Adaptive control of HPC clusters for server overload avoidance

Rosa Pagano

POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Adaptive control of HPC clusters for server overload avoidance

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria Automazione e Controllo

Author: **Rosa Pagano**

Student ID: 10629478
Advisor: Prof. Alberto Leva
Co-advisors: Bogdan Robu, Eric Rutten
Academic Year: 2022-23

Abstract

This thesis focuses on the maximal usage of an High-Performance Computing (HPC) Cluster with particular attention to the overloading of the storage, namely the file server. HPCs have become important in the scientific domain because of their computational power which allows calculation that otherwise would be impossible with a single computer. On the other hand, this powered structure has a considerable cost, and unfortunately also a short life cycle. As a consequence, it becomes important to use as best as possible all the resources that this new structure gives. In this context, Autonomic Computing and the collaboration between control theory and computer science were born.

This Mater Thesis continues a broader project where a PI controller was designed for an HPC Cluster. The aim of the controller is the minimization of idle resources inside the cluster. However, the controller is built on a model in a nominal configuration which leads to heterogeneous results for different working conditions. Moreover, conditions far from the nominal one bring the file server close to the overloading which is a challenging condition for the system.
The idea of this project is to design an identification algorithm in order to end up with an Adaptive PI that changes its own parameters to achieve specification with various working conditions. We design three different algorithms that are going to be tested in a simulation environment (SIMULINK) and then only one will be carried to the real set-up. In the end, there is a comparison between the PI and the Adaptive PI. Here it is shown how the Adaptive PI has a more homogeneous result than the case of the simple PI and how the overloading of the file server is avoided. However, the drawback is the settling time which becomes larger than in the case of classic PI.

Keywords: HPC, File server, Adaptive, Algorithm

Abstract in lingua italiana

Questa tesi si focalizza nell'usare al meglio le risorse di un High-Performance Computing (HPC) con un accento sul sovraccarico della memorial esterna.
Gli HPC sono diventati fondamentali nell'ambito scientifico per la loro potenza computazionale che sorpassa ogni limite prescritto dal singolo computer. Data la loro importanza e potenza, queste strutture vengono ad un costo non ignorabile e, sfortunatamente, anche ad un ciclo di vita relativamente breve. Di conseguenza, diventa molto importante sfruttare al meglio tutte le risorse di un HPC durante il suo ciclo vitale. Proprio in questo contesto nasce l'Autonomic computing che da' inizio alla cooperazione tra esperti di informatica e controllo.

Questa Tesi Magistrale è solo una parte di un progetto più ampio. Infatti, nel lavoro precedente, si è costruito un PI in modo da minimizzare le risorse inutilizzate in un HPC Cluster. Però, il controllore è stato disegnato su un modello specifico per una condizione di lavoro, che verrà chiamata nominale. Quindi, il sistema finale presentava diverse eterogeneità rispetto le diverse condizioni di lavoro. In più, situazioni lontane da quella nominale portavano quasi al sovraccarico della memoria esterna, una condizione da evitare assolutamente.
L'idea di questo progetto è di affiancare al precedente controllo un algoritmo di identificazione, in modo tale da avere un controllore che cambi i propri parametri per soddisfare sempre le specifiche in qualsiasi quadro lavorativo. In questo lavoro vengono discussi e testati in ambiente simulativo (SIMULINK) tre algoritmi diversi. Poi vengono mostrati anche gli esperimenti sulla vera struttura, ma qui viene testato solo uno dei tre algoritmi. Alla fine, c'è un confronto tra il PI e il PI Adattivo. In questa parte è possible vedere come i risultati del nuovo controllore sono più omogenei rispetto ai precedenti e come il sovraccarico viene evitato. Però, il nuovo sistema tende ad essere più lento del precedente.

Parole chiave: HPC, memoria esterna, Adattivo, Algoritmi

Contents

Introduction

This Master's thesis focuses on the application of control techniques to resource allocation in High-Performance Computing (HPC) systems. The presented work is part of a long-term research carried out at INRIA in Grenoble (France) that has resulted in a control technique to minimize idle resources within an HPC cluster. Building upon this result, the primary objective of this work is to make the said controller adaptive so as to face diverse operational *scenarii* and various clusters without requiring a manual re-tuning.

In the contemporary landscape, High-Performance Computing and its computational power have become crucial in the scientific domain. The HPC technology plays a fundamental role in numerous fields, including - but not limited to - weather prediction [1], the accelerated development of vaccines, large-size dynamic simulation, big data management and more. Consequently, HPC systems are beneficial for both the scientific community and the society at large by facilitating the advancement of new technologies and knowledge.

Nevertheless, such cutting-edge infrastructures come at a considerable cost for both development and maintenance, and given the highly dynamic nature of the field, they have relatively short life cycles. Thus, maximizing resource utilization within these structures is a fundamental means to optimize their efficiency and cost-effectiveness.

The scientific collocation of the research just mentioned is within Autonomic Computing, a field that IBM pioneered starting approximately in the 70s of the last century in response to a growing challenge, namely the escalating complexity of computational infrastructures. In a series of blueprints, IBM foresaw a future where computing systems would become so complicated to be manageable by a decreasing number of professionals, hence making "self-governance" capabilities a necessity.

In addition, HPC represents a substantial investment for companies, so that its suboptimal management could lead to a shortened lifespan and under-utilization of computing, potentially causing concerns for investors. Autonomic Computing offers a solution to this problem as well, by making any equipment more affective during its useful life. This approach enables machines to optimize their resources efficiently and minimize errors re-

sulting from human distractions. While human managers will continue to play a crucial role, therefore, their focus will shift toward handling larger, more complex issues that machines cannot autonomously address.

Coming to the main subject of this work, the mentioned existing controller was aimed to reduce – ideally, minimize – idle resources within an HPC cluster. To accomplish this task, the INRIA team employed the control theory to construct a system based on a Proportional-Integral (PI) controller, with the purpose of dispatching incoming work to computing units by dividing it into small computational tasks, exploiting the so gained granularity to best fill the cluster's available resources.

However, the said controller was designed based on a simplified linear model. Consequently, its performance deteriorated as operational conditions deviated from the nominal state. The core contribution of this thesis is to enhance that controller with an identification and adaptation algorithm. This new algorithm enables the controller to dynamically adjust its parameters to accommodate for varying working conditions, preventing the undesired behaviors observed in the previous solution.

0.1. Organization of the thesis

Chapter 1 Large-scale description of the system with an overview on the literature.

Chapter 2 Insightful description of the system and, in the end, the previous work of the researcher is presented.

Chapter 3 Explanation of what is Adaptive Control and Self-Tuning. Then, the design of the three different algorithms in two different contexts - evaluation of one parameter and two parameters.

Chapter 4 In this chapter the algorithms are tuned and tested. After the test, only one algorithm is implemented in the real platform and here the suitable initial condition is discussed. In the end, there is a comparison between PI and Adaptive PI.

Chapter 5 Here there are the conclusions of the work and some ideas for the future.

1 | State of the Art

This chapter provides a brief overview of High-Performance Computing (HPC), in a way tailored specifically for automation and control professionals, followed by an exploration of the primary components of HPC clusters and grids. Having introduced the said physical components and their interactions, the focus is then moved to the growing significance of control theory in *autonomic computing*. Subsequently, the potential and the challenges of integrating the control theory into this relatively new field are examined. In addition, the chapter summarises the outcome of a comprehensive review of prior research, focusing particularly on two distinct works that address themes that are similar to those discussed in this thesis.

1.1. High-Performance Computing: an overview

High-Performance Computing (HPC) encompasses various computing systems that have common capabilities to overcome limitations encountered by traditional computers in terms of complexity and time consumption. These limitations often arise when dealing with simulations, heavy computations, or the management of vast amounts of data. HPC includes systems such as supercomputers, clusters, and grids, all of which share these aforementioned properties.

Before delving into technical details, we would like to give the reader a brief historical overview of these computational structures. The first supercomputer, the CRAY-1 (Fig. 1.3), was introduced in 1975 and boasted an unprecedented computational rate of 100 million floating-point operations per second (MFlop/s) [2]. Since then, HPC has made significant progress, prompting the need to monitor achievements worldwide. Thus, in 1993, the TOP500 list was established. The TOP500 list ranks the 500 most powerful HPC systems in the world and is updated biannually [3]. As of June 2023, the most powerful supercomputer on the list achieved an astounding rate of 1.194 Exaflop/s. The continuous progress of HPC is made possible by the development of different crucial functionalities as *parallel computing* and *communication network*.

Parallel computing is a key characteristic that empowers HPC. As the development of more powerful processors became increasingly challenging and expensive, a solution emerged: instead of relying on a single, specialized, and costly processor, multiple cheaper processors could be used [4]. Then, parallel computing replaces the single processor with multiple processors, effectively addressing the limitations of relying on a single processor. Distributing the computational load among different processors reduces the burden on each individual processor and overcomes power and thermal limitations.

However, this collaborative approach introduces a new challenge, that is, the need to manage a communication network. In the past, when the number of CPUs (Central Process Unit) was small, managing data and communication was not a significant issue, and separate networks for these purposes was feasible. However, as the number of CPUs increased, the communication network's cost escalated, leading to a tradeoff between cost and performance. Then, a decision was made to merge the two networks, creating a unified network for both communication and data transfer. Various topology networks have been developed, with one of the earliest being the torus interconnection. In this arrangement, all nodes are interconnected with one another, and the topology can be realized in different dimensions. An example of a 2D torus topology is shown in Figure 1.1, giving an idea of its shape.

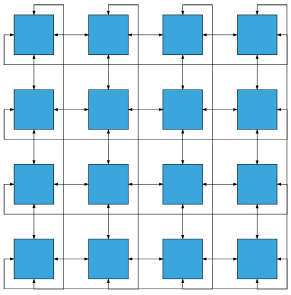

Figure 1.1: Torus design network [5].

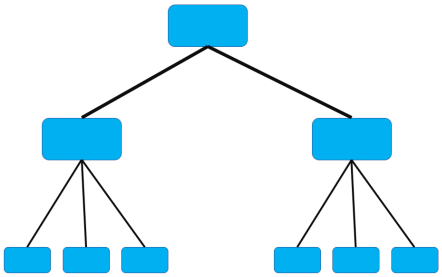

Figure 1.2: Fat tree network.

Another topology commonly used is the fat tree [6]. Unlike the torus, the fat tree does not connect each processor directly to others. Instead, it establishes a hierarchical structure (Fig. 1.2) resembling a tree, which reduces the amount of cabling required, resulting in a more cost-effective solution compared to the torus. "Fat" refers to the increasing bandwidth required as one ascends the fat tree; it starts with low bandwidth at the bottom and increases as one moves up.

Summing up, HPC distinguishes itself from traditional computers by offering incredible speed and computational power thanks to the use of several processors and network topologies. Then, because of the updating over the years of architecture and power

consumption, HPC systems have expanded beyond their initial realm of research and penetrated various fields, including aerospace, life sciences, artificial intelligence, finance, and energy [7, 8, 9].

Figure 1.3: Picture of CRAY-1 with his peculiar C shape [10].

1.2. HPC Cluster

A High-Performance Computing cluster opens up new horizons for parallel computing by embracing distributed computing models. Indeed, a cluster can be best described as an ensemble of stand-alone computers that are interconnected and collaborate smoothly to accomplish complex tasks. Each computer within the cluster is referred to as a *node*, and these nodes possess more than just processing capabilities. They also have their own dedicated memory and operating system, making them self-contained entities within the larger cluster ecosystem. Furthermore, the nodes are linked together through specialized, high-speed connections, facilitating efficient communication and data exchange [4].

The power of a cluster lies in its ability to divide the computational workload of a specific task among multiple computers, a process commonly known as workload distribution. Despite this division, the user (ideally) experiences a unified computing environment, perceiving the cluster as a single powerful machine. This perception is made possible through sophisticated coordination mechanisms and resource management techniques that seamlessly integrate the individual capabilities of each node into a cohesive whole. As a result, the cluster transcends the limits of a single machine and uses the combined strength of its constituent nodes to tackle computationally intensive problems with efficiency and speed.

1.2.1. Job Scheduling

In most scenarios, it is rare for a single job to utilize the entire cluster. As a result, the cluster needs to be divided and shared among multiple users. To facilitate this sharing, a *scheduler* is necessary to handle and manage all the job requests within the cluster. This scheduler is a critical component of a more complex system known as the Resource and Job Management System (RJMS) [11].
The RJMS can be divided into three distinct parts:

Application Manager : This component serves as the interface between users and the scheduler. Users submit job descriptions to the application manager, enabling the scheduler to find appropriate space and time within the cluster for each job.

Scheduler : The scheduler computes the job allocation and determines the timeslot for each job based on their respective priorities, if applicable.

Resource Manager : Acting as the interface between the scheduler and the cluster platform, the resource manager monitors the jobs within the cluster. It handles the deployment of applications and oversees their execution.

The scheduler serves as a bridge connecting the user interface and the platform interface within the system. It plays a crucial role in managing job scheduling and allocation. Following there is a representation of RJMS (Figure 1.4). In the picture, there is also the *file system* that will be explained later.

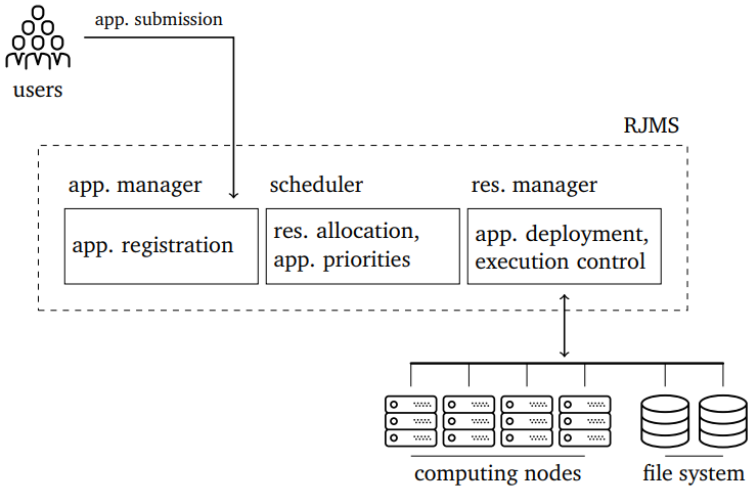

Figure 1.4: Representation of RJMS connected with both the user and cluster [11].

After providing an overview of the job scheduler's framework, we can delve into a higher-level understanding of its functioning. The following code snippet illustrates the basic

operation of the scheduler, specifically in the context of the First-In-First-Out (FIFO) discipline.

---

upon each event (arrived or terminated) run:
while *waiting_queue* is not empty
    next_job=waiting_queue.pop_front()
    if *next_job* fit now:
        launch(*next_job*)
    else
        waiting_queue.push_front()
        break

This code is executed only when jobs arrive in the queue or when the cluster finishes processing existing jobs. Two essential functions are featured in the code:

`pop_front()` : With this function, the scheduler selects the first job in the queue and plans it inside the cluster, if there is enough space.

`push_front()` : Conversely, this function adds a job to the front of the queue. This happens when there is not enough space inside the cluster. In short, the task gets picked up and put back in the queue in case the cluster cannot afford it.

The one above is a basic interpretation of the scheduler's work. More complex codes are usually employed, such as the well-known Backfill [12].

The job scheduling architecture is a crucial component of a cluster as it is in charge of efficiently managing task allocation, optimizing resource usage, ensuring fairness, and enhancing the overall performance. By intelligently coordinating job execution, the scheduler maximizes cluster efficiency and throughput, effectively utilizing available resources.

1.2.2. File System

The file system, also known as the file server, is another critical component of the cluster that works as storage. Within the cluster, each node can access two types of file systems: a local file, system exclusive to that particular node, and a shared file system, accessible by all nodes in the cluster. These two file systems exhibit different performances, with the local file system being limited in capabilities compared to the shared file system due to cost and space constraints. For the purpose of this thesis, we will focus on the shared file system.

The file system functions as an abstraction layer for the physical storage devices, enabling applications to make input and output (I/O) requests for specific portions of files [13].
One significant challenge in the relationship between the HPC cluster and the file server is the speed difference. While the cluster operates in the exascale speed, the file server lags behind due to slower devices like memory, disk, and network. Consequently, the I/O system's performance dictates the speed of the HPC system [14].
Another critical issue related to the file server is its available space. Each node in the cluster can use the file server to store, read or write data, and this may lead to a saturation of the file system's space. As a consequence, a file server's approaching saturation can significantly slow down the network between the cluster and the file server. In the worst-case scenario, the entire system may become stuck, even if the cluster is not completely full [15, 16]. Due to this substantial impact of the file server on the system, it is essential not only to monitor the state of the cluster but also the state of the file system to prevent such problematic scenarios.
One widely implemented partial solution is to execute the file system on several machines to improve performance. However, this does not entirely eliminate the aforementioned problems. Additional measures are necessary to effectively address this issue.

1.3. HPC Grid

While clusters are characterized by a distributed computing environment where all nodes are physically close to each other, grids, consist of clusters that are geographically dispersed. The primary objective of grids is to leverage the collective power of interconnected clusters, typically facilitated through the Internet. From a quantitative perspective, clusters typically consist of hundreds of nodes on average, whereas grids encompass millions of computers. Thus, it is evident that the power of a grid surpasses that of an individual cluster. However, within a grid, there can be variations in performance due to the inherent heterogeneity of the clusters [17, 4]. Moreover, HPC grids provide a distributed computing infrastructure that enables the pooling of computing resources across multiple organizations or institutions.
However, the geographic dispersion of clusters within a grid introduces a potential challenge: latency. As the workload is divided among clusters situated far apart, communication between them can introduce delays. Consequently, this latency issue must be carefully considered, as it could amplify the overall computational time.
In conclusion, HPC grids offer a powerful solution for distributed computing, leveraging interconnected clusters across geographical locations. However, the challenge of latency needs to be addressed to fully exploit the potential benefits of grid computing and avoid

significant computational time amplifications.

1.4. Autonomic Computing

In 2001, IBM identified a critical challenge in the advancement of computing systems: complexity. At that time, system administrators were entirely responsible for managing computer systems. IBM stressed that if the growth rate continued as before, there would soon be a complication to handle by hand [18, 19]. Fortunately, alongside this concern, IBM put forth a potential solution: *autonomic computing*. In reference to Parashar et al.[20], autonomic computing refers to the utilization of self-governing strategies and algorithms by the system and its applications to effectively manage complexity and uncertainties, thereby minimizing the need for human intervention, which leads to *self-adaptive* software system [21, 22]. This approach enables an HPC system to adapt and optimize its performance autonomously. Here is where the expertise of automation and control professionals becomes crucial. Automation not only enables self-management but also facilitates self-optimization from various perspectives, including energy consumption and the utilization of resources [23, 15]. By embracing autonomic computing principles, the field of computing aims to address the increasing complexity of computer systems while maximizing their performance and efficiency. These advancements have the potential to alleviate the burden on human administrators and enable computing systems to operate seamlessly in dynamic environments.

In 2005, IBM introduced an example of autonomic computing architecture, depicted in Figure 1.5. This architecture, referred to as MAPE loop, consists of four main components, each serving a distinct purpose:

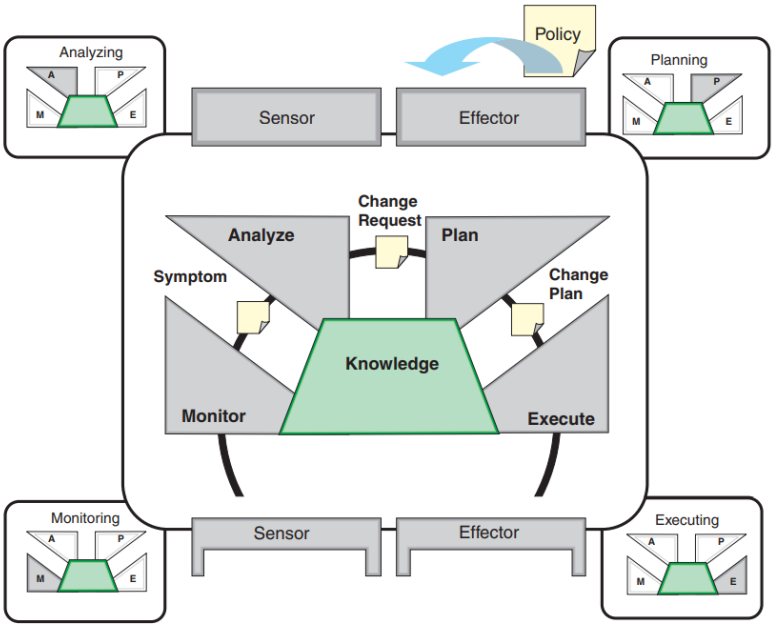

Figure 1.5: Functional detail of the autonomic manager [24].

Monitor : This component is responsible for collecting, aggregating, filtering, and reporting detailed information, such as metrics and topologies, from a managed resource.

Analyze : The analyze component utilizes mechanisms for correlating and modeling complex situations. For instance, it can perform tasks like time-series forecasting and queuing models. By analyzing data, the autonomic manager can learn about the IT (Information Technology) environment and predict future situations.

Plan : The plan component defines the necessary actions to achieve goals and objectives. It leverages policy information to guide its decision-making process.

Execute : The execute component controls the execution of the planned actions, considering dynamic updates and changes in the environment.

These four components work in concert to provide the functionality of a control loop. As depicted in Figure 1.5, all the components communicate with each other, exchanging relevant knowledge. Lastly, the policies serve as constraints and goals that the overall system must adhere to. This entire figure is commonly referred to as the MAPE-K loop, which stands for Monitor-Analyze-Plan-Execute over a shared Knowledge.

The discussion mentioned above was presented by Eric Rutten et al. in 2017 [25]. In their work, they aimed to provide a clearer understanding of the MAPE-K loop in the control theory field. They proposed reinterpreting the different phases of the loop as follows:

Monitor : This phase can be seen as the sampling phase, where data is collected from the system with a proper frequency.

Analyze : The analyze phase can be seen as signal estimation or the reconstruction of a signal based on the collected data. Thus, variables with the hat (i.e. $\hat{x}$).

Plan : The plan phase involves computing the control law using the system's knowledge held in the model.

Execute : The execute phase corresponds to the actual implementation of the control law, which involves making changes to the system through actuators.

Considering the aforementioned reinterpretation, the figure referred to as 1.5 can be updated with Figure 1.6.

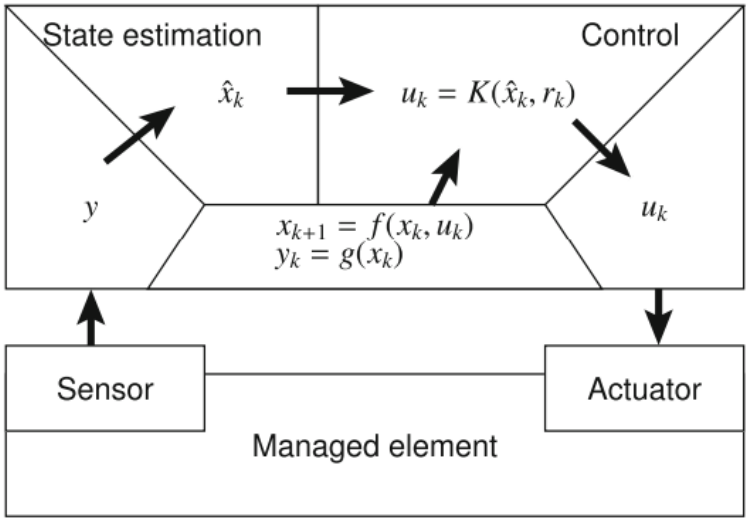

Figure 1.6: Reinterpretation of MAPE-K loop [25].

1.5. Dynamic HPC management using Control Theory

1.5.1. Assets of control theory for computing system

The application of control theory in the software field brings forth various methodologies that were not previously utilized. These approaches offer significant advantages and can greatly enhance the effectiveness of software control systems. In the following, we present how the key aspects of control theory are valuable tools for software development [26].

Input-Output Model : One fundamental concept is the Input-Output Model. This model is the fundamental brick for the system in Figure 1.7, where the control input (u) is linked to the measured output (y). By defining this relationship, we can establish a foundation for feedback-based control, leading to improved results. Neglecting this initial step could lead to issues such as failure to converge to equilibrium, overly aggressive or slow controllers, or an inability to adapt to different operating conditions.

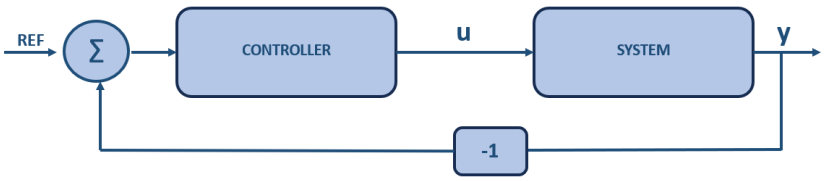

Figure 1.7: Simple control scheme.

Dynamics and Transient : Control theory models not only consider the input-output relationship but also incorporate the dynamics and transient effects. These models capture the influence of past output values on the present output, enabling the representation of memory effects within the system. For example, in equation 1.1 the coefficients a_i and b_j represent respectively the correlation between the current output and the past output, until n, and input, until $m-1$. By accounting for these dynamics, we gain a deeper understanding of the system's behavior.

$$y(k) = \sum_{i=1}^{n} a_i y(k-i) + \sum_{j=0}^{m-1} b_j u(k-d-j) + e(k) \tag{1.1}$$

Inside the above equation k is the time step, d is the delay and e is noise.

Correlation between Multiple Metrics : Another advantage of control theory is the ability to correlate multiple metrics. Indeed, it is possible to design models with different input and output variables, each with its own unique correlations. This flexibility allows us to capture complex relationships between different aspects of the system, leading to more comprehensive control strategies.

Well-Studied Control Algorithms : In the realm of control algorithms, software systems often rely on threshold-based approaches to maintain desired output values. While these algorithms are relatively straightforward to understand and implement, fine-tuning them can be challenging. The control theory offers a wide range of well-studied control algorithms that have been extensively researched and refined. Leveraging these established algorithms can greatly simplify the tuning process and improve system performance.

Stability Guarantees : Control theory also provides stability guarantees for the closed-loop system. By employing analytical methods and utilizing models of the target system and controller, we can assess the stability of the overall system. Furthermore, control theory offers guidelines for selecting appropriate parameter values to ensure system stability, enhancing the reliability of software control systems.

Nonlinear and Time-Varying Behavior : Moreover, control theory addresses the challenges posed by nonlinear and time-varying behaviors. Adaptive control theory, in particular, offers proven methodologies for designing controllers that can handle systems with varying behavior over time. Additionally, it provides techniques to control nonlinear systems, expanding the range of systems that can be effectively controlled.

In summary, the control theory gives crucial advice to model and design feedback loops in software systems. By embracing the methodologies and techniques offered by control theory, it can be feasible to enhance the performance, stability, and adaptability of software systems.

1.5.2. Challenges

However, the cultural differences between software and control engineering introduce significant challenges. Numerous studies have been conducted to establish a connection between the fields of computer science and control [27, 28]. This endeavor has been driven by the recognition that pure control specialists cannot effectively utilize their expertise without a comprehensive understanding of the processes they are dealing with. Likewise, computer science experts may lack familiarity with control theory, necessitating collabo-

ration with those who possess knowledge of the required modeling and control synthesis methodologies
For example, in the two fields, the word "adaptive" has two different meanings. In software engineering, a system is called “adaptive” if it is capable of responding to the same stimulus in different ways depending on the particular situation in which the said stimulus is received. Since in the systems theory such a property is enjoyed by any dynamic system, just naming the “situation” state, not all the systems that are adaptive in the computer sense are adaptive in the systems and control sense. For the latter fact to be true, the *law* that computes the system state and output must be subject to modifications - for example, having some of its parameters change.

In addition to the differences in terminology, there are much more intrinsic differences like the idea of modeling. In software, models are often based on architectural concepts, whereas control engineering relies on equations to estimate models in the discrete and continuous time. Therefore, creating a detailed analytical model for software becomes challenging, as software is not governed by the same law as physical systems [29]. Furthermore, even if such a model is successfully created, it may quickly become inaccurate after the first software update [30].
Another obstacle faced in the software domain is the actuator. Actuators in software systems are often subject to heavy quantization, which imposes constraints on control without clear justifications. Additionally, variable delays within the system pose a significant problem. These delays can arise from various sources, including the controller itself, which may request data from parts of the system that are not designed to respond efficiently. Another challenging issue arises from sensor noise attributed to concurrent external processes. This noise significantly affects the ultimate outcomes and results. Further problems related to control theory in the software domain are discussed in detail in A. Leva et al. [31].

1.5.3. Literature overview

Having discussed the advantages and difficulties of applying control theory to computing systems, we can examine previous applications and their results [30]. From a modeling perspective, it appears that linear models are often preferred over nonlinear models. This preference stems from the greater complexity involved in estimating nonlinear models compared to linear ones. Furthermore, the lack of available tools for identifying nonlinear models exacerbates the difficulty [32]. However, in the cases of interest of this thesis, it seems that finding a model capable of representing suitable average quantities is sufficient, as the system nonlinearities that cause deviation from averages can be compensated for

by equipping the feedback controller with adaptive or online model update mechanisms [33].

From a pure control standpoint, the most widely preferred control strategy is the Proportional-Integral-Derivative (PID) controller, as depicted in Figure 1.8. PID control is highly favored due to its ease of implementation and satisfactory performance in terms of set-point tracking and disturbance rejection. Moreover, it is worth noting that *adaptive* PID controllers are commonly used in software applications to compensate for errors resulting from model linearization. The detailed explanation of the PID controller's functioning will be provided later in this thesis.

Despite its favorable characteristics, the PID controller does have a limitation: scalability. Consequently, it is primarily preferred for Single-Input-Single-Output (SISO) systems. In contrast, for Multiple-Input-Multiple-Output (MIMO) systems, alternative techniques are preferred, with Model Predictive Control (MPC) being the most commonly used. MPC refers to a controller that employs a system model to predict future behavior and selects adaptive actions to minimize the cost associated with achieving this behavior. The adoption of MPC enables optimal management of multiple requirements, as demonstrated in various literature studies across different domains [34, 35, 36].

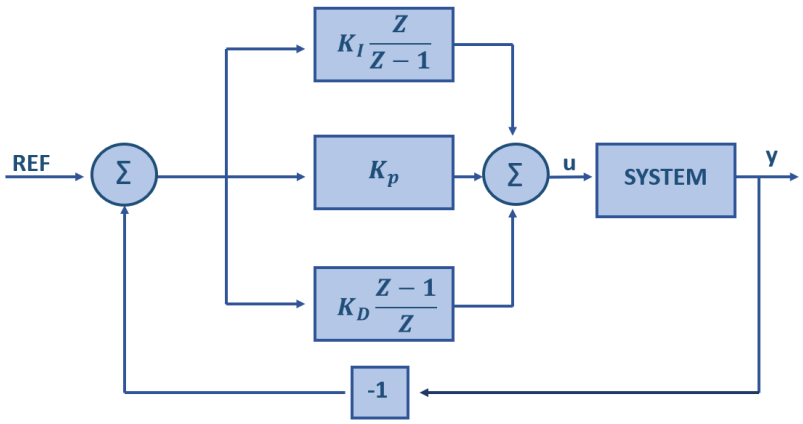

Figure 1.8: PID control loop in discrete time.

In an effort to implement control theory in the software domain, various attempts have been made. For Instance, in their work, S. Cerf et al. [23] successfully computed a linear model and used a PI controller to regulate an HPC grid. In this case, they applied the same methodology across three different clusters. The results showed satisfactory performance in two clusters, but the third cluster did not yield the desired outcomes. This case exemplified that simple models augmented with basic PI controllers can be effective; however, their applicability may be limited as linear approximations and straightforward

controllers might not always account for all relevant phenomena, as evidenced in the case of the last cluster.
To address this limitation while preserving a linear model, an adaptive controller (Ch. 3.1) can be employed. For example, in the work of A. Filieri et al. [37], researchers generated a linear model for the system and synthesized a configurable controller. This controller tackles potential nonlinearities by dynamically adapting the linear model using a Kalman filter. Moreover, to handle significant changes in system behavior, the controller incorporates a change-point detection strategy to trigger an online model rebuilding phase. By integrating these two features into the controller, the overall system performs well even with a linear model instead of a nonlinear one.

The works quoted above address the challenge of applying a linear model to a complex process from different perspectives. The first work illustrates that a controller designed exclusively for a linear model could not always be effective. Conversely, the second work overcomes this limitation by employing an adaptive controller and recalculating the model when the operational conditions significantly deviate from the previous state. In this study, we aim to solve this issue using an alternative approach. We propose enhancing the existing PI controller with an identification algorithm. In this way, the controller should meet the specified requirements irrespective of the operational conditions.

2 | System description

In the preceding chapter, we provided an overview of HPC and introduced its main components in general. In this one, we investigate the elements of the particular system used for this study. To begin, we present the HPC grid where experiments are conducted, followed by an examination of one of the main aspects of this thesis: the CIGRI controller. Within this section, we highlight the existing limitations of the current CIGRI implementation. Then, a comprehensive analysis of the scheduler is provided, along with an overview of all the interactions among the system.
Moving forward, we discuss the state of the research prior to the work presented herein. We elucidate the model's design, the underlying hypotheses, and the subsequent validation process. Finally, we introduce the controller development based on this model and conclude with a thorough performance evaluation of the controller.

2.0.1. The Experimental Infrastructure: Grid5000

The entire study is conducted within the HPC grid known as Grid5000. This grid comprises various clusters spread across nine different cities in France, as depicted in Figure 2.1, with a total of 15000 cores and 800 nodes.
Grid5000 serves as a large-scale and versatile testbed, supporting experiment-driven research in all domains of computer science, with a particular emphasis on parallel and distributed computing, including Cloud, HPC, Big Data, and AI. It is widely utilized by multiple research teams from different universities such as Université Grenoble Alpes and Université de Lorraine, as well as by esteemed organizations like CNRS and INRIA.

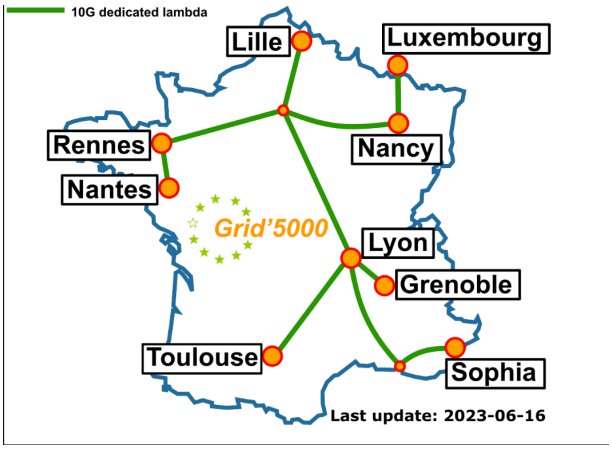

Figure 2.1: Geography representation of Grid5000 [38].

The popularity of this infrastructure stems from the wealth of information it provides, which would otherwise be unattainable through private clusters like Google Cloud Platform or Azure. This access to valuable information makes Grid5000 a highly suitable environment for research experiments, granting researchers the essential insights required to comprehensively understand the workings of the cluster.
Moreover, Grid5000 boasts a remarkable level of reconfigurability, facilitating easier execution of experiments while offering complete control over the processes. Researchers can even isolate their experiments within the network, enhancing the overall controllability of the study environment.
For all the reasons above, Grid5000 is an ideal platform for conducting this work because we are trying to test a controller that should be suitable for whatever cluster inside whatever grid. Then, all possible data that are accessible in Grid5000 becomes crucial for understanding where are the issues and giving an idea of how to solve them.

2.1. Single Cluster Components

As seen in the previous chapter, an HPC grid can be seen as a collection of clusters. This thesis focuses on the control of a single cluster, while the wider research to which the thesis belongs also comprises the management of different clusters.
In this section, we provide an overview of the crucial components that collaborate with the cluster and play a key role in this work. The first component is the controller, known as CIGRI [39]. Its main responsibility is to dispatch a specific number of jobs to the scheduler OAR [40]. Additionally, we explore various types of jobs that may arrive and the potential challenges associated with the dispatching process. By the end of this section, we show a comprehensive overview of the entire system.

2.1.1. Controller CIGRI

The CIGRI emerged as an innovative solution to address a widespread challenge faced by various laboratories – how to efficiently pool and leverage the resources of a cluster to achieve optimal performance. This led to the development of a simple, lightweight system based on a modular architecture, incorporating high-level components that enable the use of locally unutilized resources within the cluster.
The core idea behind CIGRI is to use only the idle resources of the cluster, without affecting the tasks assigned to local users. To achieve this, the concept of *Best Effort* (BE) jobs is introduced in the batch scheduler. BE jobs are jobs that require less computation and have a lower priority than the other type of jobs, namely the *High-Priority* (HP). In this study, an approximation is made - only jobs coming from CIGRI can be considered as BE, hence the HP jobs will be tasks coming directly from users. Afterward, the CIGRI functions as a server that communicates with all the batch schedulers across the clusters. It operates like a regular user, submitting jobs at the BE queue of the scheduler (Figure 2.3).

While the primary objective of CIGRI is to efficiently harvest idle resources, it is crucial to consider potential issues that might arise if this process is not carefully managed. For instance, if CIGRI indiscriminately uses the cluster's resources without considering other factors, such as the file server's capacity, it could lead to an overload. Therefore, CIGRI must be equipped with a feedback mechanism not only for the clusters but also for the file system, to prevent overload and maintain system stability [15].

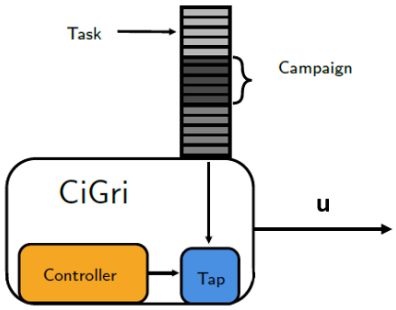

Figure 2.2: Schematic picture of CIGRI.

Figure 2.2 displays a simplified illustration CIGRI. Indeed, for the purpose of this thesis, we focus solely on the modules linked to the controller behavior.
Starting with the job queue, one can note that the jobs are organized into *campaigns*. In this context, a campaign refers to the number of jobs available with a specific file size f

(MByte).
Inside CIGRI, there are two blocks: the Controller and the Tap. The Controller plays a crucial role as it employs an algorithm to determine the optimal number of jobs to send to the scheduler queue based on the system's current status. On the other hand, the Tap is responsible for receiving the jobs selected by the Controller after this decision-making process.
An important characteristic of CIGRI is the length of the period, namely the constant time it waits before sending a new set of jobs; this is set 30 seconds. Due to this behavior, both the model and the controller will be designed in discrete time.

The research time chose to focus on CiGri due to its *current* limiting implementation:

Input : $rate$ (init.3)
$increasefactor$ (constant 1.5)
if *no running jobs* **then**:
$rate = \min(rate \times increasefactor, 100)$
submit $rate$ jobs
else
submit 0 jobs

The above algorithm involves injecting new jobs into the system, but it lacks a useful feedback mechanism regarding the system's state. Consequently, the controller will only send new jobs if all the previous batch of jobs has been executed. This approach poses a significant problem: there might be situations where CIGRI refrains from sending any new job because there is a single lingering job from the previous batch still running in the cluster. This limitation has drawn the teams attention, and it was argued that control theory could be a potential solution to overcome this issue.
By leveraging control theory, the team aimed to enhance the performance of CIGRI by introducing a feedback loop that allows the controller to make more informed decisions. Indeed, monitoring the state of the system and the progress of running jobs, the controller can dynamically adjust the injection of new jobs, ensuring a more efficient and balanced workload distribution within the cluster. Through this approach, they wanted to aim a real optimization of resources into the cluster.

2.1.2. Priorities inside the OAR scheduler

The OAR scheduler operates as described in the subsection 1.2.1. Within OAR, two different queues exist: one for jobs coming from CIGRI, known as BE jobs, and another

one for jobs submitted directly by users, referred to as the HP jobs. In the context of queueing theory, the relationship between these two types of jobs can be characterized as *preemptive priority*. This means that BE jobs can only be executed when the HP queue is empty or no HP job cannot be executed. Additionally, HP jobs have the authority to displace a BE job from the cluster if there are no available resources to accommodate the HP job [41]. The primary objective of OAR is to efficiently schedule both types of jobs. However, if an HP job needs resources and takes the place of a BE job, the preemptive priority principle comes into play, and the BE job is sent back to the BE queue to wait for available resources again.
Another notable distinction between the BE and HP jobs lies in how they book the cluster's resources. When jobs arrive from CIGRI, they enter the queue of the OAR and wait for their turn to be executed. Instead, HP jobs have the advantage of being able to schedule specific time slots for their execution. This flexibility in scheduling is governed by the internal policies of the cluster, aimed at promoting smoother resource sharing. By allowing HP jobs to book time slots, the cluster's resource allocation becomes less troublesome. For instance, an internal rule dictates that resource-intensive jobs should be scheduled during the night when the cluster experiences lower usage, ensuring that these jobs do not interfere with other user's activities.

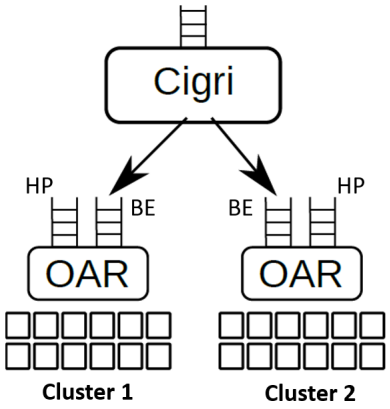

Figure 2.3: Representation of two OARs for two different clusters. The queue linked to CIGRI is the Best-Effort queue while the other is the High-Preority queue.

The job scheduler visualizes the bookings on a GANTT chart, as illustrated in Figure 2.4. In this representation, the y-axis denotes the reserved space within the cluster, the x-axis represents time, and the rectangles symbolize the scheduled jobs from different users. The specified duration for each job is typically provided by the user, which often tends to give an overestimation of the actual execution time.

Figure 2.4: Example of a GANTT chart [42].

With this scheduling approach, the cluster's resources are utilized more efficiently, and users can plan their job execution with greater flexibility while adhering to the cluster's internal policies.

In delving deeper into the working of OAR, we discover that it requires a certain amount of time to schedule a job, a process referred to as *provisioning*. Additionally, it takes some time to delete a job, known as *deprovisioning*. However, the efficiency of the OAR largely depends on the state of the job queue. If the queue is filled with numerous pending jobs, OAR will experience slower job scheduling within the cluster. Consequently, having knowledge of the waiting queue's state could prove beneficial for the controller as it would prevent overloading the OAR's waiting queue. By understanding the current status of the queue, the controller can make informed decisions to optimize job allocation and avoid potential bottlenecks, thus enhancing the overall performance of the OAR system.

2.1.3. Overview of the System's Description

In the preceding sections, we presented the controller, CIGRI, responsible for dispatching jobs to OAR. On the other hand, OAR manages two distinct queues: one for CIGRI's jobs (referred to as BE jobs) and another for jobs submitted directly by users to the cluster (referred to as HP jobs). Notably, HP jobs take precedence over BE jobs in terms of priority. To clarify, both types of jobs are sent to the cluster and executed with the simultaneous utilization of the file server.

The interaction between these components can be visualized in Figure 2.5.

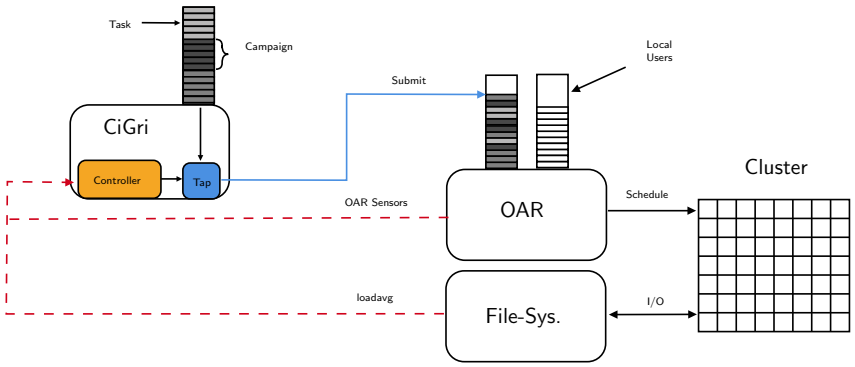

Figure 2.5: Overall system [15].

In addition, there are information exchanges between CIGRI and OAR, as well as the File System. The OAR sensor enables the controller to access essential data held by OAR, such as the cluster's status and the state of the waiting queue. Meanwhile, the other sensor provides insights into the file system's state, helping to prevent overload.
In cases where administrators lack specific information about the system's bottleneck, they may establish separate controls for three different closed loops. One closed-loop focuses on the cluster's objectives, such as energy performance or the filling level, while others aim to prevent overloading the file server and the BE queue of OAR. To achieve optimal control, a final control law may be derived from these three different control laws [15].

This work focuses on the system's limitations, with the file system identified as the bottleneck. This implies that the file system has significantly fewer resources compared to the cluster itself. Given this substantial gap between the two amounts resources, we make the assumption that we can treat the cluster as having an infinite number of resources for the purpose of the analysis. Throughout the thesis, we will consistently refer to a single controller, specifically the one designed for the file system. This controller plays a pivotal role in managing and optimizing the file system's operations to improve the overall system performance.

2.2. Prior work: a PI controller

The primary objective of this study is to prevent the file system from becoming overloaded. The strategy we will adopt involves modifying the controller algorithm while preserving its fundamental structure. Consequently, the CIGRI controller is tasked with sending jobs in a manner that avoids burdening the file system with excessive load. Then, the system we are about to discuss is depicted at high abstraction level in Figure 2.6.

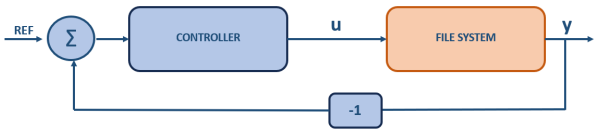

Figure 2.6: Picture of the file system closed loop.

In Figure 2.6, two crucial quantities require elucidation: the control action denoted as u, and the output labeled as y. In this context, the control action is represented as a two-dimensional vector encompassing distinct quantities. The first component, denoted as n, signifies the count of BE jobs dispatched by the controller. The second component corresponds to the weight f, namely the *file size*, of an individual BE task, measured in MByte. These two quantities are subject to specific limitations. Naturally, neither of them can take negative values due to their physical interpretation. Moreover, n is subjected to an additional constraint. Since it is not possible to dispatch part of a job, n must be integer. Consequently, it must also adhere to a quantization requirement. Instead, the output y corresponds to the load on the file server, obtained through sensor-based measurements. However, a complicating factor arises from the sensor's measurement of various quantities. As a result, the recorded measurement y becomes susceptible to substantial amounts of noise.
Figure 2.6 illustrates the presence of the file system as the unique process, but it is important to acknowledge that, as depicted in Figure 2.5, the controller's perspective of the system encompasses a trio of distinct processes: OAR, cluster, and file system. The simplification in Figure 2.6 focuses solely on the file system, aligning with the forthcoming model's specification. This scope narrowing is due to the underlying assumption that both the scheduler and the cluster's processing speeds far exceed that of the file system.

After depicting the control loop under study, in order to develop an effective controller for the system, it is essential to have a thorough understanding of the system itself. Hence, we will initiate this section by describing the model.

2.2.1. Model of File System

Modeling the static behavior

To identify the model, the team initiated the experimentation phase in an open loop setup, aiming to comprehend the underlying relationships. Importantly, they aim to model only the impact of the writing tasks, because these are considered the ones with more impact with respect to the other two operations, reading and storing.
In an open loop experiment, the team systematically manipulated the system's inputs and observed their corresponding effects on the output. The ultimate objective was to identify the optimal mathematical relationship that could effectively explain the changes in y with respect to n and f. By the use of linear regression they got

$$y = c + \beta_1 f + \beta_2 n + \gamma f \times n \tag{2.1}$$

With:

- $c = -0.5071484$
- $\beta_1 = 0.0086335$
- $\beta_2 = 0.0451394$
- $\gamma = 0.001633$

Through this experimental process, they arrived at the above model, which incorporates the two inputs, n and f, as shown in Formula 2.1. Although, It might seem odd that the result of a linear regression is a nonlinear model, the reason behind this is that in each experiment f was always fixed. However, this was an approximation made by the team. In reality, in practical scenarios, f varies in relation to the executed tasks, and its precise values is unknown.

Dynamic modeling

Now we are interested in dynamic modeling, the researchers aimed to develop a first-order model due to its simplicity and ease of study. Thus, it will be possible to predict the next value of y based on the previous value and the previous action u. Following the generic shape of a first order model:

$$y(k+1) = ay(k) + bu(k) \tag{2.2}$$

In pursuit of this objective, they considered the structure of the sensor located at the end of the file server, which is an exponential filter called *loadavg*. This particular filter was employed to obtain a smoother output result, and it is represented by the following expression, similar to the previous one:

$$y(k+1) = (1-\xi)N_k + \xi y(k) \tag{2.3}$$

Here, $y(k)$ denotes the loading of the file system at time instant k, and N_k represents the number of processes currently running on the system, namely, how many files are writing at instant k. The value of the parameter ξ is determined by the sensor's sampling time. A larger sampling time leads to a smoother output result, but it may take longer for the estimated value to converge to the actual value. Conversely, a smaller sampling time provides faster estimation but may not achieve significant smoothing of the output. After careful considerations, they opted for a small sampling time of 5 seconds (smaller than a period of CIGRI), as it struck a balance between satisfactory measurement and reasonable smoothness.
In reference to Ferrari et al. [43], the specific value of ξ was determined as follows:

$$\xi = \left(\exp\left(-\frac{5}{60}\right)\right)^{\frac{\Delta t}{5}} \tag{2.4}$$

Where Δt is 30s, the period of CIGRI.

As previously mentioned, the primary objective is to formulate a model in the format represented by expression 2.2. Up until now, the explanation has focused on the aspect that relies on the correlation with the previous output. Now, we can proceed to evaluate the correlation on the input.

Model Parameters

To do the model the researchers need to define what are the input of the system. For the nature of the system, the team decided to consider as input n because it's a controllable quantity while f will be considered as a disturbance because has an impact on the system but is not possible to control. Afterward, to be consistent with the control literature we will rename u as the number of BE jobs sent by CIGRI.
To accomplish the model they add an additional identification:

$$(1-\xi) \times N_k \approx b \times u(k) \tag{2.5}$$

With the latter expression the Eq. 2.3 becomes equal to Eq. 2.2. Then, by inverting this new formula at steady state, one gets b

$$\bar{y} \approx \xi \times \bar{y} + b \times \bar{u} \implies b \approx \frac{\bar{y}\,(1-\xi)}{\bar{u}} = \tilde{b} \tag{2.6}$$

Now, an expression for b is available, but relies on u and y. Taking into account the formula from the linear regression (Eq. 2.1) in the place of y and considering the limit to infinite of u, which means a big quantity of jobs, we can effectively untangle these two quantities:

$$\lim_{u \to +\infty} \tilde{b} \approx \lim_{u \to +\infty} \frac{(c + \beta_1 f + \beta_2 u + \gamma f \times u)\,(1-\xi)}{u} = (\beta_2 + \gamma f) \times (1-\xi) = \tilde{\tilde{b}} \tag{2.7}$$

Finally, the coefficients of the first-order model are:

$$\begin{cases} \xi = \left(\exp\left(-\dfrac{5}{60}\right)\right)^{\frac{\Delta t}{5}} = 0.6065307 \\ \tilde{\tilde{b}} = (\beta_2 + \gamma f) \times (1-\xi) \quad = 0.017761 + 6.4273488 \times 10^{-4} \times f \end{cases} \tag{2.8}$$

The significance of the second input becomes evident, and it comes out that isolating the parameter b (and also the model) from the file size of the jobs (f) is not a feasible task. In conclusion, the team uses $\tilde{\tilde{b}}$ as the value for b to build the model and afterward the controller.

Subsequently, an investigation is conducted to assess the efficacy of this approximation under various conditions. This study aligns with the general aim of this thesis, which

involves real-time estimation of the parameter b, as opposed to relying on the approximation.

Figure 2.7 illustrates the disparity between the $\tilde{\tilde{b}}$ value obtained from Eq. 2.7 and the $\tilde{b}$ value acquired through approximation in Eq. 2.6 by the use of the relative percentage error. The presented set of four graphs exhibits distinct variations as they correspond to different values of f, ranging from 25 MByte to 200 MByte. Each graph effectively portrays the error with respect to u, where u takes discrete values between 10 and 50, in increments of 10. One can note that the two approximations exhibit closer agreement when dealing with larger values of the input u, compared to cases with smaller inputs. Notably, the error term $b_{err} = \tilde{\tilde{b}} - \tilde{b}$ increases with the file size, and $\tilde{\tilde{b}}$ tends to overestimate $\tilde{b}$ only when f=25MByte, whereas in all other cases, the opposite holds true. Thus, $\tilde{\tilde{b}}$ appears to capture a distinct aspect of $\tilde{b}$, which represents a more accurate representation of b. However, it is crucial to acknowledge that $\tilde{\tilde{b}}$ becomes less reliable when f is equal to or greater than 200MByte and with a limited number of jobs (less than or equal to 20).

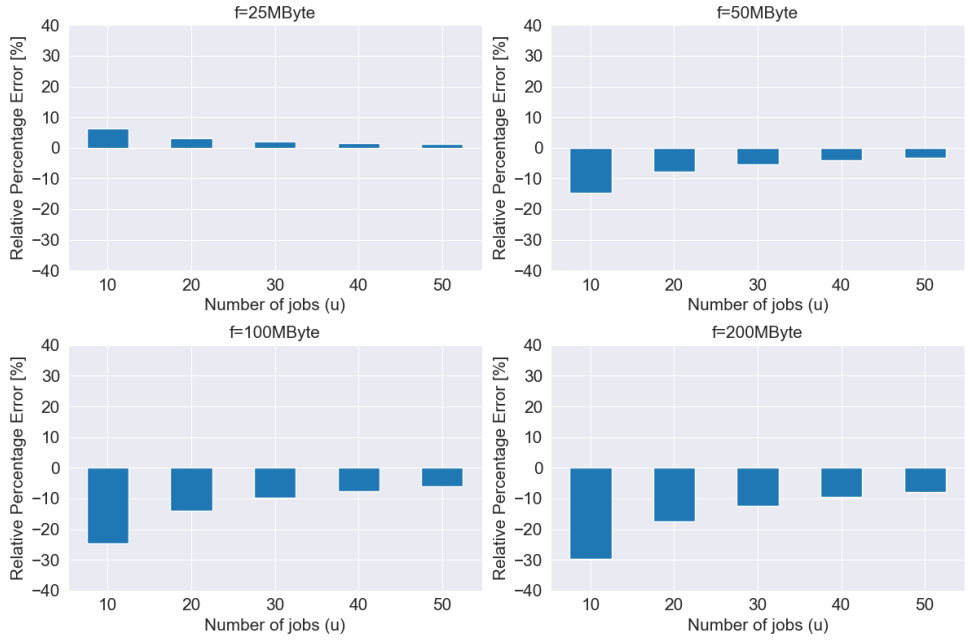

Figure 2.7: Study of the difference between the two approximations $\tilde{\tilde{b}}$ and $\tilde{b}$ for the different file size.

In Figure 2.8, we present a comparison between the real experiments conducted on the file server in an open-loop configuration (represented by the orange dots) and the model previously shown (represented by the blue line), with three different file sizes.

$$model = G(z) = \frac{\tilde{\tilde{b}}}{z - \xi} \tag{2.9}$$

The orange dot represents an average of five experiments, which helps to reduce the impact of noise on the result. Notably, the peak values differ among the three rows, which can be attributed to varying file sizes from (a) to (c). While the model provides a reasonably accurate estimation of the true system's behavior, one can note that its performance improves with higher file sizes and diminishes with smaller ones. Specifically, in the first row, the model tends to overestimate the load on the file server. However, this overestimation becomes less pronounced in the second row, and in the last row, the model appears to fit the true process very well.

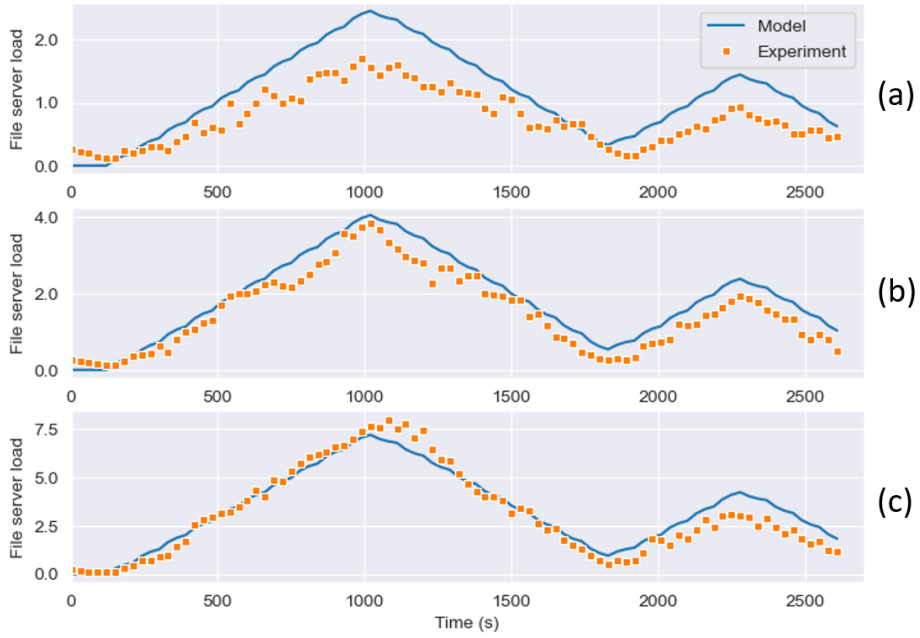

Figure 2.8: Comparison between the load of the file system in open loop and the model computed above. The orange dot represents an average of five experiments, while the blue line shows the model. (a) Case of file size = 50 MByte. (b) Case of file size = 100 MByte. (c) Case of file size = 200 MByte.

The outcomes of these two studies yield non-obvious conclusions. In the initial investigation, it appears from Figure 2.7 that as f becomes smaller, the disparity between $\tilde{\tilde{b}}$ and $\tilde{b}$ diminishes. However, an opposite observation arises from Figure 2.8, where the scenario with the smallest file size exhibits the worst result. The research team has yet to identify a definitive solution to this quandary. The disparity between these two sets of results

could stem from $\tilde{b}$ not being an accurate approximation for b, leading to a reduction in the reliability of the first study. Alternatively, the model overestimates the actual system, and in the case where $\tilde{\tilde{b}}$ underestimated $\tilde{b}$, there is a compensation which led to a good result in the last row.

Up to this point, we showed the disparities that arise among various approximations of the parameter b, by plugging in the suitable file size. However, in practical scenarios, the parameter f remains unknown. Consequently, our team needs to select a nominal file size in order to establish a single model. In this way, the controller is developed on this chosen nominal file size. For this purpose, a nominal file size of 100 Mbyte is adopted as a reference for the controller design. This particular file size was deliberately chosen due to its perceived representativeness, as indicated by J. Emeras et al. in their work [44]. Upon establishing this nominal file size as a foundation, the resulting system model takes on the following form:

$$G(z) = \frac{\tilde{\tilde{b}}}{z - \xi} \approx \frac{0.0821}{z - 0.606} \tag{2.10}$$

Subsequently, the final model incorporates an additional approximation that enhances its performance within the proximity of the 100 Mbyte nominal file size. However, its reliability diminishes as the parameter f deviates significantly from this established nominal value.

2.2.2. Controller Algorithm

After developing a first-order model for the system, the researcher's objective is to find a suitable controller to effectively regulate the system's behavior. The choice is to employ a Proportional-Integral (PI) controller due to its simplicity, robustness, and promising performance. The PI controller has two distinct parameters: the proportional gain denoted as K_p and the integral gain denoted as K_I. The control input is calculated as follows:

$$u(k) = K_p error(k) + K_I \sum_{i=0}^{k} error(i) \tag{2.11}$$

The role of the PI controller is to minimize the error between the system's output and the desired reference value. This error is determined as the difference between the reference value and the actual output of the system. By taking this error as its input, the PI controller generates a control signal to influence the system and bring the error to zero.

The proportional parameter K_p governs the strength of the controller's action in reaching the desired setpoint. However, relying solely on the proportional term is not sufficient to guarantee an error of zero in a steady state. To address this limitation, the integral part K_I is introduced. The integral gain allows the controller to accumulate the error over time, which helps to address any steady-state error that remains even after the proportional control has been applied. By combining both the proportional and integral actions, the PI controller is capable of effectively regulating the system, ensuring that it converges to the desired setpoint with minimal error. In general, another possibility could be the PID control, as shown in Figure 1.8, where the derivative action brings a quicker answer but also more sensitivity to noise.
In short, the PI controller operates by taking the error between the reference value and the actual system output and then adjusting the system's input to minimize this error. The two parameters, K_p and K_I, respectively control the proportional and integral actions of the controller, if chosen carefully, ensuring a robust and precise control strategy for the system.

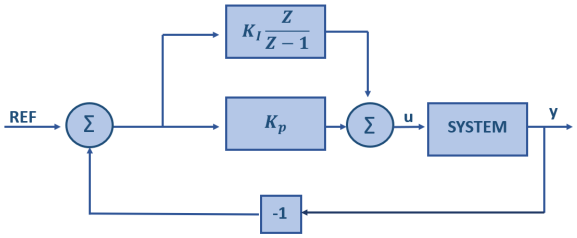

Figure 2.9: Proportional-Integrative controller

Then the PI was designed by taking into account the model 2.10 and the technique employed by J. Hellerstein et al. [45]. The latter approach allows to choose two main parameters of the PI to fulfill the two requirements K_s, the settling time in control steps, and M_p, the maximum overshoot (as a percentage of the reference).

These two desires are linked to the new poles of the system $re^{j\theta}$ and $re^{-j\theta}$, where:

- $r \approx e^{-\frac{4}{k_s}}$
- $\theta \approx \pi \frac{\log r}{\log M_p}$

Lastly, the final two parameters are directly integrated into the controller, resulting in the following parameters:

$$\begin{cases} K_p = \frac{\xi - r^2}{b} \\ K_I = \frac{1 - 2r\cos\theta + r^2}{b} \end{cases} \tag{2.12}$$

For further details, all the computations are explained in Appendix A.
In the end, the team chose as value for the PI controller $K_s = 12$ and $M_p = 0$. This selection ensures that the final system will achieve the desired output within 12 time steps, and importantly, there should be no overshoots in the response. Considering the expression of θ in light of the preceding statement, the final poles will be real.
Consequently, the two parameters of the controller are $K_p = 1.13$ and $K_I = 0.98$.

2.2.3. PI performances and limitations

To test the behavior of the closed-loop system, the team conducts an experiment using three different file sizes, with the goal of maintaining the file server's load at a constant level of three. However, their testing objective is not solely to assess the PI controller's ability to maintain the desired output; they also want to observe its interaction with the HP jobs. Then a step disturbance is designed which loads the file server of 2 units. As mentioned earlier, the HP jobs take absolute priority over the BE, and when they arrive (in this case, around 2000 s), the controller has to reduce the number of jobs sent to the file server to avoid overloading it. This event led to a significant overshoot in the system. Once the HP jobs are completed, the controller has to adjust and start sending a slightly higher number of jobs again, resulting in an undershoot.

In Figure 2.10, it can be observed that the final behavior depends on the file size scenario. The center scenario appears to be the most favorable, as the control is accurately tuned to the true file size of the tasks. However, in the other two scenarios, there seem to be different issues.
When dealing with a file size of 50 MByte, the PI controller is tuned with a weight that is twice the actual, leading to a slower settling time than expected. Conversely, when the controller is tuned with a weight (f) that is half the true one, there is a speed-up in the system but with a drawback of an overshoot at the start.
The primary limitation of this approach stems from the variability in outcomes, contingent upon the disparity between the current state of f and the one implemented.
Furthermore, an additional constraint arises from the fact that the evaluation of $\tilde{\tilde{b}}$ relies on the specific cluster. As a result, the constructed controller not only adheres to a predetermined file size of 100 MByte but is also tailored to a specific cluster choice.

The objective of this study is to devise a more versatile controller solution capable of accommodating different f and various clusters, while yielding equivalent outcomes.

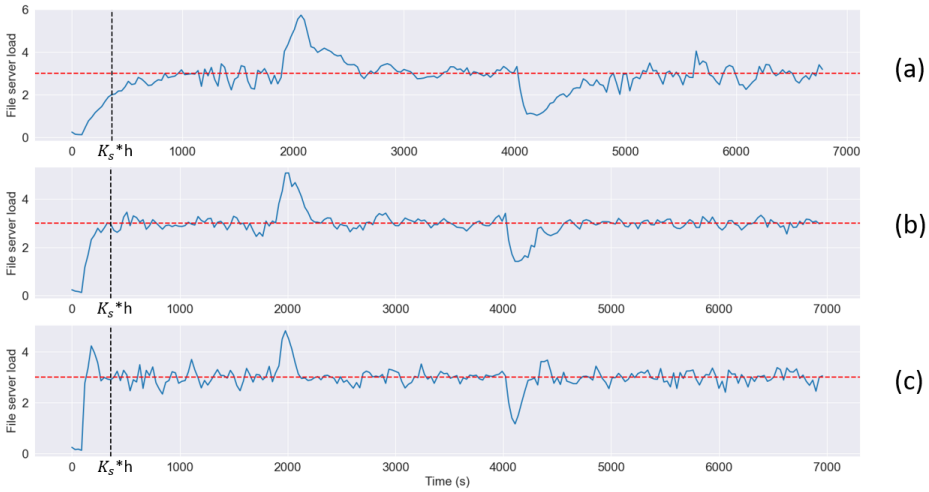

Figure 2.10: Average of five different experiments with three different file size and a step disturbance equal to 2. The dashed red line shows the reference and the dashed black line is the settling time (h is the discretization time equal to 30s). (a) Case of f = 50MByte (b) Case of f = 100MByte (c) Case of f = 200MByte

3 | Adaptive Control Design

Up to this point, our examination of the file system process has revealed the influence of two distinct factors: the quantity of BE jobs dispatched by CIGRI and the file size (f) associated with each individual task. Because only the number of BE jobs is controllable then, only this quantity is considered as input of the system.
The developed controller, designed by selecting a nominal value for f, exhibited divergent outcomes based on varying operational conditions, as elucidated in Chapter 2.2.3. For instance, when the actual f was halved from its nominal value, an observable deceleration in system performance was detected. On the other hand, when the actual file size was doubled, an initial overshoot behavior emerged, despite the team's initial design intentions to mitigate such overshooting tendencies.
In light of these observations, the primary objective becomes the formulation of an algorithm capable of dynamically adjusting the parameters K_p and K_I in response to changing operational conditions. Such a strategy aligns with the principles of *Adaptive Control* [46].

Next, we will explore Adaptive Control, with a particular focus on the Self-tuning technique and its fundamental principles. This chapter will conclude by delving into three distinct algorithms, along with their respective advantages and disadvantages.

3.1. Adaptive Control

The field of Adaptive Control addresses a critical challenge in designing control systems for situations where system parameters are either unknown or subject to changes over time. This approach offers a means to leverage system data effectively, allowing control systems to automatically adapt and meet desired conditions, even in the presence of varying process parameters. In addition, adaptive control proves beneficial in scenarios where there is a significant simplification of the system model [47, 48, 49].
Adaptive controllers can be categorized into two main types: *direct* and *indirect*. The distinction between these lies in the primary goal of the estimation process. Direct adaptive control focuses on estimating the most suitable parameters for the controller itself. In contrast, indirect adaptive control involves estimating the parameters of the underlying

process and subsequently utilizing this information to update the controller.
Considering that the controller relies on the process parameter denoted by b, the algorithm will be formulated to estimate this parameter of the system. Subsequently, we will delve into a discussion about an indirect adaptive approach.
Additionally, it is crucial for the algorithm to perform online parameter estimation due to the dynamic variations in the parameter f.

In the light of the aforementioned prerequisites, we endeavored to configure a *Self-Tuning* controller for the system. This pursuit aims to seamlessly integrate the parameter estimation process into the control framework, addressing the dynamic nature of the parameter b.

3.1.1. Self-Tuning

The concept of Self-Tuning was initially introduced by Astrom and Wittenmark in their seminal work [50]. In their article, they associated Self-Tuning with a crucial property wherein the controller parameters converge to the theoretically designed controller values if the process were fully known. Remarkably, they demonstrated that such convergence could still occur even if the estimated values differ from the actual values of the process. This approach allows the controller to adapt and optimize its performance based on real-time parameter estimations, enabling it to maintain effective control even in the presence of uncertainties and variations in the system. The utilization of the identification algorithm provides a means to continuously monitor and estimate the changing process parameters, ensuring the controller's responsiveness to dynamic environments and improving overall system performance.
However, one significant limitation of this technique lies in the assumption that the estimated parameters are entirely accurate and equivalent to the true values of the process. This assumption is known as the *Certainty Equivalence Principle* (CEP). While the CEP facilitates the convergence of controller parameters to the theoretically optimal values, it relies on the assumption that the estimated parameters are error-free and perfectly represent the true underlying process. In practice, this assumption may not always hold true, leading to potential discrepancies between the actual system behavior and the estimated model.

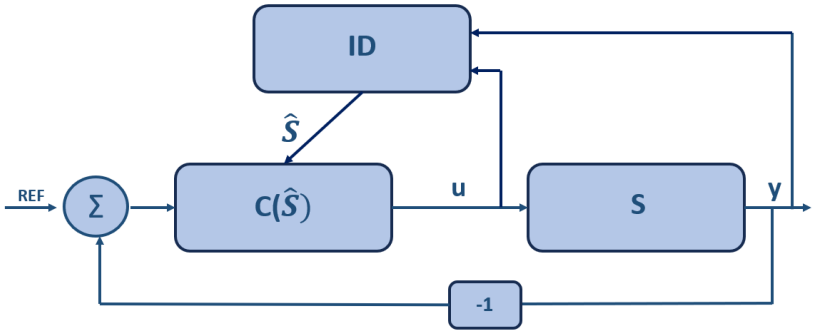

Figure 3.1: Classic scheme of self-tuning approach.

Figure 3.1 illustrates the various components involved in the self-tuning technique. Starting from the left, we have $C(\hat{S})$, where the controller relies solely on the estimated parameter of the system. This dependency on the estimated parameter is a direct consequence of CEP. The true process is represented by the variable S, while the identification algorithm (ID) operates in real-time to compute and determine the parameter values of the process ($\hat{S}$). These identified parameters are then transmitted to the controller, enabling it to update and adjust itself accordingly.

3.1.2. Fundamental Algorithm Framework: Basis for Subsequent Variations

Self-tuning regulators have been extensively studied, leading to the development of various algorithms. In this discussion, we will focus on a simpler algorithm that will be modified according to the needs. By starting with this straightforward approach, it will become easier to comprehend the underlying quantities and the associated issues. Throughout all the algorithms, we will consider θ as a vector that gathers all the unknown parameters. The simpler solution begins with a Least-Square Algorithm [51], which centers around minimizing the following cost function:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y(i) - \hat{y}(i|i-1,\theta))^2 \tag{3.1}$$

Where:

- N: number of data
- $y(i)$ measured output at instant i

- $\hat{y}(i|i-1,\theta)$= i-th estimated output by considering (i-1)-th data. Namely the formula is $\hat{y}(i|i-1,\theta) = \phi(i-1)'\cdot\theta$. Where ϕ is the *regression vector*, it collects the output and input of the system. For example being $\theta = [a,b]'$ then $\phi(i-1) = [y(i-1), u(i-d)]'$ where d is the delay.

To minimize the above const function, the derivative respect θ is computed and forced equal to zero. In this way, the optimal θ is discovered.

$$\hat{\theta} = \left[\sum_{i=1}^{N} \phi(i-1)\phi(i-1)'\right]^{-1} \cdot \sum_{i=1}^{N} \phi(i-1)y(i) \tag{3.2}$$

Tackling the singularity issue

Unfortunately, the above expression cannot be used due to the unpredictability of the information matrix ϕ, which could approach singularity and lead to large spikes in the estimated parameters. As a result, an alternative cost function is utilized:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y(i) - \hat{y}(i|i-1,\theta))^2 + (\theta-\bar{\theta})'\bar{S}(\theta-\bar{\theta}) \tag{3.3}$$

Where $\bar{\theta}$ is the initial guess for the parameters while $\bar{S}$ is a positive definite and non-singular matrix, indicating the level of confidence or trust in this initial guess. A larger $\bar{S}$ implies a higher level of trust in the initial parameter estimate. Then the new formula for $\hat{\theta}$ is:

$$\hat{\theta} = \left[\bar{S} + \sum_{i=1}^{N} \phi(i-1)\phi(i-1)'\right]^{-1} \cdot \left[\bar{S}\bar{\theta} + \sum_{i=1}^{N} \phi(i-1)y(i)\right] \tag{3.4}$$

Solving the memory and computational issue

Now that we have addressed the singularity issue and obtained a well-defined $\hat{\theta}$ using the updated formula 3.4, the data is acquired sequentially in real-time. For efficiency in memory allocation and computational processing, it is desirable to perform the computations recursively. This way, the result at time step $i-1$ can be used to obtain the estimates at time step i. In this context, it makes sense to replace the variable N with t. The change from N to t is reasonable because the Self-tuning approach is an online process where the algorithm computes the estimates at each time step. Therefore, at each instant, the algorithm repeats the computation, and the number of data points considered

in the Least-Square estimation is equal to the number of iterations of the algorithm, which corresponds to the current time step t. Afterward, we need to re-define the matrix S:

$$S(t) = \bar{S} + \sum_{i=1}^{t} \phi(i-1)\phi(i-1)' \tag{3.5}$$

$$S(t+1) = S(t) + \phi(t)\phi(t)' \tag{3.6}$$

After this modification, this matrix can be called *information matrix*, because inside has the ϕ vector, which brings the new informations. The updated optimal parameter will be:

$$\hat{\theta}_{t+1} = \hat{\theta}_t + S(t+1)^{-1}\phi(t)(y(t+1) - \phi(t)'\hat{\theta}_t) \tag{3.7}$$

To simplify the computation of $S(t+1)^{-1}$, we use a lemma from linear algebra found in Chapter 2 of [52]. This lemma enables us to obtain $S(t+1)^{-1}$ from $S(t)^{-1}$, facilitating the recursive update. Additionally, a new variable, denoted as $V(t) = S(t)^{-1}$, is introduced and referred to as the *covariance matrix*. By introducing this new variable, the algorithm can be further streamlined. The updated algorithm is as follows:

$$V(t+1) = V(t) - \frac{V(t)\phi(t)\phi(t)'V(t)}{1 + \phi(t)'V(t)\phi(t)} \tag{3.8}$$

$$\hat{\theta}_{t+1} = \hat{\theta}_t + V(t+1)\phi(t)(y(t+1) - \phi(t)'\hat{\theta}_t) \tag{3.9}$$

Before we proceed further, we would like to add some comments to the expression 3.9. In this expression, the estimated parameter at time step $t+1$ is computed based on the parameter estimate at the previous time step, $\hat{\theta}_t$, with an updating term. This updating term represents the difference between the measured output $y(t+1)$ and the estimated output $\hat{y}(t+1)$, namely the prediction error ϵ, weighted by the product of the covariance matrix and the regression vector.

Forgetting factor for time-varying parameters

When dealing with time-varying parameters, it becomes essential to account for the evolving nature of the system and give more weight to recent data while gradually forgetting older data. To achieve this, a *forgetting factor*, denoted as μ, is introduced [53]. The forgetting factor varies between 0 and 1, and when it is equal to 1, the formula presented earlier is recovered.

$$S(t+1) = \mu \cdot S(t) + \phi(t)\phi(t)' \tag{3.10}$$

$$V(t+1) = \frac{V(t)}{\mu} - \frac{V(t)\phi(t)\phi(t)'V(t)}{1+\phi(t)'V(t)\phi(t)} \cdot \frac{1}{\mu^2} \tag{3.11}$$

When μ is closer to 1, the algorithm becomes slower to follow time-varying parameters, as it gives more weight to historical data and is less sensitive to rapid changes. However, this approach also makes the algorithm less susceptible to noise, as it considers a larger amount of data. On the other hand, with a smaller value of μ, the algorithm becomes more responsive to fast-varying parameters, enabling it to track rapid changes in the system more effectively. However, a smaller μ also means that noise has a more significant impact on the parameter estimates, potentially leading to less accurate results.

The singularity problem strikes back

Despite the benefits of introducing the forgetting factor, a potential concern arises with the latter formula. The inclusion of μ may reintroduce the singularity issue, leading to the occurrence of *bursting* or *blow-up* phenomena in the control system [54]. The singularity problem manifests as sudden, large spikes in the estimated parameters, causing instability in the control performance.

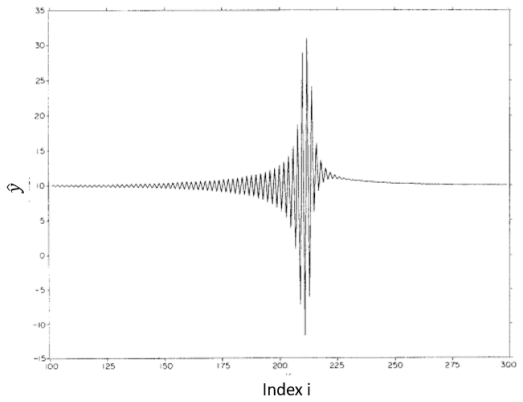

Figure 3.2: Effect of the bursting phenomena in $\hat{y}$ [54].

In this scenario, the bursting phenomena occur because the system lacks sufficient new information, leading to insufficient excitation of the system. When the system does not receive enough new data, the estimation process may become less accurate and prone to large spikes in the estimated parameters. Indeed it is usual to identify self-tuning as an approach that needs persistent excitation. To mitigate the bursting phenomena, various approaches can be implemented. For instance, the *constant trace algorithm* can be utilized to ensure that the covariance matrix, denoted as V, does not approach infinity. By limiting the growth of the covariance matrix, the algorithm remains more stable and avoids the issue of unbounded growth, which can lead to bursting behavior. Another method known as *directional forgetting* can be employed, where the forgetting factor μ is not applied uniformly to all elements of the covariance matrix. By selectively applying μ to specific elements of the matrix, the algorithm can focus more on relevant information, reducing the impact of noise or outliers. Other techniques and algorithms are also available to address the bursting phenomena, and these are discussed in detail in the work of P. Navratil and al. [55].

3.2. Exploring Algorithmic Approaches: A Triad of Implementations

In this section, we show three distinct algorithms aimed at estimating the parameter b. Since our focus is on estimation, the symbol $\hat{b}$ will denote the estimated value. In this initial exposition of the algorithms, we will disregard the influence of the HP disturbance. The mathematical representation of the process model is given by:

$$y(k+1) = \xi y(k) + bu(k) + e(k) \tag{3.12}$$

Upon inspection, a novel variable emerges within the model, signifying the noise, e, originating from the sensor due to external processes external to the file system. The research team has conducted an assessment of this noise and has determined that it conforms to a white noise with standard deviation of 0.59.
The estimated model takes the following form:

$$\hat{y}(k+1) = \xi y(k) + \hat{b}(k)u(k) \tag{3.13}$$

Given that there is just a singular parameter to be estimated,then the symbol ϕ represents a scalar, specifically equal to u, and correspondingly, V is a scalar as well.

3.2.1. Parallel Estimation

Before delving into the code and explaining the reasons for choosing this algorithm, it is important to clarify the concept of *Parallel Estimation* [52]. In computer science, parallel computation typically involves splitting a job into different resources. However, in this context, parallel computing refers to performing the same computation simultaneously with slightly different conditions. Having explained the concept of parallel estimation in this context, let's now explore the rationale behind selecting this particular algorithm.
The chosen algorithm employs a recursive least-squares estimator with variable forgetting factor. At each step, a μ is chosen among different values to minimize the discrepancy between the measured output and the estimated output. As a result, this algorithm is capable of effectively tracking both fast and slow dynamics, and it prevents the occurrence of the blowing-up phenomena. A close experiment is shown in [56].
Following the MATLAB code:

---

function $[\hat{b}_o, V_o]$ = parallel $(y, y_b, \phi, \hat{b}_i, V_i, \xi)$
define forget.factor, n=length(forget.factor)
for w=1:n
$\mu = forget.factor(w)$
$V_{vector}(w)$ =computed with 3.11
$\hat{b}_{vector}(w)$ = computed with 3.9
$residual(w) = abs(y - \xi y_b - \hat{b}_{vector}(w)\phi)$
$pos = find.minimumPosition(residual)$
$\hat{b}_o = \hat{b}_{vector}(pos)$, $V_o = V_{vector}(pos)$

One can see that the algorithm takes as input

- y: measured output
- y_b: measured output of the time step before
- ϕ: regression matrix, in this case is equal to u
- $\hat{b}_i$: is the estimated parameter of time step before. In case of inizialization is the starting condition chosen by the user.
- V_i: is the covariance value of the previus time step. In case of inizialization is the starting condition chosen by the user.
- ξ: is the value estimated from the filter with equation 2.2.1

The following algorithm, utilizing the function $find.minimumPosition$, serves to determine the position of the smallest residual within the vector. Subsequently, it strategically selects the best values for μ, consequently also the value for $\hat{b}$ and V.
A pivotal aspect of this first algorithm lies in its ability to circumvent the imposition of a fixed forgetting factor. Indeed, if $\hat{y}$ exhibits pronounced oscillations for the approaching of the bursting phenomena, the algorithm dynamically adjusts the forgetting factor to 1, thereby arresting the approach towards singularity. Simultaneously, in response to changes in f, the algorithm selects a smaller value of μ.
To sum up, this technique estimates a time-varying parameter and self-adapts to its changing rate.

3.2.2. Robust Estimation

For this second algorithm, we are taking a step back from the traditional least-squares approach. Up until now, we have been discussing a least-square algorithm, which assumes that the noise is small. However, when dealing with significant errors, the impact on $\hat{b}$ can be substantial due to the squared term in the cost function. To achieve a *Robust Estimation* [52], we need to modify the prediction error in formula 3.9. Instead of plug in the difference between the measured output and the estimated one, we will introduce a function that behaves differently for different prediction error (ϵ) magnitudes. Specifically, for larger errors, the function will be more gradual, deviating from linearity, while for smaller errors, the function will resemble a linear one. This modification allows the algorithm to handle larger errors more effectively, making it more robust in the presence of substantial noise. The specific function that we have chosen is:

$$f(\epsilon(k)) = \frac{\epsilon(k)}{1 + \alpha|\epsilon(k)|} \tag{3.14}$$

$$\epsilon(k+1) = y(k+1) - \xi y(k) - \hat{b}(k)u(k) \tag{3.15}$$

As consequence the expression 3.9 becomes:

$$\hat{b}(k+1) = \hat{b}(k) + V(k+1)\phi(k)f(\epsilon(k)) \tag{3.16}$$

In addition, the revised formula introduces a novel parameter denoted as α, which plays a pivotal role in the noise reduction process. The magnitude of α directly influences the extent of noise reduction achieved. It is important to note that as α increases, the degree of noise reduction becomes more pronounced. However, a significant increase in the value of α is accompanied by a notable deceleration of the algorithm's estimation. This phenomenon can be attributed to the fact that excessively high values of α not only mitigate the impact of noise but also diminish the algorithm's adaptability to varying working conditions. Additionally, one can note that the previous algorithm corresponds to the specific scenario where α is set to 0.
We would like to emphasize that the parallel estimation remains a fundamental aspect of the algorithm, but instead of the prediction error (ϵ), we now utilize the function $f(\epsilon)$ as outlined in the new formula 3.16.

3.2.3. Selective Memory Estimation

This latest algorithm introduces a profound departure from the previous two. Notably, a key innovation within this algorithm lies in its capability to turn off during periods without new information. The rationale behind incorporating this feature stems from the experimental context, where the parameter b is anticipated to remain constant over extended durations, punctuated by sudden and abrupt changes like a step function. Then, we used the *Selective Memory* algorithm [57]. This algorithm updates the estimated parameter only if new information is available. By doing so, it can efficiently learn from new data points while maintaining stability and reducing unnecessary updates when the system remains relatively unchanged. Implementing this algorithm in our experimental setup could provide the best of both worlds – fast learning and adaptability when needed, combined with stability and reduced oscillations in periods of stability. This approach appears promising for addressing the specific characteristics of our future implementation into the cluster, ensuring accurate and efficient parameter estimation.
Following the new algorithm split into several stages:

Step 0 : Defining $r(k)$ as an estimate of the variance of the unmodeled dynamics and M_0 as the memory length. Choose $r_0 > 0, \theta(0), V(0) > 0, 1 < M_0 < \inf$.
Set $r(0) = r_0, \sigma = 1 - \frac{1}{M_0}$ and then update k, the new value will be $k = 1$

Step 1 : Select a function for $r(k)$

$$r(k) = max[\sigma \times r(k-1) + (1-\sigma) \times \epsilon(k)^2, r_0]$$

Where $\epsilon(k) = y(k) - ay(k-1) - \hat{b}(k-1)\phi(k-1)$. Recall that $\phi(k-1) = u(k-1)$.

Step 2 : Set $B(k) = 0$ and

$$A(k) = \begin{cases} 1, & if \quad \frac{phi(k-1)'V(k-1)phi(k-1)}{r(k)} \geq 1/M_0 \\ 0, & otherwise \end{cases}$$

Step 3 : If $A(k) = 0$, set:

$$B(k) = \begin{cases} 1, & if \quad r(k) \geq max_{1 \leq i \leq k} r(k-i) \\ 0, & otherwise \end{cases}$$

Step 4 :

$$\hat{b}(k) = \hat{b}(k-1) + \Delta(k) \times \frac{V(k-1)\phi(k-1)\epsilon(k)}{r(k) + \phi(k-1)'V(k-1)\phi(k-1)} \tag{3.17}$$

Step 5 :

$$V(k) = V(k-1) - \Delta(k) \times \frac{V(k-1)\phi(k-1)\phi(k-1)'V(k-1)}{r(k) + \phi(k-1)'V(k-1)\phi(k-1)} \tag{3.18}$$

First of all, we would like to clarify that the covariance matrix $V(k)$ in this new algorithm is different from the previous ones. It does not include any forgetting factor, but instead, it incorporates an estimate of the variance in the denominator. This is because the algorithm utilizes a different information matrix called the Fisher information matrix, which is defined as follows:

$$S(k) = S(k-1) + \frac{\Delta(k)}{r(k)}\phi(k-1)\phi(k-1)' \tag{3.19}$$

With this intrinsic difference highlighted, let us shed light on the last two steps of the algorithm. The update process appears similar to the previous algorithms, but this time, the updating involves the parameter Δ. This parameter can only take values of 0 or 1, based on the outcomes of Step 2 and Step 3. The parameter will be updated only if Δ equals 1.

It is worth noting that this algorithm shares some similarities with the widely known *dead zone algorithm* [46], as both select data to use for updating the parameter. However, a key difference lies in the selection rule between the two approaches. The selective memory algorithm focuses on the degree of excitation (for parameter A), while the dead zone algorithm considers the prediction error, which is more sensitive to noise. Due to the noise in the system, the selective memory approach was preferred, as it handles noisy data more effectively and provides more reliable parameter estimates.

An additional aspect to consider for this algorithm is its initial condition. To ensure that the algorithm updates properly from the beginning, it's advisable to choose a larger initial covariance condition (greater than 100). Additionally, for robustness and optimal performance, it is recommended to set a small value for r_0 and make M_0 large, preferably in the range of 100 to 1000. This approach will help ensure both the reliability and effectiveness of the algorithm.

Nevertheless, the selective memory algorithm has a limitation—it may not be suitable for long-run adaptations as the parameter may converge and stop updating. To address this limitation, a re-start mechanism is implemented in the final code. When there is a modification of f, the algorithm is restarted to ensure it can adapt effectively to changes in the system behavior. This re-start strategy is feasible in the real system, where the condition for re-starting the algorithm would be the launch of a new campaign (as defined

in Chapter 2.1.1).

---

reinizialization function define $V(0)$
if $new\_campaign = true$
$V(i) = V(0)$

3.3. Explicitly taking into account the HP disturbance

In the previous section, we focused on the scenario where the file server contained only BE jobs. The new section investigates a more complex setup where the file system is shared between both BE and HP jobs.
As discussed in the first chapter, these jobs follow a preemptive priority rule. In addition, it is important to note that BE and HP jobs have different impacts on the file server, with HP jobs being more resource-intensive by nature. Consequently, we introduce two distinct coefficients: b_u, which represents the weight of a single BE task, and b_d, representing the weight of a single HP job into the file system. For the nature of the system, HP tasks are considered as a disturb because are beyond our control, and $d(k)$ is the number of HP tasks in the system at time instant k.
The final process is described by the following equation:

$$y(k+1) = \xi y(k) + b_u u(k) + b_d d(k) + e(k) \tag{3.20}$$

Due to the incorporation of two parameters instead of just one, there will be a consequential alteration in the dimensions of the key quantities within the algorithm. Specifically, the parameter that was initially identified will now transform into a vector encompassing two distinct parameters denoted as b_u and b_d. Consequently, the covariance matrix will adopt a square 2x2 structure, where the values for these two parameters will be situated along the main diagonal. Correspondingly, the regression matrix will evolve into a two-element vector: the first element representing the control input, and the second element representing the disturbance (referred to as HP jobs).
Another distinction arises due to the inherent dissimilarity between the two parameters, b_u and b_d. Specifically, the framework operates under the premise that the influence of HP tasks on the file server's workload remains consistently uniform. This constancy produces b_d as a parameter that remains invariant over time. Consequently, the integration of a

forgetting factor becomes unnecessary for b_d, as each data point must exert an identical influence on the estimation process, aligned with the parameter's unchanging nature. Conversely, for the parameter b_u, the scenario is distinct. In summary, the methodology involves two separate estimations: one that integrates a forgetting factor for b_u to accommodate its adaptive characteristics and another that excludes a forgetting factor for b_d due to its stable, time-invariant quality.
In the end the final estimated output in case of HP disturbance will be:

$$\hat{y}(k+1) = \xi y(k) + \hat{b}_u u(k) + \hat{b}_d d(k) \tag{3.21}$$

3.3.1. Limitation of the double estimation

It is important to emphasize that we will be directing the algorithm to estimate two parameters utilizing nearly identical data. However, it is anticipated that this adjustment might lead to a deterioration in the results. This is because estimating a greater number of parameters necessitates a higher degree of system excitation, which might not be fully accommodated by our system that is constrained by a constant reference.
Although we extensively examined the modifications required to adapt various algorithms for generating dual estimations and subsequently simulated their outcomes, optimizing the benefits of this dual estimation necessitates a controller adjustment that incorporates $\hat{b}_d$. Presently, the controller exclusively depends on $\hat{b}_u$, diminishing the significance of $\hat{b}_d$ estimation from the controller's perspective. This configuration could potentially complicate the algorithm's ability to effectively discern between the two identifications. This challenge becomes more pronounced when dealing with algorithms characterized by continuous learning patterns as the Parallel and the Robust.

4 | Validation of Algorithms

This chapter is dedicated to assessing the impact of the three previously discussed algorithms on the system. To gain insight into their performance characteristics, these algorithms underwent preliminary testing within a simulation framework. This allowed us to discern the strengths and weaknesses of each algorithm. Subsequently, we proceeded to evaluate these algorithms within the actual system. It is important to note that, for reasons explained in the following sections, only one algorithm will be discussed in the real system.
Within the second part of this chapter, we address the selection of optimal initial conditions, followed by a validation process with f which changes runtime. Finally, we will conclude with a comparative analysis between the PI and the Adaptive PI, considering various values of f.

4.1. Validation in Simulation

For the simulation frame all the system and algorithms were established inside the Simulink frame with a discrete solver with a constant step equal to 30 seconds. Each simulation represents the outcome derived from an average of five experiments. This approach is adopted to mitigate the influence of system noise. The initial phase involves tuning the algorithm to ensure a consistent basis for result comparison. Subsequently, a comparative analysis of output will be conducted under two distinct conditions: one without HP jobs, and the other involving the presence of HP jobs.

4.1.1. Choice of parameters: α

In order to facilitate a comparative analysis of various algorithms within the Simulink framework, one has to commence by fine-tuning the algorithm's parameters. In this context, the focus shall primarily be on refining the smoothing parameter α of the Robust Algorithm. This decision stems from the fact that the remaining parameters exhibit negligible influence on outcomes within the simulation environment. However, it is worth noting that in the experimental configuration, the results are intricately tied to the se-

lection of these two values. In this case, the specific choices for these parameters were established as $\hat{b}(0) = 0.03$ and $V(0) = 100$. Notably, the initial covariance condition was set to a relatively elevated value. This choice was made due to the inverse relationship between the magnitude of this value and the reliability of the initial condition of $\hat{b}$. While the forgetting factor is [0.5,0.6,0.7,0.8,0.9,1]', we select a minimum of 0.5 due to the noise of the system.
To choose the best possible α, a dedicated experiment is devised. This experimental design involved maintaining a constant reference of three for the output, coupled with the introduction of a time-varying b. To closely emulate the forthcoming experimental setup, a step function was adopted. This choice was deliberate, aligning with the experimental procedure where adjustments to parameter b necessitate the initiation of separate job campaigns with distinct f values. The step function was deemed appropriate due to its congruence with this process. Furthermore, the included visual depiction is an aggregate of outcomes from five distinct experiments. This consolidation was necessary in light of inherent sensor noise.

Subsequently, the resulting outcomes of the Robust Algorithm are depicted in the following pictures, showcasing the effects of varying α values in the range of 0.5 to 3, with increments of 0.5. The blue line represents the behavior of $\hat{b}$, while the red dashed line illustrates the behavior of $\tilde{\tilde{b}}$. An observable trend is that as α increases, oscillations within the estimation diminish. However, this improvement comes at the cost of a notable deceleration in the estimation process with higher values of the parameter. This is more clear in the final row, where a step-down change in the true parameter induces a significant gap between the red and blue lines. This gap signifies reduced agility of the algorithm in tracking the true time-varying parameter compared to the first row.

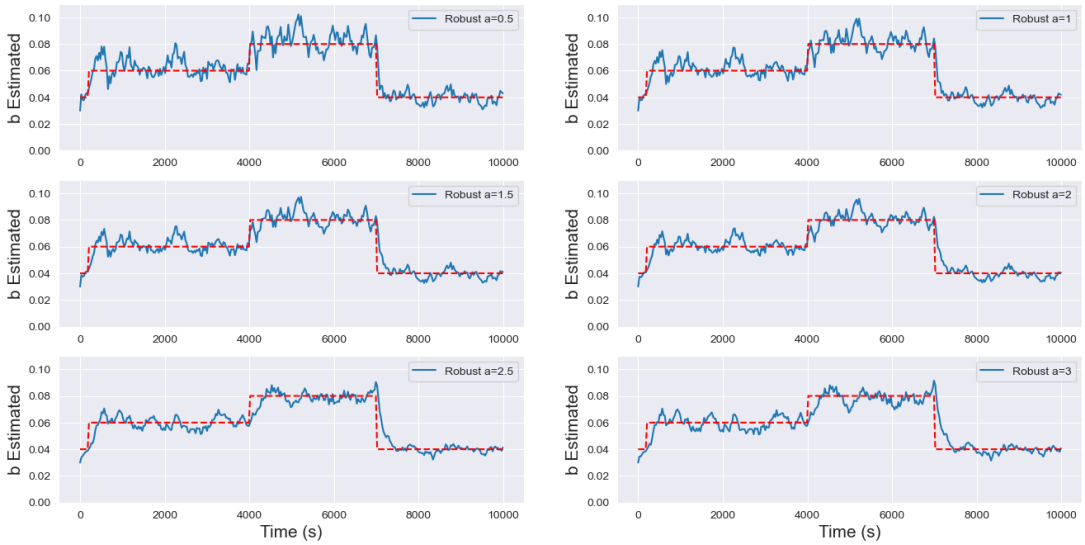

Figure 4.1: Identification results for varying α in the range of 0.5 to 3 with increments of 0.5. The blue line corresponds to $\hat{b}$, while the red dashed line corresponds to $\tilde{\tilde{b}}$.

On the other hand, the impact of α appears negligible in the tracking aspect. In this depiction, it might appear that the initial load of the file server approximates 1, but this is not the case. Despite the starting condition being 0, the noise level appears pronounced. The choice of presenting the average of five experiments is deliberate, even though the noise isn't fully mitigated. This decision aligns with the intention to maintain consistency with the forthcoming real-world setup.

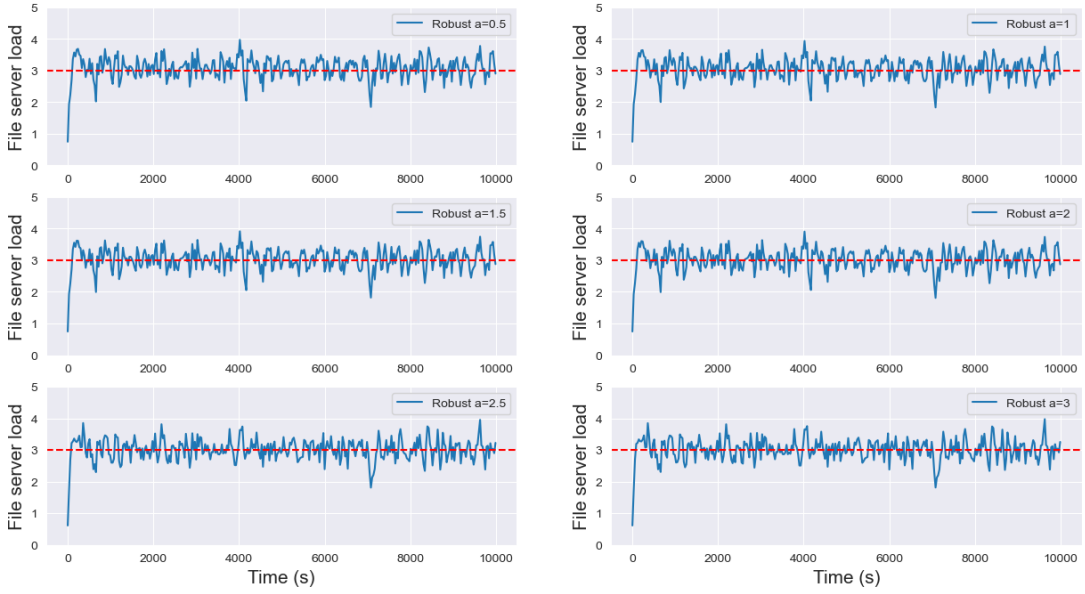

Figure 4.2: Tracking results for varying α in the range of 0.5 to 3 with increments of 0.5. The blue line corresponds to y, while the red dashed line corresponds to y_{ref}.

To gain a more insightful understanding of the impact of α on the performances, we computed boxplots to depict the relative percentage error in absolute value for both the identification error $(\tilde{\tilde{b}} - \hat{b})$ and the tracking error. The boxplots in Figure 4.3 provide a visual representation of the distribution of the results and allow us to compare the performance with different values of α. Upon analyzing picture (a), one can note that the error decreases by increasing the parameter α, however, in the case of $\alpha = 3$, it appears that the boxplot starts to expand attributable to the slowdown. On the other hand, picture (b) in 4.3 confirms the idea that α does not have a strong impact on the output. Taking into account only the boxplot result, we find the results for α values above and equal 1.5 to be particularly satisfactory, indeed 75% of data have a relative percentage error of 10% or less. However looking at Figure 4.1 it is clear that the last two cases present a slowdown that cannot be ignored. Considering the trade-off between relative error and slowdown, we would recommend selecting $\alpha = 2$ as it seems to provide a favorable compromise between smoothness and adaptability to time-varying parameters.

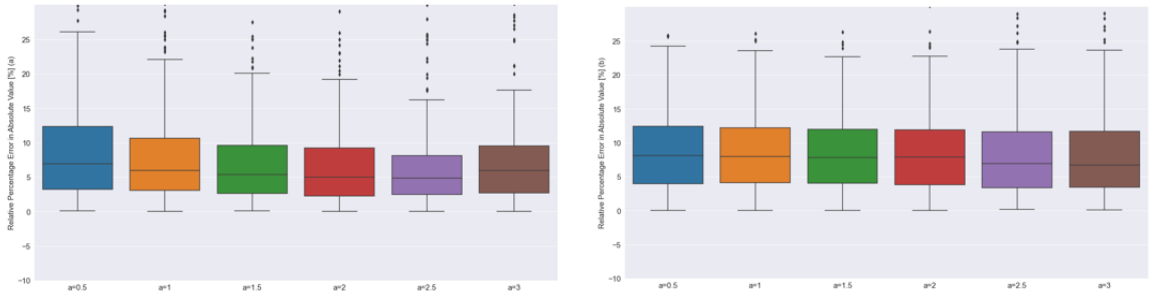

Figure 4.3: Boxplots depicting the relative percentage error in absolute value across various α values. (a) Shows the identification error, while (b) shows the tracking error.

4.1.2. Comparison of algorithms

In the preceding section, we examined the diverse outcomes associated with varying values of α. Having established our choice of alpha, we can now proceed to observe the distinct results achieved by applying the three algorithms. The Parallel and Robust Algorithms have as $\hat{b}(0) = 0.03$, $V(0) = 100$, $\mu = [0.5, 0.6, 0.7, 0.8, 0.9, 1]'$ while the initial conditions for the Selective algorithm are $r(0) = 10^{-3}$, $M_0 = 100$, $\hat{b}(0) = 0.03$ and $V(0) = 100$, where values of the Selective are the ones suggested in [57]. Illustrated in Figure 4.5, we present the identification results for three specific cases. The leftmost plot corresponds to the Parallel Algorithm, the middle plot represents the Robust Algorithm with $\alpha = 2$, and the final plot showcases the outcome of the Selective Algorithm. A noticeable trend emerges from left to right: a consistent reduction in oscillations across all cases. This

reduction is a direct consequence of the implementation of the Robust algorithm, tailored to mitigate the influence of noise through the adjustment of the parameter α. Conversely, the Selective algorithm incorporates a memory mechanism that restricts learning over a finite period, which explains the observed stabilization of the identification results beyond a certain time frame. The presence of overshooting and undershooting phenomena can be attributed to the algorithm's re-initialization process.
Instead of tracking there seem to be no noticeable improvements, probably because the impact of the noise is bigger than the one of $\hat{b}$.

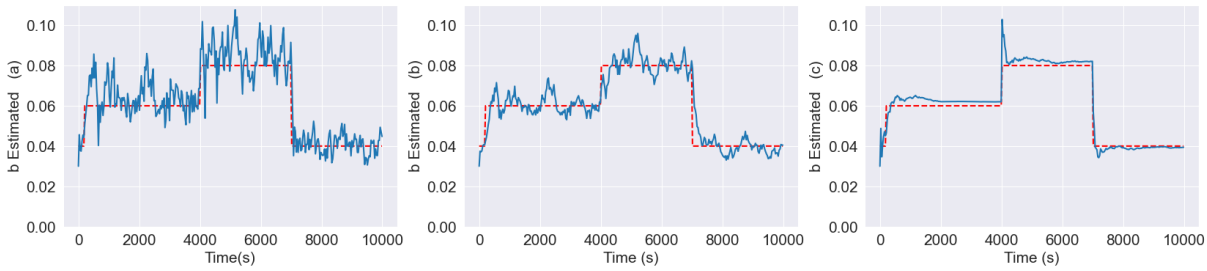

Figure 4.4: Identification results. $\hat{b}$ is shown in blue line, and $\tilde{\tilde{b}}$ in red dashed line. Figure (a) depicts results for the Parallel algorithm, (b) for the Robust, and (c) for the Selective Algorithm.

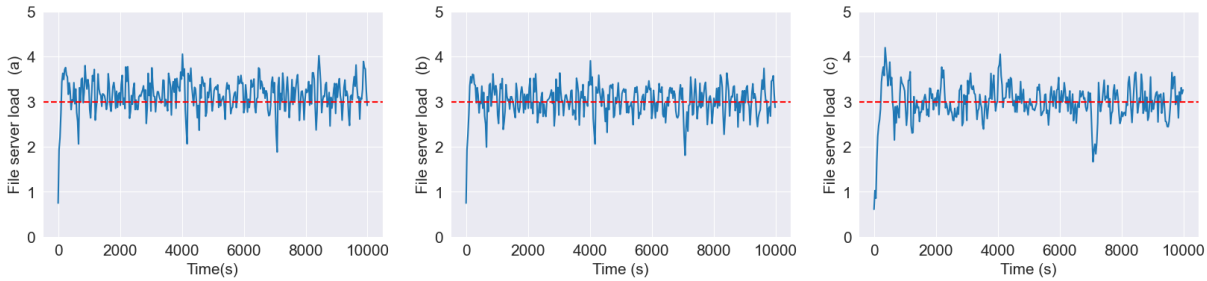

Figure 4.5: Tracking results. y is shown in blue line, and y_{ref} in red dashed line. Figure (a) depicts results for the Parallel algorithm, (b) for the Robust, and (c) for the Selective Algorithm.

To substantiate the efficacy of the Selective Algorithm and ascertain potential improvements in terms of tracking performance, we have incorporated three distinct box plots for the identification and the tracking error into the analysis. These box plots aim to offer valuable insights into the distribution of relative percentage errors in absolute values.

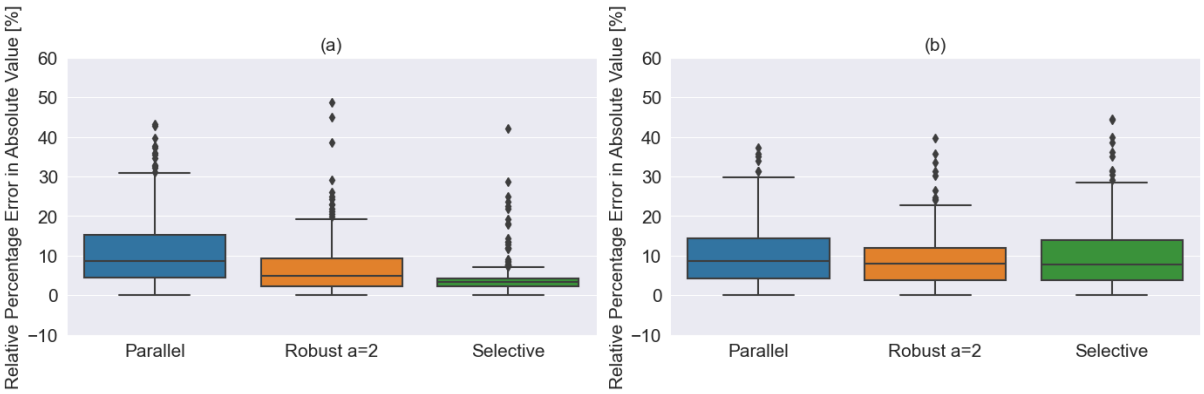

Figure 4.6: Each boxplot represents the relative percentage error in absolute value for each algorithm. (a) Represents the identification error, and (b) represents the tracking error.

By incorporating this statistical visualization, we can delve more deeply into the performance of the algorithms. The results presented in (a) of Figure 4.6 provide strong evidence supporting our assertion that the final algorithm we present is optimal in terms of identification capabilities. This assertion is reinforced by the significant reduction in error observed when transitioning from the Parallel Algorithm to the Selective Algorithm. To elaborate, the initial algorithm reveals that 75% of the data points exhibit a relative percentage error below 15%. However, with the Selective Algorithm, an equivalent proportion of data points exhibit errors below 5%. This distinct shift in the distribution of errors serves to underscore the markedly superior performance of the Selective Algorithm from the identification standpoint.
On the contrary, in graph (b) of Figure 4.6, a significant shift occurred. The Selective Algorithm, which previously occupied the first position, has now been superseded by the Robust Algorithm. In fact, looking at the error values in the graph (b) of figure 4.6, one can see that the Robust error values are smaller than the other algorithms. It is worth noting that performance disparities among algorithms are not as strong as in the case of Figure (a). This suggests that the algorithms show relatively similar levels of tracking performance, with less pronounced differences between them. The similarity in tracking errors among the results can be attributed to the findings presented in the tuning section. This analysis revealed that the influence of noise surpasses the impact of $\hat{b}$ in the controller. To gain deeper insights into the role of identification within the output, further simulations are warranted. In this work, we have selected a set of five experiments to gain consistency with the forthcoming experimental setup section.

Ultimately, considering the outcomes, one can note that the Selective Algorithm achieves

superior results in the identification phase compared to the other algorithms. However, in the context of tracking, the algorithm for achieving the most favorable outcomes remains uncertain. Given these observations, we are inclined to conclude that the Selective algorithm stands as the most favorable choice among the three.

4.1.3. Case with HP disturbance

In this section, the algorithms are tasked with evaluating two distinct parameters, which have been elucidated in chapter 3.3 - b_u and b_d. Consequently, the experimental setup in this scenario differs slightly from the other case. Specifically, we have extended the simulation duration to provide ample time for disturbance setup. Notably, while the profile of the variation in b_u remains consistent, the approach for b_d is different, as it remains constant at 0.30. To implement this, we designated the disturbance to manifest as an abrupt step change at 2000 seconds, persisting until 4000 seconds.
The visual representation in Figure 4.8 may suggest a continuous presence of the disturbance. This is due to the algorithm's characteristic behavior, which retains its previous computation even after the disturbance ceases. This perspective is pivotal in understanding the identification of b_d. The algorithm preserves its last computed value, leading to the red dashed line's persistence at the end of the disturbance interval.

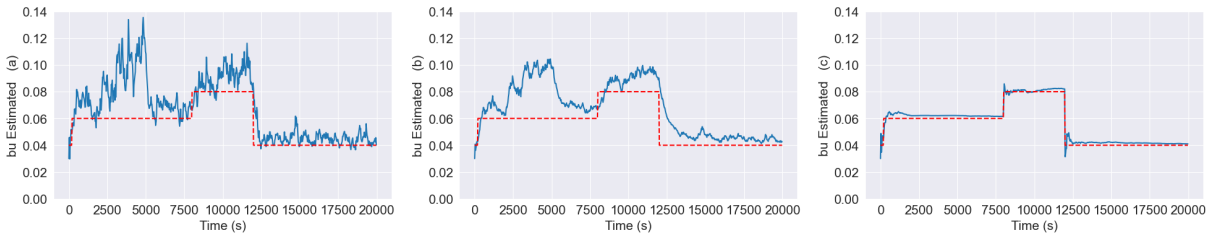

Figure 4.7: The three algorithms' identification results: $\hat{b}_u$ is shown in blue line, and b_u in red dashed line. Figure (a) depicts results for the Parallel Algorithm, (b) for the Robust, and (c) for the Selective Algorithm.

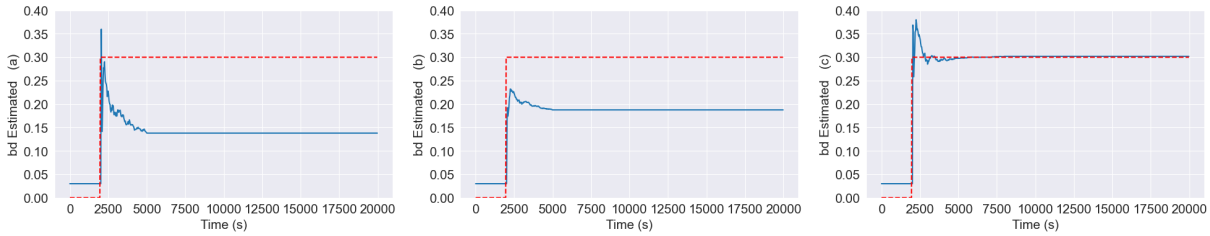

Figure 4.8: The three algorithms' identification results are displayed: $\hat{b}_d$ is shown in blue line, and b_d in red dashed line. Figure (a) depicts results for the Parallel Algorithm, (b) for the Robust, and (c) for the Selective Algorithm.

In Figure 4.7, the obtained result closely aligns with the preceding one. As depicted from left to right, a discernible reduction in oscillation becomes evident, and the Selective Algorithm maintains its prominent position. Furthermore, this scenario illuminates the influence of the disturbance on the calculation of b_u. Both the Parallel and Robust Algorithms fail to completely segregate the two calculations. Consequently, upon the disturbance's introduction, a perturbation emerges in the computation of $\hat{b}_u$ even in the absence of alterations to this parameter. This issue subsequently ripples into the determination of $\hat{b}_d$ as well. Specifically, in Figures (a) and (b) of 4.8, one can readily observe the underestimation of the actual parameter. This phenomenon appears to arise from the overestimation of b_u coinciding with the arrival of the disturbance. The updating depends on the disparity between the measured and estimated output, as depicted by:

$$\epsilon(k+1) = y(k+1) - \xi y(k) - \hat{b}_u(k)u(k) - \hat{b}_d(k)d(k) \tag{4.1}$$

Then, an overestimation of $\hat{b}_u$ precipitates an inherent underestimation of $\hat{b}_d$. Furthermore, this overestimation could be attributed to the influence of $\hat{b}_u$ within the controller's framework. The rationale lies in the fact that elevating $\hat{b}_u$ leads to a corresponding reduction in the control action—an essential adjustment considering the file server's evolving responsibility of handling both HP and BE tasks. Conversely, the parameter $\hat{b}_d$ wields no influence over the control process.
In the context of Figure (c) within 4.7, the timing of disturbance introduction becomes less apparent. This outcome stems from the distinct learning rate characteristic of the selective algorithm. Unlike the preceding algorithms that always refine their estimates, this algorithm exclusively learns upon activation, as stipulated by the triggering condition we have imposed. Given this condition, $\hat{b}_u$ remains static unless alterations occur within the BE campaign. As a result, the $\hat{b}_u$ remains unaffected by the arrival of HP jobs, effectively avoiding their influence.

In Figure 4.9, the tracking result is presented, superimposed with the impact of the HP jobs over time, indicated by the dashed black line. As the case of a single parameter, the distinction between the effects of different algorithms in the tracking is less evident. Furthermore, an observation of significance arises when examining the moment of disturbance introduction into the system; specifically, a heightened impact is noticeable upon the disturbance's arrival, in contrast to the case of its disappearance.

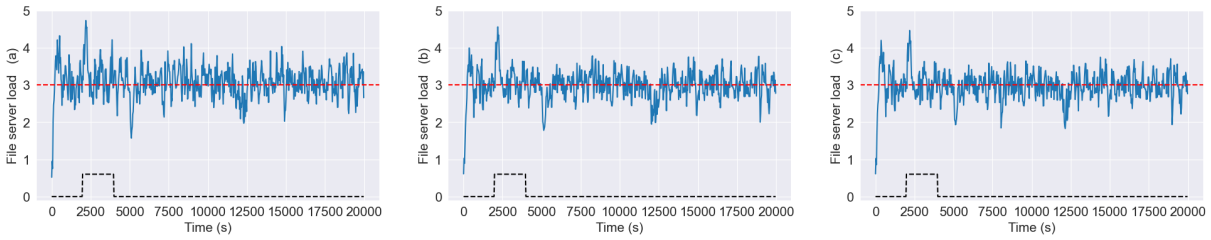

Figure 4.9: The three tracking results are displayed: y is shown in blue line, and y_{ref} in red dashed line, and the impact of HP jobs is depicted in black dashed line. Figure (a) depicts results for the Parallel Algorithm, (b) for the Robust, and (c) for the Selective Algorithm.

4.2. Evaluation with experiments on the real platform

Only the Robust Algorithm will be implemented on the actual platform. The decision to exclude the Parallel Algorithm from implementation is rooted in the insights provided in Chapter 3. As demonstrated, the Robust algorithm represents an evolution of the Parallel Algorithm, sharing a common underlying structure. However, the Robust Algorithm specifically addresses the limitation of noise impact.

Regarding the Selective Algorithm, its omission is not a matter of preference but rather a consequence of observed behavior misalignment with initial expectations. The system encounters a delay during initiation, a factor not accounted for in the original model. This delay poses a significant challenge for the Selective Algorithm due to its narrow learning window. As a result, the algorithm prematurely halts its learning process, yielding suboptimal estimations. Efforts to mitigate this issue through alterations in initial conditions were unsuccessful in resolving the problem.

Another deviation from the simulation phase arises due to time constraints, preventing the implementation and real-platform testing of estimations for the two distinct parameters. Consequently, the forthcoming sections will exclusively present the estimation of only one parameter. Nevertheless, the disturbance will be introduced following the same

methodology outlined in the previous section. It will be subsequently demonstrated that $\hat{b}$ serves the dual purpose of estimating the disturbance as well.

4.2.1. Experimental setup

In this master thesis, we focus on utilizing one of the clusters available at Nancy, specifically Grisou. Grisou comprises 46 nodes, each equipped with 2 CPU Intel Xeon E5-2630 v3. Each CPU contains 8 cores, resulting in a total of 16 cores per node [58]. For our experiments, we reserved 4 nodes, with each node serving a specific role: one for the controller CIGRI, one for the scheduler OAR, one for simulating the cluster itself, and the last one for simulating the file system. The experiments are conducted using the programming language *Ruby* on a Linux platform.
Due to the limitation on booking the entire cluster, we needed to find an alternative approach to conduct large-scale experiments. The inspiration for our solution comes from the article [59], which proposes the use of a *sleep mode*. With this approach, the cluster does not actively compute the jobs; instead, the jobs wait for a specific amount of time within the cluster. By doing so, the computing power of a single CPU can be efficiently utilized to host several jobs in sleep mode. This allows us to treat the single *physical resource* as multiple *virtual resources*. In our context, physical resources refer to the physical nodes, while virtual resources denote the number of resources from the perspective of OAR. The sleep mode duration is set to match the period of CIGRI for simplicity.

In this section, we intend to highlight the significant challenges associated with the actual platform. These challenges serve to provide rationale for conducting only five experiments for each simulation iteration. It is important to note that the execution time of a single experiment spans several hours. Consequently, limitations stemming from the user policy prevent us from conducting simulations in quantities of our choosing. Furthermore, each experiment is subject to pronounced heterogeneity due to variations in the utilized nodes and the scheduler employed. Specifically, the scheduler utilized in the experiments represents an updated version of the one currently deployed in the cluster. Regrettably, this scheduler component is prone to overload occurrences with insufficiently clear justifications, largely attributed to its *beta* version status which renders it susceptible to bugs.
Having established the simulation of the cluster, we now turn our attention to the simulation of the file server. In light of the main objective of this thesis - the overloading of the file server, in the experiments we use a real file server with some approximations. Indeed considering that the real bottleneck of this system is the writing task, in the experiment the fill of the file system will represent only this process. This is justified by the lighter

impact of the other two operations of the file system, namely reading and storing.

4.2.2. Initial parameter condition

Initial Condition for $\hat{b}$

Now that we have the opportunity to assess the algorithms within the actual experimental framework, we can also explore alternative optimal initial conditions. Our first objective is to determine the optimal $\hat{b}_0$ value. In order to structure the experiments, a singular campaign was devised, resulting in a fixed data volume of $f = 200$ MBytes. During the testing phase, three distinct initial conditions were employed: 0.5, 0.15, and 0.03. The selection of the initial condition of 0.5 facilitates a less aggressive control action, as both K_p and K_I are contingent on the inverse of the estimation. In the case of $\hat{b}(0) = 0.15$, we opted to assign a value equivalent to $\tilde{\tilde{b}}$. Lastly, for the third initial condition, we utilized the value previously employed in the simulation setup. This approach aims to investigate the impact of different initial conditions on the algorithm's performance within a real-world context.

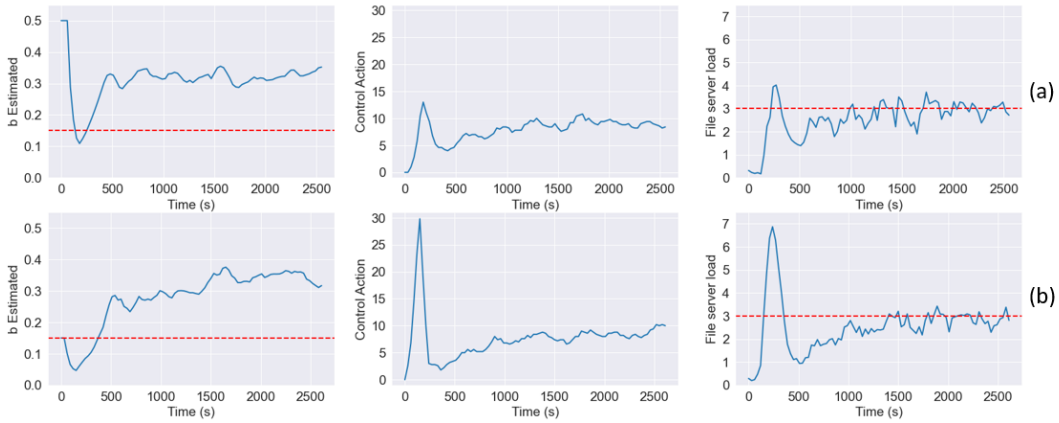

Figure 4.10: Results with different initial conditions. In row (a), we examine the scenario where $\hat{b}(0) = 0.5$, while in row (b), we observe the case where $\hat{b}(0) = 0.15$.

One can see from Figure 4.10 that the simulation comprises only two cases, deviating from the initially anticipated three simulations. This unexpected outcome arises due to the simulation halting when $\hat{b}(0) = 0.03$. This particular circumstance was not predicted in advance, as the existing model of the file server fails to account for the initial delay for the system to turn on.

As illustrated in Figure 4.11, the value of $\hat{b}$ initially decreases, mirroring the behavior observed in the other two experiments. However, due to the initial small value of $\hat{b}$, the control action rapidly escalates to an extremely high value, notably 100. This corresponds

to an instance where, within a single time step, the CIGRI system dispatches an excessive 100 BE jobs. Consequently, the file server load surpasses the value of 8 (where the overloading experience starts with the value of 7), when the reference value stands at three. At this juncture, a cascade of issues emerges. With the file server experiencing overload conditions, it initiates a self-preservation mechanism, resulting in system-wide deceleration. Consequently, the number of jobs in the waiting queue surges, further compounding the system's slowdown due to the scheduler's responsiveness being inversely proportional to the queue's length. Evidently, the value of $\hat{b}(0) = 0.03$ proves to be unviable within the real-world setup, as it triggers a series of unfavorable events leading to system performance degradation. This underscores the complexity and unpredictability of system behavior under certain initial conditions.

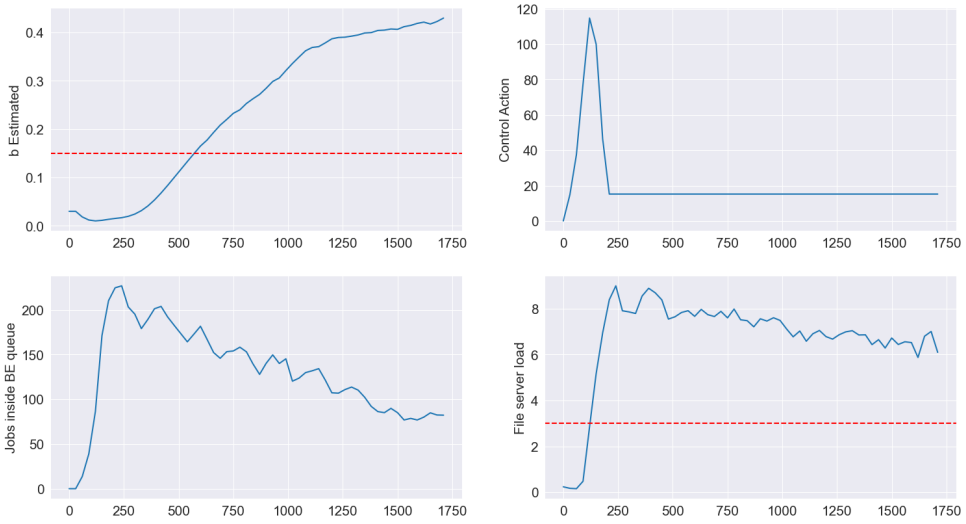

Figure 4.11: Results for the scenario where $\hat{b}(0) = 0.03$. The blue line represents the average across five distinct experiments. Within the estimation visualization, the red dashed line corresponds to $\tilde{\tilde{b}}$, while in the file server load illustration, the red dashed line signifies the reference load.

Now, focusing our attention back on Figure 4.10, we can discern a consistent pattern in both cases: an initial decrease in the estimated value. This insight has been gleaned from the experimentation process. It has come to light that the system requires a certain amount of time to initiate, manifesting as an initial delay. This delay subsides after the first 3-4 time steps. However, owing to this initial delay, the estimation is notably underestimated during this phase. This underscores the reason why opting for an excessively small initial condition for $\hat{b}(0)$ is inadvisable. The consequences of this underestimation are twofold. Firstly, it results in an overshoot in the control action. Notably, the overshoot is more pronounced in Figure (b), primarily due to the estimated value reaching lower

levels compared to the scenario depicted in Figure (a). This discrepancy in the overshoot can be attributed to the different initial conditions. Secondly, this underestimation also affects the tracking process, leading to overshoot in the early stages due to the reduction in the estimation.
Given these observations, a deliberate decision was made - select an initial value of 0.5. This choice facilitates a soft starting of the control action, effectively preventing overshoots that could otherwise lead to system overloads. It's worth noting that while 0.5 was the chosen solution in this context, it's not the exclusive solution. It is believed that any initial condition greater than 0.5 would yield satisfactory results, thereby offering flexibility in the choice of an appropriate starting point for $\hat{b}(0)$.

Initial Covariance condition

Having selected the optimal initial value for the estimated value, our focus now shifts to determining the suitable initial condition for the covariance parameter.

Figure 4.12: Outcomes based on variations in initial Covariance values. Figure (a) corresponds to an initial condition of 100, while Figure (b) employs an initial value of 10^4, and Figure (c) utilizes 10^6 as its starting point.

As evident from Figure 4.12, the impact of altering the second parameter, namely the initial covariance, is relatively modest. The observed discrepancy primarily surfaces in the final overshoot within the tracking outcome. It is worth noting that the system exhibits a substantial degree of inherent noise, and our analysis is based on averaging results from just five experiments. Therefore, it is plausible that the observed improvements are not

solely attributed to the covariance modification.
Nevertheless, considering the outcomes of this study, we have opted to select an initial value of $V(0) = 10^4$. This choice is underpinned by Figure 4.12, where this particular initial covariance value displays the least pronounced overshoot. It is important to acknowledge that the decision to increase this value is based on the consideration that higher initial covariance values may render the initial condition on $\hat{b}$ less dependable.

4.2.3. Experiment Result with Varying File Size

Once the algorithm has been fine-tuned ($\alpha = 2, \hat{b}_0 = 0.5, V(0) = 10^4$), the next phase involves an examination of the conclusive outcome through an experiment encompassing with two distinct campaigns. The initial campaign has a file size of 100 Mbyte with 2000 jobs, while the other has f equal to 200 Mbyte with 1000 tasks. In the context of the experimental setup, the beginning of the second campaign is not deterministic. Indeed the second campaign will start only when all the jobs of the previous campaign are sent, then in different experiments, the second campaign starts at a different time.

As one can see from the initial row, we refrained from displaying solely the average of the five experiments, as such an approach would oversimplify the analysis because of the motivation above. Relying solely on the average could potentially yield misleading outcomes due to the divergence in experiment completion times for the first campaign. One can note this variability in Fig. 4.13, where certain experiments initiate the new campaign around 3000 seconds, while others commence closer to 4000 seconds.

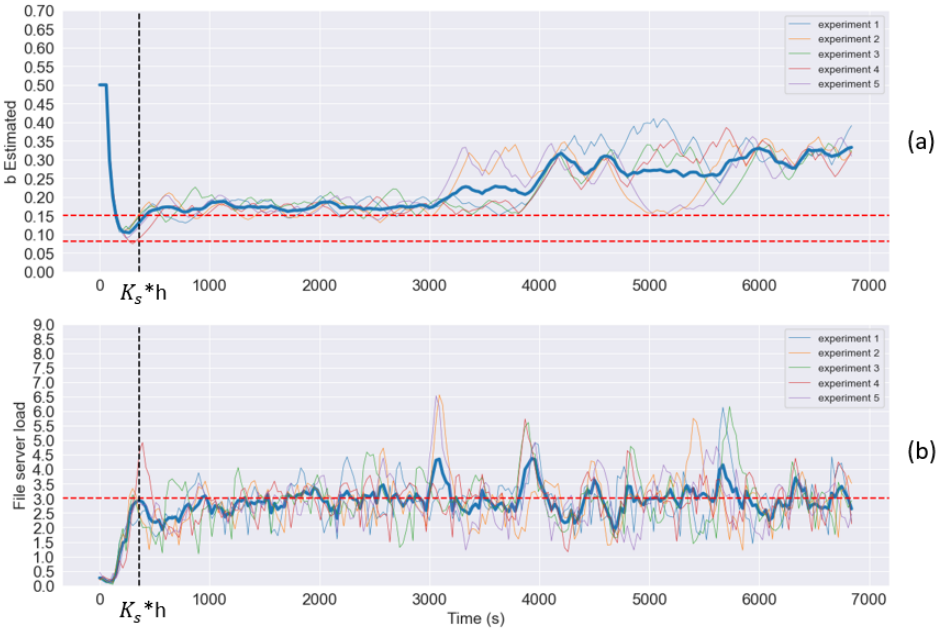

Figure 4.13: Results of the system using the adaptive PI. In Figure (a), the average of five $\hat{b}$ estimations is denoted by the blue line, while individual experiment outcomes are depicted using thinner lines. The red dashed lines correspond to $\tilde{\tilde{b}}$ values for the distinct f settings. In Figure (b), the blue line represents the average output from the five experiments, and the specific outcomes for each experiment are indicated by the thinner lines. Both figures incorporate a black dashed line designating the settling time.

Furthermore, it is essential to evaluate whether the system, when coupled with the algorithm, adheres to the overshoot requirement and time constraint. Analyzing Figure (b), we observe the tracking behavior, revealing an absence of overshooting. However, the ultimate system performance seems to be notably affected by the algorithm's presence. This observation is supported by the black dashed line in both plots, representing the anticipated time for the system to reach the reference value. In practice, the system attains this reference value after more than double the initially projected time. This deceleration appears to stem from the controller dispatching numerous jobs to achieve the reference value within the designated timeframe. Nevertheless, it is crucial to note that the identification algorithm must still converge to a specific value for $\hat{b}$, causing a delay. As depicted in Figure (a), $\hat{b}$ exhibits an increasing trend, resulting in a reduction of the control action and subsequently influencing the output in a similar manner.
The last key elements to address within Figure 4.13 pertain to the red dashed lines featured in (a). These lines correspond to the values of $\tilde{\tilde{b}}$ for the two distinct f values. Specifically, for $f = 100$MByte, the approximation value is $\tilde{\tilde{b}} = 0.082$, while for $f = 200$MByte, it equates to $\tilde{\tilde{b}} = 0.15$. Importantly, in Figure (b), these values are lower than the parameter

identified by the algorithm. It is important to note that the self-tuning technique does not guarantee the precise replication of the actual system parameter; rather, it aims to determine the parameter yielding optimal control. On the other hand, this outcome aligns with the findings in Chapter 2.2.1, where a comparison between $\tilde{\tilde{b}}$ and $\tilde{b}$ highlighted underestimation when f exceeds 25MByte. Additionally, the observed trend, wherein larger file sizes lead to greater underestimation, remains consistent.

4.3. PI vs. Adaptive PI

In this section, we present an analysis that involves the comparison of outcomes obtained from two distinct systems. The first system, illustrated by the orange line, employs the classic PI control, while the second system, represented by the blue line, incorporates the Adaptive PI. Within the disturbance plot, we observe an additional dashed green line, indicating the disturbance applied to the system, namely the HP jobs. Notably, the appearance of this disturbance at the 2000s time mark is attributed to the scheduling of HP jobs on the platform. In this scenario, a substantial quantity of HP jobs arrives, significantly occupying more than half of the file server's reference. This phenomenon is particularly evident in all instances of picture (b), where different file sizes of BE tasks are considered. Across these cases, a conspicuous spike emerges due to the application of the disturbance, accompanied by an undershoot as the disturbance subsides.

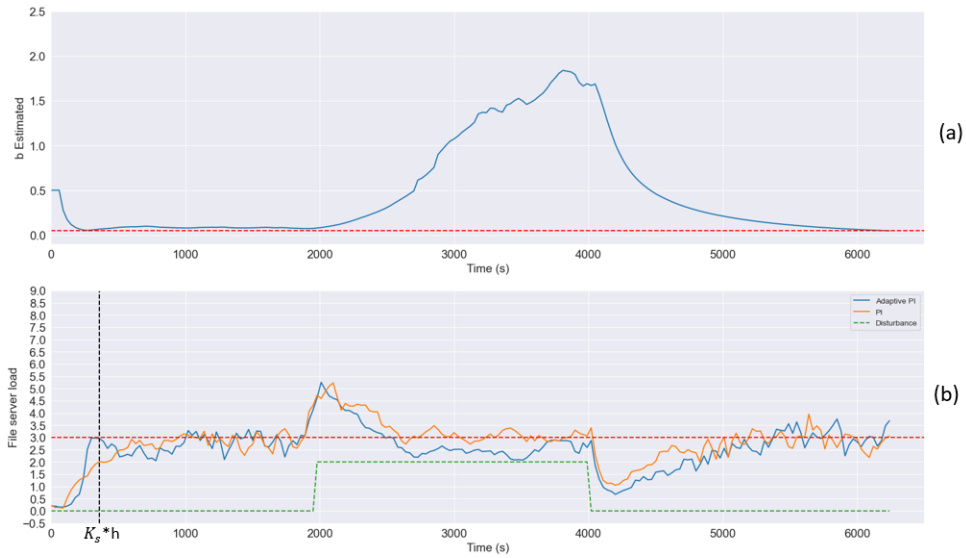

Figure 4.14: Experiment results with a f=50MByte for BE tasks: (a) Identification algorithm results (blue line) vs. $\tilde{\tilde{b}}$ (red dashed line). (b) Tracking comparison: Adaptive PI (blue), simple PI (orange), reference (red dashed), disturbance (green dashed), theoretical settling time (black dashed).

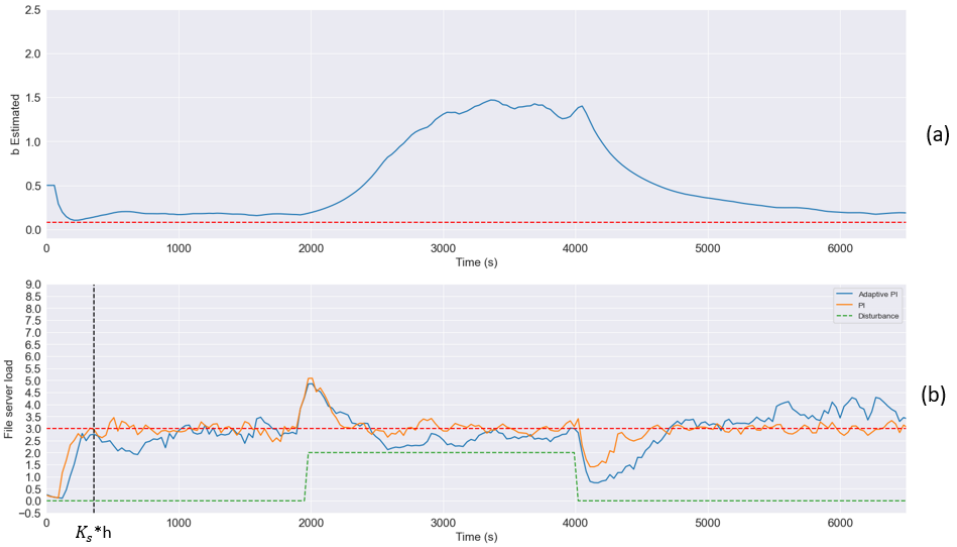

Figure 4.15: Experiment results with a f=100MByte for BE tasks: (a) Identification algorithm results (blue line) vs. $\tilde{\tilde{b}}$ (red dashed line). (b) Tracking comparison: Adaptive PI (blue), simple PI (orange), reference (red dashed), disturbance (green dashed), theoretical settling time (black dashed).

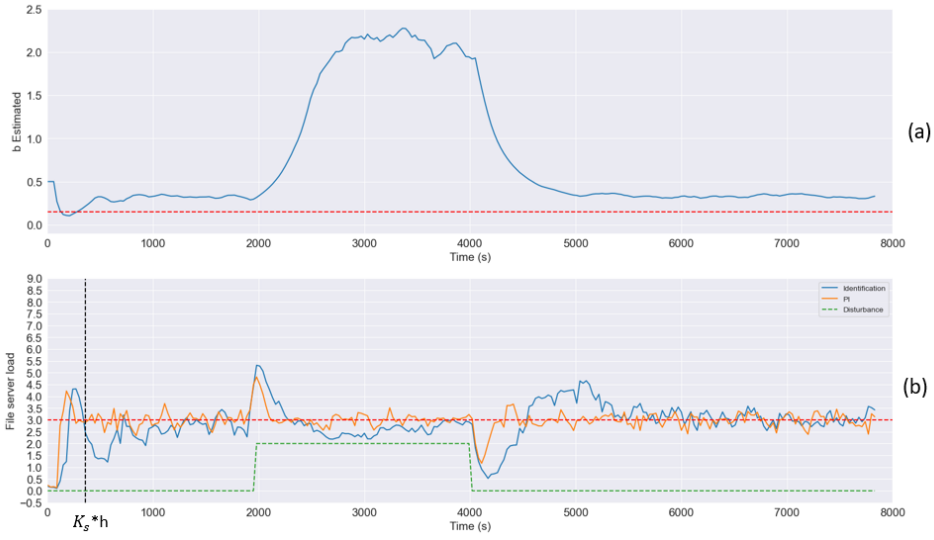

Figure 4.16: Experiment results with a f=200MByte for BE tasks: (a) Identification algorithm results (blue line) vs. $\tilde{\tilde{b}}$ (red dashed line). (b) Tracking comparison: Adaptive PI (blue), simple PI (orange), reference (red dashed), disturbance (green dashed), theoretical settling time (black dashed).

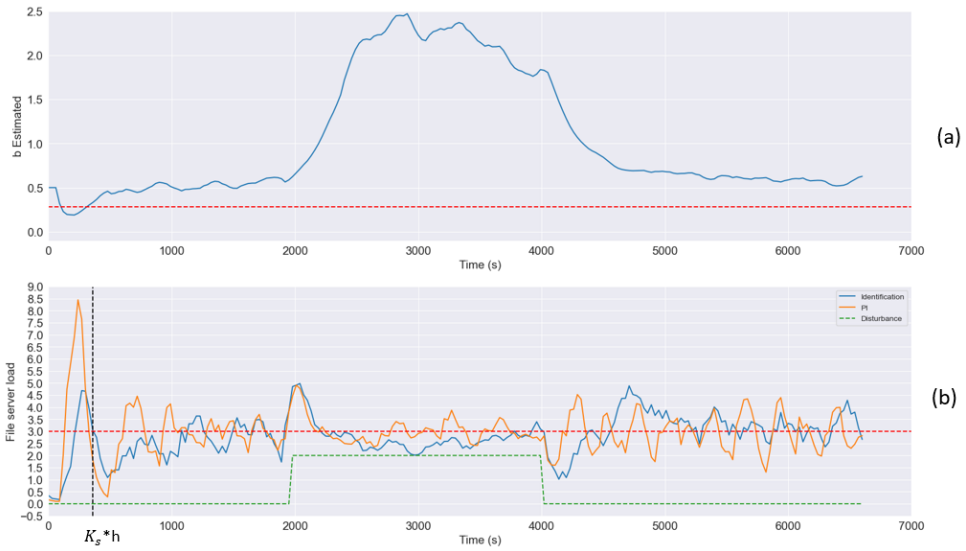

Figure 4.17: Experiment results with a f=400MByte for BE tasks: (a) Identification algorithm results (blue line) vs. $\tilde{\tilde{b}}$ (red dashed line). (b) Tracking comparison: Adaptive PI (blue), simple PI (orange), reference (red dashed), disturbance (green dashed), theoretical settling time (black dashed).

Starting an in-depth analysis of the tracking results across varying file sizes. Our objective is to comprehensively dissect the outcomes, shedding light on disparities in terms of speed and initial overshoot. The culmination of our investigation will be summarized in Table 4.1. This table encapsulates and delineates the key findings of our analysis, providing a comprehensive overview of the convergence times and system behaviors across different file sizes.

Building upon the insights from the preceding section, we observed that the system, as identified by the combined identification algorithm and PI controller, exhibits an inertia exceeding our performance expectations. In the PI controller's design phase, we aimed for a settling time of 12 time steps, equivalent to 360 seconds. However, our empirical findings reveal that the new system's settling time is more than double this projected value. Moreover, a visual examination of the preceding diagram highlights a notable challenge even in the case of the system without the algorithm. Fulfilling the time requirement across all file sizes, denoted as f, proves to be intricate, except for the nominal case and the case of f=200MBytes.
Nonetheless, a discernible trend emerges – as the file size (f) diverges from its nominal value, the disparity between the convergence times of the two systems appears to narrow. However, the new control seems to take always more time to finish the transient.

Another crucial aspect of comparison between the two systems revolves around their ini-

tial overshoot characteristics. In the design phase, we carefully choose the parameter M_p to prevent overshooting and to steer clear of challenging overloading scenarios. This intention aligns with our observations discussed in Chapter 2.2.3. As previously demonstrated, the original system successfully met this no-overshoot criterion for f equal to or less than the nominal value of 100MByte. However, this requirement remained unmet for cases involving larger file sizes, as illustrated in Figures 4.16 and 4.17. Additionally, upon examining the latter graph, one can see that the overshoot exceeds the critical threshold of 8. This implies a grave concern – the system could potentially experience severe overloading, resulting in deceleration or, in extreme scenarios, a complete halt to the experimental process.
In light of this, the newfound perspective reveals that the new control mechanism delivers more favorable outcomes. Indeed, even when it experiences overshoot in the same situations as the previous controller, it appears that the magnitude of overshoots does not increase significantly, thereby mitigating potential issues.

Further insights emerge when investigating the transient behavior, including a distinctive observation within Figure 4.17. Curiously, the system employing the simple PI control exhibits intricate transient characteristics, possibly attributed to the presence of complex poles. This peculiar transient phenomenon is visually depicted in the image. In contrast, the new control system demonstrates a lack of such transients.
Before transitioning to the discussion on the identification behavior, a final noteworthy point should be highlighted. This refers to the system's response when subjected to disturbances. Notably, in the majority of cases, the adaptive system exhibits a slower recovery in returning to the reference value compared to its counterpart utilizing the simple PI control. Thus, the Adaptive PI addresses the initial overshoot problem and provides an improved transient response, although it takes a bit more time.

f [MByte]	**Settling Time**	**Initial overshoot**
50	The Adaptive PI's delay is 50% of the PI's settling time. Whereas the final PI has a delay of 67% of the theoretical settling time.	Both the PI and the Adaptive PI have no overshoot.
100 (nominal case)	Adaptive's delay is 177% of the standard PI's settling time. The conventional PI converge 17% faster than the theoretical settling time.	Both the PI and the Adaptive PI have no overshoot.
200	Adaptive's delay is 260% of the standard PI's settling time. The conventional PI converge 30% faster than the theoretical settling time.	Both the control have an overshoot of 40% of the reference.
400	The Adaptive PI has a 5% delay compared to the conventional PI. The traditional PI converges with a time delay of 178% of the predicted settling time.	The PI has an overshoot of 180% of the reference value, while the Adaptive PI has an overshoot of 53% of the reference.

Table 4.1: A comparison of the two separate systems' outcomes for a different file size.

An examination of the identification process reveals that the earlier segment, preceding the disturbance, closely resembles the observations detailed in the preceding section, as one can see in Figure 4.18.

As the disturbance is introduced around the 2000s mark, the identification algorithm promptly gauges the disturbance's effect on the file system. Notably, the estimated impact of the disturbance appears to converge at approximately 2 units. This outcome arises due to the omission of disturbance consideration within the model. Consequently, the algorithms interpret the disturbance's arrival as a modification in the weight of BE tasks within the file server. This interpretation, in turn, contributes to the extended delay exhibited by the new system compared to the previous one.

Figure 4.18: Zoomed-in estimation results prior to the disturbance's arrival. In picture (a), we observe the scenario with a data size of f=50 MByte. In picture (b), the scenario with f=100 MByte is depicted, in (c), the scenario with f=200 MByte is presented, and in (d) the case with f=400 MByte .

5 | Conclusion and Future work

This master thesis focuses on the saturation problem of the file system within a cluster. The idea was to develop an identification algorithm that could change the main parameters of the controller for each working condition. Additionally, our goal was to achieve more consistent outcomes across a range of operational scenarios. Notably, the proportional-integral (PI) controller exhibited divergent performance outcomes depending on the file size, with particularly problematic behavior observed when dealing with larger values of f.
As previously discussed, when f reached 400 Mbytes, the PI controller displayed a substantial overshoot, potentially leading to system lock-ups. In contrast, the introduction of the Adaptive PI controller effectively mitigated this issue. In fact, it was observed that the overshoot remained relatively constant for values of f exceeding 100 Mbytes.
Nevertheless, it is important to acknowledge that this solution is not without its drawbacks. In practice, while the Adaptive PI controller effectively addresses the overshoot problem, it tends to slow down the system. Consequently, results obtained using this algorithm consistently indicate a slower system performance compared to the non-algorithmic approach. This implies that during system initialization or transitions in working conditions, such as the arrival of high-priority jobs or changes in the value of f, the system supported by the algorithm experiences a longer period of idle resources compared to the traditional PI controller.

In addition to the findings presented in this thesis, there are several potential solutions that were not implemented in the actual system but hold promise for future improvements. First, let's discuss the Selective algorithm. Initially, we had high expectations for this algorithm because it closely aligned with the system's specifications. It was designed to learn only when necessary. However, this algorithm had limited time to adapt, and this limitation stemmed from our lack of knowledge about the real platform's startup delay. In light of this, we propose a possible solution: initiating the algorithm a few steps after the overall system startup. To achieve this, we can set an initial condition for $\hat{b}$ to remain constant for the first three steps, after which the algorithm can commence its learning process. This adjustment, we believe, can effectively address the problem

of underestimation, and the available learning time for the Selective Algorithm will be sufficient.

Another technique that we developed for simulations that we did not test on the real platform is the use of a two-parameter model. We think that the two-parameter formula is better suited for the system than the single-parameter model. By considering two parameters, we incorporate information from the GANTT chart into the model, making more effective use of available data. Additionally, when relying solely on a single parameter, the theoretical interpretation of that parameter becomes muddled, as it combines the weight of only one BE task on the file server (b_u) with the weight of all HP jobs within the file server ($b_d \times d$). Therefore, we believe it would be advantageous to include other parameters in the model to ensure greater consistency with the actual processes occurring within the system.

Furthermore, we would like to reflect on a broader aspect that emerged from our experience during this research. In engineering we often focus on equations, expressions, and numerical results, sometimes overlooking the people behind the work. In the course of this study, collaboration between individuals from control theory and computer science was essential. It became evident that bridging the gap between these two fields required more than just merging different knowledge domains; it necessitated an understanding of the distinct thought processes and approaches to problem-solving that each field brings. This underlying difference in thinking patterns is not always obvious but can present significant challenges.

In conclusion, we believe that the algorithm contributes to achieving more consistent and homogeneous results. By estimating $\hat{b}$ in each iteration, it effectively addresses the challenge of pre-tuning the model for different clusters and helps prevent overloading of the file server when dealing with high values of f. However, it is important to acknowledge that this improvement comes at the cost of a system slowdown.

A | Appendix A

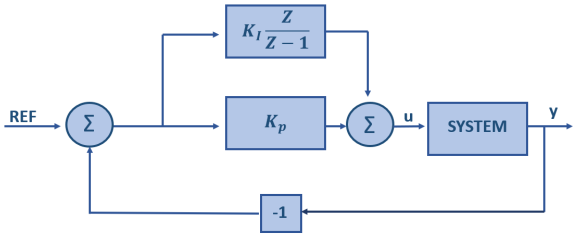

Figure A.1: Close loop with a PI controller.

The idea behind the choice of k_p and k_i is build a sort of pole placement with a PI. To do this the book [45] considered the close-loop transfer function

$$C(z) = \frac{G(z)R(z)}{1 + G(z)R(z)} \tag{A.1}$$

Where $G(z)$ is the system and $R(z)$ the controller that define the control action u, in this case

$$R(z) = K_p + \frac{K_i z}{z-1} = \frac{(K_p + K_i)z - K_p}{z-1} \tag{A.2}$$

Afterward, the close loop transfer function can be defined as

$$C(z) = \frac{[(K_p + K_i)z - K_p]G(z)}{z - 1 + [(K_p + K_i)z - K_p]G(z)} \tag{A.3}$$

The next step will be the computation of K_p and K_i in order to plug in the desired poles for the closed loop system. Then now there will be the estimation of the desired poles, by the assumption these poles are complex conjugates in the form of $r \exp^{j\theta}$. Then it considers the 2% criterion for the settling time K_s in discrete time

$$K_s \approx \frac{-4}{\ln r} \tag{A.4}$$

Then, in this specific case where the poles are complex conjugate the maximum overshoot M_p can be computed as

$$M_p \approx r^{\pi/|\theta|} \tag{A.5}$$

After the definition of these two fundamental parameters, it is possible to inverse the above relationship to compute the value of the poles by establishing the desire characteristic for the close loop

$$\begin{cases} r \approx \exp^{-4/K_s} \\ \theta \approx \pi \frac{lnr}{lnM_p} \end{cases} \tag{A.6}$$

Then, one can determine the characteristic polynomial of the system

$$(z - r\exp^{j\theta})(z - r\exp^{-j\theta}) = z^2 - 2rcos(\theta)z + r^2 \tag{A.7}$$

Then, by matching the denominator of A.3 and the right part of expression A.7, K_p and K_I are found

$$\begin{cases} K_p = \frac{a-r^2}{b} \\ K_I = \frac{1-2r\cos\theta+r^2}{b} \end{cases} \tag{A.8}$$

Bibliography

[1] John Michalakes. "HPC for Weather Forecasting". In: *Parallel Algorithms in Computational Science and Engineering*. Ed. by Ananth Grama and Ahmed H. Sameh. Cham: Springer International Publishing, 2020, pp. 297–323. ISBN: 978-3-030-43736-7. DOI: 10.1007/978-3-030-43736-7_10. URL: https://doi.org/10.1007/978-3-030-43736-7_10.

[2] Richard M. Russell. "The CRAY-1 Computer System". In: *Commun. ACM* 21.1 (Jan. 1978), pp. 63–72. ISSN: 0001-0782. DOI: 10.1145/359327.359336. URL: https://doi.org/10.1145/359327.359336.

[3] *TOP500*. https://www.top500.org/lists/top500/2023/06/. I visited the site on 13.07.2023.

[4] Marián Vajteršic, Peter Zinterhof, and Roman Trobec. "Overview – Parallel Computing: Numerics, Applications, and Trends". In: *Parallel Computing: Numerics, Applications, and Trends*. Ed. by Roman Trobec, Marián Vajteršic, and Peter Zinterhof. London: Springer London, 2009, pp. 43–80. ISBN: 978-1-84882-409-6. DOI: 10.1007/978-1-84882-409-6_1. URL: https://doi.org/10.1007/978-1-84882-409-6_1.

[5] By Yixuan Li. "Picture of 2D Torus topology". In: (). URL: https://commons.wikimedia.org/w/index.php?curid=53610657.

[6] Charles E. Leiserson. "Fat-trees: Universal networks for hardware-efficient supercomputing". In: *IEEE Transactions on Computers* C-34.10 (1985), pp. 892–901. DOI: 10.1109/TC.1985.6312192.

[7] Daniel Reed, Dennis Gannon, and Jack Dongarra. *Reinventing High Performance Computing: Challenges and Opportunities*. 2022. arXiv: 2203.02544 [cs.DC].

[8] P. S. Kostenetskiy, R. A. Chulkevich, and V. I. Kozyrev. "HPC Resources of the Higher School of Economics". In: *Journal of Physics: Conference Series* 1740.1 (Jan. 2021), p. 012050. DOI: 10.1088/1742-6596/1740/1/012050. URL: https://dx.doi.org/10.1088/1742-6596/1740/1/012050.

[9] Rahil Parmar et al. "18 - 5G-enabled deep learning-based framework for healthcare mining: State of the art and challenges". In: *Blockchain Applications for Health-*

care Informatics. Ed. by Sudeep Tanwar. Academic Press, 2022, pp. 401–420. ISBN: 978-0-323-90615-9. DOI: https://doi.org/10.1016/B978-0-323-90615-9.00016-5. URL: https://www.sciencedirect.com/science/article/pii/B9780323906159000165.

[10] Rama. *CC BY-SA 2.0 fr*. https://commons.wikimedia.org/w/index.php?curid=345858.

[11] Raphael Bleuse. "Apprehending heterogeneity at (very) large scale". PhD thesis. Université Grenoble Alpes, 2017.

[12] "Backfill scheduling". In: *https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=jobs-backfill-scheduling* ().

[13] Francieli Zanon Boito et al. "A Checkpoint of Research on Parallel I/O for High-Performance Computing". In: *ACM Comput. Surv.* 51.2 (Mar. 2018). ISSN: 0360-0300. DOI: 10.1145/3152891. URL: https://doi.org/10.1145/3152891.

[14] D. Patterson and J. Hennessy. *Computer Organization and Design: the hardware/software interface*. 3rd edition. Elsevier, 2005, pp. 468–610.

[15] Quentin Guilloteau et al. "Controlling the Injection of Best-Effort Tasks to Harvest Idle Computing Grid Resources". In: *2021 25th International Conference on System Theory, Control and Computing (ICSTCC)*. 2021, pp. 334–339. DOI: 10.1109/ICSTCC52150.2021.9607292.

[16] Agustín Gabriel Yabo et al. "A control-theory approach for cluster autonomic management: maximizing usage while avoiding overload". In: *2019 IEEE Conference on Control Technology and Applications (CCTA)*. 2019, pp. 189–195. DOI: 10.1109/CCTA.2019.8920473.

[17] Géza Ódor and Bálint Hartmann. "Heterogeneity effects in power grid network models". In: *Phys. Rev. E* 98 (2 Aug. 2018), p. 022305. DOI: 10.1103/PhysRevE.98.022305. URL: https://link.aps.org/doi/10.1103/PhysRevE.98.022305.

[18] P Horn. "IBM, Autonomic Computing: IBM's Perspective on the State of Information Technology". In: *URL: https://people. scs. carleton. ca/soma/biosec/readings/autonomic computing. pdf* ().

[19] J.O. Kephart and D.M. Chess. "The vision of autonomic computing". In: *Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055.

[20] Manish Parashar and Salim Hariri. "Autonomic Computing: An Overview". In: *Unconventional Programming Paradigms*. Ed. by Jean-Pierre Banâtre et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 257–269. ISBN: 978-3-540-31482-0.

[21] Danny Weyns. "Software Engineering of Self-adaptive Systems". In: *Handbook of Software Engineering*. Ed. by Sungdeok Cha, Richard N. Taylor, and Kyochul Kang.

Cham: Springer International Publishing, 2019, pp. 399–443. ISBN: 978-3-030-00262-6. DOI: 10.1007/978-3-030-00262-6_11. URL: https://doi.org/10.1007/978-3-030-00262-6_11.

[22] Tarek Abdelzaher et al. "Introduction to Control Theory And Its Application to Computing Systems". In: *Performance Modeling and Engineering*. Ed. by Zhen Liu and Cathy H. Xia. Boston, MA: Springer US, 2008, pp. 185–215. ISBN: 978-0-387-79361-0. DOI: 10.1007/978-0-387-79361-0_7. URL: https://doi.org/10.1007/978-0-387-79361-0_7.

[23] Sophie Cerf et al. "Sustaining Performance While Reducing Energy Consumption: A Control Theory Approach". In: *Euro-Par 2021: Parallel Processing*. Ed. by Leonel Sousa, Nuno Roma, and Pedro Tomás. Cham: Springer International Publishing, 2021, pp. 334–349. ISBN: 978-3-030-85665-6.

[24] IBM. "An architectural blueprint for autonomic computing". In: (2005). URL: https://users.cs.fiu.edu/~sadjadi/Teaching/Autonomic%20Grid%20Computing/CIS-6612-Summer-2006/AC-Blueprint-WhitePaper-V7.pdf.

[25] Eric Rutten, Nicolas Marchand, and Daniel Simon. "Feedback Control as MAPE-K Loop in Autonomic Computing". In: *Software Engineering for Self-Adaptive Systems III. Assurances*. Ed. by Rogério de Lemos et al. Cham: Springer International Publishing, 2017, pp. 349–373. ISBN: 978-3-319-74183-3.

[26] Xiaoyun Zhu et al. "What Does Control Theory Bring to Systems Research?" In: *SIGOPS Oper. Syst. Rev.* 43.1 (Jan. 2009), pp. 62–69. ISSN: 0163-5980. DOI: 10.1145/1496909.1496922. URL: https://doi.org/10.1145/1496909.1496922.

[27] Antonio Filieri et al. "Software Engineering Meets Control Theory". In: *2015 IEEE / ACM 10th International Symposium on Software Engineering for Adaptive and Self - Managing Systems*. 2015, pp. 71–82. DOI: 10.1109/SEAMS.2015.12.

[28] Yuriy Brun et al. "Engineering Self-Adaptive Systems through Feedback Loops". In: *Software Engineering for Self-Adaptive Systems*. Ed. by Betty H. C. Cheng et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 48–70. ISBN: 978-3-642-02161-9. DOI: 10.1007/978-3-642-02161-9_3. URL: https://doi.org/10.1007/978-3-642-02161-9_3.

[29] Alessandro Vittorio Papadopoulos et al. "A dynamic modelling framework for control-based computing system design". In: *Mathematical and Computer Modelling of Dynamical Systems* 21.3 (2015), pp. 251–271. DOI: 10.1080/13873954.2014.942785. eprint: https://doi.org/10.1080/13873954.2014.942785. URL: https://doi.org/10.1080/13873954.2014.942785.

[30] Stepan Shevtsov et al. "Control-Theoretical Software Adaptation: A Systematic Literature Review". In: *IEEE Transactions on Software Engineering* 44.8 (2018), pp. 784–810. DOI: 10.1109/TSE.2017.2704579.

[31] Alberto Leva et al. "How control-friendly is a computing system? And how control-friendly could it be?" In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 7857–7864. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.1962. URL: https://www.sciencedirect.com/science/article/pii/S240589632032591X.

[32] M. Kristic, Kokotovic P.V., and kanellakopoulos I. *Nonlinear and Adaptive Control Design*. 1st edition. New York: Wiley, 1995.

[33] Tharindu Patikirikorala et al. "An evaluation of multi-model self-managing control schemes for adaptive performance management of software systems". In: *Journal of Systems and Software* 85.12 (2012). Self-Adaptive Systems, pp. 2678–2696. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2012.05.077. URL: https://www.sciencedirect.com/science/article/pii/S0164121212001628.

[34] Konstantinos Angelopoulos, Alessandro Vittorio Papadopoulos, and John Mylopoulos. "Adaptive Predictive Control for Software Systems". In: *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*. CTSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, pp. 17–21. ISBN: 9781450338141. DOI: 10.1145/2804337.2804340. URL: https://doi.org/10.1145/2804337.2804340.

[35] Tharindu Patikirikorala, Liuping Wang, and Alan Colman. "Towards optimal performance and resource management in web systems via model predictive control". In: *2011 Australian Control Conference*. 2011, pp. 469–474.

[36] Guangyi Cao and Arun A. Ravindran. "Energy Efficient Soft Real-Time Computing through Cross-Layer Predictive Control". In: *9th International Workshop on Feedback Computing (Feedback Computing 14)*. Philadelphia, PA: USENIX Association, June 2014. URL: https://www.usenix.org/conference/feedbackcomputing14/workshop-program/presentation/cao.

[37] Antonio Filieri, Henry Hoffmann, and Martina Maggio. "Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees". In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 299–310. ISBN: 9781450327565. DOI: 10.1145/2568225.2568272. URL: https://doi.org/10.1145/2568225.2568272.

[38] . "Picture of Grid5000". In: (2023). URL: https://www.grid5000.fr/w/File:G5k-backbone.png.

[39] Yiannis Georgiou, Olivier Richard, and Nicolas Capit. "Evaluations of the Lightweight Grid CIGRI upon the Grid5000 Platform". In: *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. 2007, pp. 279–286. DOI: 10.1109/E-SCIENCE.2007.32.

[40] N. Capit et al. "A batch scheduler with high level components". In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Vol. 2. 2005, 776–783 Vol. 2. DOI: 10.1109/CCGRID.2005.1558641.

[41] U. Narayan Bhat. *An Introduction to Queueing Theory*. Birkhauser, 2008.

[42] Ali El Hadi Noura. *Integration of Scheduler Knowledge into CiGri Control Loop*. 2022.

[43] Domenico Ferrari and Songnian Zhou. "An Empirical Investigation of Load Indices For Load Balancing Applications". In: *Proceedings of Performance '87* the 12th International Symposium on Computer Performance Modeling.Measurement, and Evaluation (1988), pp. 515–528. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/1987/CSD-87-353.pdf.

[44] Joseph Emeras et al. "Analysis of the Jobs Resource Utilization on a Production System". In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Narayan Desai and Walfredo Cirne. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–21. ISBN: 978-3-662-43779-7.

[45] Joseph L. Hellerstein et al. *Feedback Control of Computing Systems*. WILEY INTERSCIENCE, 2004.

[46] Ioan Doré Landau et al. *Adaptive Control: Algorithms, Analysis, and Applications*. Springer, 2011.

[47] Stepan Shevtsov and Danny 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. Seattle, WA, USA: Association for Computing Machinery, 2016, pp. 229–241. ISBN: 9781450342186. DOI: 10.1145/2950290.2950301. URL: https://doi.org/10.1145/2950290.2950301.

[48] Martina Maggio, Cristian Klein, and Karl-Erik Årzén. "Control strategies for predictable brownouts in cloud computing". In: *IFAC Proceedings Volumes* 47.3 (2014). 19th IFAC World Congress, pp. 689–694. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20140824-6-ZA-1003.00669. URL: https://www.sciencedirect.com/science/article/pii/S1474667016416940.

[49] Stepan Shevtsov, M. Usman Iftikhar, and Danny Weyns. "SimCA vs ActivFORMS: Comparing Control- and Architecture-Based Adaptation on the TAS Exemplar". In: *Proceedings of the 1st International Workshop on Control Theory for Software*

Engineering. CTSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, pp. 1–8. ISBN: 9781450338141. DOI: 10.1145/2804337.2804338. URL: https://doi.org/10.1145/2804337.2804338.

[50] K.J. Åström and B. Wittenmark. "On self tuning regulators". In: *Automatica* 9.2 (1973), pp. 185–199. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(73)90073-3. URL: https://www.sciencedirect.com/science/article/pii/0005109873900733.

[51] L. Ljung and T. Soderstrom. *Theory and practice of recursive identification*. MIT Press, 1983.

[52] Karl J. Astrom and Wittenmark Bjorn. *Adaptive Control*. second edition. DOVER PUBLICATIONS, 2008.

[53] K.J. Åström et al. "Theory and applications of self-tuning regulators". In: *Automatica* 13.5 (1977), pp. 457–476. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(77)90067-X. URL: https://www.sciencedirect.com/science/article/pii/000510987790067X.

[54] Brian D.O. Anderson. "Adaptive systems, lack of persistency of excitation and bursting phenomena". In: *Automatica* 21.3 (1985), pp. 247–258. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(85)90058-5. URL: https://www.sciencedirect.com/science/article/pii/0005109885900585.

[55] P Navrátil and V Bobál. "Recursive identification algorithms library". In: *Proceedings17th International Conference on Process Control*. 2009, pp. 516–523.

[56] T.R. Fortescue, L.S. Kershenbaum, and B.E. Ydstie. "Implementation of self-tuning regulators with variable forgetting factors". In: *Automatica* 17.6 (1981), pp. 831–835. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(81)90070-4. URL: https://www.sciencedirect.com/science/article/pii/0005109881900704.

[57] Jennifer Hsu Hill and B. Erik Ydstie. "Adaptive control with selective memory". In: *International Journal of Adaptive Control and Signal Processing* 18.7 (2004), pp. 571–587. DOI: https://doi.org/10.1002/acs.819. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/acs.819. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/acs.819.

[58] Grid5000. "Documentation about the cluster in NAncy". In: (). URL: https://www.grid5000.fr/w/Hardware.

[59] Quentin Guillentau et al. "Folding a Cluster containing a Distributed File-System". In: *HAL* (2023). URL: https://hal.science/hal-04038000.

List of Figures

List of Tables

List of Symbols

Variable	Description	SI unit
$'$	Transpose	
$\boldsymbol{\alpha}$	Parameter for the robust algorithm	
$\boldsymbol{\Delta}$	Parameter for select data	
$\boldsymbol{\epsilon}$	Prediction error	
$\boldsymbol{\theta}$	Vector which gather all the unknown parameter	
$\hat{\boldsymbol{\theta}}$	Estimation of Θ from the algorithm	
$\boldsymbol{\mu}$	Forgetting factor	
$\boldsymbol{\xi}$	Parameter of the exponential filter	
$\boldsymbol{\sigma}$	Time constant for the variance r	
$\boldsymbol{\phi}$	Regressor vector	
$\boldsymbol{b}$	Weight of a single BE job inside the file system	
$\tilde{\boldsymbol{b}}$	Eq. 2.6 for the computation of b	
$\tilde{\tilde{\boldsymbol{b}}}$	Eq. 2.7 for the computation of b	
$\hat{\boldsymbol{b}}$	b estimated by the algorithm	
$\boldsymbol{b_d}$	Single HP job file server weight	
$\hat{\boldsymbol{b}}_{\boldsymbol{d}}$	Estimation of the algorithm of b_d	
$\boldsymbol{b_u}$	Single BE job file server weight, in the case of double estimation	
$\hat{\boldsymbol{b}}_{\boldsymbol{u}}$	Estimation of the algorithm of b_u	
$\boldsymbol{d}$	Disturbance (number of HP jobs)	
$\boldsymbol{e}$	Sensor's noise	
$\boldsymbol{f}$	File Size	MByte

Variable	Description	SI unit
$\boldsymbol{h}$	Discretization time	s
$\boldsymbol{K}_D$	Derivative coefficient for the PI controller	
$\boldsymbol{K}_I$	Integrative coefficient for the PI controller	
$\boldsymbol{K}_p$	Proportional parameter for the PI controller	
$\boldsymbol{K}_s$	PI Settling Time in Iterations	
$\boldsymbol{M}_0$	Memory length	
$\boldsymbol{M}_p$	Initial overshoot allowed as a percentage of the output	
$\boldsymbol{n}$	Noise of the sensor	
$\boldsymbol{r}$	Estimating unconsidered model variance	
$\boldsymbol{S}$	Information matrix	
$\boldsymbol{u}$	Control action of the controller, number of BE jobs	
$\boldsymbol{V}$	Covariance matrix	
$\boldsymbol{y}$	Measured load of the file system	
$\hat{\boldsymbol{y}}$	Estimated load of the file system	

Acknowledgements

Firstly, I would like to thank the people who followed me at Grenoble, Sophie Cerf, Bogdan Robu, Raphael Bleuse, and Eric Rutten. Furthermore, I would like to thank also my relator, Alberto Leva, who gave me the chance to do this thesis abroad.

Afterward, I would like to thank all my friends in Milan both the ones from physics engineering and automation engineering. Your friendship was fundamental for me to continue my studies and follow what I like the most. Then, I would like to thank my friends from Civitanova Marche, you are my home when I think that I have no one. Thank you for helping me when I needed it.

Then I would like to thank my family. Moreover, I would like to thank my parents for their patience when I was a teenager. Without your choices, I couldn't be here, so thank you. Then I would like to mention also my brother, Luigi (Giggi), who is very important to me.

Lastly, I would like to thank my boyfriend, Stefano. Thank you for listening to me every time that I complain and I do not believe in myself. I do not know how I could handle all the changes without you. And I'm sure that whatever happens, you will make me smile, then thank you for being with me in this experience and the next.

Ringraziamenti

Vorrei iniziare con i ringraziamenti per le persone che hanno reso questo lavoro possibile. Vorrei ringraziare Sophie Cerf, Bogdan Robu, Raphael Bleuse e Eric Rutten per avermi seguito a Grenoble durante i mesi che sono stata lì. Inoltre vorrei ringraziare anche il mio relatore, il professor Alberto Leva, per avermi dato la possibilità di fare la tesi all'estero.

Vorrei ringraziare tutti i miei amici che ho conosciuto a Milano, sia quelli di ingegneria fisica che quelli di automazione. La vostra amicizia è state fondamentale per il normale conseguimento dei miei studi e mi avete dato sempre la forza di seguire ciò che più mi piacesse.
Poi, vorrei ringraziare i miei amici storici di Civitanova Marche, siete la mia casa quando penso di non averne nessuna. Grazie per esserci sempre.

Vorrei ringraziare la mia famiglia. Soprattutto i miei genitori per la pazienza che hanno avuto nei miei confronti quando ero un'adolescente. Senza le vostre scelte non sarei mai diventata ciò che sono. Grazie. Vorrei anche ringraziare mio fratello, Luigi (Giggi), che è davvero molto importante per me.

Infine, vorrei ringraziare il mio fidanzato Stefano. Grazie per aver sempre ascoltato tutte le mie lamentele e tutte le volte che non credevo in me. Non so come avrei affrontato questa nuova esperienza senza di te. Sono sicura che qualsiasi cosa accada, mi farai sempre sorridere, quindi grazie per avermi seguito in questa esperienza ed esserci anche nella prossima.