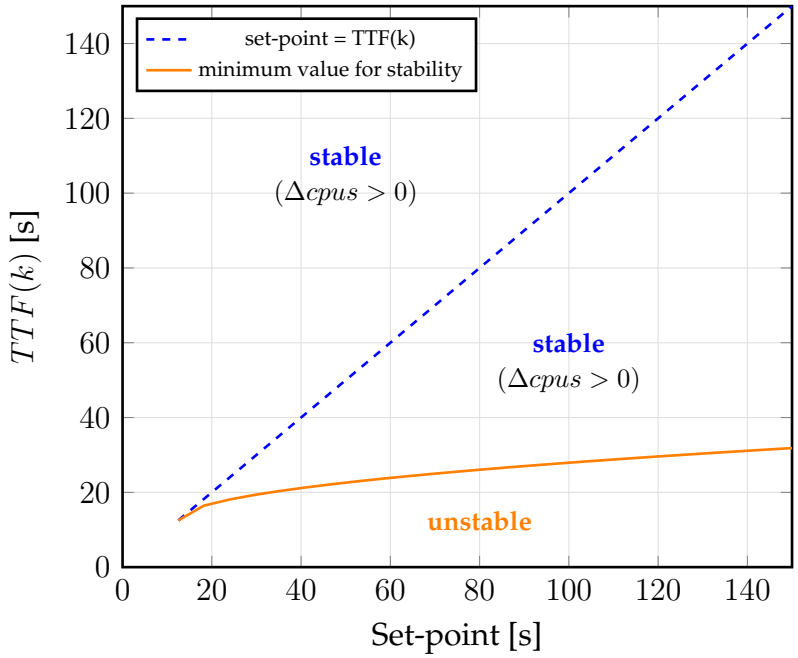


Figure 3.8 – Delimited zones of stability in the sense of Lyapunov Theorem

(ii) $(\Delta\beta_c(k+1))^2 - (\Delta\beta_c(k))^2 < 0$
The same conclusions made in the beginning of this section for the above condition still valid in this case.

3.5 Synthesis of all control strategies

In the former sections of this chapter, the design of four control strategies was fully described. Their stability analysis were also conducted, establishing conditions to the estimation process that drive each strategy to achieve the main goal of set-point tracking.

The real-time execution of these control strategies follows almost the same Sequence Diagram introduced in Figure 1.3. The only difference is verified by the inclusion of the Algorithm 3 after the moment where the parameters are estimated.

Tables 3.3 and 3.4 regroup the main aspects of each control strategy mentioned in this chapter, whose performance under some test scenarios are examined on the next chapter.

Table 3.3 – Key aspects of the control strategies based on a hyperbolic function

Aspects		**Control Strategies**	
		1	**2**
Parameters Estimation	Regression model	$TTF(k) = \boldsymbol{\alpha} + \boldsymbol{\beta}\frac{1}{cpus(k)+\gamma_0}$	
	Parameters to be estimated	α and β	
Control Law Design	*Macro Plant* Dynamics	$TTF(k+1) = TTF(k) - \frac{\beta * \Delta cpus}{[cpus(k)+\gamma_0]^2}$	$TTF(k+1) = TTF(k) - \beta\frac{\Delta cpus(k)}{(cpus(k)+\gamma_0)^2+\Delta cpus(k)[cpus(k)+\gamma_0]}$
	Control Law	$\Delta cpus = -\frac{[cpus(k)+\gamma_0]^2}{\hat{\beta}}[set\_point - TTF(k)]$	$\Delta cpus(k) = -\frac{(cpus(k)+\gamma_0)^2(sp\_error)}{(cpus(k)+\gamma_0)(sp\_error)+\hat{\beta}}$

Table 3.4 – Key aspects of the control strategies based on an exponential function

Aspects		**Control Strategies**	
		3	**4**
Parameters Estimation	Regression model	$ln[TTF(k) - \gamma_0] = \boldsymbol{ln}[\boldsymbol{\alpha}] - \boldsymbol{\beta} cpus(k)$	
	Parameters to be estimated	α and β	
Control Law Design	*Macro Plant* Dynamics	$TTF(k+1) = TTF(k) - \beta[TTF(k) - \gamma_0]\Delta cpus$	$ln[TTF(k+1) - \gamma_0] = ln[TTF(k) - \gamma_0] - \beta\Delta cpus$
	Control Law	$\Delta cpus = -\frac{1}{\hat{\beta}_c}\frac{set\_point - TTF(k)}{TTF(k)-\gamma_0}$	$\Delta cpus = -\frac{1}{\hat{\beta}} ln\left[\frac{set\_point-\gamma_0}{TTF(k)-\gamma_0}\right]$

Chapter 4

Deployment of a single container using the Adaptive Control Strategies

Contents

4.1 Main goals for testing 70
4.2 Design of test scenarios 71
4.2.1 Tests environment 72
4.3 Performance Analysis 74
4.3.1 Control strategies based on a hyperbolic function for the workload characteristics 75
4.3.2 Control strategies based on an exponential function for the workload characteristics 76
4.4 Conclusion 77

This chapter describes how the control strategies presented in Chapter 3 had their performances assessed through a testing campaign. It also presents an analysis allowing to compare their performances in order to choose an unique control strategy for the further development of this work.

4.1 Main goals for testing

In view of the design of a resource allocation algorithm able to handle multiple applications at the same time, testing control strategies developed for the execution of only a single container is an important task. This allows that some aspects implemented at the level of each container can be validated:

1. Specific aspects regarding the implementation of a self tuning regulator
   1.a) Verify the importance of γ_0 for the response of each control strategy. As said in Chapter 3, not all parameters from the candidate functions can be estimated at the same time. The solution proposed to overcome

this problem was to set one parameter as a constant value, represented as γ_0 in all strategies.

However, considering the context of this work, aiming to handle the execution of multiple containers with different workloads, this can turn into a complex task when implemented in large scale. Hence, the testing campaign must verify if the absence of this parameter on the control input evaluation produces important damage to the system response.

1.b) Verify if the desired behavior - set-point tracking - is confirmed in every situation.

In this case, the main objective is to validate that a control law which takes only the dynamics of the plant drives the set-point tracking error to zero.

1.c) Verify if the estimation performance is in agreement to its expected behavior - constant value in steady state.

2. Control system performance under different situations.

Due to the unpredictable changes that may occur at run-time, it is necessary that the designed control strategies handle them without turning into an unstable condition.

4.2 Design of test scenarios

Table 4.1 – Characteristic of each phase composing a test scenario

phase	**number of epochs**	**set-point value (initial value: *sp*)**	**constraint applied**
1	25	*sp*	None
2	20	*sp*	imposed limitation of resources at the first five epochs
3	20	2*sp*	None
4	20	*sp*	None

The validation of the control strategies presented in Chapter 3 considers more complex scenarios than those performed in section 1.2. The designed test scenario consists of four main phases defined by different conditions applied to the

container. All phases require the set-point reference provided as an input. The Table 4.1 presents some details of each one of them.

These phases correspond to some possible features that may occur at run-time. The purpose of testing such conditions are described as follows:

Phase 1: Verify the performance of the whole control strategy when it takes its first measures into the estimation process;

Phase 2: Analyze two different aspects:

(a) the effect on the estimated parameters values brought by the use of data from the epochs in which a resource usage constraint is applied;

(b) the performance of the control strategy after this constraint is released.

Phase 3: Verify how the strategy drives the execution time in transient and steady state parts to a new operating point when demanding for less resource usage;

Phase 4: Same of the precedent phase, but for the case in which the resource allocation requires the same set-point value of Phases 1 and 2.

The initial set-point values considered on test cases are the same performed on the testing campaign described in section 1.2: 12.5s, 25.0s, 50.0s and 100.0s. For each set-point value tested for each control strategy, two test cases were designed, by choosing two different values for γ_0. One of the test cases assumes $\gamma_0 = 0$ and the other one, the value provided by the Curve Fitting Process analyzed in section 3.1.

The reason each test scenario considered only the changing in the set-point value after the resource usage restriction is because this represents the worst-case scenario for the estimation process. The estimated parameters provided after that specific time present higher values than whether this constraint is not applied. This is due to the fact that the estimator fits all available data, and from the moment the points with the applied constraint are taken into account, the estimation is no longer performed locally. High values for $\hat{\beta}$ induces a slow system response, which is not desirable in terms of mastering its requirements.

4.2.1 Tests environment

The test cases previously described were performed on a single machine from NACAD. It consists of 8 cores with Intel(R) Core(TM) i7-3770, running at 3.40 GHz, with total memory of 12 GB and with the operating system CentOS Linux 7 (Core).

Due to the important number of tests and the required time for their execution, they were implemented using an automated approach, using Jenkins [1]. It is an open-source automated server used for different software development phases, including testing. Specially in this work, Jenkins provided to the control strategies development a faster deployment of adjustments necessary to tune the whole solution according to the system requirements.

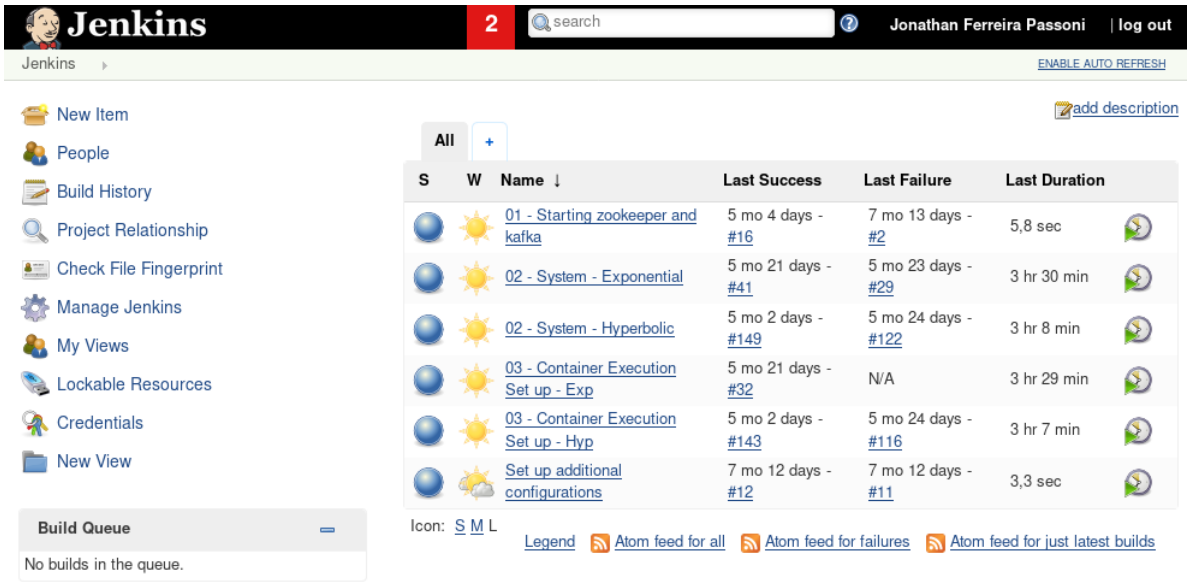


Figure 4.1 – Jenkins's main user interface used in this work.

Six different jobs were implemented in this environment to make the test cases run automatically after being set up. Figure 4.1 shows all jobs implemented. The first consists of some machine configurations that must be performed before the execution of the control strategy. To start a test case, two different jobs runs at the same time: one is dedicated to set up a container execution and the other is dedicated to the control strategy, whose real-time operation can be simply understood in Figure 1.3. For the execution of this last job, some parameters are required, aiming to identify which test case is performed. Figure 4.2 shows how a test case can be manually set-up before its execution. Test cases were previously configured to run in a specific moment and also according to a certain set of parameters defined.

1. https://www.jenkins.io/

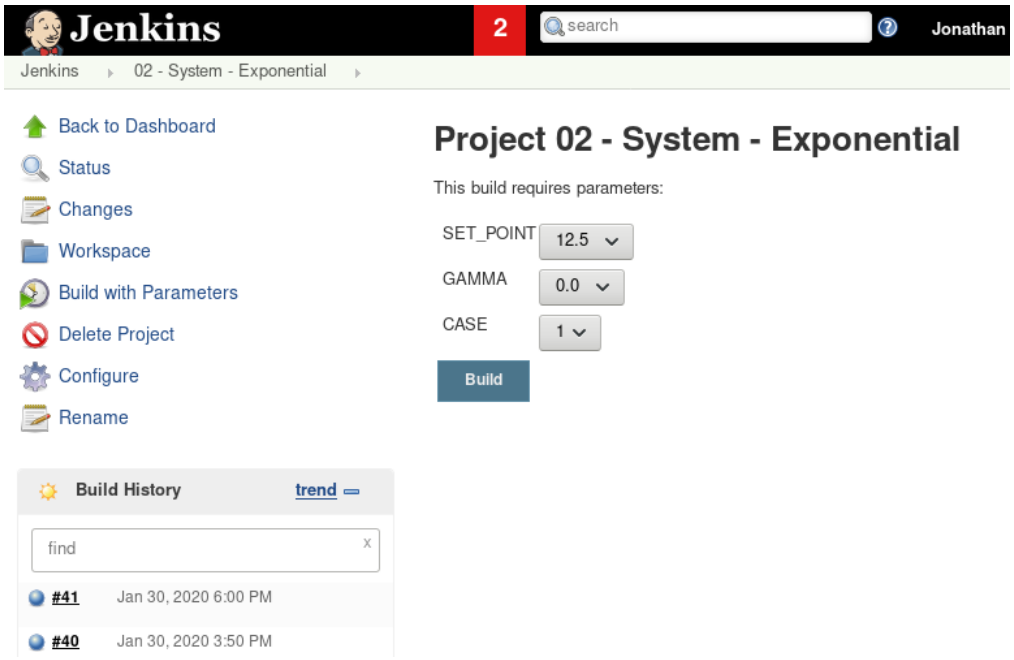


Figure 4.2 – Jenkins's set-up page in order to build a test case.

4.3 Performance Analysis

For the matter of simplicity due to the large amount of data recovered during the tests, this section presents only the important performance aspects verified to establish sufficient conclusions about which control strategy must be chosen for further development steps of this work. All data which based the following analysis are available in Appendix B.

First, a point that relates all performed test cases concerns the significance of an initial parameters identification phase, implemented in the control strategy analyzed in section 1.2. The tests performed for the new control strategies revealed that an initial identification is not necessary or relevant to accomplish the system requirements. When a set-point changes and the operating point is driven to a still not explored zone of the workload characteristics, the system response is penalized in only one epoch, in average.

The following analysis compares control strategies based on the same equation for the workload characteristics. After that, a conclusion regroups all of them to determine the best strategy.

4.3.1 Control strategies based on a hyperbolic function for the workload characteristics

In this section, **Control Strategy 1**, fully described in section 3.4.1.1, and **Control Strategy 2**, equally described in section 3.4.1.2, have their performance analyzed.

In all phases for each test case of these control strategies, the main goal of set-point tracking was quickly reached.

Transient response

Important overshoots in execution time were verified in phase 3 and also important undershoots in phases 2 and 4 for **Control Strategy 2**.

When demanding for more resources, behavior verified at the beginning of phase 4, smaller overshoots of resources were verified for **Control Strategy 1**.

Use of a complementary logic (Algorithm 3)

The use of the algorithm to determine the best estimated values revealed its importance. This was verified by a decrease in the estimation error, especially when the container asks for more resource capacity, and even more important for the right allocation of a small amount of resources.

Behavior of $\hat{\beta}$

The parameter estimation was affected by the data provided from the epochs whose amount of resources was set to the minimum value, comproving the significance of Algorithm 3. The cases most affected were the ones with small set-point references, as expected.

Control Strategy 2 drives quickly $\hat{\beta}$ to a constant value, even after transient response when the operating point changes. The alternation of values verified in some cases, specially among phases 2, 3 and 4, was due to the action of Algorithm 3.

Although the expected behavior of $\hat{\beta}_c$ in **Control Strategy 1** is a convergence in steady state to the value of $\hat{\beta}$, this was only verified in epochs from phase 1. This can be due to the use of $\hat{\beta}$ on the computation of $c\hat{p}us(k+1)$ in equation (3.34). Nevertheless, convergence was verified in steady state for most of cases, in values close to the estimated value found in phase 1.

Absence of γ_0 for control input evaluation

The only difference evidenced was that by choosing $\gamma_0 = 0.0$, more important overshoots and undershoots in response time were verified when compared to cases with non-zero value of γ_0.

4.3.2 Control strategies based on an exponential function for the workload characteristics

In this section, **Control Strategy 3**, fully described in section 3.4.2.1, and **Control Strategy 4**, equally described in section 3.4.2.2, have their performance analyzed.

First, it is worthy mentioning two important aspects verified on the tests cases for these two control strategies. As said in the last chapter, estimating parameters assuming an exponential function for the workload characteristics and giving the same importance to all available data may result on estimated values which present high estimation errors for the boundaries of the inputs set. This can be seen on both Figures B.9 and B.13, whose related test cases corresponds to the smallest set-point reference. Although with a stabilization around the set-point value of each phase, some oscillatory response was verified after the transient time from one operating point to another.

Transient response

Control Strategy 3 presented the smallest overshoots in amount of resources in phase 2. Both strategies had a similar performance in phase 4. For the test case with set-point $= 25.0$s using **Control Strategy 3**, no overshoot/undershoot was verified in all phases.

Behavior of $\hat{\beta}$

The estimation error remained small, except from test cases whose phase intended to drive the set-point value to 200s. In this case, the estimator tries to compensate the appearance of important estimation errors on $\hat{\alpha}$, whilst the proper way to overcome it was by changing also $\hat{\beta}$, what was not verified.

Through all steps, $\hat{\beta}$ did not present important variations in its value and it remained constant and close to the value verified on phase 1.

Use of a complementary logic (Algorithm 3)

The use of the algorithm to determine the best estimated values seemed irrelevant to both **Control Strategies 3** and **4**. As the value of $\hat{\beta}$ was not affected when the estimator took into account the data from epochs with a constraint in the amount of resources, the estimation error remained small.

4.4 Conclusion

In total, the testing campaign performed 16 test cases, each one of them consisting of 4 different phases. They provided information about the performance of the designed control strategies under certain normal conditions and also a critical scenario when the amount of resources is limited to its minimum value.

The control approach implemented on these strategies considers a self tuning regulator, whose architecture embodies an estimation process. The analysis focused on both performance of this process and its implication in the resource allocation. As it provides to the control input evaluation the estimated parameters of the Plant, this process affects directly the performance of the whole system.

As seen by the last section and also by the Figures displayed on Appendix B, most control strategies presented both pros and cons. Considering all aspects previously mentioned, the choice of the best control strategy to proceed to the development of an algorithm for multiple containers basically depends on two aspects.

The first one corresponds to the perspective of resources usage. In this case, the best control strategy presents the minimum overshoots and undershoots as possible until it drives the set-point tracking error to zero at steady state. The other one corresponds to the perspective of not violating SLA metrics. The best control strategy must present the minimum number of SLA violations - higher execution time than the reference value - as possible. Table 4.2 presents the best control strategies considering these both perspectives for each set-point value performed on testing campaign.

Table 4.2 – Best control strategy (CS) regarding two different approaches

Approach	Set-point values			
	12.5s	25.0s	50.0s	100.0s
SLA-oriented	**CS 1**	**CS 3**	**CS 1**	**CS 1**
resource usage-oriented	**CS 1**	**CS 1**	**CS 1**	**CS 1**

When an application starts running, the control system does not know where the operating point is located on the workload characteristics. Therefore, the control strategy to be chosen must handle the biggest set of viable operating points as possible. For this reason and considering the results of Table 4.2, the **Control Strategy 1** presents the best operation.

It is worth mentioning that, as seen on Table 4.2, the strategies whose dynamics is based on a Taylor expansion approximation were the best at responding to the conditions in which the system was subject to. This can be associated with two design aspects of these control strategies. One of them is the absence of an initial Identification Phase dedicated to bring enough information about the estimated parameters to, finally, use them in the control input evaluation. The other one is that these strategies take into account a different parameter, $\hat{\beta}_c$, instead of applying directly $\hat{\beta}$. Although the computation of this new value depends of the estimated parameter $\hat{\beta}$, it also relies on the operating point and the set-point value. With these both factors, the local operation turns to be more important to the system response and the same is not more exclusively reliant on the estimator.

Part II

Optimal Adaptive Control Strategy considering multiple containers execution

Chapter 5

Optimal Adaptive Control Strategy

Contents

5.1 System Structural Architecture 81
5.1.1 The new context of multiple containers execution 81
5.1.2 The Kafka issue 83
5.1.3 Overview of System Modules 85
5.2 Optimal Adaptive Control Strategy 86
5.2.1 Problem formulation update 86
5.2.2 Proposed Solution 87
5.2.3 Optimal Control Problems 88
5.2.4 The real-time operation 94
5.3 Synthesis of the global optimal adaptive control strategy algorithm 98

Until the previous chapter, this work has focused on verifying whether it is possible to satisfy the set-point tracking requirement using the self tuning regulator technique for the process of only a single container. As a result, the best association between an adaptation mechanism and a dynamic model was found. From this chapter on, the aspects of the global solution - a resource allocation algorithm which deals with multiple containers running simultaneously - are presented.

This chapter provides a complete description of the optimal adaptive control strategy designed. First, a new system structural architecture is introduced. Additionally, the constrained and the unconstrained optimal problem are specified. Some algorithms are presented aiming to deal with some aspects specially related to the adaptation mechanism introduced by the use of an estimator.

5.1 System Structural Architecture

This first section of this chapter presents the structural architecture of the system. As a new context, the multiple containers execution, is introduced, a new architecture must be conceived to deal with some issues not verified in a single container environment. This section also underlies the choices defined for some of the critical aspects regarding the recommended system performance.

5.1.1 The new context of multiple containers execution

The real-time operation of a single container execution represents the simplest configuration that integrates a control module, Docker and Kafka. Using some specific variables of these structures for the purpose of container identification, data can be sent through Kafka, which redirects it to the control module. After finished all expected evaluations, the new resource allocation is updated using a Docker command. In all these steps, the container identification is mandatory and only possible by the use of predefined variables.

When multiple containers run in a same environment, the integration of these structures faces some challenging aspects in view of the scaling effect. The first and major one concerns the container identification.

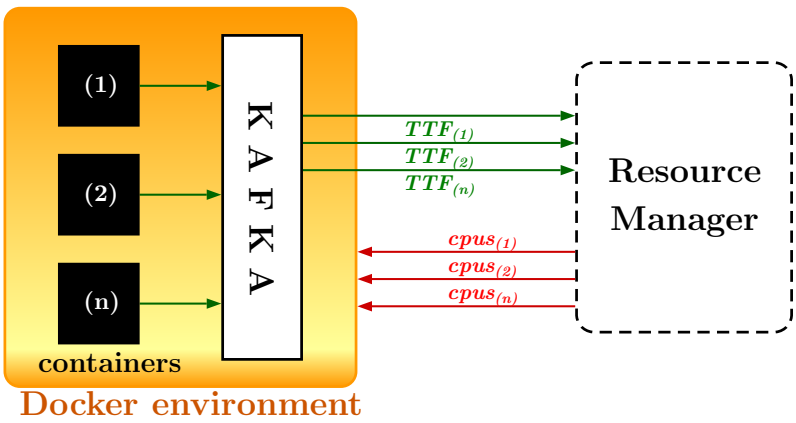


Figure 5.1 – Simple system architecture extended to multiple containers environment

As seen in Figure 5.1, the Resource Manager module provides to Docker the resource allocation value for each container running in its environment. Compared to the system structural architecture of Part I, the Resource Manager is the control module equivalent.

Figure 5.1 also reveals that this component deals with data coming from all containers, in a centralized approach, requiring the use of variables to identify

each running container. Different from Part I, a novel approach must be designed allowing these variables to be autonomously created before the deployment of a new container. This is a paramount characteristic to ensure that all further system functionalities make the right association of data from the correct container analyzed with other system modules.

Aiming the correct performance of the whole system, three container variables were identified to be the set of unique IDs:

— **the container name**: the variable intended to identify all data related to the same container at run-time. This variable is also employed when Docker commands are invoked, which require container identification;

— **the container topic**: the Kafka topic (a messages repository) in which a container sends all its messages;

— **the container image**: the specification of which template with the right application must be read to build a container.

All three variables must be assigned before a container starts, as it is indicated in the Sequence Diagram for the deployment of a new container shown in Figure 5.2.

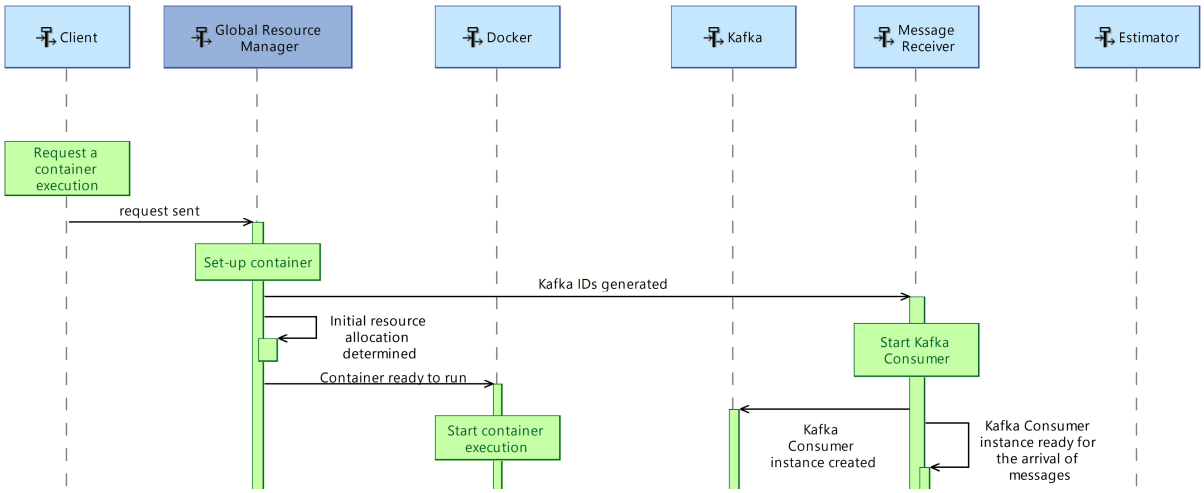


Figure 5.2 – Sequence Diagram (SysML representation) for the deployment of a new container

As part of the container set-up, another challenging aspect concerns the definition of the initial resource allocation when a container is about to start its execution. This parameter is also mandatory to build a container inside the Docker environment. Keeping in mind that no previous information is required for the system operation, the initial allocation is defined taking only the available amount of resources at the moment a newcomer container is set up. Algorithm 4 presents the simple mechanism designed to provide this initial amount. As **cpus**$(k = 1)$

does not take all available amount of resources, this allows other containers to be deployed during the initial epoch execution of the first one.

Algorithm 4: The initial resource allocation algorithm

Input : The available amount of resources **total_cpus**

Output: The initial amount of resources **cpus**$(k = 1)$ dedicated to a new container

```
1 if total_cpus <= 4.0 then
2 |   cpus(k = 1) ← 0.5total_cpus
3 else
4 |   cpus(k = 1) ← 0.3total_cpus
5 if cpus(k = 1) <= 0.2 then
6 |   The new container is not allowed to start its execution. Cause:
  |    Insufficient amount of resources
```

The deployment of multiple containers also introduces another vital aspect. The Resource Manager module receives requests for resource allocation update from all running containers. In this case, this module takes into account the value of the current available amount to implement a viable solution. As multiple requests demand an updated information of this same variable (**total_cpus**) to proceed to its evaluation, this event characterizes a classic synchronization problem, easily solved by the introduction of semaphores.

A semaphore is a protected variable used for restraining the permission to a certain part of the code for a predefined number of accesses at the same time [33]. Looking to the case mentioned before, as a container requires a resource allocation update, the part of the code in which the available amount is updated must be accessed only for a container at a time. As the method **acquire** is called at the beginning of the evaluation, another container can just access this functionality after the method **release** is called, when the restriction is no longer necessary.

To guarantee the uniqueness of containers IDs, another semaphore is positioned inside the functionality responsible for creating them.

5.1.2 The Kafka issue

Important changes in performance can be verified when multiple containers run at the same time. As a container epoch ends, data are sent as a message through a Kafka Producer method to perform all necessary evaluations after this data arrive at the Resource Manager. The container must specify in which Kafka topic, a sort of messages repository, all of them will be sent. In a same topic, messages can be allocated to different partitions.

As seen in Figure 5.3, the inclusion of partitions ensures that the Kafka structure does not present an excessive number of topics. Despite that, some problems may arise during the expansion or decrease of that structure. When a consumer is added or removed, the partition ownership, *i.e.* from what consumer method a partition sends a message from now on, changes. When this effect, called **rebalance** [34], happens, some consumers become unable to receive and treat messages. After this period of time, depending on how Kafka is configured, messages can be re-processed or even missed.

Taking Figure 5.3 as an example, the arrows indicate the association of messages sent from containers and their corresponding Kafka Topic/Partition and Consumer instance. Messages sent to Partitions 0 and 1 from Topic T1 are addressed to the same Kafka Consumer instance, Consumer 1. The introduction of Consumer 2 to take all further messages from Partition 1 (dotted arrow) causes a rebalance. As highlighted in this figure by a red border, this affects messages sent to Consumer 1 and, consequently, damages the arrival of data from containers (1) and (2).

The simplest solution implemented is to not consider partitions. The deployment of a new container automatically generates a new topic and only a single Kafka Consumer instance is addressed to receive messages from this topic. With this configuration, the effect of rebalance and its consequences are avoided. This approach also presents another advantage since the allocation of an unique topic and an unique Kafka Consumer method to a container allows that the rate of incoming messages to this method remains in a level in which they are treated as soon as they arrive.

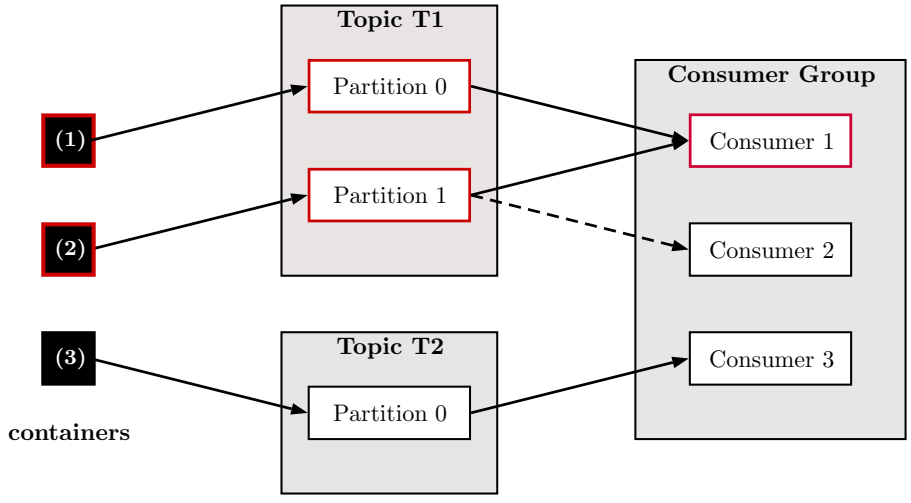


Figure 5.3 – Example of the message flow from inside the container until its arrival at the Kafka Consumer method

5.1.3 Overview of System Modules

The way that each system components are integrated to the global solution is as important as their individual behavior at run-time. For this reason, this integration must be carefully designed.

The system architecture was formulated to consider multi-processing aspects such that an adequate system performance can be ensured in large scale. First, containers run in Docker environment and, as expected, use all computational capacity provided by their last resource allocation update. As previously presented at the beginning of this manuscript, containers are thought to run independently. From the viewpoint of each container running at the same time, every other running container is generally seen as a black box, with some or even no interaction among them.

Regarding the use of computational resources, the performance of each container is strongly attached by the configuration of the provided infrastructure. With multi-cores CPUS, the parallel capacity can be better used by an application by taking part of each available core. Containers can take advantage of it, when they are implemented as processes. The use of threads in this case does not allow the allocation using this partitioning approach.

In the Message Receiver, another system component, a Kafka Consumer method is initialized to enable the receipt of messages from a running application. The Message Receiver also treats all data enclosed in a message and asks for the resource allocation update. As they run in parallel with their corresponding container execution process and present an intensive I/O behavior with the other components, they are implemented as Threads.

The final component corresponds to the Resource Manager, which deals with the control input evaluation. Besides that, it is responsible for preparing all initial identification variables to build a container and deploy its execution. The functionalities that allow data to be stored and analyzed after tests being conducted are also implemented on it. The way the Resource Manager was conceived supposes that it does not need to run in parallel with the container execution and the Message Receiver. However, it must be available when needed. For this reason, it is designed as a Class and its functionalities, as methods.

The Figure 5.4 shows how these modules are integrated, forming the system architecture capable to manage the parallel execution of multiple containers. Although only two containers are illustrated in this figure, it can be easily extended to any number of applications running at the same time.

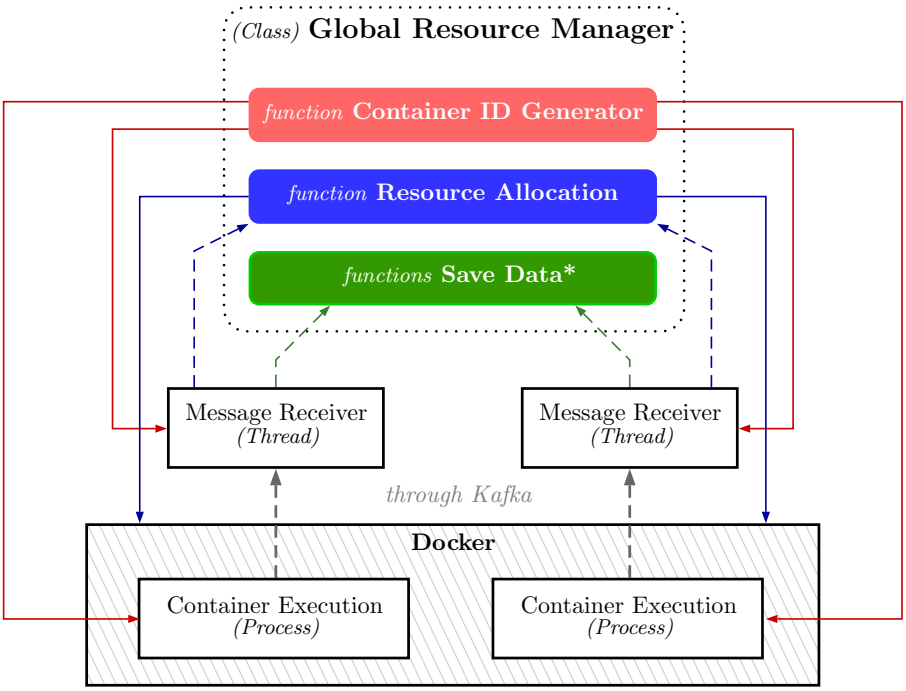


Figure 5.4 – System architecture

5.2 Optimal Adaptive Control Strategy

In Part I, adaptive control strategies were designed to satisfy the set-point tracking goal in an environment where only a single container is deployed. Such specific condition allowed to understand the peculiarities about the introduction of an adaptation mechanism into the control input evaluation.

In an environment where multiple running containers demand for resource allocation update, a new control strategy can be designed by readjusting the best solution provided in the last chapter. With requirements coming from different containers, the solution must achieve the best compromise even in cases where not all demands can be fully satisfied. Using optimal control techniques, the control input is evaluated by minimizing a cost function.

5.2.1 Problem formulation update

In a data center, different applications run in parallel by taking advantage of the computational infrastructure provided by it. Each one of them take some part of the computational resources to perform their workloads with a certain QoS.

The optimal use of this infrastructure is achieved when every single application uses a minimal amount of resources that allows to fulfill their requirements. A larger number of applications can then make use of the same infrastructure without violating any of their QoS commitments.

The resource allocation problem for which a solution is presented in this chapter has the following characteristics:

— Each alive container has its own SLA that must be respected. The SLA stills the same considered in Part I: an upper bound in execution time of a container epoch;

— The computation of the amount of resources to be allocated to each container will not use any *a priori* knowledge about the parameters of its workload characteristics;

and the following constraints:

Constraint 1 *The amount of resources to be allocated to each container must be a positive value;*

Constraint 2 *The sum of all allocated resources in a given time cannot exceed the total value provided by the infrastructure.*

5.2.2 Proposed Solution

The first aspect that must be contemplated in the design of a solution for this sort of problem is the scaling effect brought by the introduction of several containers in the running environment.

Assuming that each container has an independent workload execution, they all can be considered as different SISO plants without coupling variables. In this way, they can be seen as several smaller subsystems to be managed.

Despite the absence of coupling variables, constraint 2 presents a condition that reunites the control input of each one of them. For this reason, every resource allocation request takes all running containers into account to determine the control input. Therefore, in each request, data from all running containers must be retrieved.

The simplest way to manage it is by introducing a centralized control input evaluation module, the **Resource Manager**, presented in the last section. It takes the necessary data coming from all running containers to determine a feasible solution.

The remainder of this section describes the design of the optimal control law. At the beginning of the project, it was thought that the implementation of the model predictive control (MPC) technique would be better for managing model uncertainties and cases with resource usage limitation. On the other hand, this technique is known by its high computational effort while the control action is

calculated. So, this section examines the relevance of implementing a model predictive controller instead of a standard optimal one, without contemplating a predicted horizon on it.

5.2.3 Optimal Control Problems

In this work, the design of a control law using MPC considered the approach implemented in [35]. There, the discrete-time state-space model of the plant was extended to its augmented model to design the control law. On the other hand, this work does not consider that augmented model approach, which provides the solution in terms of the increment of control input. The objective is to provide a solution that gives directly the control input to be applied on a container, since all constraints are functions of it. This also simplifies the control input computation, reducing the order of all matrices.

The control law design first develops the solution for the simplest case, with an unconstrained resource usage, and then extends it for the case in which a limited resource usage is imposed by looking mainly at condition 2.

As no coupling variables are supposed, the development is first presented for a single container execution, a SISO system, and extended to the multiple containers case.

5.2.3.1 Brief reminder of the model chosen from the work developed in Part I

Considering the dynamic model in equation (3.26) for a plant characterized by a single container and assuming that the relationship presented in equation (3.34) can be extended to the real values of β_c and β, it leads to:

$$TTF(k+1) = TTF(k) - \beta \frac{cpus(k)}{cpus(k+1)} \frac{\Delta cpus}{[cpus(k)]^2} \tag{5.1}$$

This equation can be rewritten as follows:

$$TTF(k+1) = TTF(k) - \beta \frac{1}{cpus(k)} + \beta \frac{1}{cpus(k+1)} \tag{5.2}$$

Let $u(k) = \frac{1}{cpus(k)}$. So, the dynamics are expressed as:

$$TTF(k+1) = TTF(k) - \beta u(k) + \beta u(k+1) \tag{5.3}$$

Different from the approach led in Chapter 3, the above dynamic model for a single container plant is described in a linear parameterized form. This parameterization is required because most of the optimization tools impose a matrix formulation of the problem.

5.2.3.2 Unconstrained control problem

Consider the following discrete-time state-space system:

$$x(k+1) = x(k) - \beta u(k) + \beta u(k+1) \tag{5.4a}$$
$$y(k) = x(k) \tag{5.4b}$$

where $x(k)$ represents the execution time of a container epoch and $u(k)$ the corresponding inverse of the resource allocated to that same epoch.

Assuming that at the sampling instant k_i, the measurement of the state variable vector $x(k_i)$ is available, it provides the most recent information for the plant. With this data, the future state variables can be predicted for N_P number of samples, where N_P is called the prediction horizon.

Based on the state-space model presented in equation (5.4), the future state variables are evaluated sequentially:

$$\begin{aligned}
x(k_i+1|k_i) &= x(k_i) - \beta u(k_i) + \beta u(k_i+1) \\
x(k_i+2|k_i) &= x(k_i+1|k_i) - \beta u(k_i+1|k_i) + \beta u(k_i+2) \\
&= x(k_i) - \beta u(k_i) + \beta u(k_i+2) \\
&\vdots \\
x(k_i+N_P|k_i) &= x(k_i) - \beta u(k_i) + \beta u(k_i+N_P)
\end{aligned}$$

where $x(k_i+1|k_i)$ represents the prediction of the instant k_i+1 evaluated given the sampling instant k_i. Then, the predicted output variables can be obtained:

$$\begin{aligned}
y(k_i+1|k_i) &= x(k_i+1|k_i) \\
y(k_i+2|k_i) &= x(k_i+2|k_i) \\
&\vdots \\
y(k_i+N_P|k_i) &= x(k_i+N_P|k_i)
\end{aligned}$$

Expressing the predicted output variables in a compact matrix form, whose

terms are in function of the current state variable information $x(k_i)$, the current inverse of the resource allocation applied $u(k_i)$ and the future control input $u(k_i + j)$, leads to:

$$\mathbf{y} = \mathbf{f}x(k_i) - \beta\mathbf{f}u(k_i) + \beta\mathbf{I}_{N_P}\mathbf{u} \tag{5.5}$$

where

$$\begin{aligned}
\mathbf{y} = &\quad [y(k_i+1|k_i) \quad y(k_i+2|k_i) \quad \dots \quad y(k_i+N_P|k_i)]^T, \\
\mathbf{f} = &\quad [1 \quad 1 \quad \dots \quad 1]^T, \\
\mathbf{u} = &\quad [u(k_i+1|k_i) \quad u(k_i+2|k_i) \quad \dots \quad u(k_i+N_P|k_i)]^T
\end{aligned}$$

and $\mathbf{I}_{N_P}$ is the identity matrix of dimension N_P.

Optimization

When dealing with a set-point signal $r(k_i)$ at sample time k_i, the main goal of the predictive control system is to drive the predicted output as close as possible to this value. For the optimization window given by the prediction horizon N_P, it is considered that the set-point value remains constant.

Let $\mathbf{r_s}$ be the vector that contains the set-point value for each prediction performed with the knowledge brought at sample time k_i:

$$\mathbf{r_s}^T = [1 \quad 1 \quad \dots \quad 1]r(k_i) \tag{5.6}$$

The cost function J that aims to fulfill the control system goal is then:

$$J = (\mathbf{r_s} - \mathbf{y})^T(\mathbf{r_s} - \mathbf{y}) + \mathbf{u}^T\bar{\mathbf{R}}\mathbf{u} \tag{5.7}$$

where $\bar{\mathbf{R}}$ corresponds to a diagonal matrix in the form that $\bar{\mathbf{R}} = r_w\mathbf{I}_{N_P}$, with $r_w >= 0$ used as a tuning parameter for the desired closed-loop performance. This parameter reflects on how the cost function pays attention to the magnitude of the control input.

Considering equation (5.5) in the cost function expressed in equation (5.7), it leads to:

$$
\begin{aligned}
J =&(\mathbf{r_s} - \mathbf{f}x(k_i) + \beta\mathbf{f}u(k_i) - \beta\mathbf{I}_{N_P}\mathbf{u})^T(\mathbf{r_s} - \mathbf{f}x(k_i) + \beta\mathbf{f}u(k_i) - \beta\mathbf{I}_{N_P}\mathbf{u}) + \\
&+ \mathbf{u}^T\bar{\mathbf{R}}\mathbf{u} \\
=&(\mathbf{r_s} - \mathbf{f}x(k_i) + \beta\mathbf{f}u(k_i))^T(\mathbf{r_s} - \mathbf{f}x(k_i) + \beta\mathbf{f}u(k_i)) - 2\beta\mathbf{u}^T(\mathbf{r_s} - \mathbf{f}x(k_i) + \\
&+ \beta\mathbf{f}u(k_i)) + \beta^2\mathbf{u}^T\mathbf{u} + \mathbf{u}^T\bar{\mathbf{R}}\mathbf{u}
\end{aligned}
$$

The optimization purpose is to provide a solution that minimizes the resource usage. The first derivative of the cost function J regarding $\mathbf{u}$ gives:

$$
\frac{\partial J}{\partial \mathbf{u}} = -2\beta(\mathbf{r_s} - \mathbf{f}x(k_i) + \beta\mathbf{f}u(k_i)) + 2\beta^2\mathbf{u} + 2\bar{\mathbf{R}}\mathbf{u} \tag{5.8}
$$

Considering that $u(k_i) = \frac{1}{cpus(k_i)}$, the resource usage minimization is only achieved when u is maximized. This is verified when:

$$
\frac{\partial J}{\partial \mathbf{u}} = 0 \tag{5.9}
$$

which provides the optimal solution for the control input, as follows:

$$
\mathbf{u}^* = \beta(\beta^2\mathbf{I}_{N_P} + \bar{\mathbf{R}})^{-1}(\mathbf{r_s} - \mathbf{f}x(k_i) + \beta\mathbf{f}u(k_i)) \tag{5.10}
$$

Although $\mathbf{u}^*$ contains the control input for every prediction considered, only the first predicted value is implemented as the control input.

To do so, let:

$$
\begin{aligned}
u(k+1)^* = & \underbrace{[1 \quad 0 \quad \dots \quad 0]}_{N_P}\mathbf{u}^* \\
= & \frac{\beta}{\beta^2 + r_w}(r_s - x(k_i) + \beta u(k_i))
\end{aligned}
$$

Considering that the value of β is not available, but only its estimated value $\hat{\beta}$, the optimal control law becomes:

$$
u(k+1)^* = \frac{\hat{\beta}}{\hat{\beta}^2 + r_w}(r_s - x(k_i) + \hat{\beta}u(k_i)) \tag{5.11}
$$

This result provides an important conclusion about the MPC relevance for this

problem in particular. As seen on equation (5.11), the optimal solution does not consider any contribution given by the prediction horizon. The developed solution is equivalent to a standard optimal control law, using an adaptation mechanism to provide the estimated parameter to the control input evaluation. For this reason, from now on, this solution is called only as an optimal adaptive control strategy (OACS).

Additionally, the value of the tuning parameter r_w must be defined. The closed-loop dynamics of the system is expressed by substituting equation (5.11) into (5.4(a)):

$$x(k+1) = x(k) - \beta u(k) + \beta \frac{\hat{\beta}}{\hat{\beta}^2 + r_w}(r_s - x(k_i) + \hat{\beta} u(k_i)) \tag{5.12}$$

To make $x(k+1) \to r_s$, two conditions should be verified:

$$\begin{aligned} &\text{(i)} \quad \tfrac{\beta\hat{\beta}}{\hat{\beta}^2 + r_w} = 1 \\ &\text{(ii)} \quad \tfrac{\beta\hat{\beta}^2}{\hat{\beta}^2 + r_w} = \beta \end{aligned}$$

Both conditions are only satisfied when $r_w = 0$ and $\hat{\beta} = \beta$.

Therefore, the optimal control solution can be simplified to:

$$u(k+1)^* = \frac{1}{\hat{\beta}}(r_s - x(k_i)) + u(k_i) \tag{5.13}$$

Extending the unconstrained optimal solution to the multiple containers case

When a resource allocation update is requested, the resource management system must consider all running containers to build a viable solution. This request arrives when a container epoch ends and with it, the most current data from that specific container. The other containers are still running independently their workloads, and their most recent data also correspond to their finished epochs. Although the current resource allocation for these containers are known, their corresponding execution time are not available due to their unfinished execution at the moment where another container asks for resource allocation update. Therefore, an execution time estimation must be executed for these containers to consider their most recent amount of resource.

The optimal solution considering multiple containers at the sampling time k_i

can be computed by regrouping sequentially in a matrix form all SISO optimal solutions. This leads to:

$$\mathbf{u}(k_i+1)^* = (\mathbf{I}_j\mathbf{b}(k_i))^{-1}(\mathbf{r_s} - \mathbf{x}(k_i)) + \mathbf{u}(k_i) \tag{5.14}$$

where

$$\begin{aligned}
\mathbf{b}(k_i) =& [\hat{\beta}_1(k_i) \quad \hat{\beta}_2(k_i) \quad \dots \quad \hat{\beta}_j(k_i)]^T, \\
\mathbf{r_s} =& [r_{s1} \quad r_{s2} \quad \dots \quad r_{sj}]^T, \\
\mathbf{x}(k_i) =& [x_1(k_i) \quad x_2(k_i) \quad \dots \quad x_j(k_i)]^T, \\
\mathbf{u}(k_i) =& [u_1(k_i) \quad u_2(k_i) \quad \dots \quad u_j(k_i)]^T,
\end{aligned} \tag{5.15}$$

with $\hat{\beta}_j(k_i)$ being the estimated value of β for a given container, r_{sj} its corresponding set-point value, $x_j(k_i)$ its most recent execution time measured (for the container which required the resource allocation update) or estimated (for the other ones) and $u_j(k_i)$ the current resource allocation applied on it. The index j in (5.15) represents the number of running containers at the sampling time k_i.

Although the optimal solution gives the resource allocation for each running container, only the one which required for a new allocation are updated. As the other running containers are still performing an epoch, a resource allocation update in the middle of it would invalidate the use of the running epoch data on their estimation process.

5.2.3.3 Constrained control problem

In this section, the constrained discrete-time optimal control problem, which is written as a nonlinear programming problem, has its objective function defined by a sum of squared errors. The dynamics of each container are treated as constraints and the computing capacities appear as upper bound constraints. The problem formulated in section 5.2.1 can be expressed as:

$$\begin{aligned}
\min_{cpus_n} \quad & \sum_{n=0}^{j} (r_{sn} - x_n(k_i+1))^2 \\
\text{s.t.} \quad & x_n(k+1) = x_n(k) - \hat{\beta}_n u_n(k) + \hat{\beta}_n u_n(k+1) \\
& u_n(k_i) = \frac{1}{cpus_n} \\
& cpus_n \geq min_cpus \\
& \sum_{n=0}^{j} cpus_n < total_cpus
\end{aligned} \tag{5.16}$$

that is equivalent to:

$$\begin{aligned} \min_{cpus_n} \quad & \sum_{n=0}^{j} \hat{\beta}_n^2 u_n^2(k_i+1) - 2\hat{\beta}_n u_n(k_i+1)(r_{sn} - x_n(k) + \hat{\beta}_n u_n(k)) \\ \text{s.t.} \quad & u_n(k_i) = \frac{1}{cpus_n} \\ & cpus_n \geq min\_cpus \\ & \sum_{n=0}^{j} cpus_n < total\_cpus \end{aligned} \tag{5.17}$$

where $cpus_n$ is the amount of resources allocated to a given container, min_cpus is the minimum value allocated to each running container and $total\_cpus$ is the total amount of resources provided by the infrastructure where containers are running.

The cost function in equation (5.17) only takes the terms that are a function of the decision variable to be optimized. In this case, the resultant cost function presents the form required to apply a numerical solution using Quadratic Programming.

However, this method supposes that both cost function and constraints should be expressed in a linear parameterization. Even though the cost function is expressed in this way, the same is not verified for the constraints. As the problem is characterized as a non-linear constrained optimization, the solution is obtained using the Sequential Least Squares Programming algorithm.

5.2.4 The real-time operation

This section discusses some issues related to the real-time operation of the resource management system, as well as describes how each one of them is managed by the introduction of some mechanisms.

5.2.4.1 The influence of the estimation process on the system performance

Both equations (5.14) and (5.17) make use of the estimated value of parameters to compute the optimal control solution. The use of the RLS algorithm presumes an initialization of the estimated parameters. In the approach led in Part I for a single container, a random choice or even setting all their values to zero has revealed to be harmful for the system performance. The initial values for the estimated parameters were tuned close to the values provided by the Curve Fitting Process described in section 3.1.

However, in a multiple containers environment, with applications presenting distinct workloads, this tuning event must be executed autonomously. As the optimal control only requires $\hat{\beta}$, a simple algorithm was developed to determine an initial guess for its value. For each container alive, Algorithm 5 is only performed once, after the Message Receiver module receives the data from the first epoch and before the initialization of the estimation class instance.

The idea behind this algorithm is to find quickly a value for $\hat{\beta}$ that gives a positive value for α, which is the offset in execution time verified on the workload characteristics. This algorithm is expected to perform a small number of iterations and because of this, its output provides an underestimated value on the neighborhood of the operating point employed on its computation.

Algorithm 5: The initial guess for $\hat{\beta}$ algorithm

Input : The amount of resources allocated **cpus(k=1)** to the first epoch the container in analysis and its corresponding execution time **ttf(k=1)**

Output: The initial guess $\hat{\beta}_0$

```
1 β̂0 ← 10.0
2 while True do
3     α = β̂0 / cpus(k=1) − ttf(k=1)
4     if α > 0 then
5         β̂0 -= 10.0
6         break
7     else
8         β̂0 += 10.0
```

As mentioned in the beginning of the section 3.4.1 and reinforced by Figure 3.4, local estimation cannot reflect the behavior of the whole set of possible input values. This problem was managed in Part I by including Algorithm 3, where specific conditions are detected to provide the best available estimated parameters value to the control module. The OACS also benefits from this algorithm to manage such conditions.

5.2.4.2 Obtaining a fast optimal control input evaluation

As said in previous sections, the unconstrained optimal problem provides a straightforward solution. On the other hand, the constrained one requires an optimization algorithm to be executed and provide a viable solution. To do so, its performance demands high computational effort.

Although it is important to always consider the constraint 2, this one is relevant to the optimal control problem only when the critical condition of insufficient amount of resources is verified. Otherwise, the unconstrained control problem is sufficient to provide the optimal solution.

The first tests performed using this approach revealed that when the relative set-point error is negative and with a high magnitude, an unfeasible solution is provided, with $u(k+1)^*$ negative (See equation (5.13)). This happens even with the choice of the highest available value for $\hat{\beta}$, provided by Algorithm 3. To overcome this problem, an additional mechanism is introduced specially to deal with this kind of situation.

Dealing with an unfeasible solution : a negative amount of resources

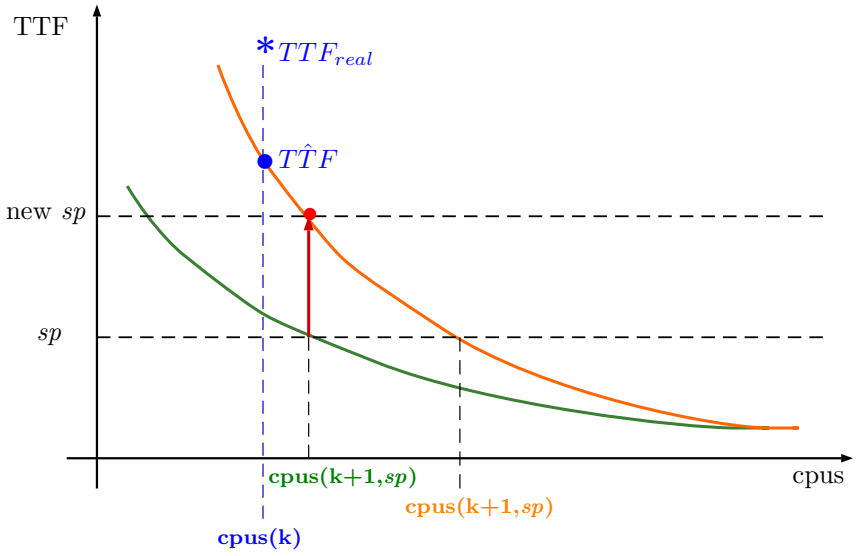


Figure 5.5 – Test scenario

As pointed out previously, the negative value for the resource allocation is due to a negative set-point error with high magnitude. This happens when the most recent data corresponds to an operating point which is far from the desired one, around the set-point value.

When the estimator takes this point into account to provide the estimated parameters, their resultant values present a strong change to better fit the most recent data introduced on the model. Consequently, this can make estimation errors which were once small before the inclusion of this point become quite significant. The introduction of Algorithm 3 helps to overcome higher changes in $\hat{\beta}$. However, this new problem concerns the strong differences between the characteristics before and after the estimator includes points which present high absolute set-point error.

To illustrate what happens in such case, let consider Figure 5.5. Before the introduction of the point $(cpus(k), TTF_{real})$ on the estimation process, the estimated parameters described a workload characteristic given by the curve in green. Once this specific point is taken into account, the parameters provided by the estimator now describes a characteristic given by the curve in orange, decreasing the current estimation error. Although this novel curve fits better the local neighborhood of the current point, it can overestimate the resource allocation in areas where the curve in green is their best representation. Considering both curves, each one of them provide a different resource allocation to drive the execution time to the set-point value *sp*.

Suppose that the value $\mathbf{cpus(k+1, sp)}$ is the one which better satisfy this requirement. Even though this value can be easily provided in an open-loop process, all control input evaluations in this work consider a closed loop approach to guarantee system stability. Because of this, the current data $(cpus(k), TTF_{real})$ must be used on the computation of the next resource allocation, through an additional mechanism.

As seen in Figure 5.5, the estimated parameters from the orange curve provide an overestimation in resources to satisfy the desired set-point value sp. However, the set-point value applied on the control input computation must be adapted to not result on an overestimated solution. This is possible by using the value $\mathbf{cpus(k+1, sp)}$ provided by the curve in green to evaluate the new set-point value $newsp$ to be integrated into the computation. Algorithm 6 describes the way this new value is obtained.

Algorithm 6: Correcting the set-point value algorithm

Input : The set-point value sp;
The current point $(cpus(k), TTF_{real})$;
The set of estimated parameters ($\hat{\alpha}$ and $\hat{\beta}$) given before and after the estimator took the current point in its evaluation.

Output: The right value **newsp** to be applied on the control input evaluation

1 $cpus_{before} \leftarrow \frac{\hat{\beta}_{before}}{TTF_{real} - \hat{\alpha}_{before}}$
2 $cpus_{after} \leftarrow \frac{\hat{\beta}_{after}}{TTF_{real} - \hat{\alpha}_{after}}$
3 **if** $cpus_{before} < cpus_{after}$ **then**
4 $\quad newsp \leftarrow \hat{\alpha}_{after} + \frac{\hat{\beta}_{after}}{cpus_{before}}$
5 **else**
6 $\quad$ No compensation is available

5.3 Synthesis of the global optimal adaptive control strategy algorithm

The previous sections of this chapter presented all mechanisms designed to provide the right resource allocation at any condition the system might experience.

Figure 5.6 shows all different modules of the OACS and the association with one another. The figure makes reference to two containers execution to emphasize how multiple containers interact within this architecture.

Figure 5.7 provides the Sequence Diagram for the whole control strategy operation. It refers to Figure 5.2 in the block *Start container execution*, as it provides the particular diagram for the deployment phase of a container. The function *Perform control input evaluation* is fully characterized in Algorithm 7.

Additionally, *Perform estimation process* is also mentioned as an external block. To this functionality, Figure 5.8 presents its corresponding Sequence Diagram, clearly describing different operations according to what container epoch the Message Receiver is processing.

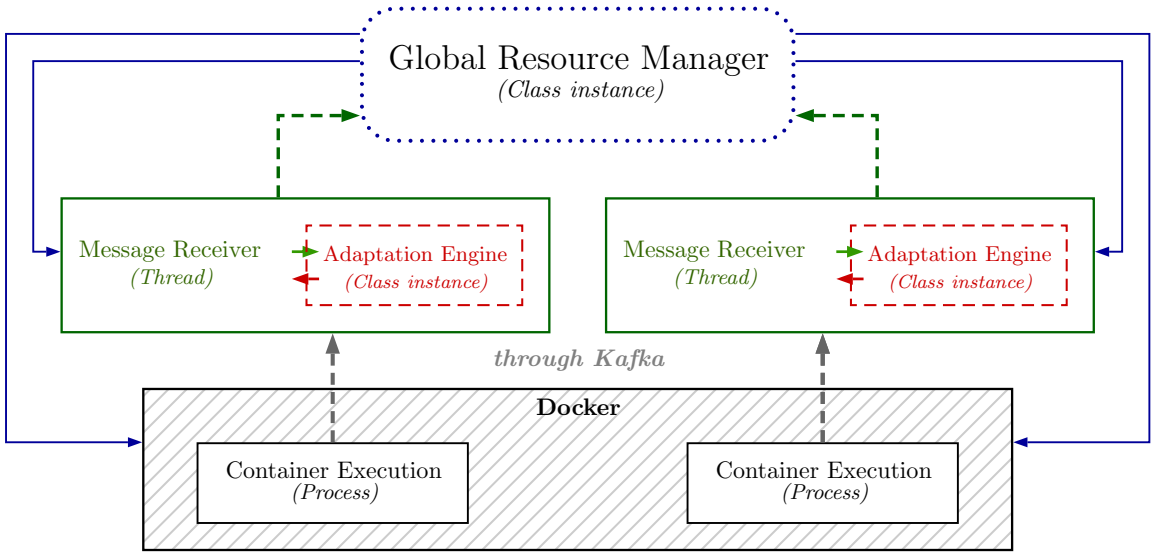


Figure 5.6 – Global system architecture, integrating the estimation process

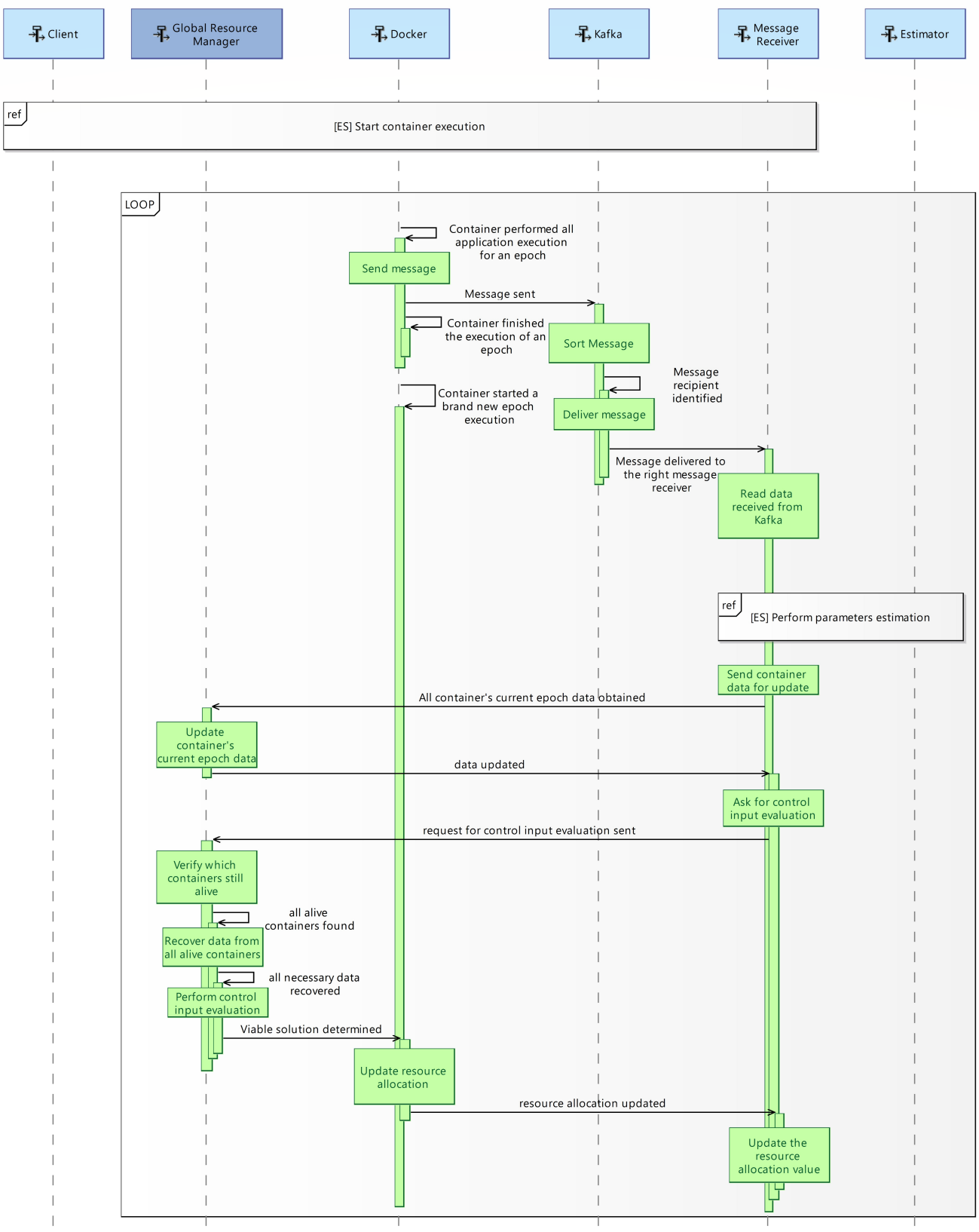


Figure 5.7 – Sequence Diagram (SysML representation) for the global control strategy operation

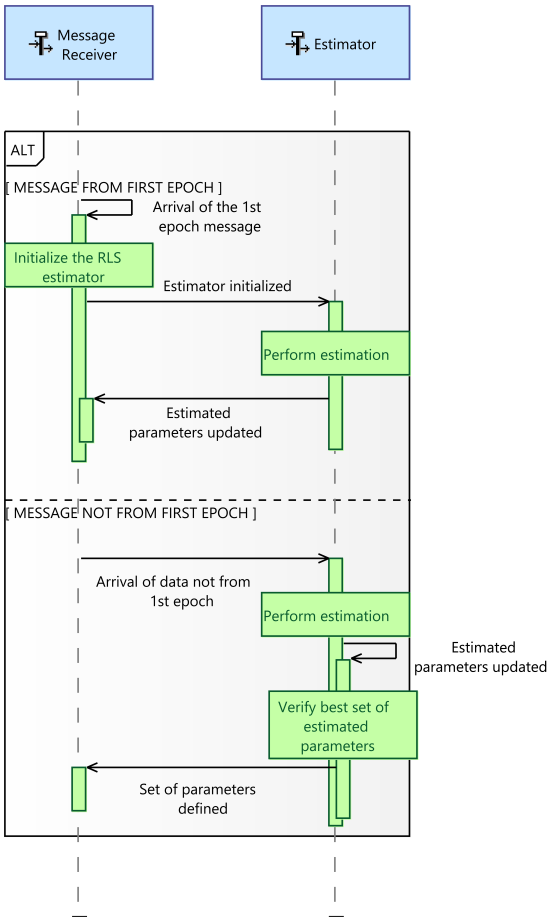


Figure 5.8 – Sequence Diagram (SysML representation) for the operation of the estimator

Algorithm 7: The optimal adaptive control input evaluation algorithm

Input : The identification of the container **container_id** whose resource allocation is about to be updated;
The total amount of resources available **total_cpus**;
The data $(cpus(k-1), ttf(k-1))$ from **container_id** ;
For the other running containers, their current resource allocation **cpus(k)** ;
For each running container: the estimated value $\hat{\beta}$ and the set-point value sp.

Output: The amount of resources **cpus(k+1)** to be allocated to **container_id**

1 Estimate the execution time for the running containers with **cpus(k)**, using equation (3.1) with $\gamma_0 = 0.0$
2 Form vectors $\mathbf{b}(k_i)$, $\mathbf{r_s}$, $\mathbf{x}(k_i)$ and $\mathbf{u}(k_i)$ from (5.15)
3 Perform unconstrained optimal control law (Equation 5.14)
4 **if** $\boldsymbol{u}(k_i+1)^* < 0$ **then**
5 Perform Algorithm 6
6 **if** $\sum 1/\boldsymbol{u}(k_i+1)^* > \boldsymbol{total\_cpus}$ **then**
7 Perform constrained optimization given by (5.17)
8 **cpus(k+1)** $\leftarrow 1/\mathbf{u}(k_i+1)^*$

Chapter 6

Deployment of multiple containers using the OACS

Contents

6.1 Main objectives for the OACS validation 103
6.2 OACS validation for the deployment of a single container . . . 103
6.2.1 Design of test scenarios 104
6.2.2 Performance Analysis 105
6.2.3 Conclusion 106
6.3 OACS validation for the deployment of multiple containers . . 107
6.3.1 Test case 1: Constant set-point with changes in environment 108
6.3.2 Test case 2: Varying set-point with no change in environment 113
6.3.3 Test case 3: Varying set-point with changes in environment 118
6.3.4 Conclusion 123

This chapter is devoted to fully describe and analyze the outcomes of testing campaigns designed to validate the performance of the optimal adaptive control strategy presented in Chapter 5.

The validation of the OACS is organized in an incremental difficulty approach of the system operation at run-time. First, tests are carried out for a single container execution. In contrast to Part I, this testing campaign covers all applications described in Appendix A. With the assurance brought by a proper performance in these cases, tests with multiple containers running at the same time are finally executed. Three test cases were conceived to cover diverse scenarios and provide guarantees about the system behavior when managing normal and critical situations.

6.1 Main objectives for the OACS validation

The OACS validation has the following goals:

1. Verify the correct deployment and operation of containers under a new structural architecture;

   The deployment of multiple containers corresponds to a multi-processing problem. Due to this, it is important to guarantee the right association among the different modules presented in Figure 5.6 with the data related to the right container which requested a resource allocation update.

2. Verify the performance of the control strategy:

   2.a) The resource allocated to each container must allow them to achieve set-point tracking in an environment where there is enough resources to satisfy their SLA;

   2.b) The resource allocated to each container must provide an optimal solution that minimizes the set-point tracking error, when there isn't enough resources to satisfy all SLA.

3. Verify the performance of all algorithms designed to deal with some particular behaviors;

4. Verify if the initial hypothesis about no coupling variables between containers affects the system performance.

6.2 OACS validation for the deployment of a single container

The first testing campaign considered during the OACS validation corresponds to the simplest possible one: its performance in an environment where only a single container runs an application. Compared to the strategies validated in Part I, the control strategy described in Chapter 5 has its design reformulated, as well as new mechanisms integrated on it. Due to this, it is necessary to validate the same aspects once verified in Chapter 4, but also taking the new system structure into account.

This part of the validation process can be seen as a preliminary step for the execution of tests with multiple containers. For such case, applications with different sorts of workloads are fed into the same environment. Because of this, it is important to certify the control strategy robustness regardless of the type of iterative workloads. Each application presented in Appendix A is individually tested in this part.

6.2.1 Design of test scenarios

A broader test scenario than the one described in Chapter 4 was designed, covering the same main aspects:

— when a constant set-point step signal is provided as the input;

— when the amplitude of the set-point step signal changes;

— when a resource usage limitation is imposed to a container.

Additionally, a new aspect includes the change in the amplitude of the set-point step signal before the resource usage limitation. In this specific situation, the estimator has not considered yet the zone of the workload characteristics whose new set-point belongs to. Moreover, Algorithm 3 does not provide a better way to manage the transient phase for this situation, giving the lack of knowledge about the new zone.

Figure 6.1 shows how the set-point value changes over the different steps composing each test case. The set-point value for each step is defined given an initial value represented as *sp*, provided as the input at the moment a new container execution is required.

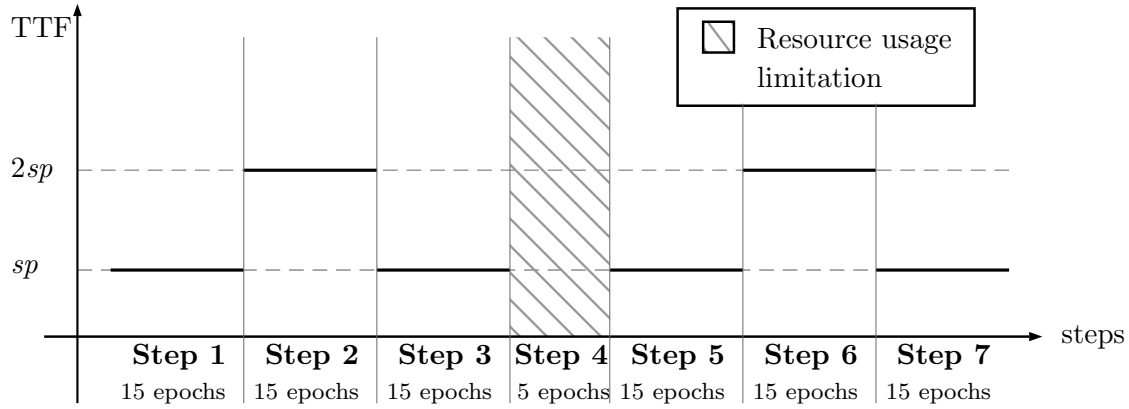


Figure 6.1 – Test scenario

Each step was designed to verify a certain aspect of the whole strategy under a specific circumstance, described as follows:

Step 1: the effect of the first measures taken into account by the estimator;

Step 2: the effect on the whole strategy when a change in the operating point is required, and there is no *a priori* knowledge about $\hat{\beta}$ on the new point;

Step 3: the effect on the whole strategy when the system must return to its initial set-point value;

Step 4: the effect of the resource usage limitation on the estimated parameters, specially on $\hat{\beta}$;

Step 5: the system's recovery from a constrained environment;

Step 6: the effect on the whole strategy when there is enough knowledge about the workload characteristics;

Step 7: the same situation of Step 3.

For each application running inside a container, four different initial set-point values are considered to generate test cases. For the deep learning training processes described in A.1.1 and A.1.2, the initial set-point values are 12.5s, 25.0s, 50.0s and 100.0s. For the video encoding process described in A.2, they are 15.0s, 30.0s, 60.0s and 120.0s.

A total of 12 test cases are performed in this testing campaign. Each one are deployed on the same single machine from NACAD described in section 4.2.1.

6.2.2 Performance Analysis

The following performance analysis of the OACS with only a single running container considers the test cases whose data can be found in Appendix C.

Initial operation of the control strategy

Regarding the initial resource allocation, provided by Algorithm 4, for a container which is about to start its execution, high overshoots in execution time are verified when the first allocation isn't near to the set-point value. In spite of that, the system converges quickly around the set-point value.

The introduction of Algorithm 5 to autonomously provide to the estimator a guess for $\hat{\beta}$ confirms the supposition that the use of the same initial value for all applications is not convenient. Different workloads present distinct characteristics. Therefore, the value of the initial guess for $\hat{\beta}$ is also different as the workload changes, which can be seen in Figure C.1. Adopting the value provided by the algorithm allows the transient phase of the control action to be shortened, specially in cases with small set-point values. In such cases, the value provided by the algorithm is close enough to the one verified in steady state. Even when this is not confirmed, the control strategy achieves its main goal and the estimated parameters converges in steady state.

Changing the set-point value before the resource usage constraint

For the steps in which the set-point value changes before the constraint being applied, the system does not have enough knowledge at this specific moment to drive directly to the new operating point.

Regarding the change of set-point value, the two possibles scenarios are individually analyzed, as follows:

— when the set-point changes from a small value to a higher one (Step 2):
In this case, almost all test cases present an important overshoot in execution time, specially in operating points located in the neighborhood of the knee point of the workload characteristics. The estimated parameters, in this phase, continuously change their values to fit the recent data integrated on the method. Even without the estimated value convergence, the execution time converges around the set-point value.

— when the set-point changes from a higher value to a small one (Step 3):
The execution time tries to return to its initial set-point value. Algorithm 3 provides the highest available $\hat{\beta}$ value to deal with the transient phase. However, the overshoot in resource allocation cannot be avoided without a more general knowledge about this parameter. In spite of that, the execution time converges around the set-point value.

Changing the set-point value after the resource usage constraint

In this phase, the estimator has already dealt with a large range of operating points. With the use of Algorithm 3, different sets of estimated parameters values can be applied depending on the current execution time and the desired set-point value.

After the end of the resource usage constraint, where the highest set-point error of each test case is verified, the execution of Algorithm 6 results on the absence of overshoot in resource allocation. In cases with the highest set-point value considered, where the sets of estimated parameters taken into account in this algorithm had similar values, no improvement in performance is verified.

The control strategy performance after the resource constraint emphasizes the importance of Algorithm 3 to provide the correct estimated value in every moment. The results provided by the online estimation stills relevant even after so many epochs performed. It specially provides to the control input evaluation the values allowing to better manage change in set-point value and even some possible fluctuations in the workload characteristics.

6.2.3 Conclusion

The OACS succeeds in providing a satisfactory resource allocation for all three workloads considered in this work. The times in which a SLA requirement could not be reached were due to the uncertainties about the characteristics. Even

though, the mechanisms developed in this work helped to drive quickly the execution time of each container to the desirable set-point value.

6.3 OACS validation for the deployment of multiple containers

In this new phase of the OACS validation, the main purpose is to verify how the strategy manages all containers requirements, including changes at run-time.

Different from the past section, each test case has its description, results and performance analysis presented altogether. This aims to facilitate the comprehension since each one of these cases includes several scenarios.

The following test cases were designed to consider two major changes: the variation of the total amount of resources available for the running containers and the change in set-point values.

The results from each test case are no longer displayed referring to a container epoch as a time unit. As different containers are considered in a same test case, each moment where a resource allocation is updated is called as a *Step*. The scheduled changes in environment take place according to a given pre-defined step.

Associated with the deployment of the OACS, each test case also includes a CPU Clock Speed monitor to verify its performance level. This monitor takes every second the CPU Clock Speed provided by the command `lscpu | grep MHz` called by a python script.

The outcomes of each test case are displayed in two figures. The first one is dedicated to the performance of the OACS, showing the execution time and the corresponding resource allocation for each container, as well as the total allocated number of resources and the CPU Clock Speed for each Step. These data are gathered together due to the correlation between them in important aspects of the global strategy performance. Some pertinent events to be presented in the following are only possible to be analyzed by looking closely to this data all at once.

The other figure corresponds to the estimation performance of each running container, showing both estimated parameters given as a result of these processes.

6.3.1 Test case 1: Constant set-point with changes in environment

In this first test, each container is assigned to satisfy a given SLA - a constant set-point value - during all its execution. The containers are set to run in different moments (steps) of the control strategy operation. In this way, the effect induced by the introduction of new containers can be verified and validated.

6.3.1.1 Test case description

In this test case, the strategy must handle four containers presenting different workloads. The specifications for their deployment are presented in Table 6.1.

Table 6.1 – Configuration of each container designed to run at Test Case 1

Container ID	Workload Reference	Constant set-point value	Total number of epochs	Initialization Step
container_1	MNIST-ML	15s	50	1
container_2	Fashion-MNIST-ML	20s	65	4
container_3	Video encoding	25s	80	14
container_4	MNIST-ML	15s	50	110

Environment changes are applied at run-time. The following changes in the total amount of available resources seen in Table 6.2 are applied according to the current step in which the Resource Manager module deals with.

Table 6.2 – Steps in which the total amount of available resources is changed in Test Case 1 and its related value to be applied

Step	Total amount of resources available
1	6.5
57	4.5
80	5.0
110	6.0
145	6.5

6.3.1.2 Test results

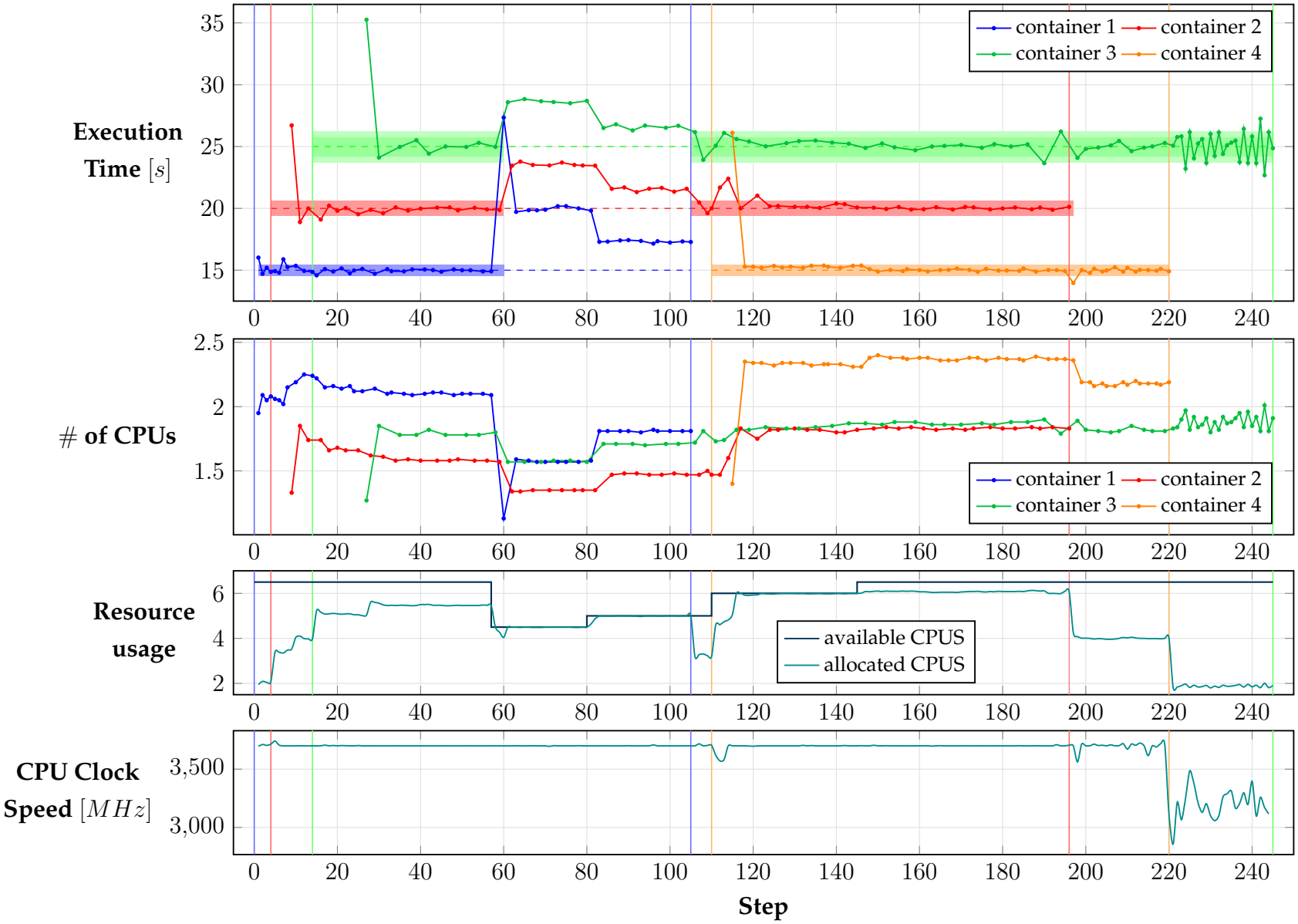


Figure 6.2 – Performance of the OACS in Test Case 1

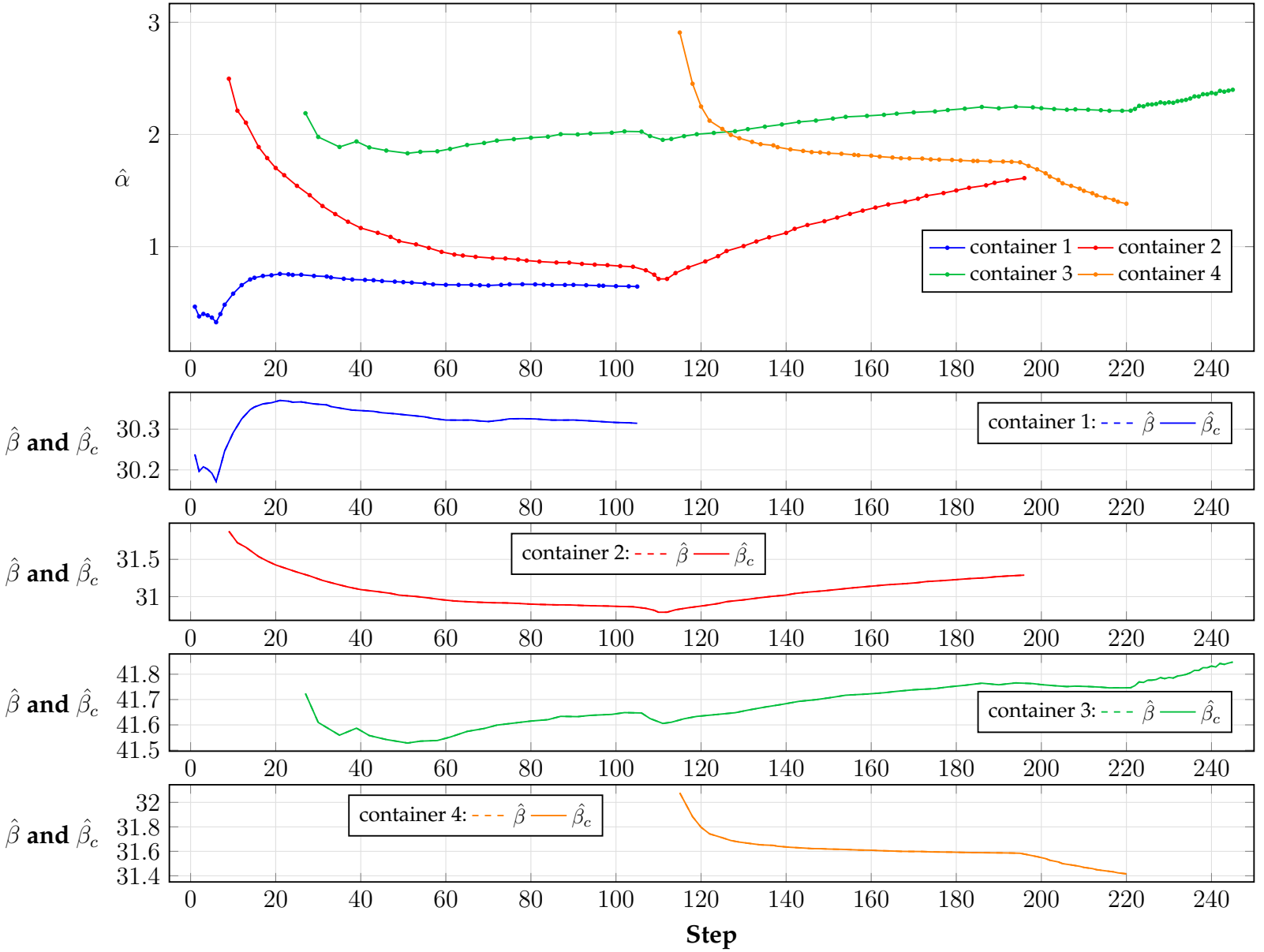


Figure 6.3 – Performance of each estimation process performed in Test Case 1

On Figure 6.2 and for the following ones which also present test results in this section, some specific marks displayed on them allow the reader to verify some performance aspects.

First, the sub-figures displaying the Execution Time, # of Cores, Resource usage and CPU Clock Speed present some vertical lines. For each container considered in a test, there are two lines at the corresponding color seen on the label: the first, in the smallest step number, marks the step in which a container is set to run. The last one marks that its final performed epoch.

In addition, dotted horizontal lines seen in the Execution Time sub-figure represent the set-point value of containers in a given step. The stability margin is also included and represented by small rectangles only displayed when no constraint is verified, for the sake of clarity.

6.3.1.3 Performance Analysis

The following analysis chronicles the most important events of this test case, mainly verified in Figure 6.2.

Steps 1, 4, 14 and 110: Initialization of containers

With the initial resource allocation, provided by Algorithm 4, all four containers start with high execution time, with a negative set-point error. Except from the first container, the remaining ones had their values with a high amplitude. The first three initialized containers reached their SLA in an unconstrained resource usage environment, what can be seen until Step 57.

Steps 57 and 80: First changes in the total available amount of cores

The first container to face the constraint in Step 57, container_1, is the most penalized in its performance, presenting an elevated execution time. However, this momentary behavior is surpassed only one epoch later, allowing the other containers to be driven directly to their new operating point, representing the constrained optimal solution. At Step 80, where the total amount of resources has a slight increase, each container reacts as fast as during the first constraint to reach their new operating point.

Step 105: container_1 finishes its execution

A strong decrease of the total amount of allocated resources is verified and the remaining containers direct quickly their execution time to the stability zone.

Step 110: container_4 is initialized

The CPU Clock Speed presents, for the first time during this test case, a slightly reduction in its value. As a consequence, the execution time for the other

containers presents an increase on their set-point tracking errors for a few epochs. The strategy itself manages this situation, as seen in Figure 6.3 in the value provided by the estimator of each running container.

Step 118: container_4 reaches set-point tracking

In this moment, requirements of all containers are satisfied, even using the total available amount of resources.

Looking specifically to the resource allocation of container_2 and container_4, two different operating points, in distinct moments of this test case, allow them to achieve set-point tracking. For container_2, a small amount of resources was allocated between steps 20 and 60 and a higher one, starting in step 120 until its final epoch execution.

To understand what causes this change of operating points in different moments of the test case operation, it is necessary to look at the resources availability. The highest total amount of allocated resources applied on this test case is verified during the last referred moment. With a large amount of cores being used, their parallel capacity also reaches an upper bound, which explains the growth in the number of resources in this case in order to satisfy the same SLA.

Step 195: container_2 finishes its execution

There is a strong increase of resource availability, enlarging as a result the parallel capacity and decreasing the amount of resources assigned to satisfy a SLA, which is clearly seen for container_4.

Step 220: container_4 finishes its execution

The amount of allocated resources is reduced to the smallest value seen on this test case. Such event is also followed by a decrease of the CPU Clock Speed value and the start of oscillations verified around an inferior value than the one in all previous steps.

As the CPU Clock Speed changes, it affects directly the workload characteristics. At a given speed, each iterative workload has an unique characteristic. Reflecting the oscillation on the clock speed, the estimator tries to adapt their parameters to the new context, which can be seen by the increase verified on the value of $\hat{\beta}$ for container_3 in Figure 6.3. But despite that, the resource allocation does not stabilize as in the previous steps. The oscillation on the amount of cores also reflects on the execution time, which oscillates around the set-point value and in most of the steps, inside the stability zone.

To sum up, this test case revealed three different events which affect directly running containers, impacting on the OACS performance:

(i) the introduction of a new container in the same infrastructure in which containers are already running;

(ii) the end of a container execution;

(iii) the presence of oscillations in the CPU Clock Speed value, specially when there is only a container left running.

The OACS in this case, manages to properly satisfy its main goal when the first two events take place. Although not so performant as for these two events, the strategy also reaches an acceptable performance for the last event, considering that it does not control the new variable identified.

6.3.2 Test case 2: Varying set-point with no change in environment

This test aims to verify the OACS performance at satisfying changes in SLAs. In this case, varying set-point is assigned to each container considered. In order to analyze the influence of this specific condition, no change in the total available amount of resources is included in this case.

6.3.2.1 Test case description

Table 6.3 presents the initial configuration of each container considered in this test case. In addition, Table 6.4 shows the changes in set-point for each container. They were previously defined to happen one at a time, allowing to verify how the global strategy deals with the introduction of a new context. Each change is applied considering which epoch a given container executes.

Table 6.3 – Configuration of each container designed to run at Test Case 2

Container ID	Workload Reference	Total number of epochs	Initialization Step
container_1	MNIST-ML	65	1
container_2	Fashion-MNIST-ML	75	4
container_3	Video encoding	65	14
container_4	MNIST-ML	65	120

Table 6.4 – Changes in set-point for containers designed to run at Test Case 2

Container ID	Initial set-point value	Intermediate set-point value	Epoch to execute the first change	Epoch to execute the final change
container_1	17.5s	25.0s	25	55
container_2	15.0s	30.0s	32	50
container_3	22.5s	35.0s	23	55

6.3.2.2 Test results

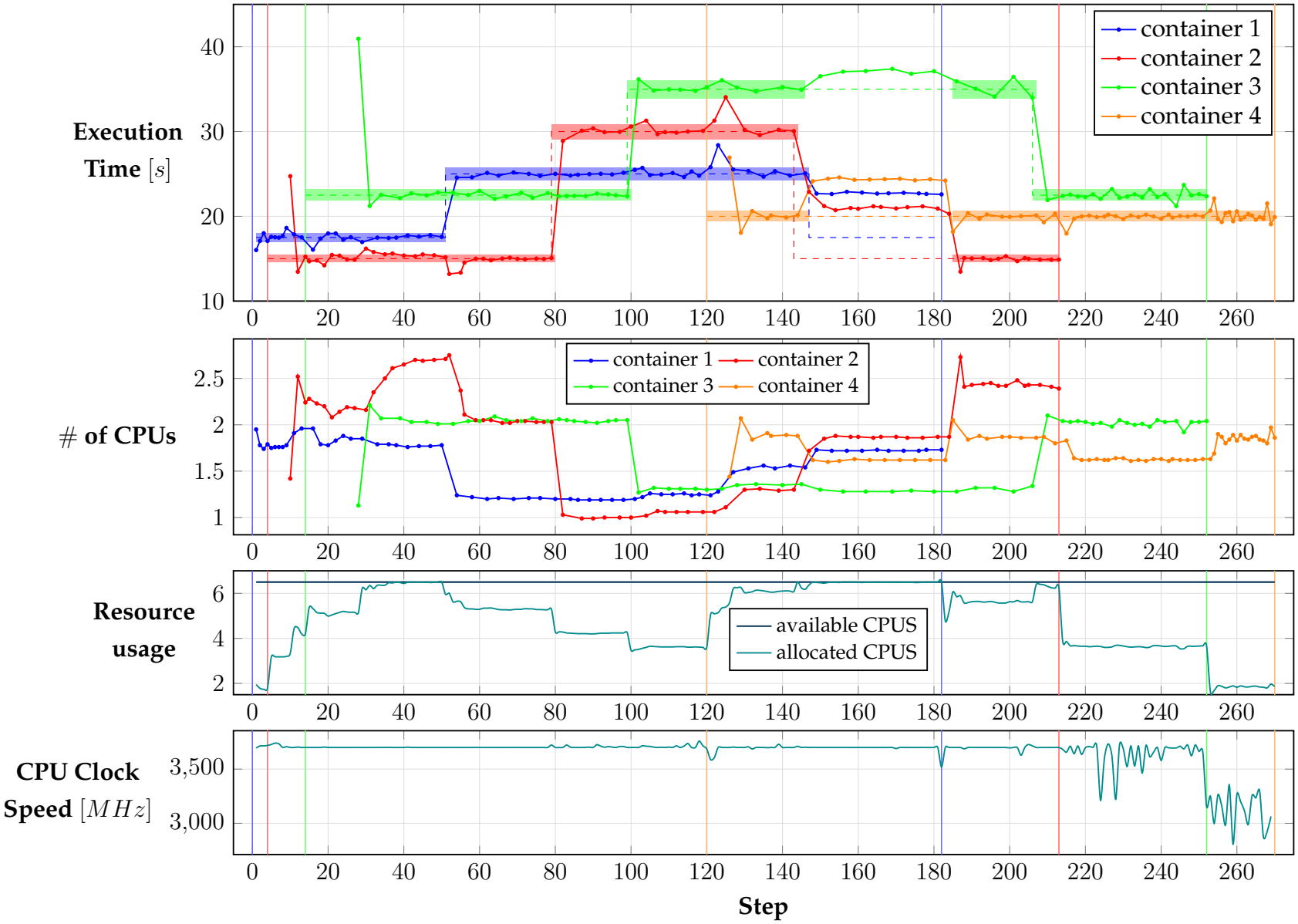


Figure 6.4 – Performance of the OACS in Test Case 2

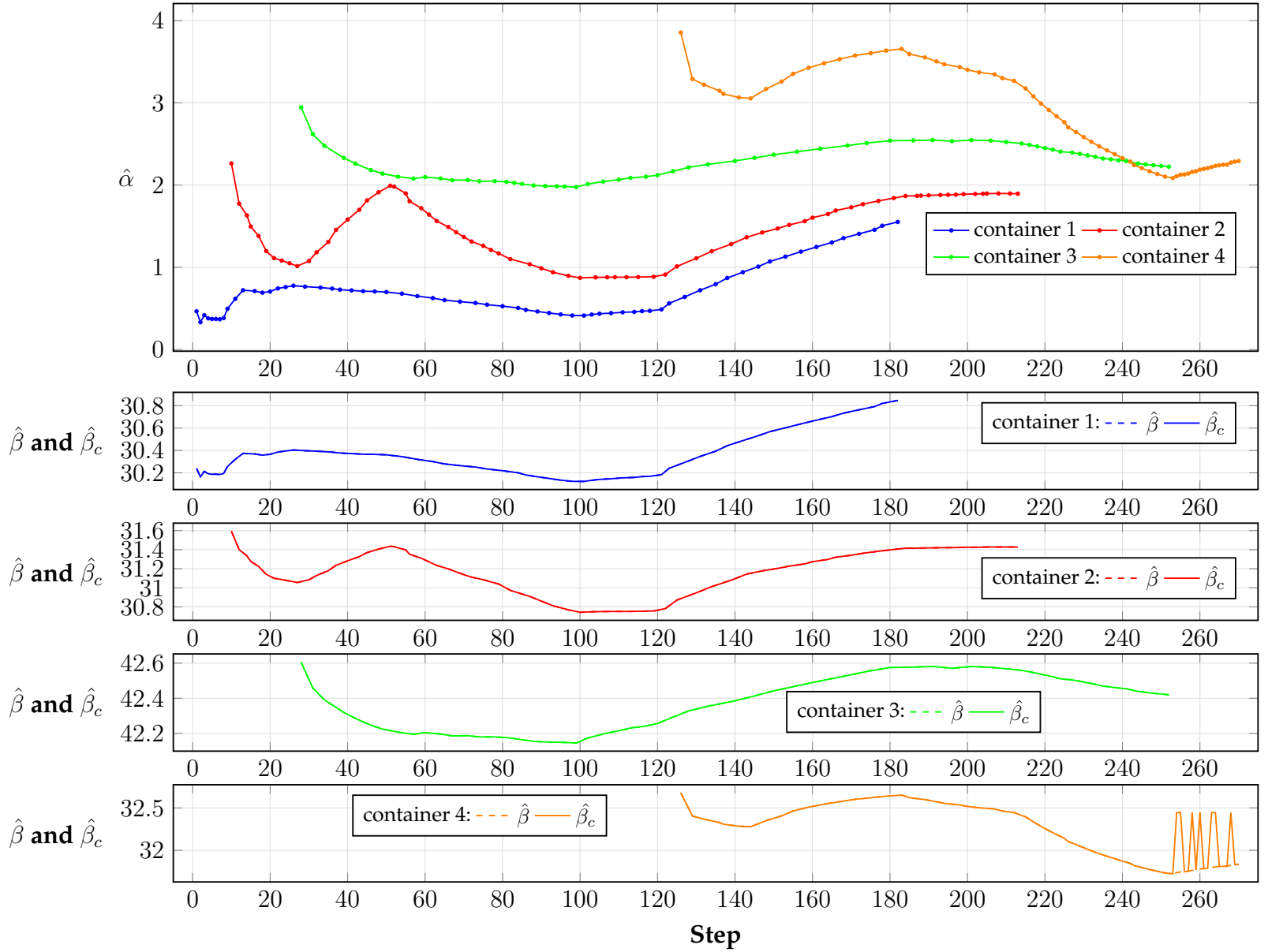


Figure 6.5 – Performance of each estimation process performed in Test Case 2

6.3.2.3 Performance Analysis

The following analysis chronicles the most important events of this test case, mainly verified in Figure 6.4.

Steps 1, 4 and 14: Initialization of container_1, container_2 and container_3, respectively

The initial allocation value assigned to container_1 is really close to the one capable to satisfy its SLA. Due to this, the estimator captures at the beginning of its operation the workload characteristics for this specific location. As soon as the execution time pulls away of the stability zone, around Step 10, its estimated values begin to change, as seen in Figure 6.5. This has an impact on future resource allocations and finally, on the execution time.

An unexpected behavior is verified on the resource allocation for container_2. After the resource allocation update for the second epoch of container_3, around Step 30, a significant change is verified in the amount of resources assigned to that container. Moreover, after container_3 starts running, the OACS deals with a considerable amount of resources allocated to all running containers. The parallel capacity is then extremely reduced, which explains the strong decrease of CPU efficiency. In addition, the performance of container_1, in this same moment, seems to not be affected as for the aforementioned container.

Step 50: Change in the set-point value for container_1

This first set-point change occurs in a moment with a extremely reduced parallel capacity. When this container reached its new SLA, more resources became available for the other running containers. As a result, container_2, which had a considerable increase of resources allocated to satisfy its SLA, experiences a strong reduction on it due to the boost in parallel capacity.

Steps 79 and 99: Change in the set-point value for container_2 and container_2, respectively

This changes are perfectly managed by the OACS, driving them quickly to their new defined requirement. In terms of resources availability, all of them unlocks an important amount of resources. However, only the first increase in parallel capacity, around Step 50, is significant for the other containers performance. This is comproved by the considerable drop in the amount of resources verified by container_2.

Step 120: Initialization of container_4

The introduction of container_4, in Step 120, also places the total allocated amount of cores on a high threshold. The reduction of parallel capacity is, in this

case, perceived by the other running containers, given that all of them present unusual set-point tracking errors. They have these errors quickly reduced by the action of the OACS, but in new operating points. Their new resource allocation verified around Step 130 indicate a change in their workload characteristics.

Steps 143 and 146: Change in the set-point value for container_2 and container_1, respectively

With these changes, more cores are required to specific containers. Additionally, the total allocated amount is already very close to the upper bound initially defined. As a result, these changes are not satisfied because of the lack of available resources. The solution provided by the OACS drives all containers in an optimal and stable condition. Once more resources are released by the end of container_1 execution, all remaining containers are quickly directed to reach their SLA.

Steps 213 and 253: container_2 and container_3 finish their execution, respectively

As the number of running containers decreases and the total amount of resources returns to not high values, some oscillations on the measured CPU Clock Speed are observed. The characteristics of container_4 is modified, given the change in the amount of cores to satisfy the same SLA. When this one is the only remaining container, the decrease on the clock speed is handled by the use of Algorithm 3. It provides to the control input evaluation the most suitable value of $\hat{\beta}_c$, as seen in Figure 6.5.

This test case confirms the final conclusions established on the previous test case. It is even more evident that the amount of cores allocated to a single container also depends on which level of resources availability and parallel capacity the infrastructure provides.

6.3.3 Test case 3: Varying set-point with changes in environment

This final test case aims to verify the performance of the OACS given that containers present SLAs defined by varying set-points. In addition, this test case also applies a limitation of the total amount of available resources.

6.3.3.1 Test case description

Only three containers were designed to perform this test case. Table 6.5 presents their initial configuration. Table 6.6 shows the changes in their set-point value through their corresponding epochs. Finally, Table 6.7 exposes the environment changes applied during the test case operation.

Table 6.5 – Configuration of each container designed to run at Test Case 3

Container ID	Workload Reference	Total number of epochs	Initialization Step
container_1	MNIST-ML	150	1
container_2	Fashion-MNIST-ML	80	4
container_3	Video encoding	70	14

Table 6.6 – Changes in set-point for containers designed to run at Test Case 3

Container ID	Initial set-point value	Intermediate set-point value	Epoch to execute the first change	Epoch to execute the final change
container_1	15.0s	20.0s	44	120
container_2	17.5s	15.0s	42	60
container_3	22.5s	30.0s	37	49

Table 6.7 – Steps in which the total amount of available resources is changed in Test Case 3 and its related value to be applied

Step	Total amount of resources available
1	6.5
70	5.0
190	6.5

6.3.3.2 Test results

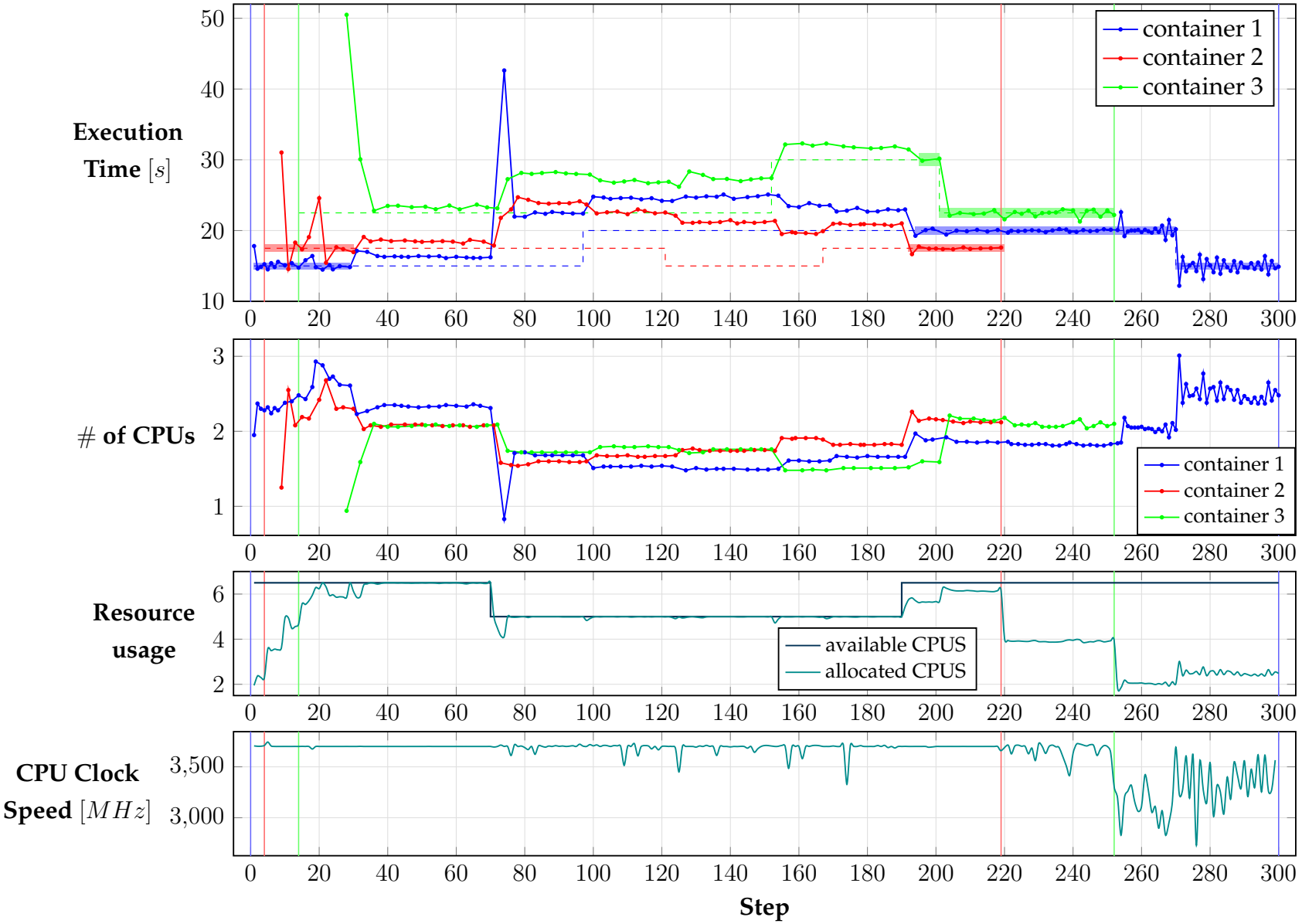


Figure 6.6 – Performance of the OACS in Test Case 3

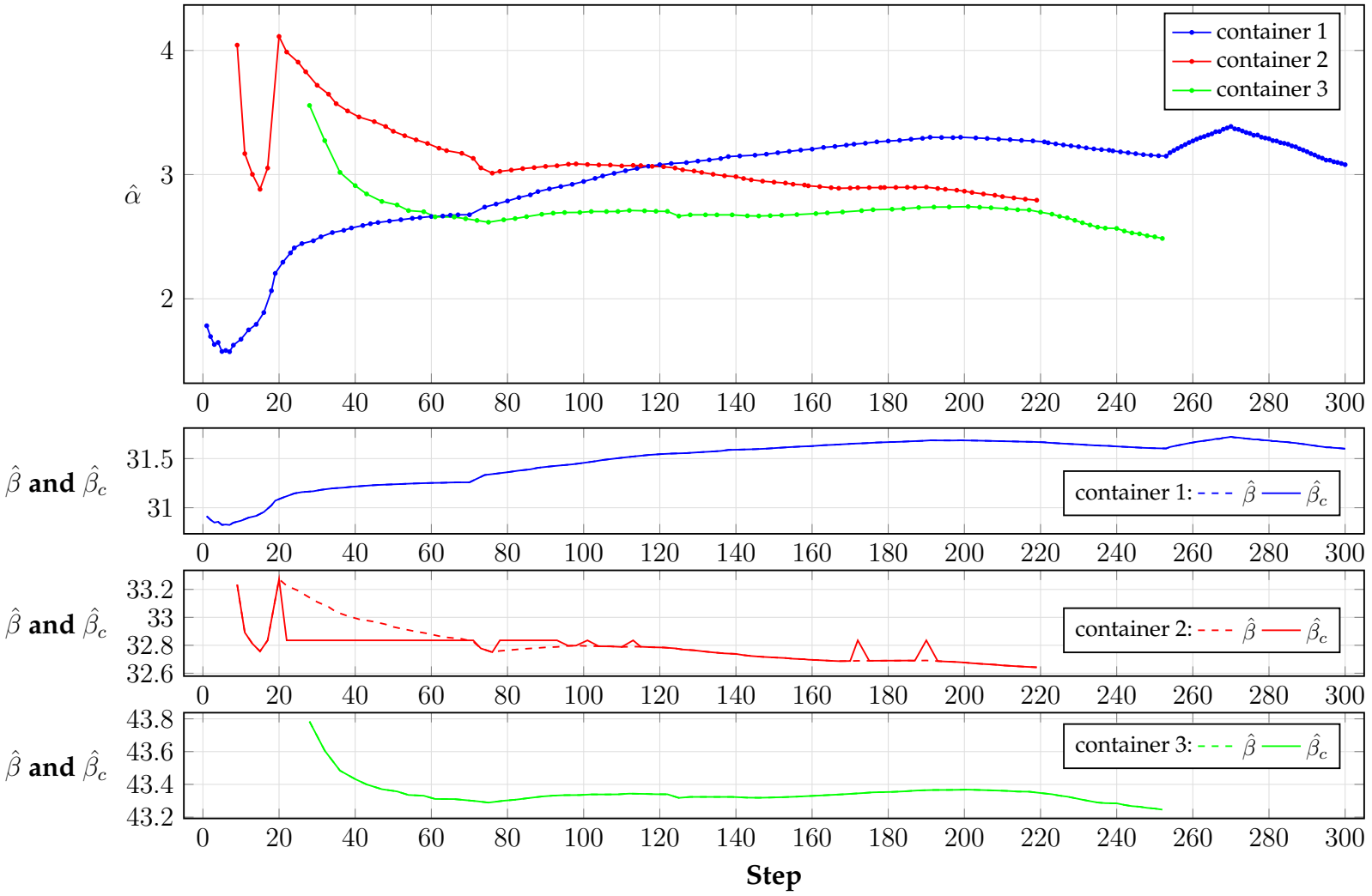


Figure 6.7 – Performance of each estimation process performed in Test Case 3

6.3.3.3 Performance Analysis

The following analysis chronicles the most important events of this test case, mainly verified in Figure 6.6.

Steps 1, 4 and 14: Initialization of containers

The introduction of container_3, in Step 14, disturbs the performance of the other running containers. The strong variation in execution time presented by container_2 is handled by the action of Algorithm 3, which selects the most suitable value for $\hat{\beta}_c$, as seen in Figure 6.7. As the execution time for this container approaches its desirable set-point value, the main problem turns to a constrained optimal one. All containers are quickly driven to a new operating point.

Step 70: Decrease of the total amount of resources available

As seen in Test Case 1, the first container to be in contact with this new restriction, container_1, is the most penalized one. On the other hand, considering all running containers, this approach allows the other ones to reach faster their new operating point.

Steps 97, 121, 152 and 167: Changes in set-point values in an environment with a constrained amount of resources

Each change in set-point value causes an update on the implemented cost function. As a result, in these Steps, each change directed the optimal solution to be updated to new operating points. Once the resources constraint is released, in Step 190, all containers return to their own stability zones.

Steps 220 and 252: container_2 and container_3 finish their execution, respectively

Differently from the other test cases, the strong release of resources around Step 220 does not provoke any changes on the operating point for the running containers, even with the small oscillations verified on the CPU Clock Speed value.

Only after the second strong reduction on the allocated resources, with a more important oscillation verified on the clock speed, a change in the workload characteristics of container_1 can be seen.

Step 270: Change in the set-point value for container_1

Even with significant oscillatory behavior on the CPU Clock Speed, the OACS succeeds at driving it to the operating point nearer to the desirable set-point value, but also with an oscillatory response.

6.3.4 Conclusion

The test cases presented in this section reveal some important behavior when multiple containers run in the same environment. For the first time, the following aspects were identified as having a strong influence on containers performance:

(i) the initialization/ending process of a container in the same environment where there are containers already running their workload;

(ii) the available parallel capacity provided by the computational infrastructure seems to affect directly the workload characteristics for a container;

(iii) the variation on the CPU Clock Speed also presents a strong influence on the workload characteristics.

Even though none of these behavior were taken into account at the development phase of the OACS, it succeed at:

(i) Providing the right resource allocation for each application after few epochs, without a-priori parameters identification phase;

(ii) Being robust when changes in workload characteristics happens. However, only for constant set-point values.

(iii) Redirecting all running containers to a viable and optimal resource allocation regarding their SLAs when not all of them can be satisfied at the same time.

Concerning the time required to the whole system in order to reach a new operating point, the changes in environment or SLA requirements demands at least the same number of epochs than the number of running containers.

Conclusion

The main objective of this work was to propose a solution to the problem of resource allocation to applications containing iterative workloads in a cloud computing environment. Each container running an application is to be assigned a certain number of cores (the computational resource) in order that it be executed in no more than a certain upper bound on time, specified as the SLA.

Different models were developed by choosing well-known curves that reflect the relationship between number of cores and execution time for a given workload. These models were then used to develop adaptive control strategies for a prior approach taking only a container execution. One of them, considered the best at reaching system requirements, was then reformulated to extend the solution to the case in which multiple containers run in the same environment. This global solution consisted of an optimal adaptive control strategy, capable to manage requirements from different containers and provide a viable resource allocation even when not all SLAs can be reached.

The final tests performed in this work, deploying multiple containers, revealed that fluctuations in some infrastructure properties induce changes in the workload characteristics for the running containers. Although these events were not considered at the design phase of the global solution, the implemented adaptive approach was capable to manage them under some conditions. Therefore, the global strategy can be considered as an self-adaptive one, able to handle independently the several scenarios evoked in this work.

Perspectives

For the use of the OACS as a resource allocation mechanism in large scale, some other aspects should be considered. The following ones constitute a good indication of possible future works using all development already validated:

1. Scaling up to a larger number of containers running in the same environment. Another important aspect to be considered is how the computational infrastructure is organized. Two possible scenarios can be explored:

(i) a machine with more resource capacity. In this case, all conclusions made in this work must be validated in this new context;

(ii) a cluster of machines, commonly employed in cloud data centers. In this case, some work still need to do in order to manage the network concept.

It is also paramount to conduct Integration Testing to validate the communication among OACS modules and their correct synchronism of events with a more important number of running containers.

2. The possibility of developing another system model, also considering the CPU Clock Speed and the parallel capacity to provide the right amount of cores to a given container;
3. The extension of this work to other types of workloads. In this case, it might be necessary to adapt the estimation to deal with a more variant workload characteristic.

Appendix A

Profile of applications performed on tests and their workload characteristics

In this appendix unit, the different applications that were implemented inside Docker containers and considered in this work for the purpose of validation are introduced. Additionally, their workload characteristics are presented, determined by the relationship Number of Cores versus Execution Time considering a given computational infrastructure.

Trace study configuration

The workload characteristics for each application which is introduced in the following sections were obtained thanks to a trace study. By taking advantage of the structural infrastructure developed for a single container execution (See Section 1.2.1.2), the control module was deactivated and in its place, a simple algorithm was implemented to allocate the amount of resource to the application at the moment in which a Kafka message is received, indicating the start of a new epoch.

This study was conducted using the same computational infrastructure applied to the validation processes in this work.

A.1 Deep learning training processes

In deep learning processes, patterns are extracted from data by using a hierarchy of concepts, with each concept defined through its relation to simpler concepts [36]. A main problem can be then broken into smaller and simpler parts,

making less complicate for the algorithm to establish understandings that will lead to an intuitive solution. The Figure A.1 shows an example of a deep learning neural network (DNN). A series of hidden layers allow to extract features in a crescent level of complexity to accomplish the final objective of producing an output.

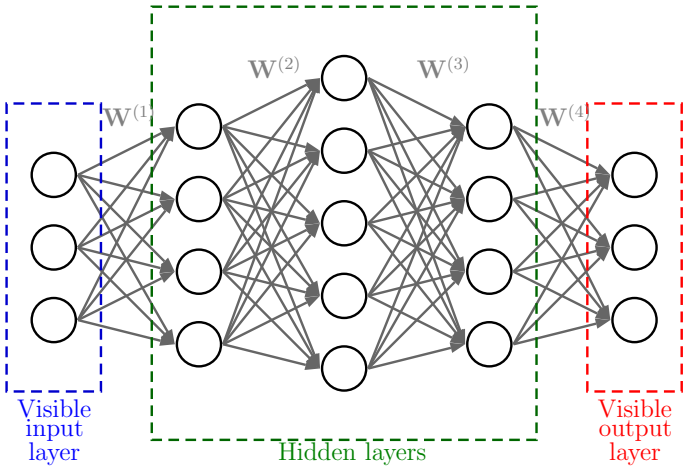


Figure A.1 – An example of a deep learning neural network

Both deep learning training processes to be described present the same network: there is only a hidden layer, in which some activation of nodes are randomly dropped. At the end, they are intended to classify the data into ten different outputs.

A.1.1 Case using MNIST database

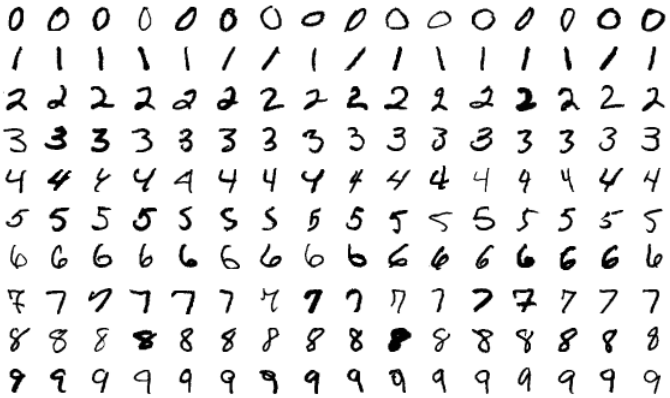


Figure A.2 – A few samples from the MNIST test data set.

In this case, the data set consists of examples of handwritten digits [37], as the ones presented in Figure A.2 [1]. The workload characteristics of the proposed

1. https://en.wikipedia.org/wiki/MNIST_database

DNN for that case is presented in Figure A.3.

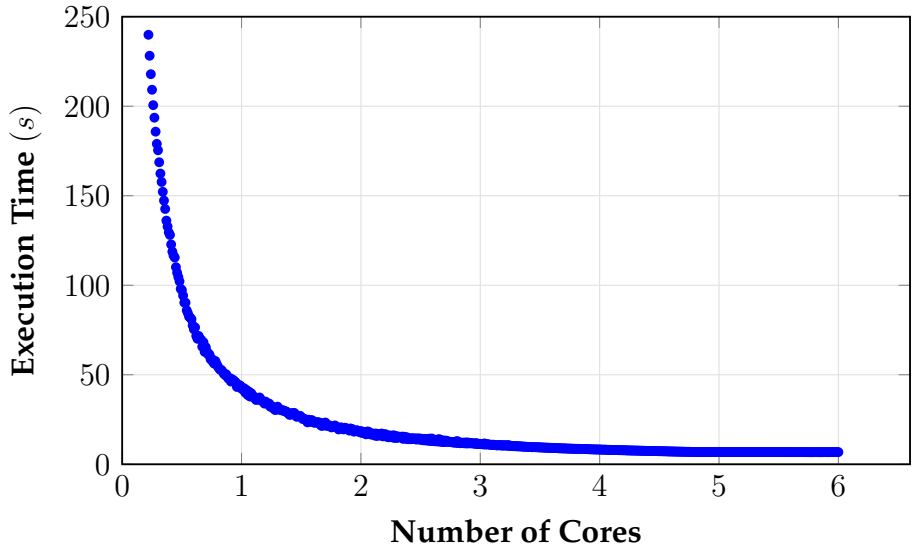


Figure A.3 – Workload characteristics for a DNN process using a MNIST database.

A.1.2 Case using Fashion-MNIST database

In this case, the data set consists of examples of Zalando's article images [38], as the ones presented in Figure A.4[2]. The workload characteristics of the proposed DNN for that case is presented in Figure A.5.

Figure A.4 – A few samples from the Fashion-MNIST test data set.

2. https://github.com/zalandoresearch/fashion-mnist

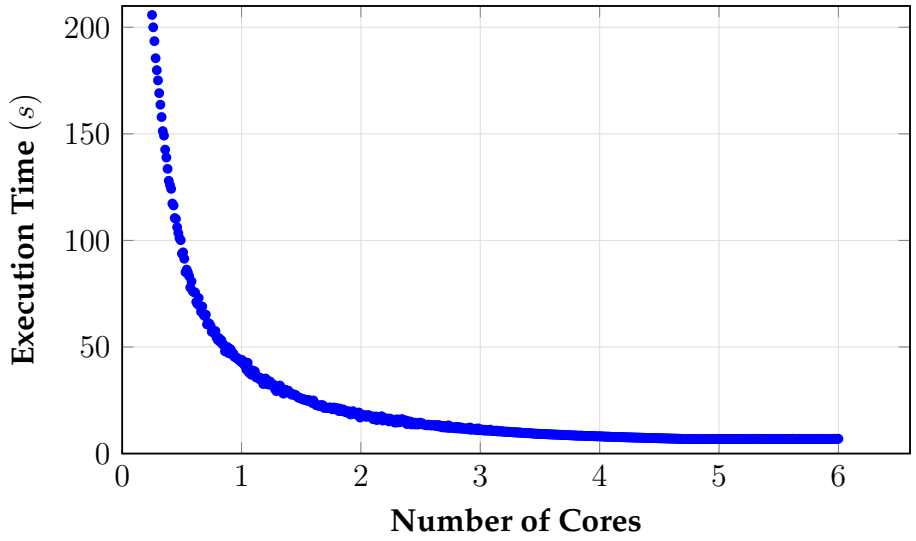


Figure A.5 – Workload characteristics for a DNN process using a Fashion-MNIST database.

A.2 Video encoding process

The video communication comprehends several steps to ensure that a video captured by a camera will be displayed with a certain QoS requirements by the receivers. One of these steps corresponds to the task of video compressing, called **video encoding process**, converting it into a given format. As a result of this process, a video becomes compatible to several devices and platforms. This is a major concern in the domain of video streaming industry, once it contributes to the bit rate reduction of the digital video signal.

Although the wide-spread video encoders use complex algorithms, the application considered in this work took advantage of a simpler encoder [3] developed as a case study in [39], which uses the function `convert` to manipulate the frames from a video. The control aspect of that work was not taken into account for the development of the application described bellow.

Development of an iterative workload container

As in streaming applications, the real-time video processing is implemented by splitting the video into multiple "segments", sets of video frames, which are progressively handled on this process.

The iterative workload execution implemented in this work corresponds to the encoding process performance in a segment, defining a step of the container execution.

3. The code is available at `https://github.com/martinamaggio/save`

In each step, a video segment consisting of 33 video frames has the quality of each frame adjusted. The quality argument of the `convert` function comprehends values between 1, indicating lowest image quality and highest compression, and 100, for best quality but least effective compression [40]. In this work, the quality argument value is set to 10.

Described as an important feature for fast performances [41], the algorithm takes advantage of parallelization capabilities to compute as much frames as possible in a given time considering also the available number of cores allocated to that task.

For the case of the container developed in this work, the video frames were extracted from the Youtube video of the Visit of the President of France, Emmanuel Macron, to the new campus of the Engineering School CentraleSupélec, during the Inauguration of its new facilities at the Paris-Saclay campus. [4] The Figure A.6 presents the workload characteristics considering this video as the source of the video frames applied on the encoding process.

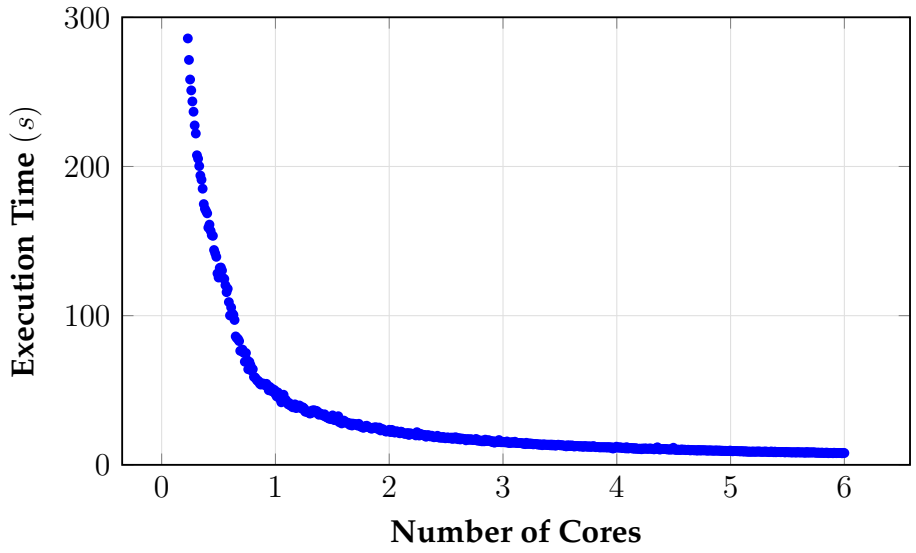


Figure A.6 – Workload characteristics of a video encoding process performed in a Youtube video

4. `https://www.youtube.com/watch?v=xJz6OJgrDnA`

Appendix B

Detailed results from tests of the adaptive control strategies for single container execution

This appendix unit aims to provide all data recovered from the validation process described in Chapter 4 and which based all conclusions established on the same chapter.

For each test scenario performed, the following data is displayed for each epoch:

— the execution time (TTF);

— the number of CPUs allocated to the application;

— the estimated parameters $\hat{\alpha}$ and $\hat{\beta}$;

— the parameter β_c;

— the estimation error.

The data to be presented is organized by type of control strategy implemented on a test case.

B.1 Control strategies which used an hyperbolic model for the input-output relationship

B.1.1 Control Strategy 1: System Dynamics given by the Taylor expansion of an hyperbolic function

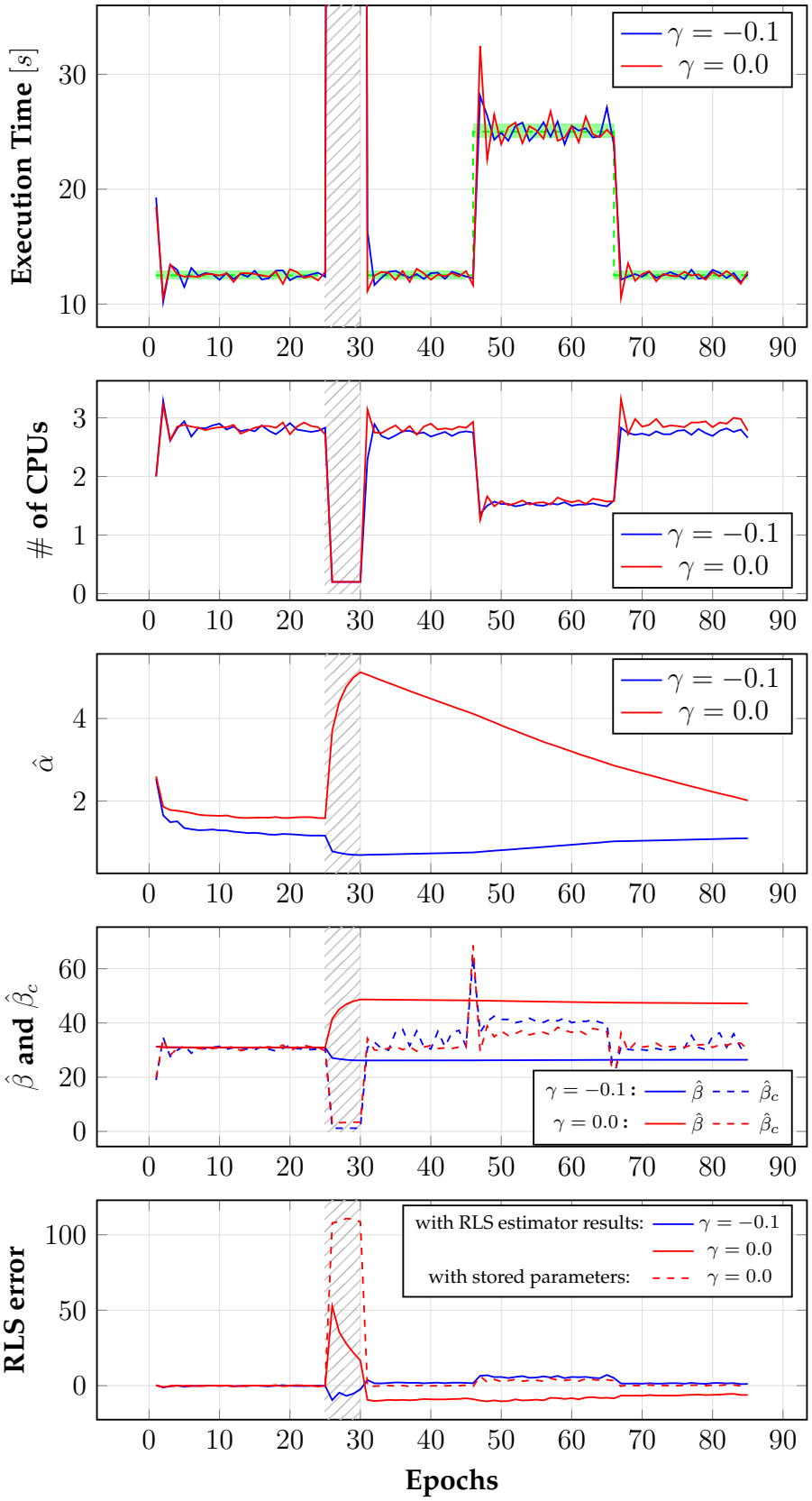


Figure B.1 – Performance of Control Strategy 1 for a initial setpoint value of 12.5s

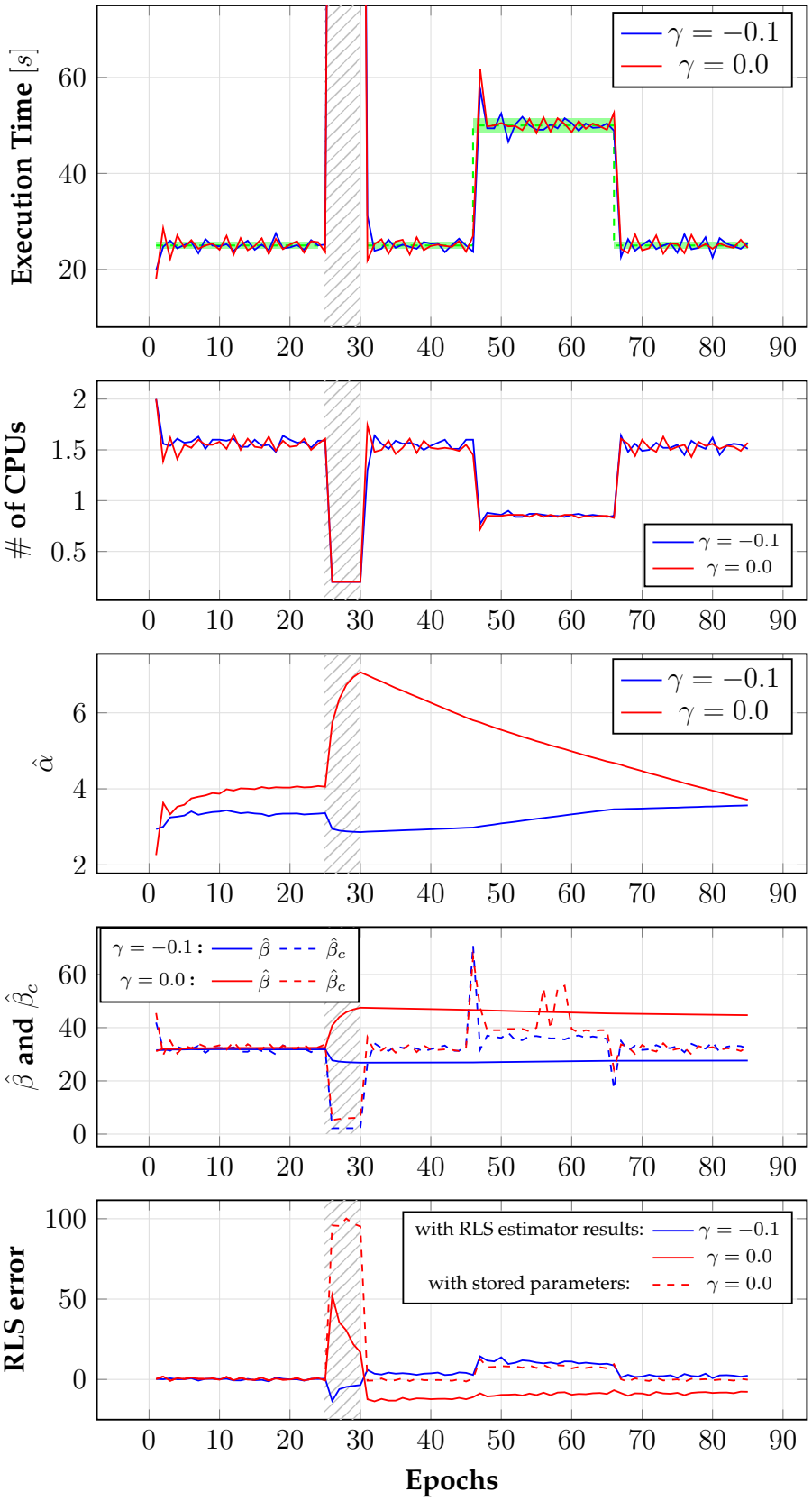


Figure B.2 – Performance of Control Strategy 1 for a initial setpoint value of 25.0s

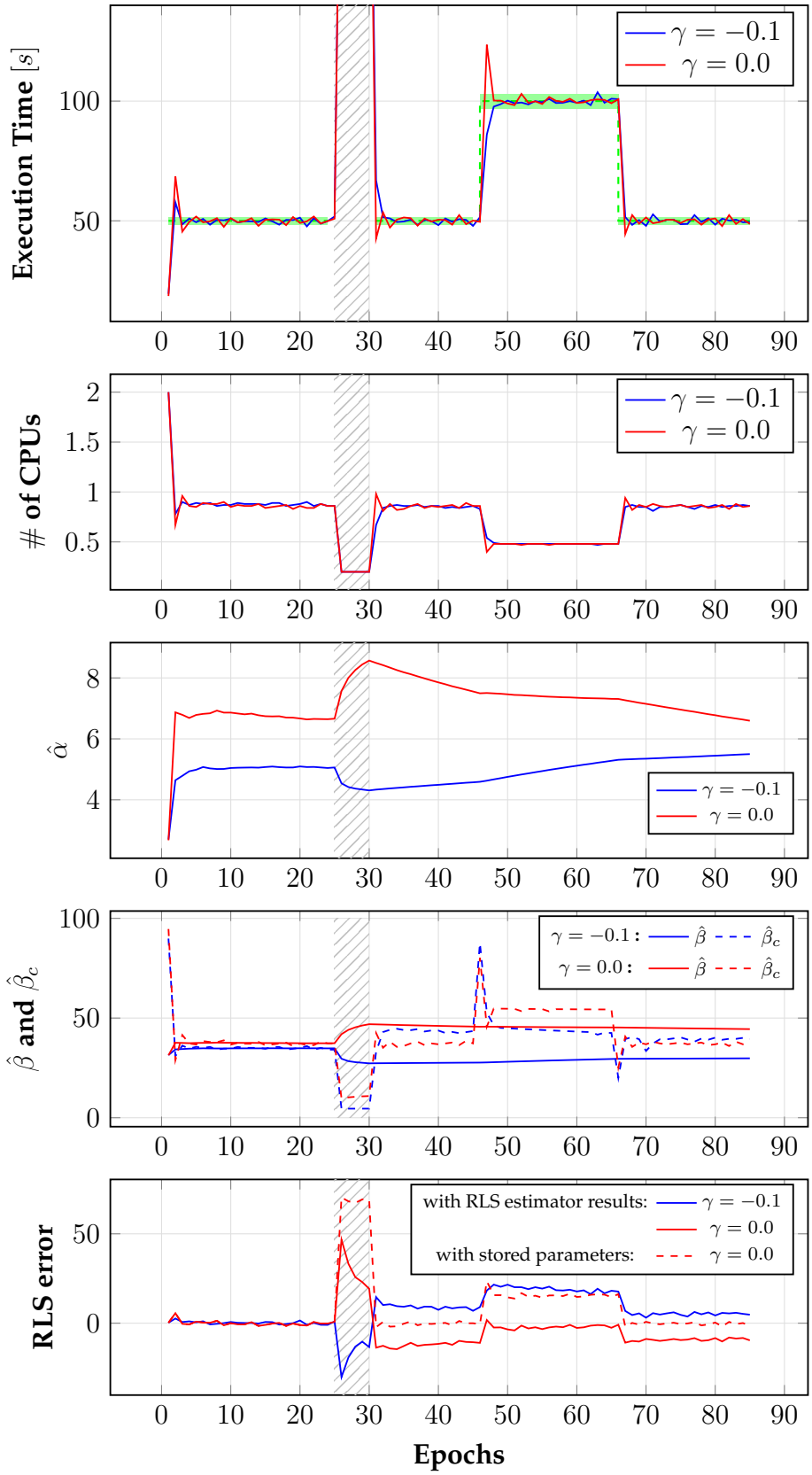


Figure B.3 – Performance of Control Strategy 1 for a initial setpoint value of 50.0s

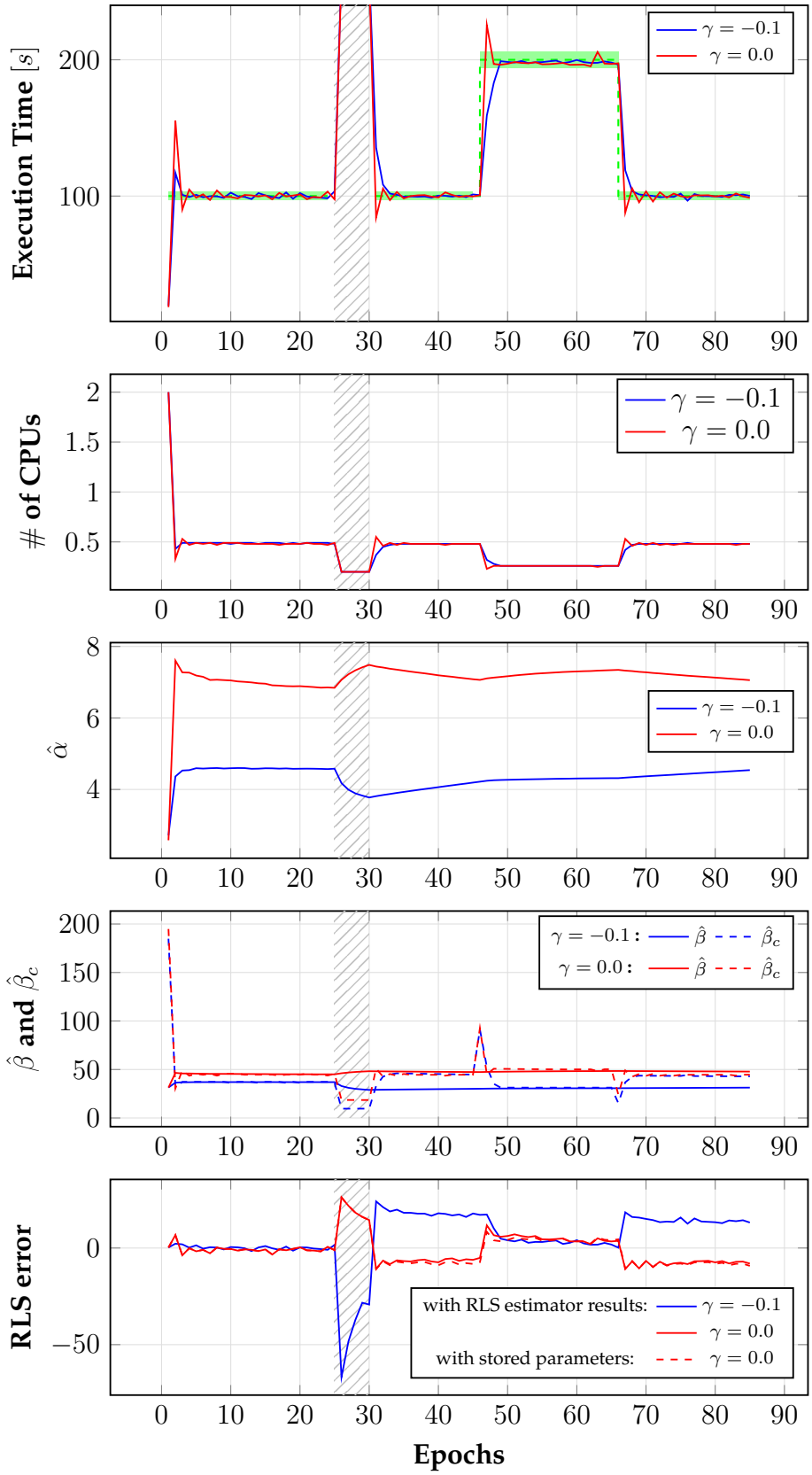


Figure B.4 – Performance of Control Strategy 1 for a initial setpoint value of $100.0s$

B.1.2 Control Strategy 2: System Dynamics given by a manipulation of an hyperbolic function

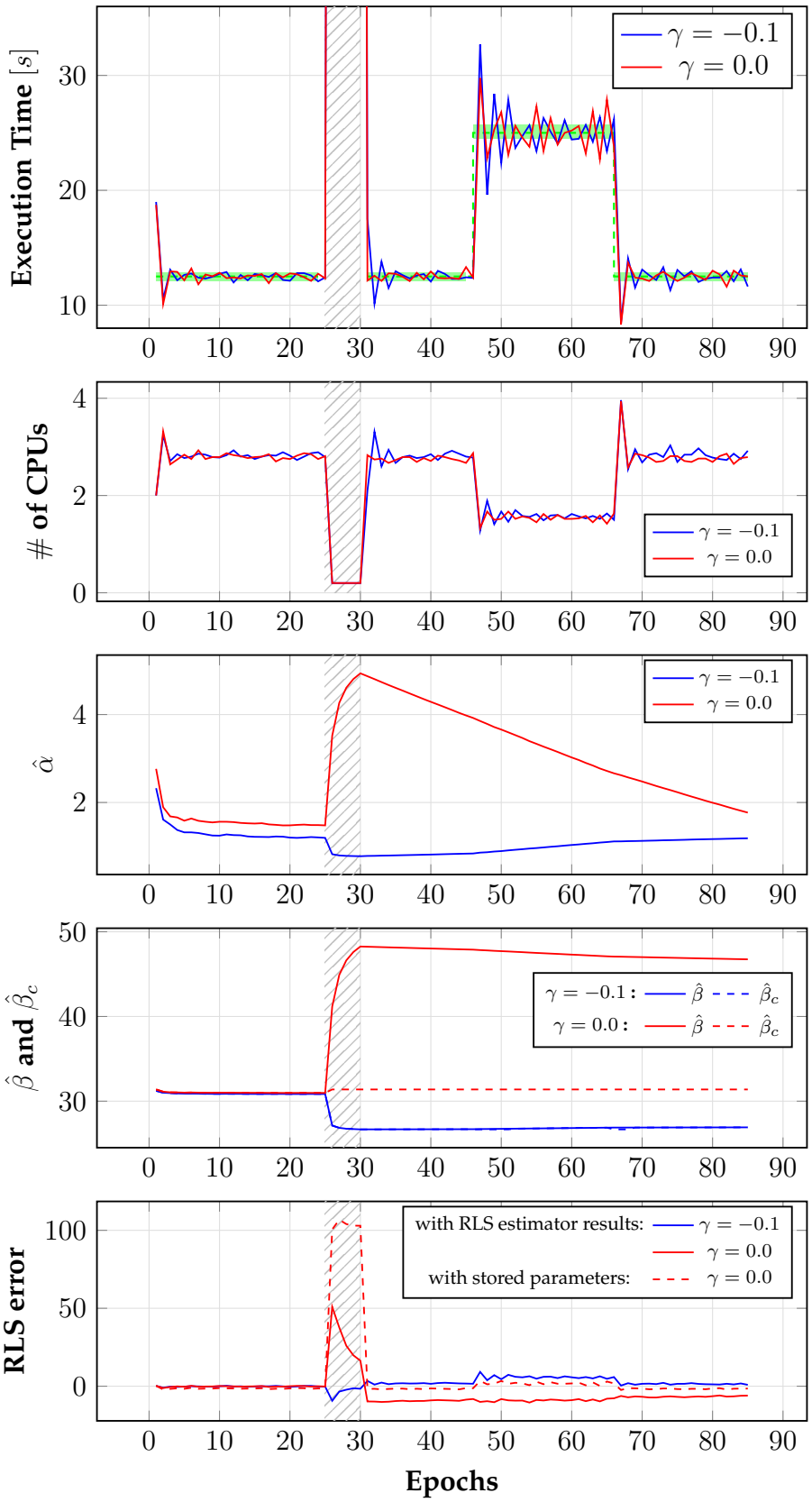


Figure B.5 – Performance of Control Strategy 2 for a initial setpoint value of 12.5s

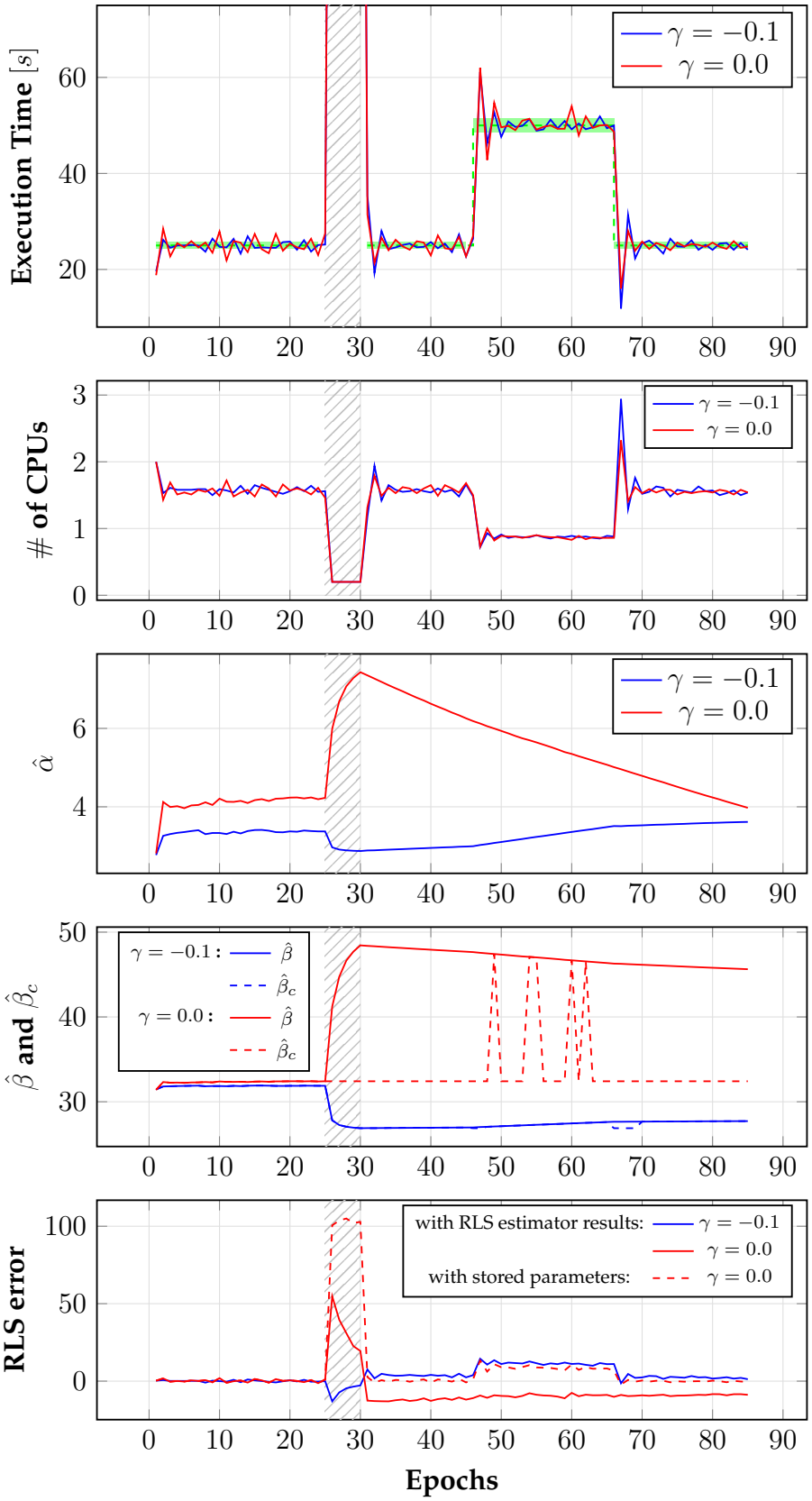


Figure B.6 – Performance of Control Strategy 2 for a initial setpoint value of 25.0s

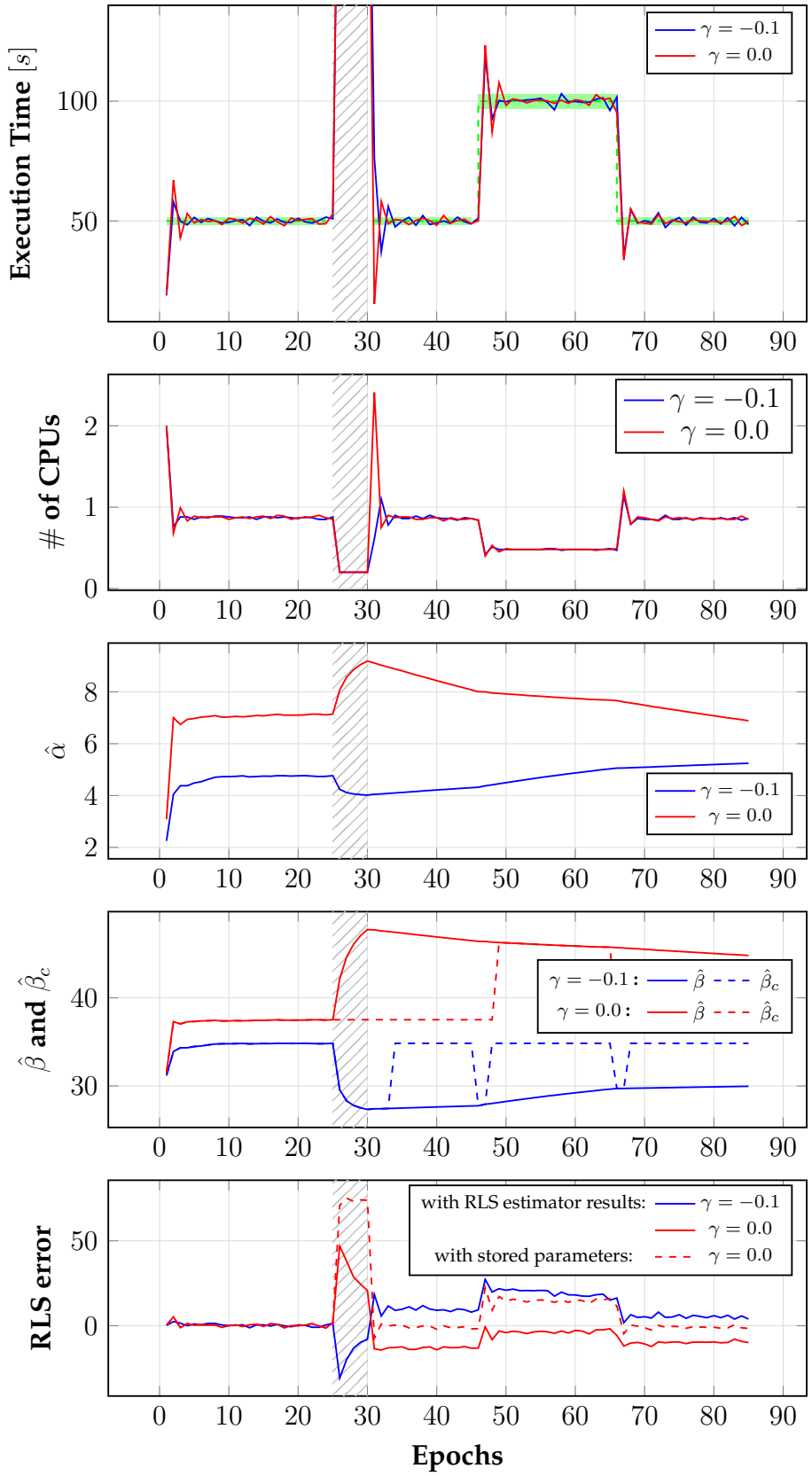


Figure B.7 – Performance of Control Strategy 2 for a initial setpoint value of 50.0s

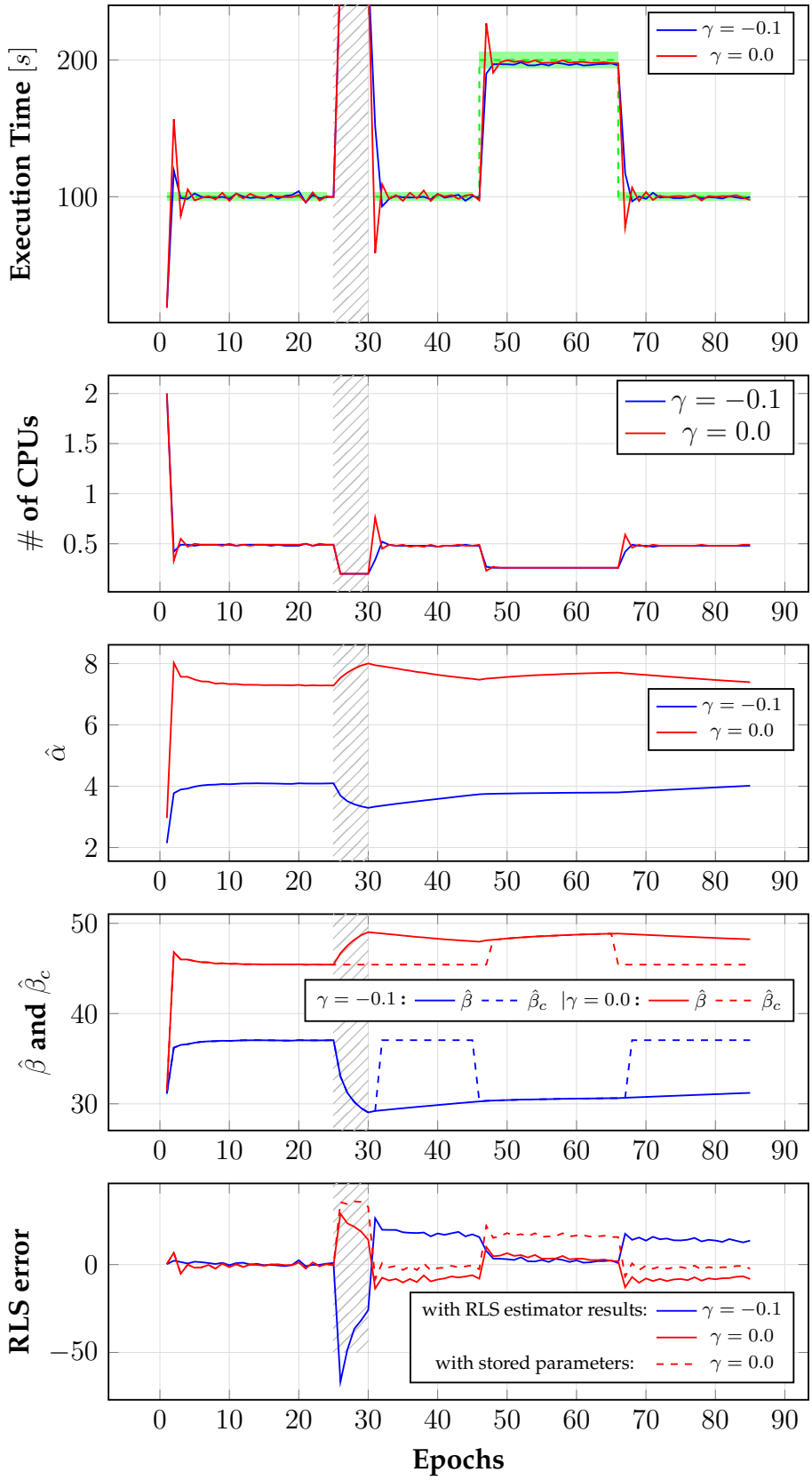


Figure B.8 – Performance of Control Strategy 2 for a initial setpoint value of 100.0s

B.2 Control strategies which used an exponential model for the input-output relationship

B.2.1 Control Strategy 3: System Dynamics given by the Taylor expansion of an exponential function

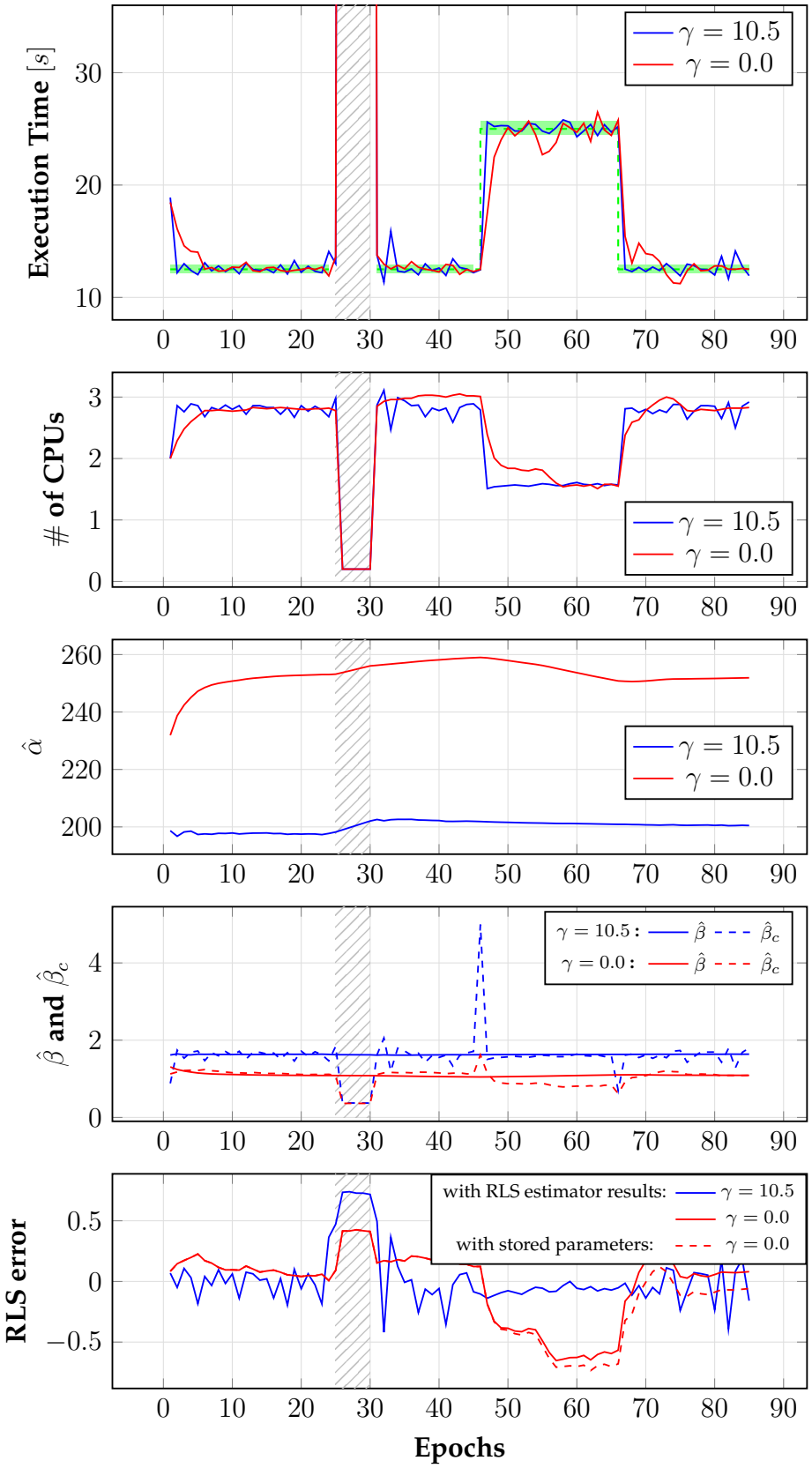


Figure B.9 – Performance of Control Strategy 3 for a initial setpoint value of 12.5s

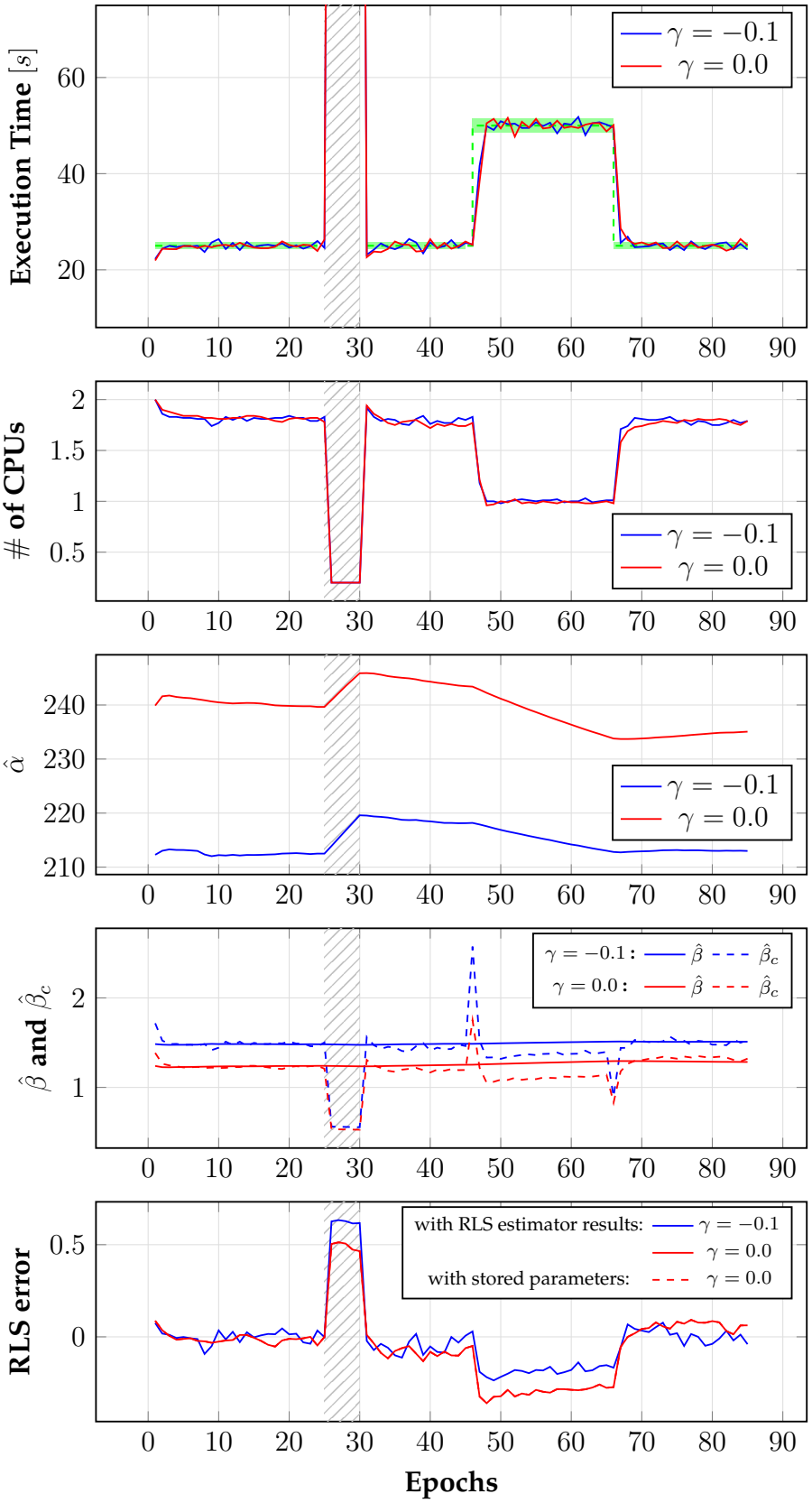


Figure B.10 – Performance of Control Strategy 3 for a initial setpoint value of 25.0s

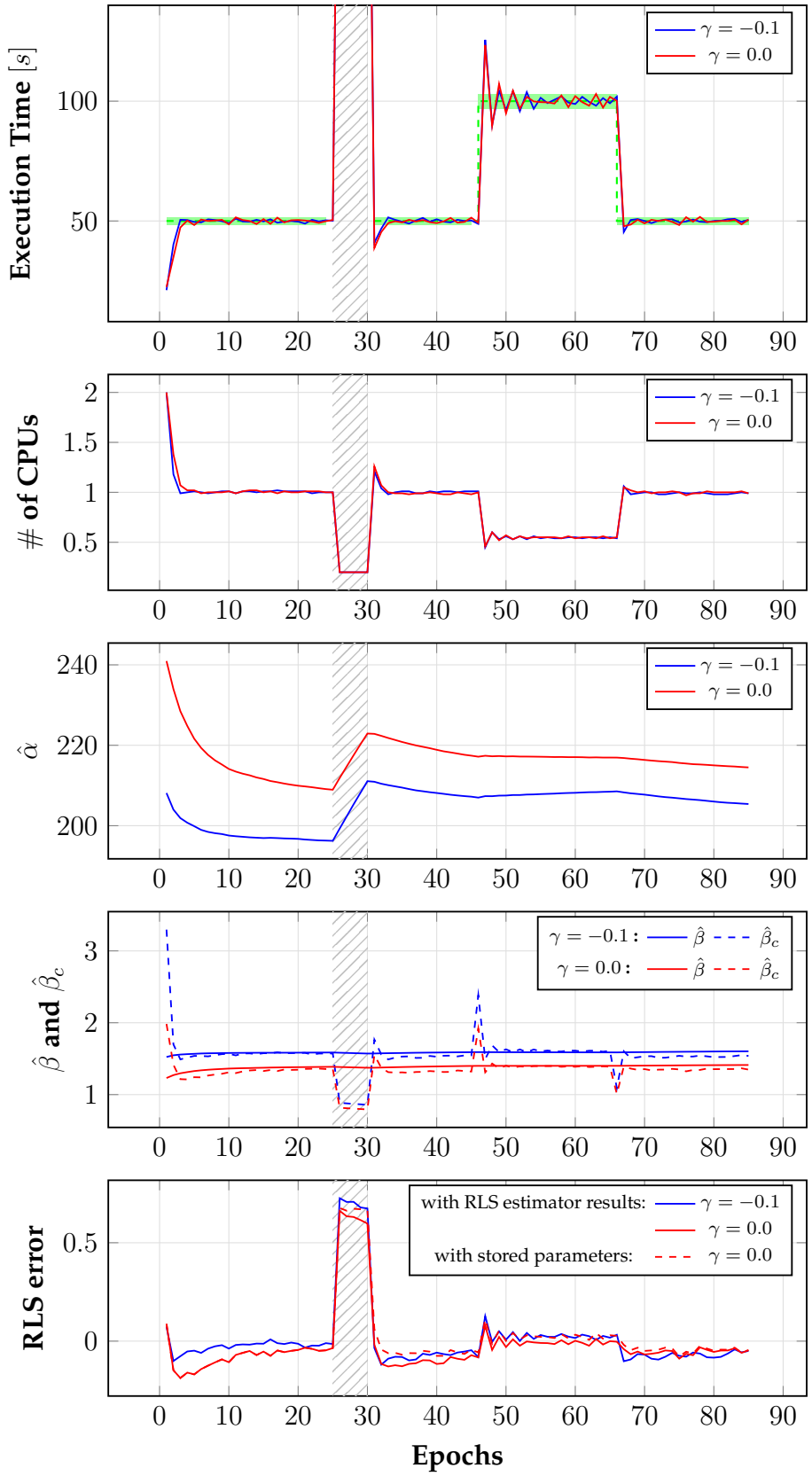


Figure B.11 – Performance of Control Strategy 3 for a initial setpoint value of 50.0s

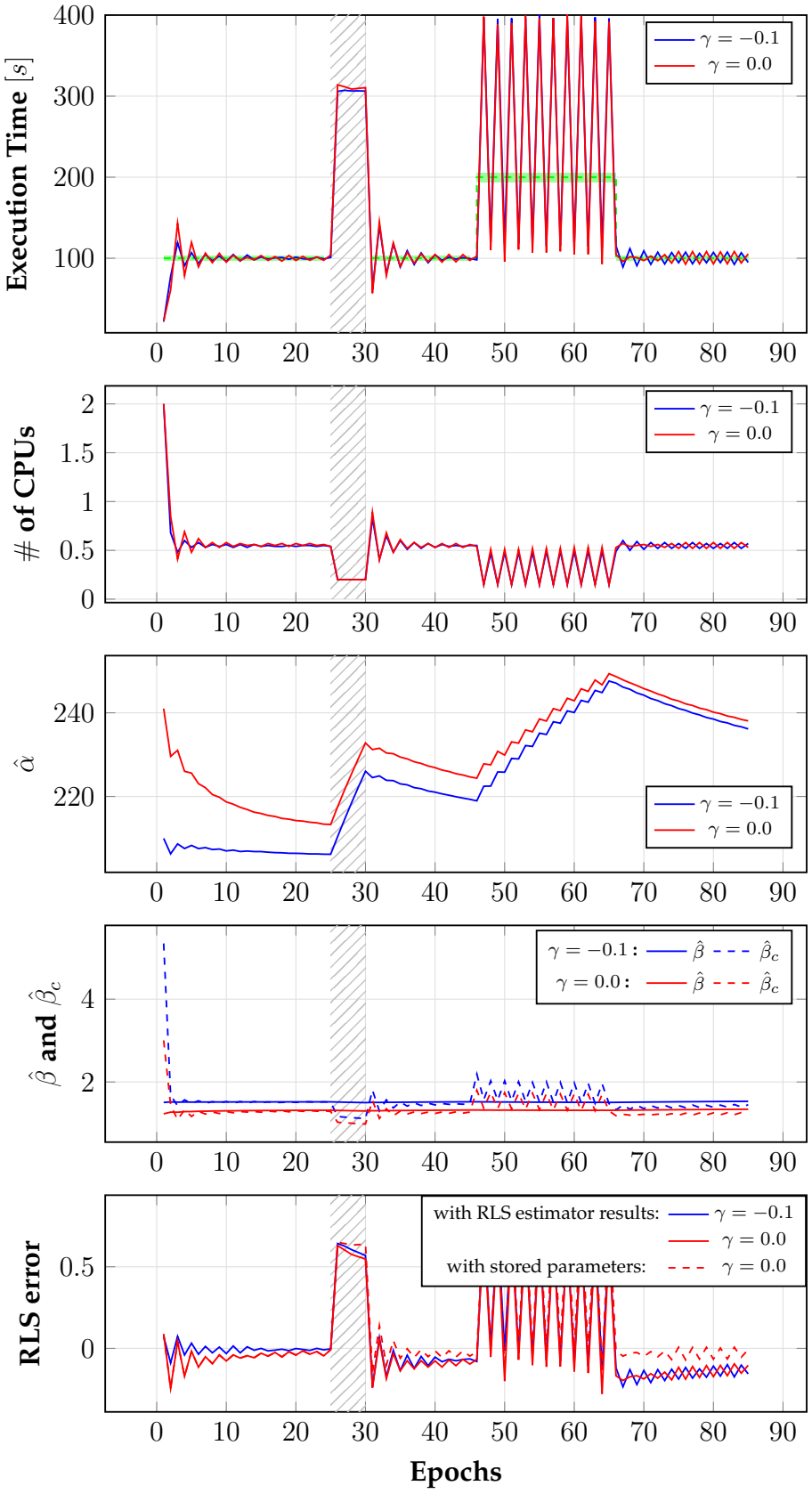


Figure B.12 – Performance of Control Strategy 3 for a initial setpoint value of 100.0s

B.2.2 Control Strategy 4: System Dynamics given by a manipulation of an exponential function

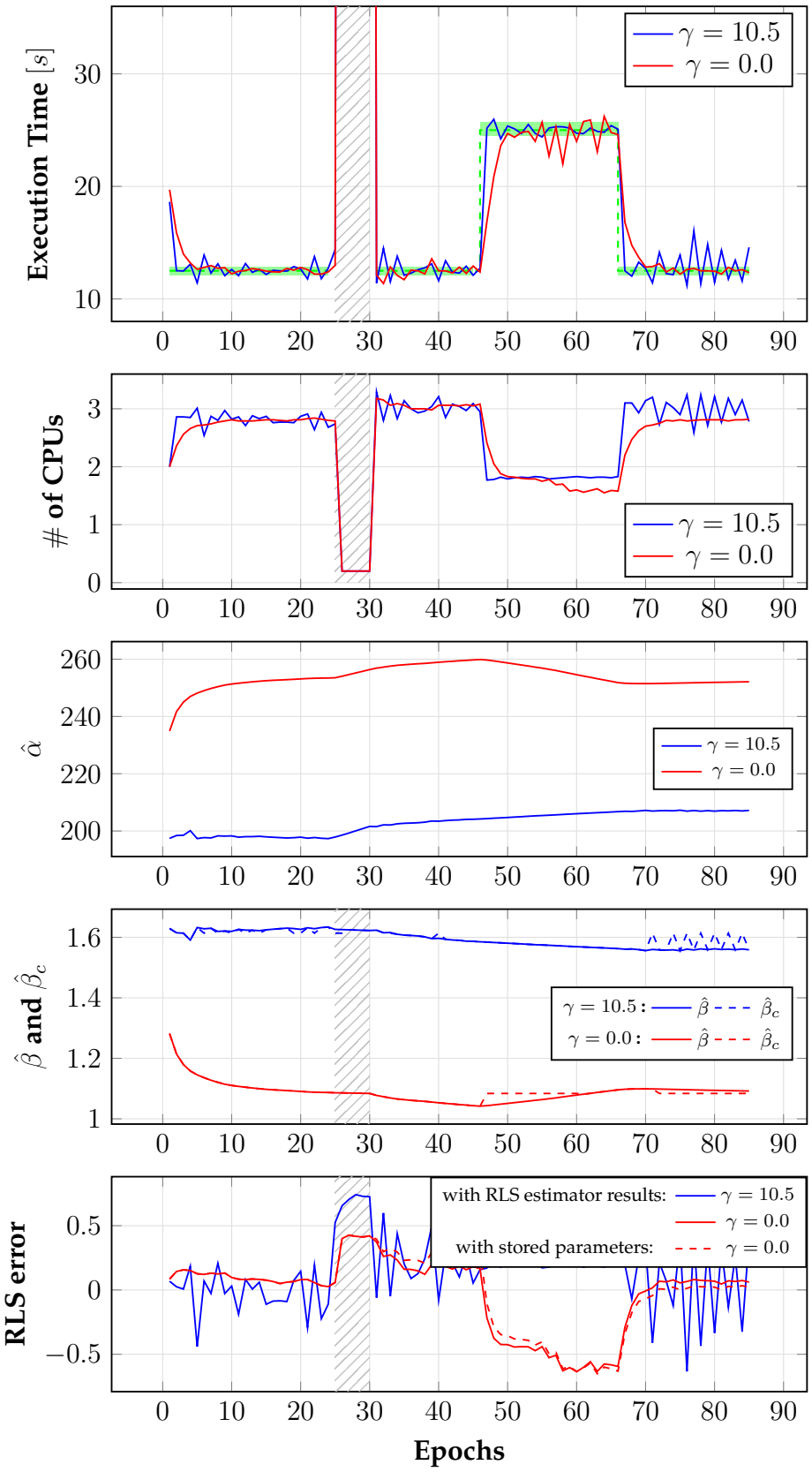


Figure B.13 – Performance of Control Strategy 4 for a initial setpoint value of 12.5s

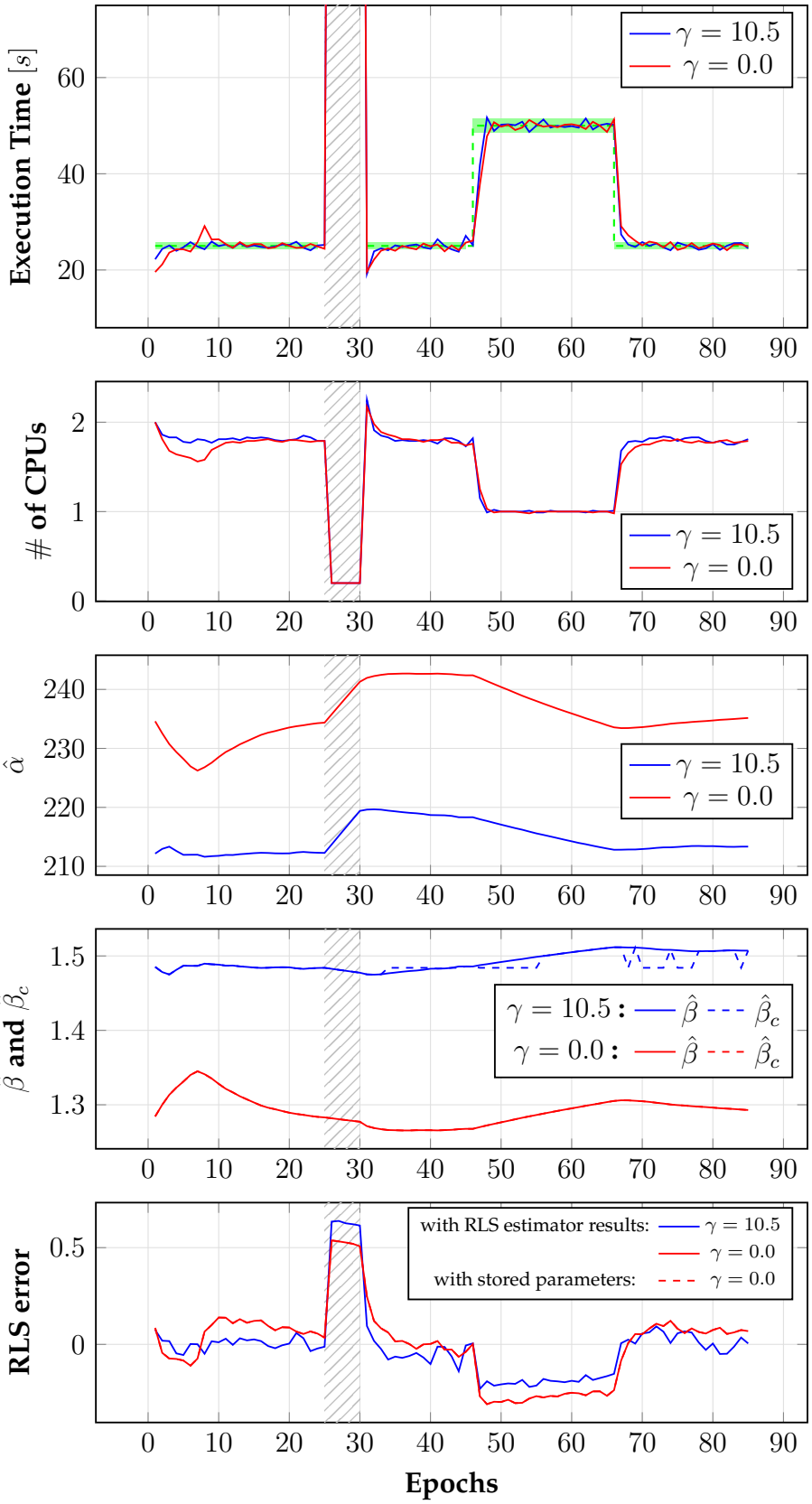


Figure B.14 – Performance of Control Strategy 4 for a initial setpoint value of 25.0s

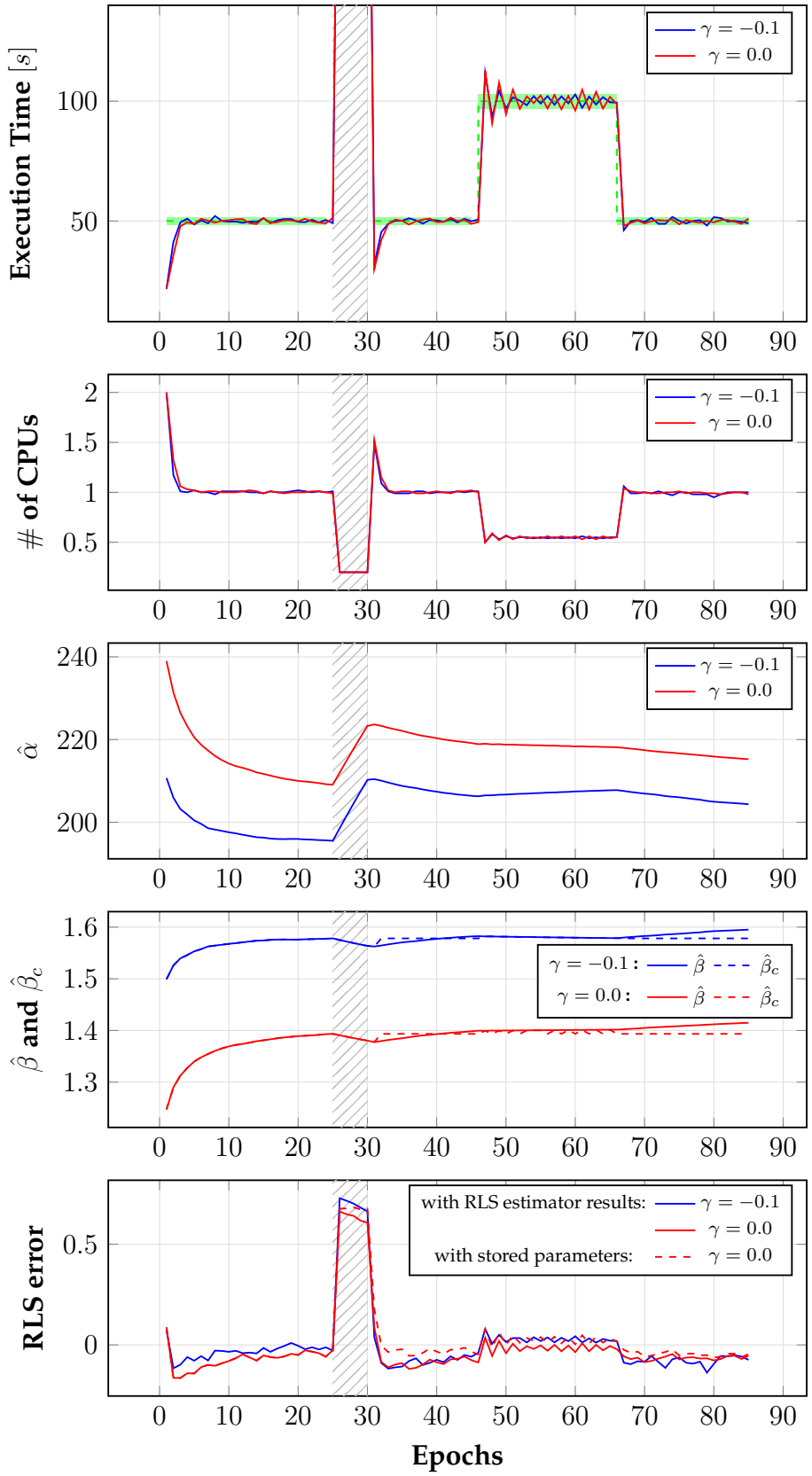


Figure B.15 – Performance of Control Strategy 4 for a initial setpoint value of 50.0s

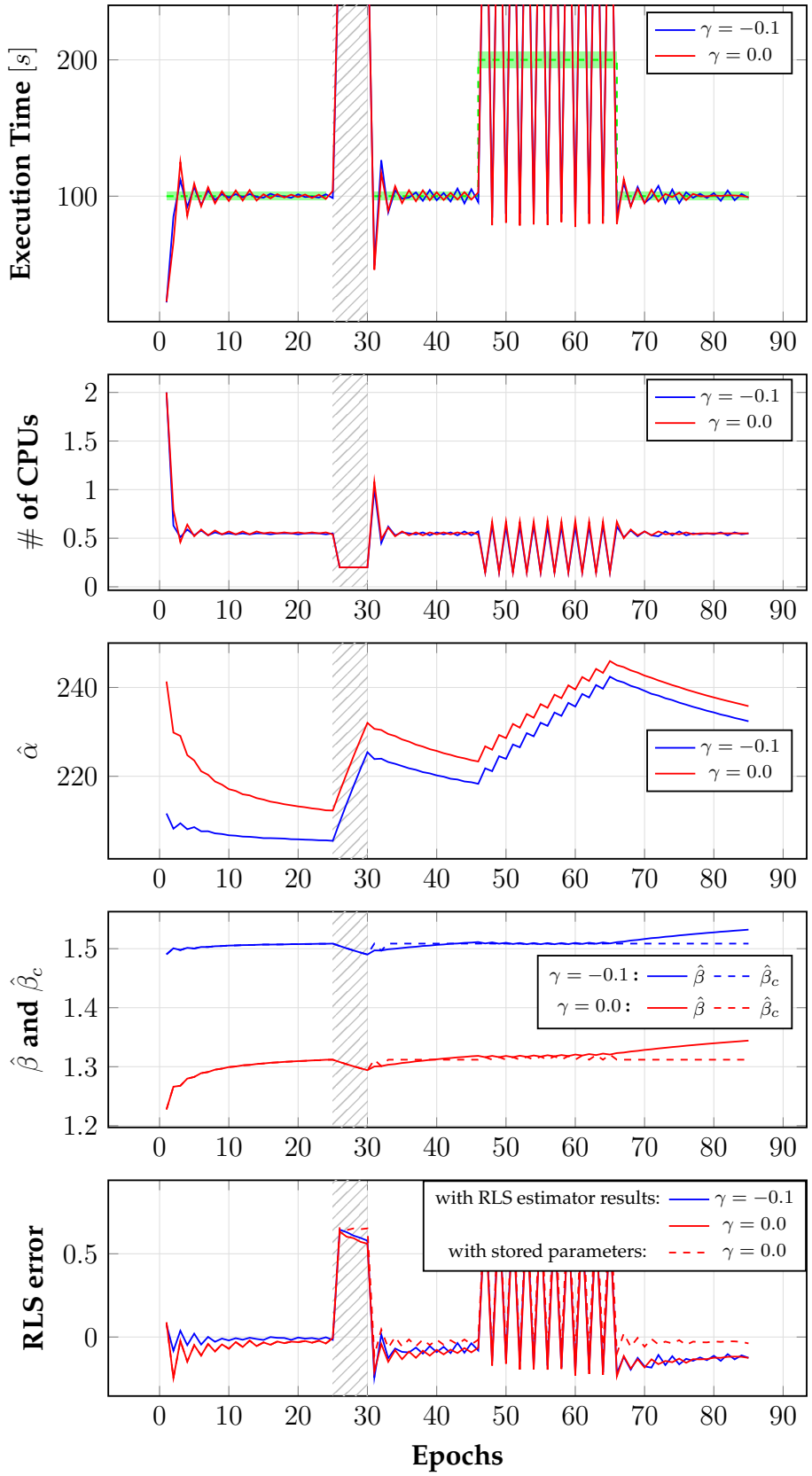


Figure B.16 – Performance of Control Strategy 4 for a initial setpoint value of 100.0s

Appendix C

Detailed results from tests of OACS in a single container execution environment

This chapter presents the data from the validation process of the Optimal Adaptive Control Strategy for the case in which only a single container is deployed.

A full description of test scenarios covering all test cases presented below can be found in section 6.2.

First, Figure C.1 shows the value of initial guess of β given as a result of Algorithm 5 for all containers considered in this work.

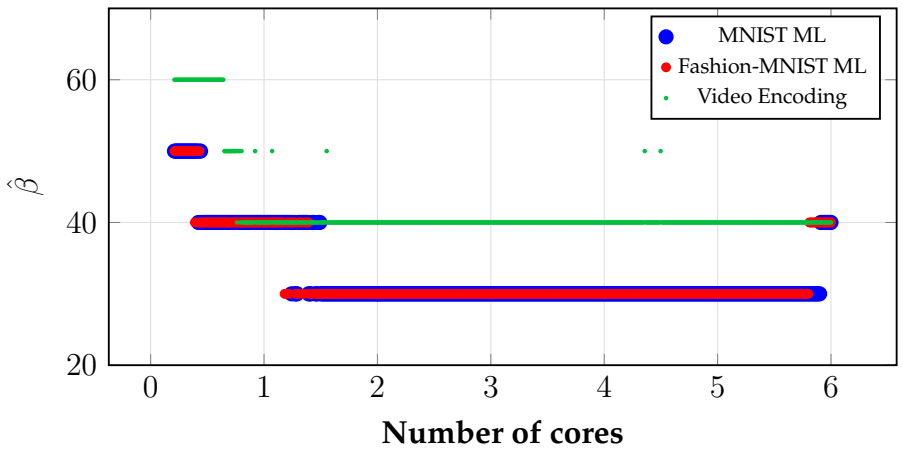


Figure C.1 – Guess for the initial value of $\hat{\beta}$

C.1 Test cases using MNIST ML application

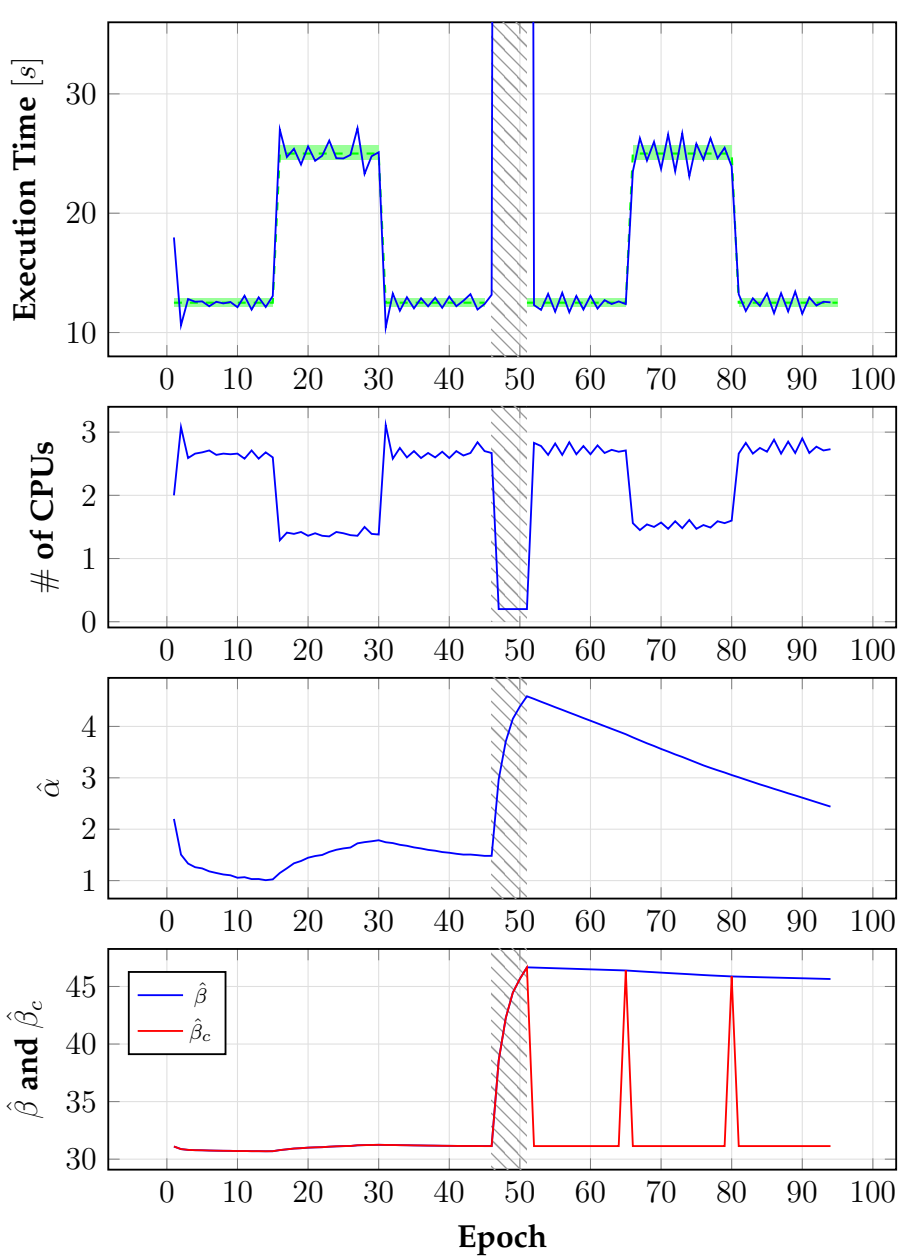


Figure C.2 – Performance of the OACS.
Single container environment with MNIST ML application.
Initial set-point value: 12.5s

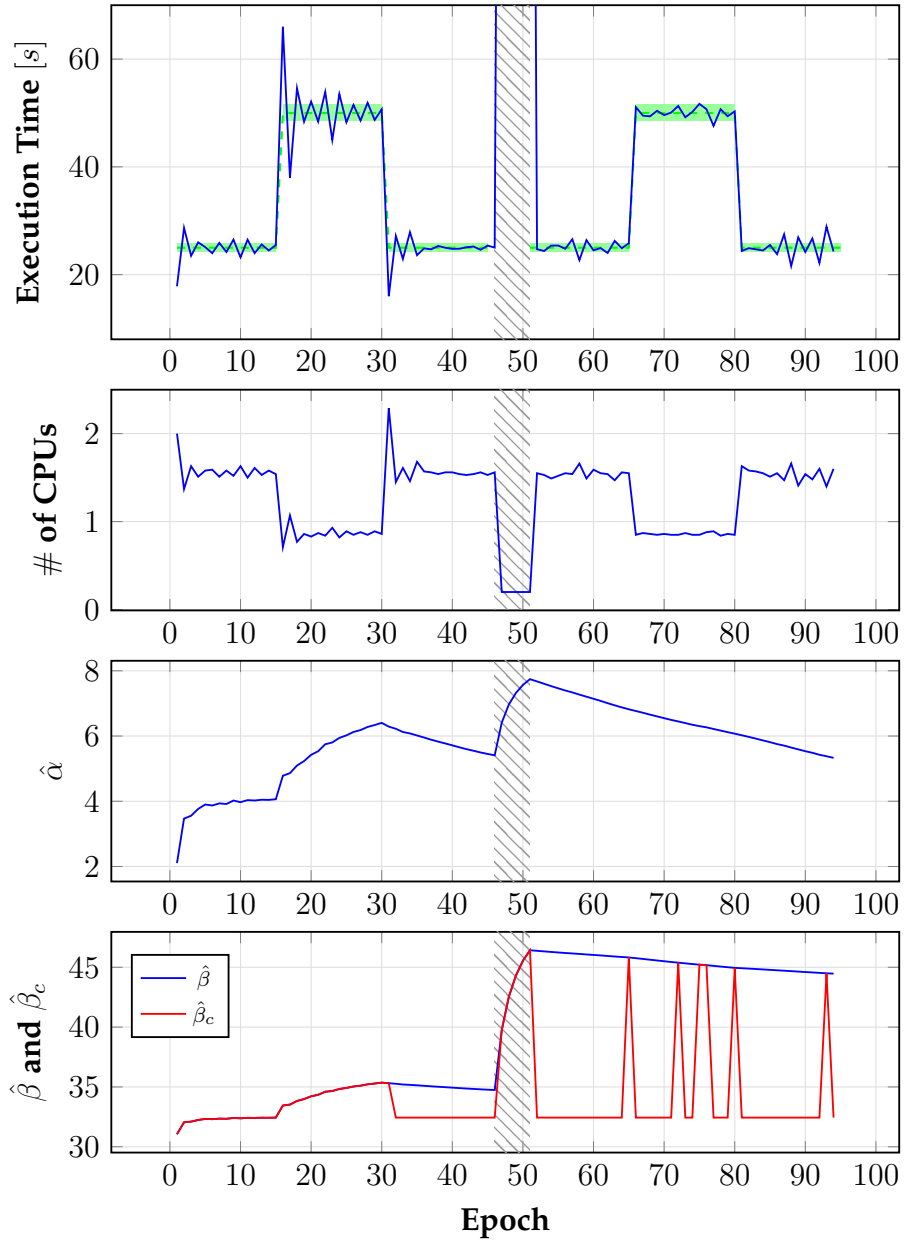


Figure C.3 – Performance of the OACS.
Single container environment with MNIST ML application.
Initial set-point value: 25.0s

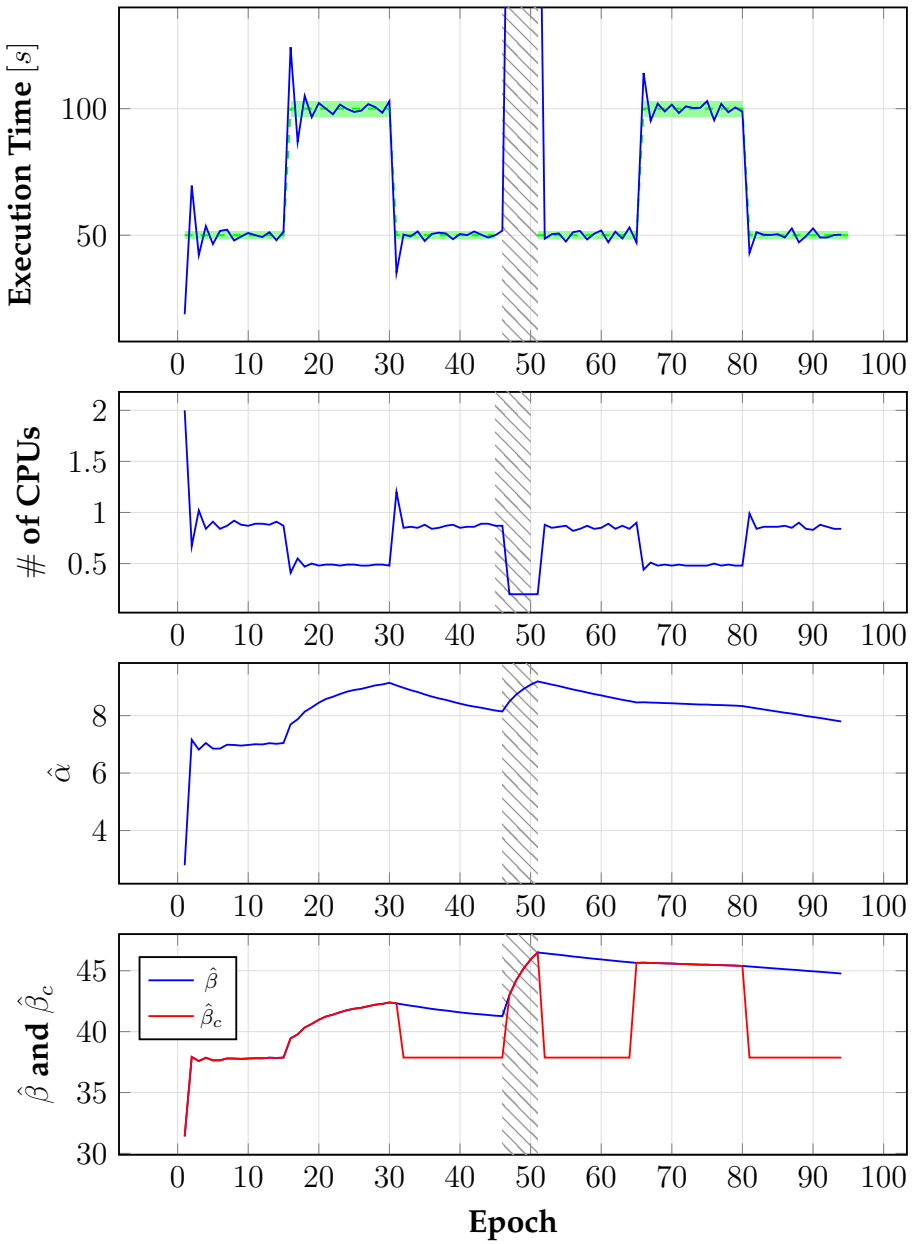


Figure C.4 – Performance of the OACS.
Single container environment with MNIST ML application.
Initial set-point value: 50.0s

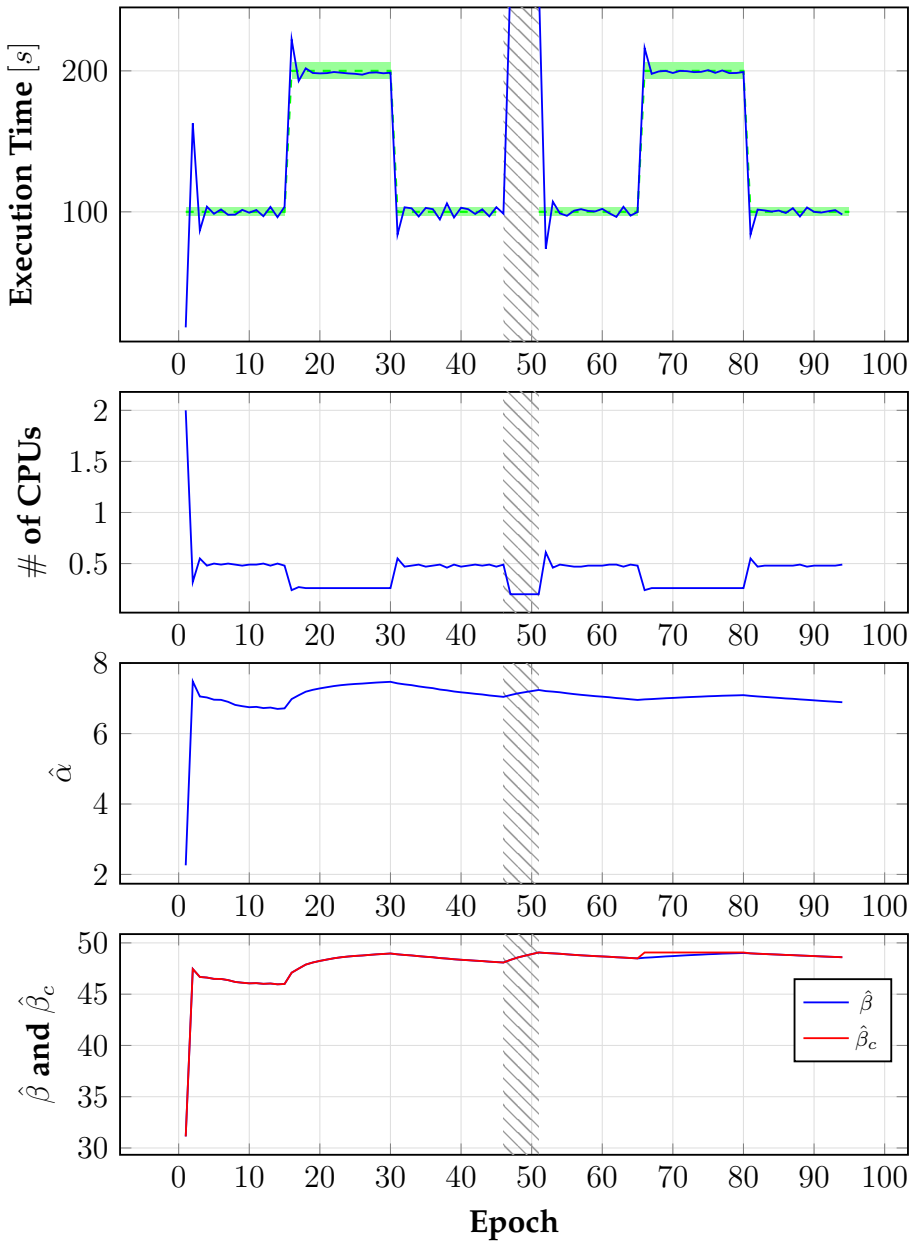


Figure C.5 – Performance of the OACS.
Single container environment with MNIST ML application.
Initial set-point value: 100.0s

C.2 Test cases using Fashion-MNIST ML application

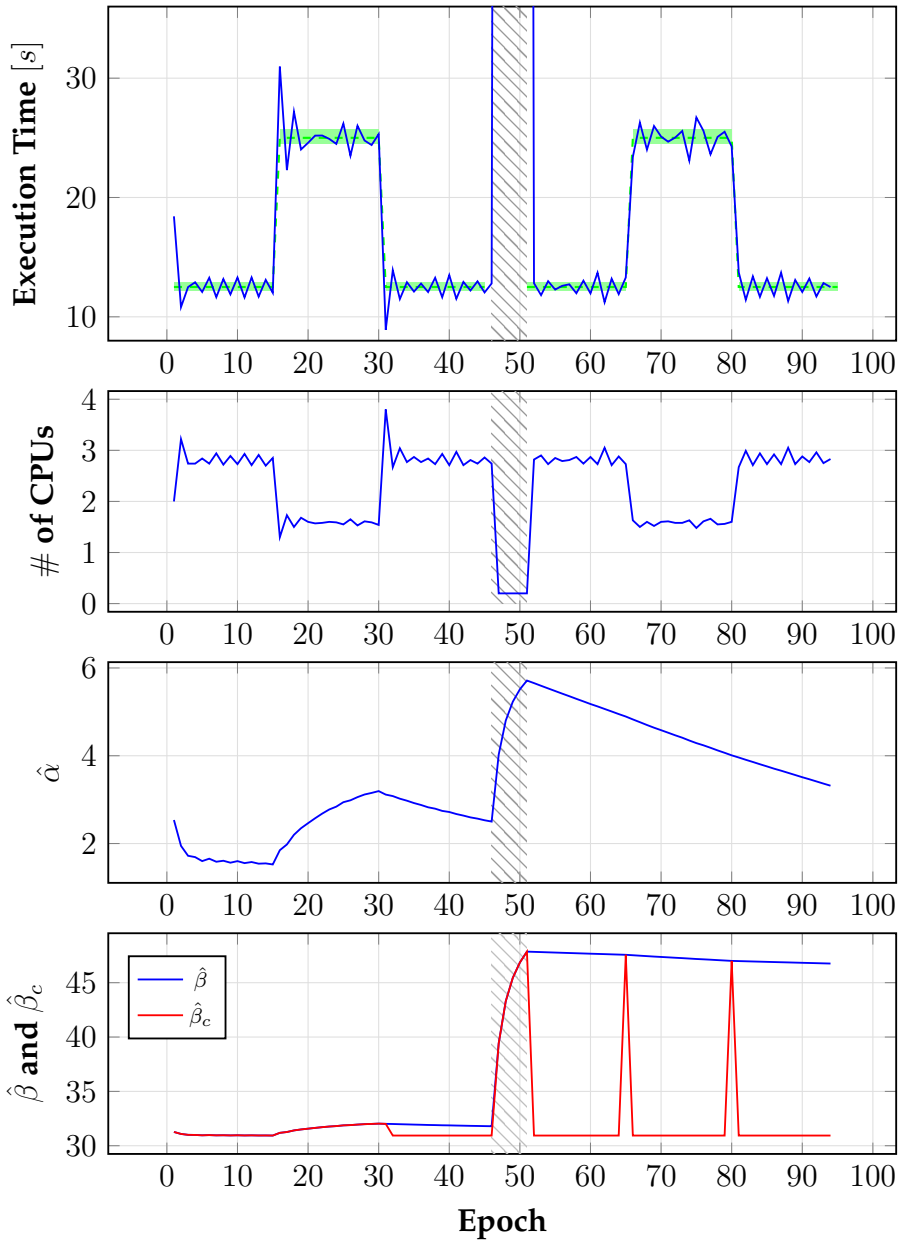


Figure C.6 – Performance of the OACS.
Single container environment with Fashion-MNIST ML application.
Initial set-point value: 12.5s

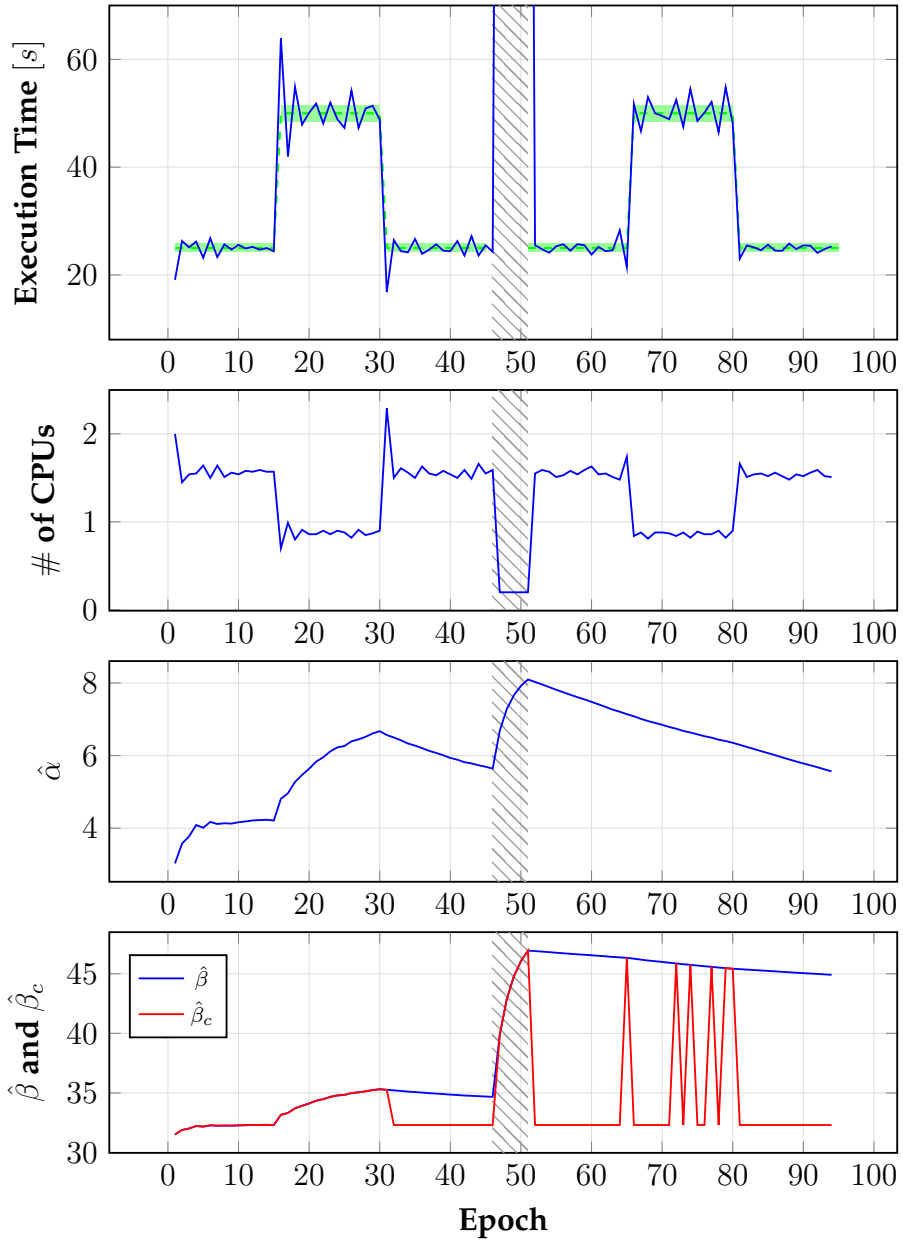


Figure C.7 – Performance of the OACS.
Single container environment with Fashion-MNIST ML application.
Initial set-point value: 25.0s

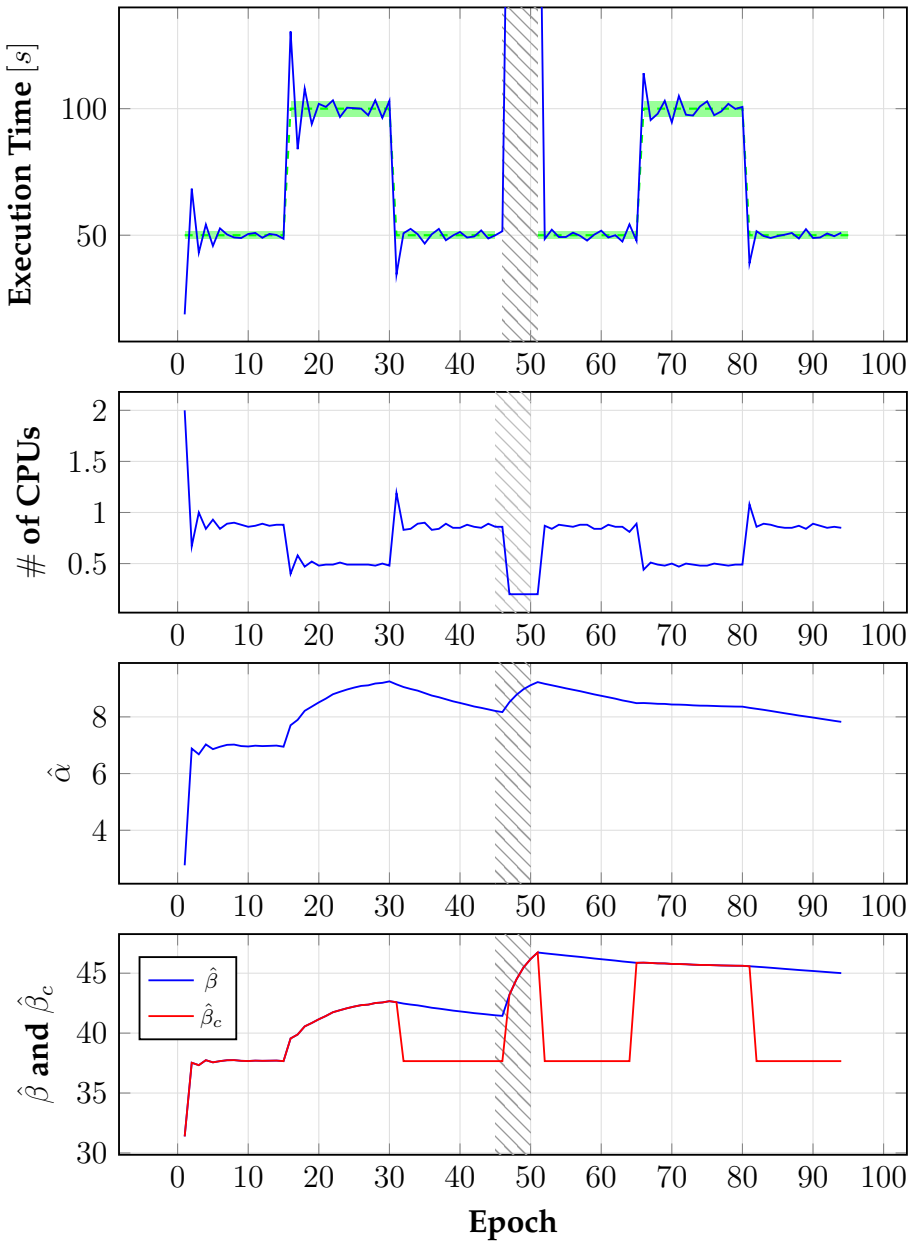


Figure C.8 – Performance of the OACS.
Single container environment with Fashion-MNIST ML application.
Initial set-point value: 50.0s

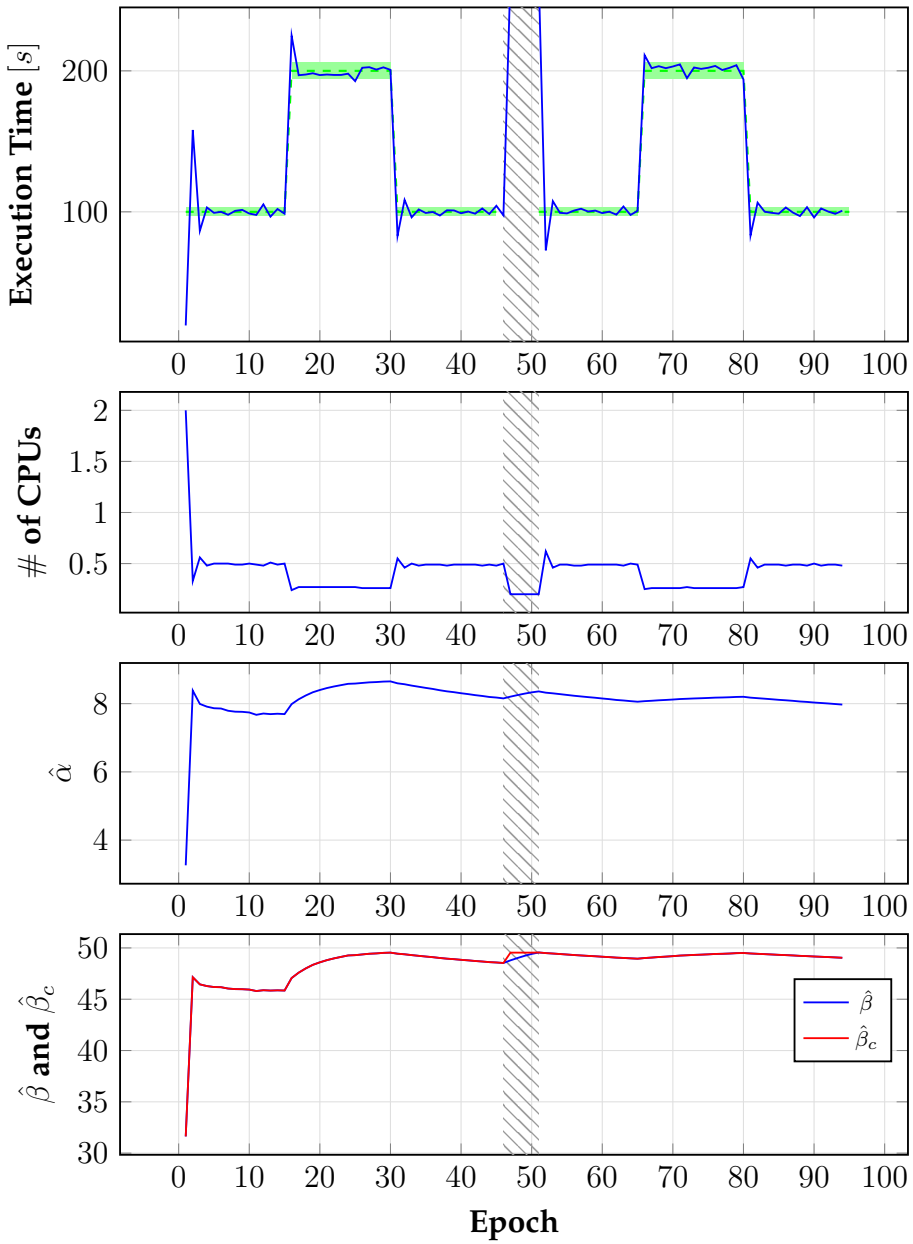


Figure C.9 – Performance of the OACS.
Single container environment with Fashion-MNIST ML application.
Initial set-point value: 100.0s

C.3 Test cases using Video Encoding application

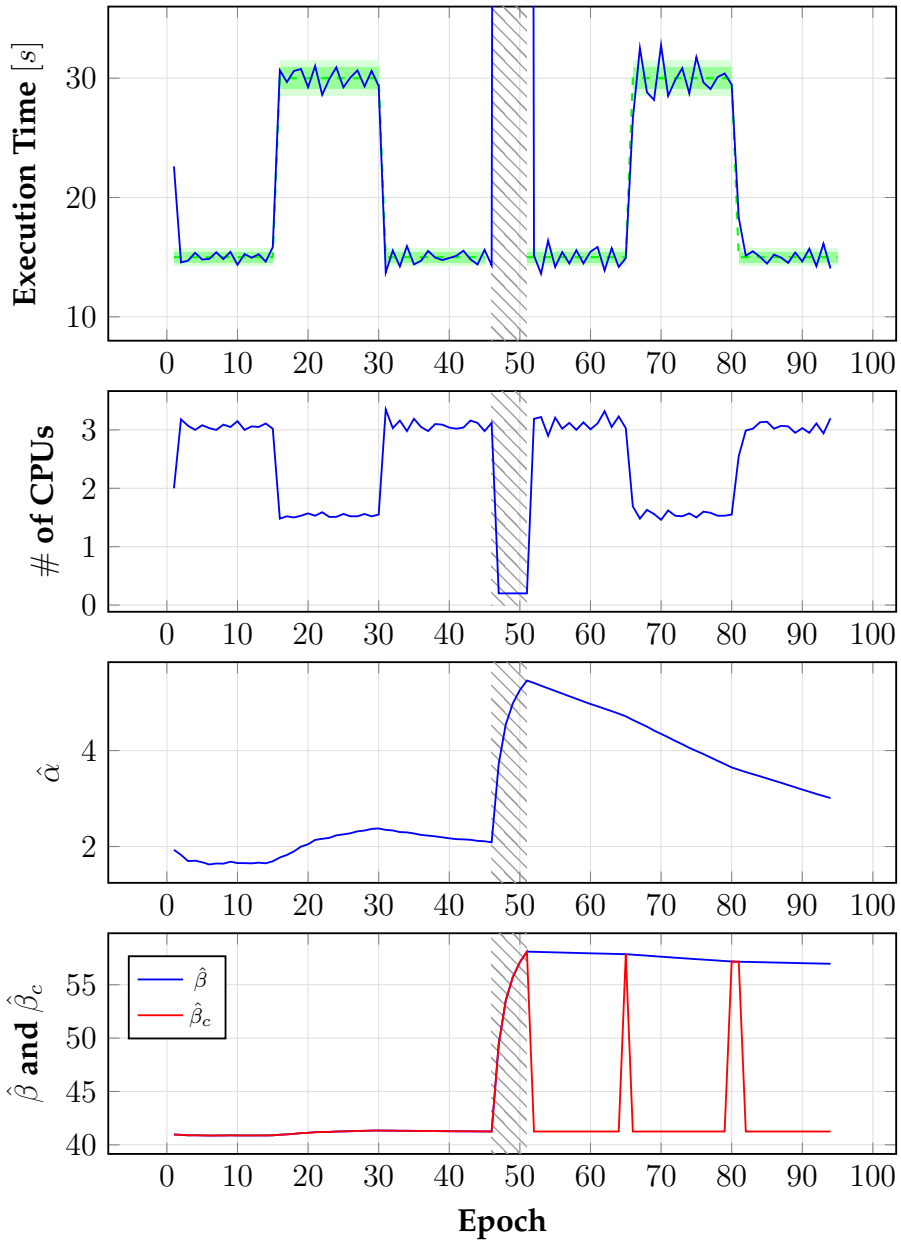


Figure C.10 – Performance of the OACS.
Single container environment with Video Encoding application.
Initial set-point value: 12.5s

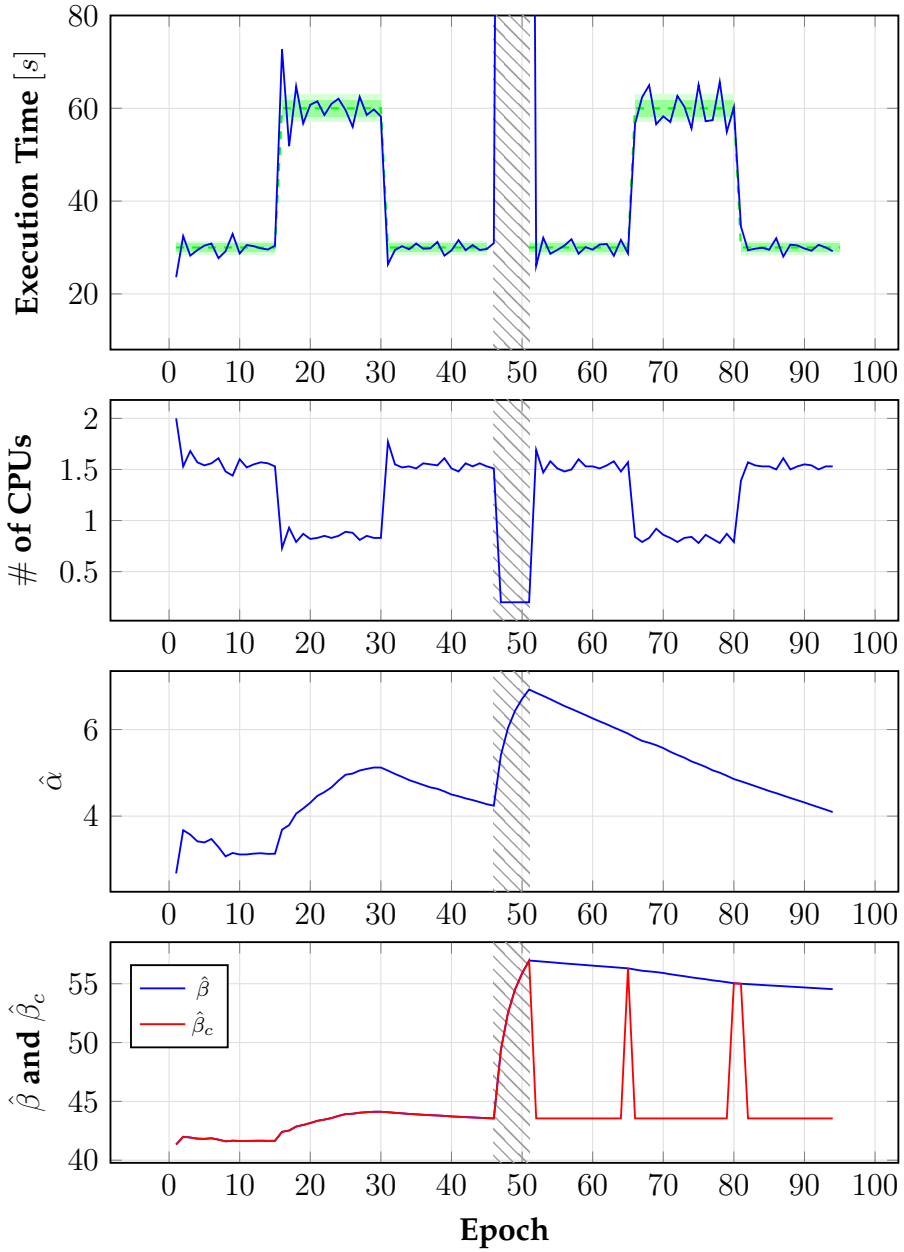


Figure C.11 – Performance of the OACS.
Single container environment with Video Encoding application.
Initial set-point value: 25.0s

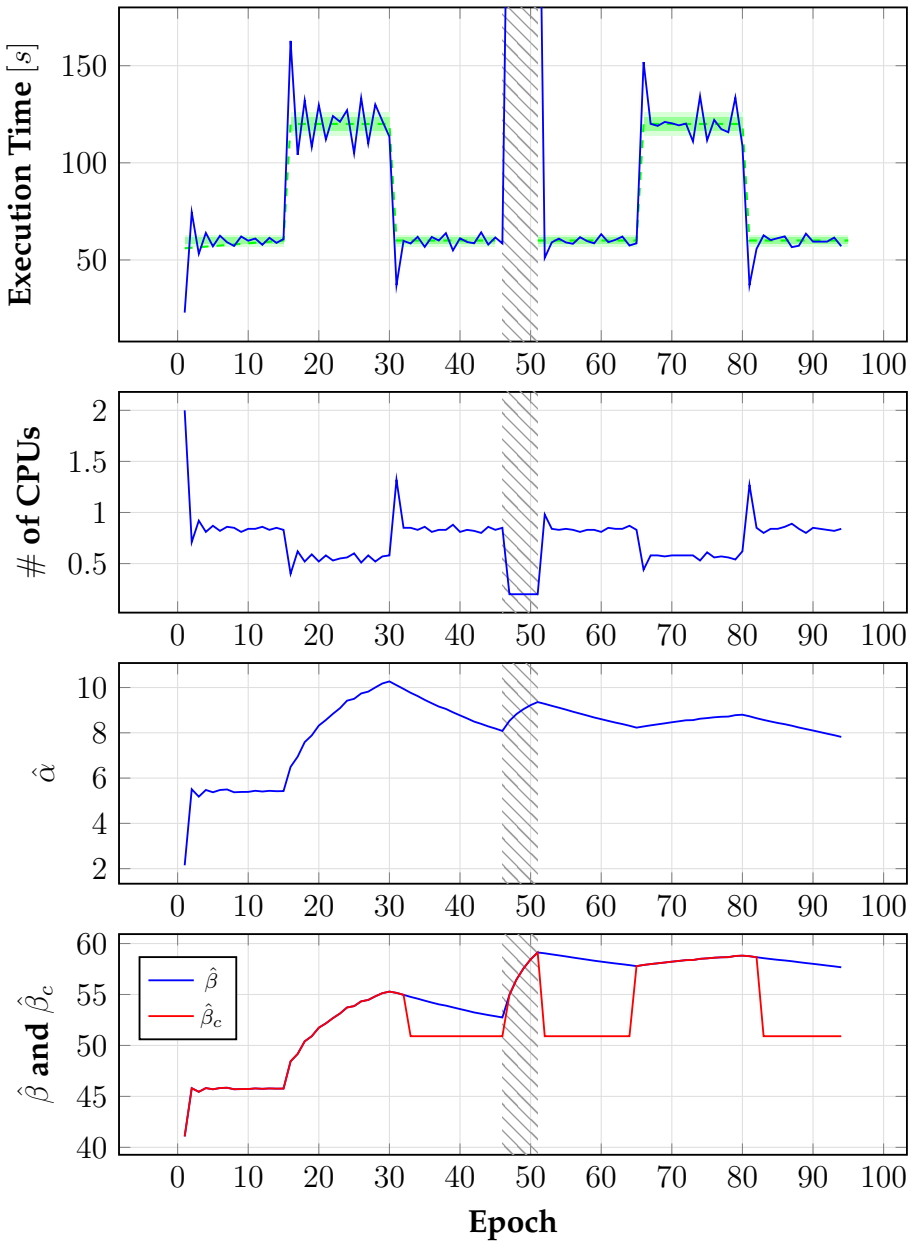


Figure C.12 – Performance of the OACS.
Single container environment with Video Encoding application.
Initial set-point value: 50.0s

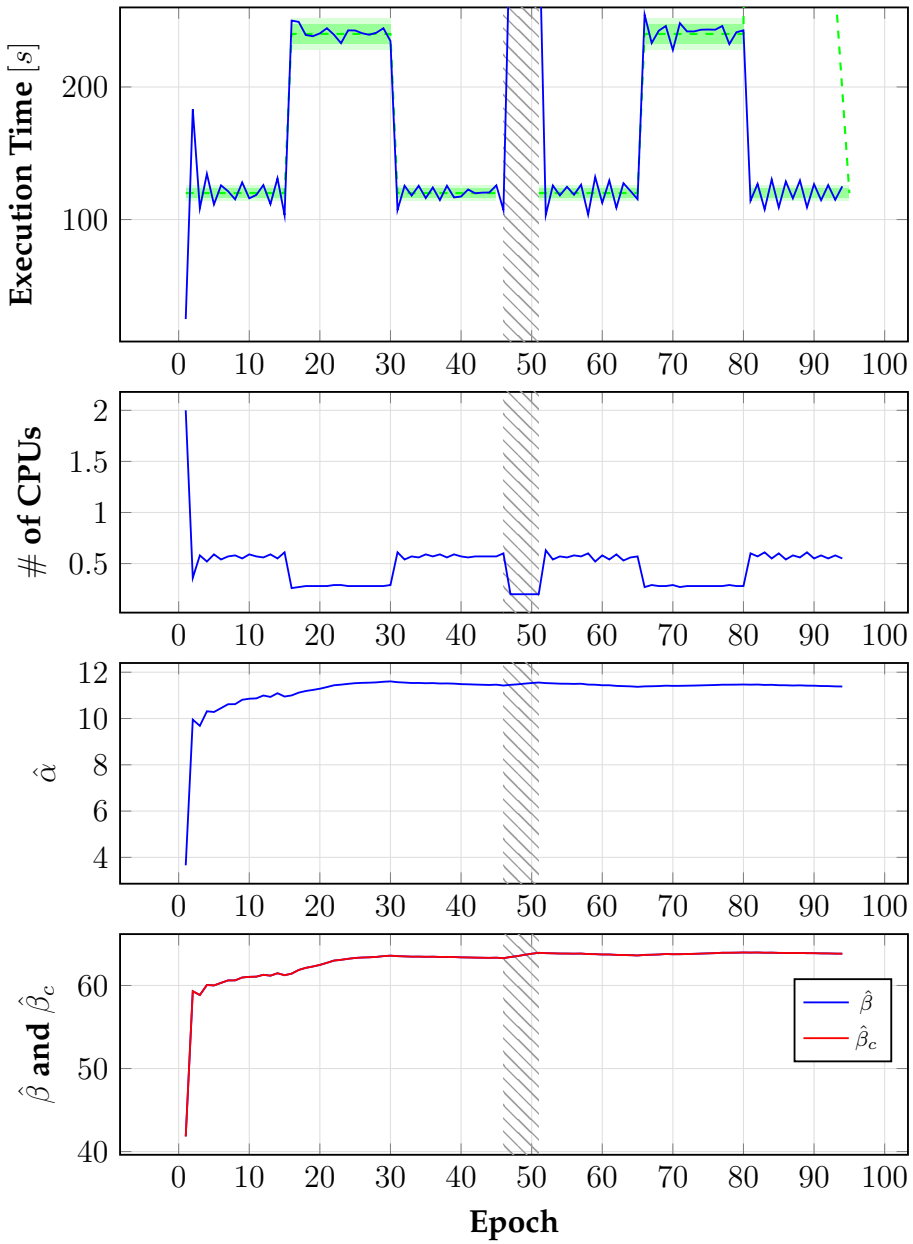


Figure C.13 – Performance of the OACS.
Single container environment with Video Encoding application.
Initial set-point value: 100.0s

Bibliography

[1] S. Sastry and M. Bodson, *Adaptive Control: Stability, Convergence, and Robustness*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[2] J.-J. Slotine and W. Li, *Applied Nonlinear Control*. Pearson, 1991.

[3] P. Anglard, *Automatique, modélisation et controle des systèmes*. Cours de l'Ecole Centrale Paris, 1996.

[4] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *2008 International Conference on Autonomic Computing*, pp. 3–12, June 2008.

[5] A. Paschke and E. Schnappinger-Gerull, "A categorization scheme for sla metrics.," in *Service Oriented Electronic Commerce* (M. Schoop, C. Huemer, M. Rebstock, and M. Bichler, eds.), vol. P-80 of *LNI*, pp. 25–40, GI, 2006.

[6] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, (New York, NY, USA), Association for Computing Machinery, 2012.

[7] S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, "Control-theoretical software adaptation: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 44, pp. 784–810, Aug 2018.

[8] A. Filieri, M. Maggio, K. Angelopoulos, N. D'ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel, "Control strategies for self-adaptive software systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 11, Feb. 2017.

[9] C. Pautasso, E. Wilde, and R. Alarcón, eds., *REST: Advanced Research Topics and Practical Applications*. Springer, 2014.

[10] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967,*

Spring Joint Computer Conference, AFIPS '67 (Spring), (New York, NY, USA), pp. 483–485, ACM, 1967.

[11] J. L. Gustafson, "Reevaluating amdahl's law," *Commun. ACM*, vol. 31, pp. 532–533, May 1988.

[12] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, pp. 1–26. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[13] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, *Engineering Self-Adaptive Systems through Feedback Loops*, pp. 48–70. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[14] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Automated control of multiple software goals using multiple actuators," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, ACM Press, 2017.

[15] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, ACM Press, 2014.

[16] B. Sohlberg and E. Jacobsen, "Grey box modelling – branches and experiences," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 11415 – 11420, 2008. 17th IFAC World Congress.

[17] S. Shevtsov and D. Weyns, "Keep it SIMPLEX: satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, ACM Press, 2016.

[18] "AWS auto scaling." `https://aws.amazon.com/autoscaling/`. Accessed: 2020-06-30.

[19] "HP-UX workload manager overview." `https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c01919398`. Accessed: 2020-06-30.

[20] "Workload manager." `https://www.ibm.com/support/knowledgecenter/ssw_aix_72/devicemanagement/wlm.html`. Accessed: 2020-06-30.

[21] A. Filieri, H. Hoffmann, and M. Maggio, "Automated multi-objective control for self-adaptive software design," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, ACM Press, 2015.

[22] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource containers on shared servers," in *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005.*, pp. 163–176, 2005.

[23] K. Angelopoulos, A. V. Papadopoulos, V. E. S. Souza, and J. Mylopoulos, "Model predictive control for software systems with cobra," in *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 35–46, May 2016.

[24] D. Arcelli, V. Cortellessa, A. Filieri, and A. Leva, "Control theory for model-based performance-driven software adaptation," in *2015 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*, pp. 11–20, 2015.

[25] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel, "Software engineering meets control theory," in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 71–82, 2015.

[26] "The definitive guide to docker containers." `https://www.innovate-systems.de/downloads/docker/The_definitive_Guide_to_Docker.pdf`. Accessed: 2019-06-06.

[27] "Runtime options with memory, cpus, and gpus." `https://docs.docker.com/config/containers/resource_constraints/`. Accessed: 2019-06-06.

[28] P. A. Ioannou, *Robust Adaptive Control*. Prentice Hall, sep 1995.

[29] H. K. Khalil, *Nonlinear Systems (3rd Edition)*. Pearson, 2001.

[30] A. Iggidr and M. Bensoubaya, "Stability of Discrete-time Systems : New Criteria and Applications to Control Problems," Research Report RR-3003, INRIA, 1996.

[31] S. Haykin, *Adaptive Filter Theory (3rd Edition)*. Prentice Hall, 1995.

[32] V. Bobal, *Digital Self-tuning Controllers: Algorithms, Implementation and Applications (Advanced Textbooks in Control and Signal Processing)*. Springer London, 2005.

[33] A. Downey, *The little book of semaphores*. United States: SoHoBooks, 2008.

[34] N. Narkhede, G. Shapira, and T. Palino, *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. O'Reilly Media, 2017.

[35] L. Wang, *Model Predictive Control System Design and Implementation Using MATLAB*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[37] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[38] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[39] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Self-adaptive video encoder: Comparison of multiple adaptation strategies made simple," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 123–128, May 2017.

[40] "Imagemagick - command-line tools: Convert." `https://imagemagick.org/script/command-line-options.php#quality`. Accessed: 2020-08-29.

[41] "Svt-av1: open-source av1 encoder and decoder." `https://netflixtechblog.com/svt-av1-an-open-source-av1-encoder-and-decoder-ad295d9b5ca2f`. Accessed: 2020-05-11.