Universitatea Națională de Științe și Tehnologie POLITEHNICA
București

Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare

TEZĂ DE DOCTORAT

Rezumat

Îmbunătățirea execuției aplicațiilor în cluster, Grid și cloud

Conducător Științific:
Prof. Dr. Ing. Nicolae ȚĂPUȘ

Autor:
drd. Maria-Elena MIHĂILESCU

București, 2024

National University of Science and Technology POLITEHNICA Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

PHD THESIS

Summary

Improvements to application execution in cluster, Grid and cloud environments

Scientific Adviser:
Prof. Dr. Ing. Nicolae ȚĂPUȘ

Author:
drd. Maria-Elena MIHĂILESCU

Bucharest, 2024

Abstract

With each new scientific discovery that advances the research world, new research topics arise. Thus, they generate large computational problems that require many resources for fast and reliable solving. At the same time, people shift from maintaining their own server, storage and application at home to using cloud resources due to their advantages (easy management, pay only for the used resources etc.). In consequence, there is a need for large computing architectures that can provide sufficient computing power for the user's needs. Usually, these resources are grouped in clusters, and clusters in Grids. Adding an abstraction layer on top of these, cloud architectures are obtained.

Nevertheless, complex grid architectures require proper management to achieve and maintain high-performance standards. From design choices and infrastructure's security to users' needs, system administrators must decide what are the best tools to be used on top of the given hardware resources. This thesis gives an in-depth analysis of two complex architectures, focusing on their needs: a cluster and cloud architecture existent in the University Politehnica of Bucharest (UPB) and a Grid infrastructure used in CERN's ALICE (A Large Ion Collider Experiment) Grid middleware. Based on the identified needs, this thesis proposes solutions that can be used to improve the cluster, Grid and cloud architecture's management.

ALICE is one of the four main experiments that run at the flagship CERN Large Hadron Collider (LHC). To process the accumulated physics data, it uses a considerable amount of computing power, up to 200K CPU cores, and the processing payload execution must be optimized for efficient use of these resources. Each payload on the Grid is assigned a maximum time to run (TTL = Time To Live) and all execution hosts have a maximum predefined time during which a job can be executed, usually 24 hours. Due to the heterogeneity of the Grid resources (different hardware, software, network and system load), the TTL is statically configured to fit all sites and hosts across the Grid and is commonly set near the time defined by the slowest execution host. This is a sub-optimal approach, as the same payload will considerably underuse the defined TTL and some of the hosts with better performance cannot schedule payloads with inflated TTL. By eliminating the 'lowest common denominator' approach and tuning the TTL to match the individual host's capabilities, each computing node could potentially execute the maximum amount of payloads and thus decrease the overall execution turnaround time. This thesis presents a profiling of the MonteCarlo productions that run in the ALICE Grid. Furthermore, it shows the factors that influence job performance and derives an algorithm to determine a suitable TTL for MonteCarlo jobs, tuned to match the performance of the executing host. Furthermore, it shows the experimental results of the proposed algorithm.

In the last years, research and education institutes are shifting towards (fully or partially) online learning platforms and are improving their research infrastructure to accommodate researcher's needs. Nevertheless, the management of a complex infrastructure can become troublesome for its users since courses need to be created, students and teachers need to be enrolled to courses. When the number of users increases, the manual labour does not scale. Thus, automatic courses

and enrolment methods are needed. This thesis presents the way UPB automatically handles 35000 users divided across 11000 courses.

Developing new features for the core of an operating system is a difficult task that needs many resources and the proper infrastructure for building, running, reverting/recovering from failure or rebasing the code. This thesis presents cluster and cloud infrastructure improvements added to support research activities on FreeBSD in UPB. Furthermore, this thesis proposes improvements for the live migration feature in bhyve, that can be later used as a means for load-balancing cluster nodes.

Research and education centres have many users to whom they provide Internet access. While users can bring their own devices and connect them to the network, new potential attack vectors arise either by malicious attackers that connect to the public research networks or by non-malicious users that are unknowingly a pivot during an attack (through phishing or other malware campaigns that spread through the network). Ensuring network security is, however, a difficult task. Usually, multiple tools (or toolkits) are used to protect the network: there is no perfect tool that can protect against all kinds of malware; thus, adding multiple tools with different scopes enhances the network protection. Nevertheless, as new security tools are created, their real-world testing is difficult. This thesis presents two complex network architectures that integrate two machine-learning-based tools (BDE Engine and Punch Platform) from the SIMARGL (Secure Intelligent Methods for Advanced Recognition of Malware and Stegomalware) toolkit. Furthermore, it presents how controlled attacks were used to test the tools' detection correctness.

Contents

Chapter 1

Introduction

As the technological world evolves so does the need for computational resources. Complex research problems (simulations of complex chemical or physical processes, machine learning training, big data processing) require large physical hardware resources to run on. Furthermore, the digitalization process has an ascendant trend in all industries: public or private sectors as well as individuals move towards using digital resources to improve their activity by using e-platforms, online storage or online communication tools. All these digital solutions need large hardware resources as well.

The technological world tends to move from vertical scaling (one single machine with many resources) to horizontal scaling (more machines with fewer resources): one powerful machine may not be able to solve complex problems; instead, gathering multiple machines and splitting the problem in smaller task may improve the solving time. In consequence, cluster, and, then, Grid, architectures started to be adopted. Furthermore, by abstracting the hardware through a cloud solution, system administrators could easily provide people access to digital resources.

1.1 Thesis Scope

This thesis dives into cluster, Grid and cloud architectures, studying their topologies, requirements and security concerns. The scope of this thesis is to discover the needs of cluster, Grid and cloud architectures and to improve their usability and performance.

The thesis concentrates on the network, middleware and applications that compose, manage and, respectively, run in cluster, Grid and cloud architectures. As case studies, it analyses the cluster and cloud architecture used by the University Politehnica of Bucharest (UPB) and the Grid architecture used in ALICE's Grid (CERN's A Large Ion Collider Experiment). Moreover, it identifies their needs and proposes solutions for them. Nevertheless, the contributions of this thesis can be adapted to other cluster, Grid and cloud architectures.

1.2 Thesis Contributions

The thesis studies the literature and presents a general view of the topology of a cluster infrastructure. Multiple clusters are interconnected through a middleware and form a Grid structure. Furthermore, the thesis summarizes, based on the literature review, the needs in cluster, Grid and cloud computing architectures or the domains that use them: the need for heterogeneity, proper scheduling and resource management, cost or energy efficient infrastructures, security and privacy, support for research and learning activities, datasets and resource variety, and support for job and application diversity. The thesis also presents two case studies: a cluster and cloud computing architecture in UPB and a Grid middleware in ALICE.

Furthermore, the thesis presents contributions for solving parts of the needs of cluster, Grid and cloud architectures we have identified. Even though the contributions are applied in specific environments, the analysis and the proposed solutions can be replicated as well in other architectures.

The thesis describes the **improvements for scheduling MonteCarlo production in the ALICE Grid** - MonteCarlo jobs are CPU-intensive applications and are used for simulation. In CERN's ALICE experiment, they are used to simulate particle collision in conditions that resemble LHC's (Large Hadron Collider) physics events. A production can contain thousands of identical MonteCarlo jobs (same application, same condition, different entropy) that will be executed in the ALICE's Grid. In the ALICE Grid, the central services require a slot of 24h on one of the clusters that adhere to the WLCG Grid (Worldwide LHC Computing Group). For better efficiency and resource utilization, in this 24h slot, as many jobs as possible must be run. However, due to the high heterogeneity of the Grid (different CPU models, different configurations), it was observed that the MonteCarlo jobs need a very high TTL (Time to Live) so that can fit on all sites. Nevertheless, this means that sites that are more powerful for CPU intensive jobs will finish the job much faster. However, due to the high TTL, jobs might not be scheduled to these sites even though they would have sufficient time for execution. Thus, this thesis presents a method of profiling the MonteCarlo jobs: it analysis and plots two large productions (LHC19_cpubench_pp and LHC22e1_extra) that ran in the ALICE Grid based on their traces.

After identifying the CPU model name, the hyperthreading option, and the site and host configurations as the most important factors, the thesis proposes a TTL prediction algorithm that is based on the running history of similar jobs. The similarity is given by splitting jobs based on a certain key. We propose two keys (<production, site, CPU Model Name and hyperthreading> and <production, site, host>) and simulate the algorithm behaviour when data collected from the ALICE Grid is split by these keys. Furthermore, the thesis presents the results we obtained when combining the two keys. Finally, we conclude that <production, site, CPU Model Name, hyperthreading> is the key that generates the best results.

After that, the thesis describes the **improvements** made for **facilitating the research and**

education activities. The improvements concentrated on e-learning platforms and making FreeBSD available in Universities' cluster environments.

In addition to being an e-learning platform, Moodle can be used for other **collaborative tasks**. Through multiple plugins and features, Moodle can help teachers and University staff to overview the learning activities or generate new means of internal organization as presented in this thesis. This thesis proposes a method for **automatic course creation and student enrolment in an e-learning platform**. The proposed solution uses the students' learning contracts and the faculties' learning plans to generate a course structure, and then upload the courses. The solution relies on this information being digitalised and easily accessible to system administrators. The proposed solution was implemented for UPB's Moodle platform.

The thesis also shows how the **FreeBSD operating system can be used in cloud environments to facilitate software development activities** (from compilation to fault tolerance). There are a large number of development projects based on FreeBSD in UPB. Thus, the thesis shows how FreeBSD can be integrated into a heterogeneous architecture. It also shows the changes made to the operating system to facilitate integration into OpenStack, as well as the tests performed.

Moreover, the thesis also presents **the changes made to the virtual machine migration procedure using bhyve**, the hypervisor in FreeBSD. Migrating virtual machines is an administrative procedure performed in the cluster for load balancing of systems or for maintenance operations. The thesis details improvements to bhyve virtual machine migration: adding support for virtual machines with non-wired memory (memory is not pre-allocated when starting the virtual machine) and reducing the number of copy operations of the same data. In addition to the implementation, the thesis also shows the results obtained.

After that, the thesis shows the **testing procedures for network security tools** we have used for testing SIMARGL, a security toolkit. While network architectures become more complex, their attack surface increases. In consequence, multiple tools (and toolkits) are developed to detect malicious attacks using machine-learning techniques. However, one major drawback of these security tools is that they are not tested in real-life scenarios, in real network infrastructures. This thesis proposes two network architectures (Network A and Network B) used for testing two machine learning-based tools that are part of a security toolkit, SIMARGL. Besides the network architectures used to integrate the tools, the thesis presents the attacks that were run and their results.

1.3 Thesis Structure

This thesis is structured as follows:

1. The first chapter highlights the scope of this thesis and its objectives.
2. The second chapter presents the differences between cluster, Grid and cloud infrastructures. It gives an overview of a cluster and Grid topology, its needs, its performance

requirements and the security concerns that may arise in a complex network. Moreover, the chapter presents the requirements that must be met by an operating system to be considered for a cluster infrastructure. Furthermore, the chapter shows two case studies: one for a cluster and cloud infrastructure and one for a Grid infrastructure. By analysing their architectures, we present their needs, needs that are further considered for improvement in the following chapters.

3. Chapter three presents the profiling of MonteCarlo productions in a heterogeneous Grid infrastructure, CERN's ALICE Grid. Furthermore, the chapter proposes a prediction algorithm meant to forecast a maximum value that is needed by MonteCarlo jobs to finish their execution in the ALICE Grid. The proposed algorithm aims to reduce the TTL (Time To Live) value required by jobs and improve the job scheduling mechanism from ALICE's Grid middleware. Then, the chapter presents the results of the proposed algorithm, by considering multiple prediction keys. The simulations shown in this chapter were run on data collected from the ALICE Grid.

4. The fourth chapter presents improvements for learning and research activities. It starts by showing the automatic management operations we have developed for UPB's e-learning platform, Moodle. The chapter describes the advantages of using Moodle for collaborative tasks, as well as the course generation and student enrolment mechanisms we have developed for UPB. After that, we present how FreeBSD is used by UPB students for learning and research projects. Thus, we show how FreeBSD can be used in a cluster and Grid Environment. We show that FreeBSD meets all operating system requirements for running in a distributed environment. Furthermore, we test bhyve under various operating systems and in FreeBSD's bare-metal in UPB's cluster. Then, we present improvements for the live migration feature for bhyve.

5. Chapter five shows the development and testing process of a security toolkit, SIMARGL, used for protecting complex network architecture. The chapter also describes how two machine learning-based tools from the toolkit were tested in real life scenarios, in two production networks. Moreover, it presents how the attacks were run without affecting the normal network activity. Furthermore, it presents the results we have obtained.

6. The last chapter presents this thesis' conclusions.

Chapter 2

Cluster, Grid and cloud infrastructures

Based on the literature review, this chapter presents an overview of cluster, Grid and cloud infrastructures and their needs. This chapter also presents two case studies. The first one describes a hybrid cluster and cloud architecture in University Politehnica of Bucharest. Even though the University's cluster is part of a Grid architecture (WLCG - Worldwide LHC Computing Group), our case study focuses on its cluster architecture. The second case study depicts a Grid architecture and its middleware functionality: CERN's ALICE Grid.

2.1 General Topology of a Cluster and Grid Infrastructure

Cluster, Grid and cloud [1, 2, 3] are three terms used to describe the architectures of physical and software resources used by institutions for computing activities and problems: research topics in biology, chemistry, physics[2], learning activities [4], training machine learning algorithms, semantic annotation [5], parallel processing [6], hyperspectral dimensionality reduction [7] .

While cluster, Grid and cloud are all used for computing, they differ from various points of view: cluster usually refers to the resources from a single physical building, usually with one team of system administrators, whereas the Grid is usually is locality-distributed and is composed by multiple clusters. Consequently, each location has its own system administration team [8]. Even though cluster may have heterogeneous resources, the Grid is, by far, the most heterogeneous type of shared resources [9, 8].

In contrast with the cluster and Grid architectures, the cloud is an abstract environment from the user's point of view [1, 2, 9, 6]. It offers end users the possibility to customize their resources without the need of knowing the underneath infrastructure and with minimal interaction. Moreover, end users can pay only the resources they use [1] and scale up or down according to their needs.

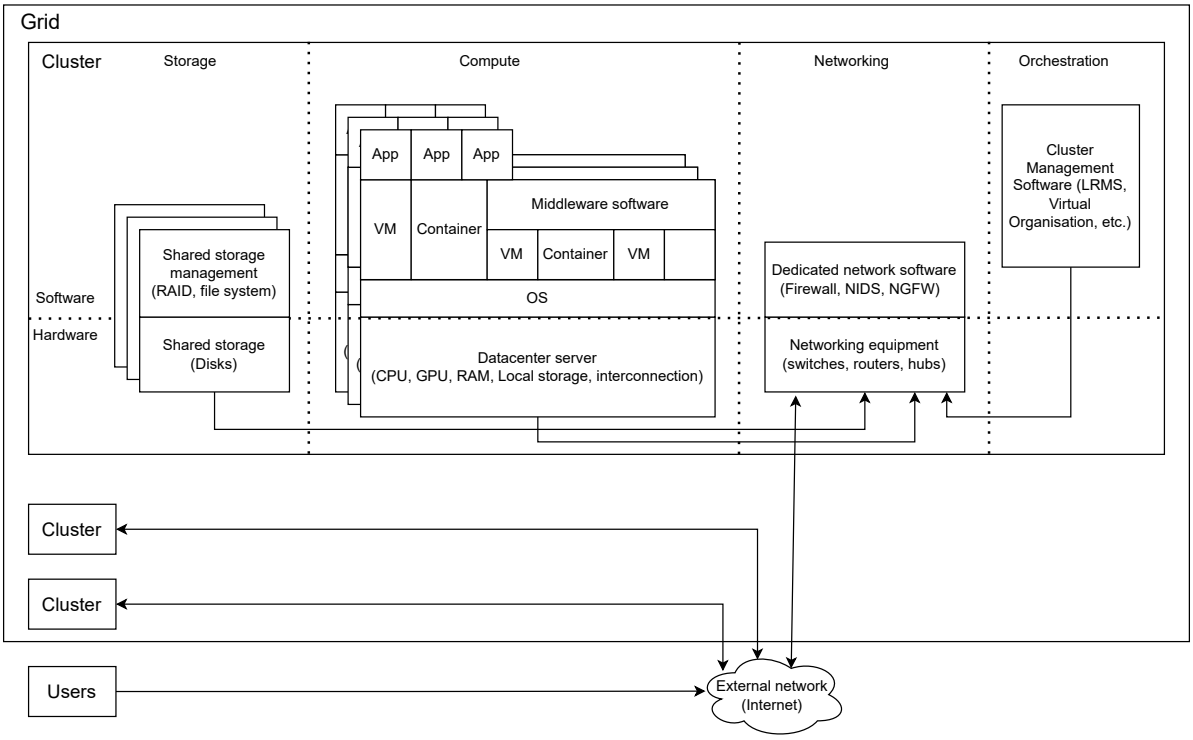


Figure 2.1: General view of a Cluster and Grid Architecture

2.2 Needs in Cluster, Grid and Cloud Computing architectures

The technological domain is continuously evolving and new problems that need to be researched or solved using computing facilities arise. Thus, infrastructures need to be constantly updated to provide the necessary means so that their users to improve their work.

Based on the literature review, we identify the following major categories of needs in cluster, Grid and cloud environments:

- **the need for many (heterogeneous) resources**. Large, computational-intensive or IO-intensive problems (biobanking [10], big data [11], data processing [7, 5, 12, 13]) usually need many resources to run on (CPUs, memory, storage). Thus, powerful clusters and Grids are needed by communities.
- **scheduling and resource management**. From efficiency [1], fairness across users [1], cost or energy optimizations [14, 15, 3], resource availability [16, 15], scheduling remains one of the most complex problems in computing architectures. In [1], the authors state that finding the most suitable scheduler for a given infrastructure is a complex task, while in [17] the authors state that scheduling is an NP-hard problem.
- **cost or energy efficient infrastructures**. Computing architectures tend to use many resources, especially energy which leads to higher costs for institutions or users (pay-as-you-go services). As such, researches towards creating a cost or energy efficient infrastructure [3, 18, 15] are made. In [18], the authors demonstrate that energy efficiency

can be improved by migrating the virtual machines across cluster nodes based on their proposed algorithm.

- **security and privacy**. A large topology with many users is exposed to security threats [19, 20, 21, 22, 23, 24].
- **support for research and learning activities**. Extensive studies show the need for cluster, Grid or cloud computing setups in universities or research environments [16, 25, 11, 24, 26, 10]. In [24], the authors present the advantages of using a cloud computing infrastructure in educational fields: from flexibility and availability to intensive computing and scalability. Moreover, [27] presents how OpenStack (a cloud management solution) can improve the learning activities; Various universities use e-learning solutions (e.g., Moodle, Blackboard) that were adjusted to better fit their needs [5, 28, 29, 30, 31, 32].
- **datasets and resources variety**. Infrastructures need to accustom and integrate or generate a variety of resources. In [33], the authors state that cluster workload (job traces) are needed by research groups to validate their algorithms; however, very few sources were available, thus, they propose new traces from four HPC clusters. Another study [7] uses various datasets for testing their parallel and distributed implementation for hyperspectral data.
- **support for job and application diversity**. Especially in education and research, multiple applications must be deployed or available [34, 27, 24, 35]. Furthermore, running multiple kinds of jobs helps researchers test their implementation in various environments [33, 4, 25].

2.3 Case Study: Computing and Network Architecture in UPB

Figure 2.2 shows an overview of the University Politehnica of Bucharest network infrastructure. The network has an entry point (main router with firewall) that connects UPB's internal network with the external network. The local network is split in multiple subnetworks that connects the following components: **Central Services** - identity management, name services (e.g., LDAP, DNS), dedicated security equipment, identity provider, single sign on solutions; **Research and Study Buildings** (laboratories, library, offices, classes); **Cluster** - heterogeneous resources that are used for computing [23, 4]; **personal devices** - the UPB campus has multiple access points to allow its users to connect to Internet.

Adhering to WLCG, UPB's cluster is part of a Grid infrastructure and shares its computing resources with the ALICE Experiment from CERN. It acts both as storage and computing element, proving the experiment almost 6PB of storage, 150 worker nodes, 2400 CPUs and almost 10TB of RAM.

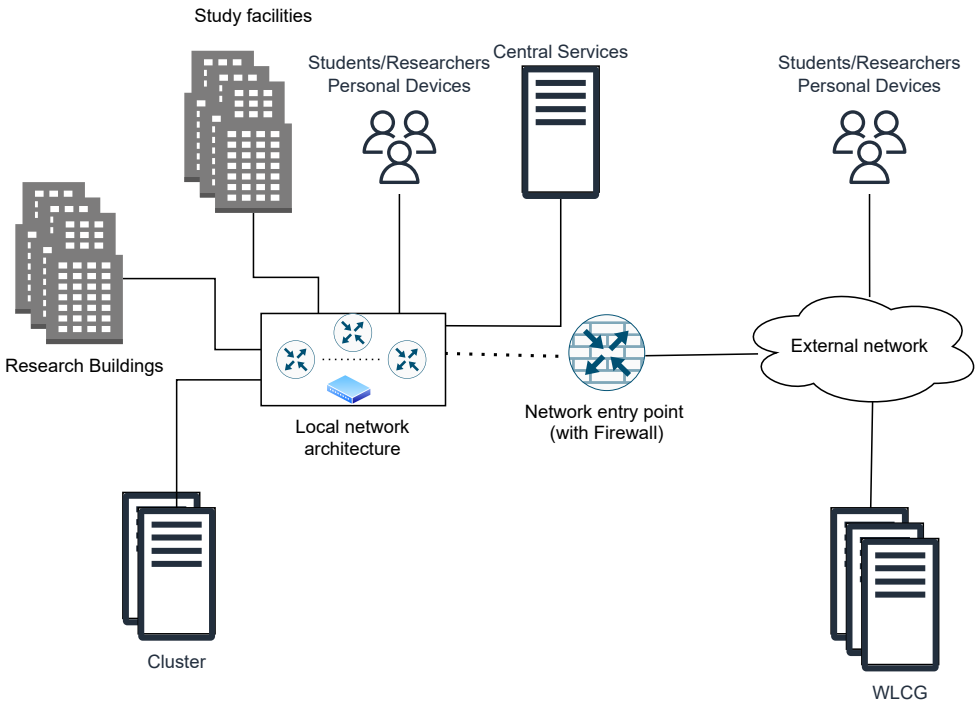


Figure 2.2: General view of UPB network architecture

The UPB cluster architecture is a heterogeneous one, having 8 CPU models (Intel Xeon E5 and Intel Gold) that are provided to the ALICE experiment, and various CPU models (AMD Opteron, AMD Quad, Intel Xeon) and GPUs provided to students and researchers. For orchestration, SLURM, OpenStack and Microsoft Hyper-V are used.

As presented in Figure 2.2, UPB has a complex network architecture. On its premises, it hosts management services (LDAP, DNS, mail, SSO, VPNs) and e-learning services (Moodle, OpenStack, wikis, generic sites). These services can be accessed by both internal and external users. Moreover, students, teachers, researchers and guests connect to the internal network from labs, course rooms or by connecting to the available WiFi access points.

Being a higher education facility, UPB has an e-learning platform, Moodle, used by more than 35000 users. Moreover, UPB has 15 faculties, each with multiple domains, directions, series and course structures. On average, in UPB's Moodle platform, around 11000 courses are created. Thus, a manual course generation means a tremenduos work and is not feasible (not enough human resources). Thus, an automatic management for the e-learning platform is needed.

2.4 Case study: Distributed computing architecture in ALICE, CERN

The ALICE Grid is composed of various components that work together to allow users to submit jobs and Grid sites (and consequently worker nodes) to execute those jobs. Figure 2.3 presents an overview of the ALICE Grid architecture and the interactions between its elements [36, 37,

13, 38, 12, 39, 40]. The architecture has three main logical components: the **Central Services**, which manage and monitor the Grid activity, **Grid Sites** which represent the Grid's resource providers, including the wide area network, and **Users** which submit jobs on the Grid.

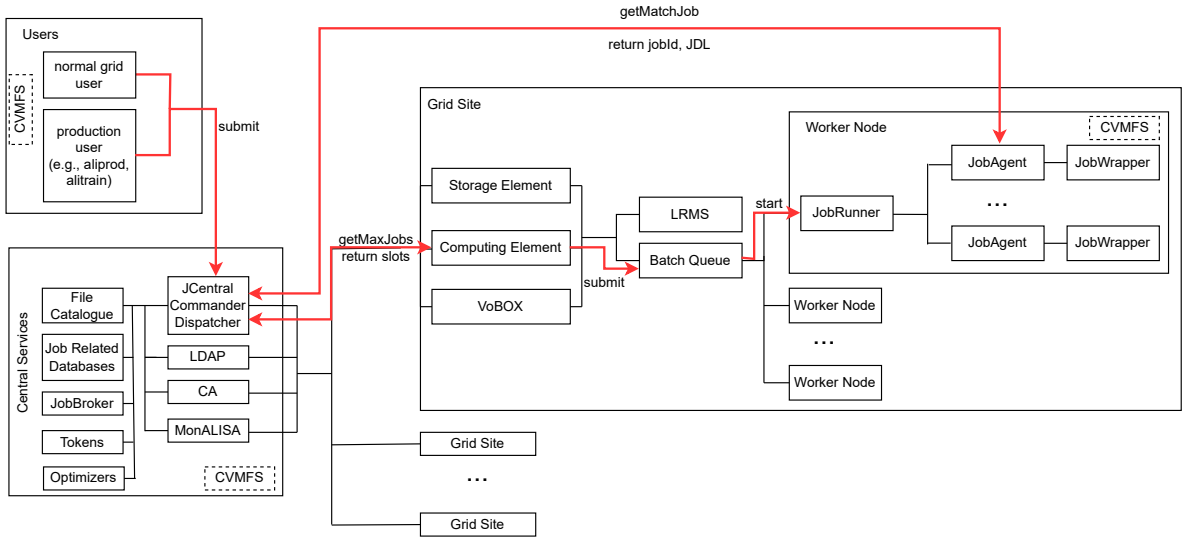


Figure 2.3: ALICE Grid Architecture

The Central Services run on the CERN premises and have various components with which the user can directly interact such as JCentral, LDAP, CA, MonALISA. These services are also a gateway to key modules such as a File Catalogue, Job related databases or tokens. The Central Services components that are presented in Figure 2.3 are the following: **JCentral** - the gateway command dispatcher [39, 37, 13]; **File Catalogue** - a module that annotates the persistent data (files and metadata) on the Grid; **Job Related Databases** - set of databases containing information necessary for the job management.; **JobBroker** - the scheduler; **Optimizers** - modules that analyses the job databases and runs various optimizations; **Tokens** - module that manages the token (X.509 based certificates) of the jobs;**LDAP** - contains configurations and information for the sites and the users.; **CA** - a Certificate Authority that signs and validates the X.509 certificates used for authentication, authorization and access; **MonALISA** - a monitoring and accounting framework; **CVMFS (CERNVM File System)** - a distributed file system that contains the experiment software.

The ALICE **users** (from Figure 2.3) are divided into two types: individual users and production Grid accounts. Each user has quotas on maximum disk space and maximum number of jobs submitted and running. Production accounts are used to run the large loads on the Grid: simulation, reconstruction of experimental data and organized analysis.

A production is a class of jobs of the same type that needs to be executed on the Grid. Each production has a unique tag (e.g., "LHC23i1" or "LHC23d6b") to allow for easy identification and is "anchored" to a specific data-taking interval with identical conditions. In a simulation production, each run within a period is represented by a single job named master job. A master job is identified by a JDL directive "Split" and numerical argument, which directs the Job Optimizer module to split it in the required number of subjobs. For each subjob, a new JDL

file is automatically generated starting from the original JDL file. The subjobs's JDL is almost identical to the master job's JDL except some contextualization data that specifies the job's seed number (the initial number of the pseudorandom number generator for the Monte Carlo jobs) and the output directory. This particularity of practically identical job generation within a production makes possible the analysis presented in Section 3.1.

The most important checkpoints (states) of a job along its lifetime on the Grid are the following: *INSERTED* - the job was added in the QUEUE database; *RUNNING* - the job is running the executable specified in the JDL; *DONE* - The job is completed (without errors); *ERROR_** - various errors.

The ALICE experiment has around 60 sites that pool their resources into the Grid. **The Grid sites** have complex architecture (see Figure 2.3) with multiple modules, as follows: **Storage Element (SE)** - a module that manages the storage resources provided by the site; **Computing Element (CE)** - a module that manages the job related activities on the site.; **VoBOX (Virtual Organization BOX)** [41, 37] - services that manage the site's accounting and monitoring activities within an experiment by directly interacting with the CE and receiving the necessary information from the running jobs. ; **Local Resource Management System (LRMS)** - a module that handles the available resources and manages the worker nodes.; **Worker Nodes (WN)** - the computers which execute jobs on a site.

While the CE submits requests to the local job scheduler comprising of the LRMS and the Batch Queue, the worker node starts a **Job Runner**, which is the JAliEn module managing the job execution on the WNs [12]. The JobRunner receives a 24-hour certificate to directly communicate with the Central Services. Once started on the WN, the Job Runner launches one by one **JobAgents**. Each JobAgent receives a job from the Central Services and starts a **Job Wrapper** which sets up the environment, mounts CVMFS and starts the job execution. The JobWrapper, if allowed by the system, uses containerization to isolate the payloads.

The certificate generated for the Job Runner and its Job Agents also takes into account the time slot the Central Services requested from the site. Since the site can be part of multiple virtual organisations (VOs), i.e., sharing the same resources between multiple projects, the slot requested by the Central Services must be used efficiently. Thus, the central services cannot end a Job Runner before the deadline, nor let it run empty when there are no matching jobs, since it would reduce the efficiency of the requested Grid resources. The 24-hours interval was chosen as optimal in which the experiments can do their work without blocking the site and the other projects.

SiteSonar [42, 43] is a module that probes the Grid site and host configurations. Based on a series of scripts published in CVMFS, SiteSonar collects information from each host that runs a job. The collected information is sent to the Central Services and kept in a database.

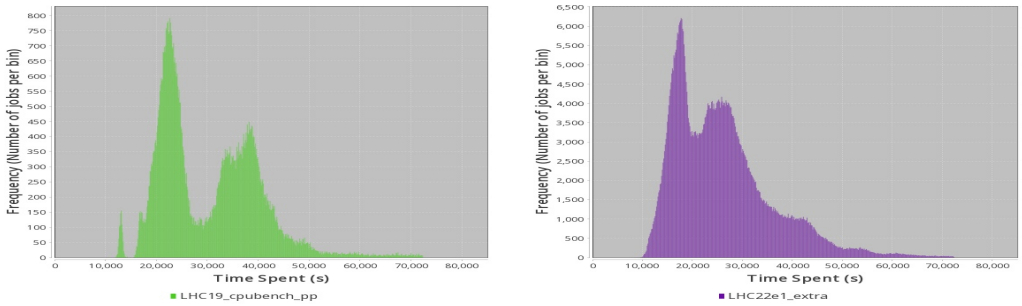
Table 2.1: Statistics of Grid activity during one week generated with data extracted from Site Sonar. The table presents the number of distinct considered keys from the Grid.

Hosts	Sites	Operating Systems	CPU Model Name
11676	55	15	131

Table 2.1 is based on data extracted using Site Sonar and present statistics during a week of grid activity. This table showa that the Grid's resources are diverse, highlighting its heterogeneity.

While collecting and monitoring the Grid activity, it was observed that Job Agents on some sites do not match any job from the Grid, even though they still have left more than half of their certificate validity. This happens since most of the jobs require almost all the time frame that can be assigned for a job (e.g., 72000 or 80000 seconds, almost 20 and 22 hours, respectively). However, it was observed that most jobs do not use all the TTL they require. Nevertheless, from the Grid perspective, reducing the production global TTL is not a valid approach: jobs must finish their execution regardless of the worker node configuration they may land on. Considering the heterogeneity of the grid and the diversity of jobs (some are CPU intensive, some require data from the local storage, some require data from remote storage), lowering the TTL for an entire production without a proper analysis may lead to an inefficient use of the Grid and increased job failures.

Figure 2.4 presents the jobs' time spent distribution for two productions running in the ALICE Grid. The figure highlights the wide range of time values jobs take to execute: from 10000 to 72000 seconds in the case of these two productions. Thus, the production's TTL cannot be empirically reduces to a smaller value since it would kill jobs.



(a) LHC19_cpubench_pp (b) LHC22e1_extra

Figure 2.4: Time spent by jobs running in two MonteCarlo Production on the ALICE Grid. The values for time spent range from 10000 to 72000 (the latter is the required TTL).

Taking into consideration the results from Figure 2.4, we need to improve the scheduling mechanism to better fit the jobs actual execution times. This paper focuses only on MonteCarlo jobs since they are CPU-intensive jobs and no I/O interaction influence the execution time. Nevertheless, improving the scheduling mechanism is a non-trivial task since it was observed that the same MonteCarlo jobs can have widely different values of execution time (shown in Figure 2.4). Thus, we must first determine all factors that influence the job execution time. Then, we can develop and propose an estimation algorithm to predict the maximum real job duration.

2.5 Proposed improvements for computing architectures

Based on the identified needs, the thesis proposes solutions that improve:

- **scheduling in Grid environments** through the algorithm that is applied for MonteCarlo productions in the ALICE Grid. However, the algorithm can be applied in other clusters as well, as long as the suitable key is found.
- **support for research and learning activities** through the improvements added for the course generation and students' enrolments in Moodle and the improvements in the setup of running FreeBSD in cluster and cloud environments.
- **energy efficient infrastructures** through improvements for the live migration feature for bhyve. As stated in [18], live migration can be used for load-balancing which leads to energy efficient clusters.
- **security** through adding improvements for bhyve, the FreeBSD's hypervisor and through proposing network architectures and generating datasets for validating the functionality of machine-learning based security tools.
- **dataset variety** through creating new network attacks datasets.

Chapter 3

Job Scheduling Optimizations for Monte Carlo simulation jobs in Heterogeneous Grids

The first step in optimizing Monte Carlo simulation job scheduling by estimating a TTL value based on job execution history in the Grid is to study existing productions and determine the factors that influence job execution time. This chapter presents the analysis of the Monte Carlo productions within the ALICE Grid. Based on the identified factors and a proposed estimation algorithm, the TTL is estimated and the results obtained when the algorithm is applied to real data extracted from the ALICE Grid are presented.

3.1 Monte Carlo Job Duration Profiling in Heterogeneous Grids

From a Grid perspective, they are CPU-intensive jobs and the minimal amount of I/O operations does not influence the execution time. Therefore, we consider that there are three types of factors that may impact a MonteCarlo job performance: **CPU and hardware configuration** - since the jobs are CPU-intensive, it means that they will mainly be influenced by the CPU/hardware type; **software and site configuration and activity** - even though the jobs are running on identical hardware, the software configuration layer may add overhead or improve execution on different site, and sites can be part of multiple VOs that can simultaneously request resources on the site's Grid; **production features** - over time, the software version of the executable can change, as well as its arguments (e.g., the number of events), impacting the job performance.

In order to have a better view of the global production behaviour, we must extract many jobs and statistically analyze their individual behaviour. Since the data represents identical jobs, its statistical representation should be in form of a normal distribution, baring normalization factors.

To create our data sample, we have chosen two productions with many jobs running on the Grid at the time of the analysis (`LHC19_cpubench_pp` and `LHC22e1_extra`). For these productions, we extracted the jobs that finished their execution with the status `DONE`. the job traces (logs from the job execution), the actual time the job has run, the site and the host it has landed on, and the description for the host from SiteSonar. From SiteSonar, we extract the host CPU configuration (CPU Model Name, CPU family, frequency, number of CPU execution units, number of cores), the job management tool (e.g., HTCondor or Slurm), the containerization tool (e.g., Docker, Singularity/Apptainer, no containerization), if virtualization is used and patches are applied to mitigate various CPU vulnerabilities (e.g., Spectre/Meltdown).

To preserve privacy, the data in this paper is anonymized by assigning a generic name to each site (e.g., SiteXYZ). Table 3.1 presents the data set we have used for this study.

Table 3.1: Dataset used for this paper

Production	**Date & number of jobs extracted**		
LHC19_cpubench_pp	**June 2022** 77623	**July 2022** 82733	**Oct 2022** 858532
LHC22e1_extra	**October-November 2022** 634088		**April 2023** 66930

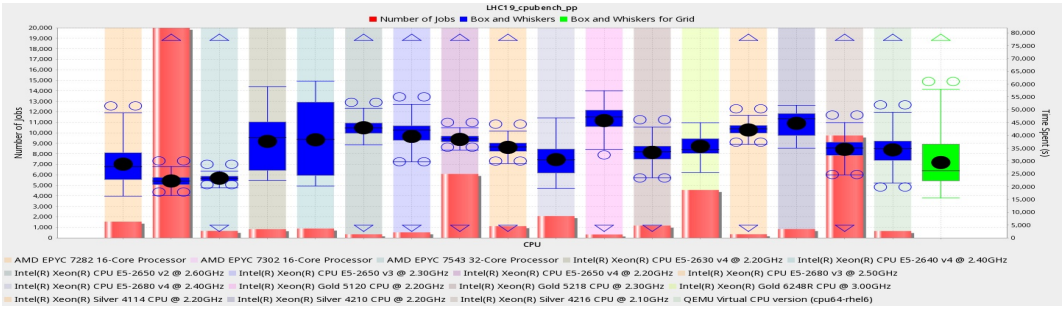
To analyse the data samples (Section 3.1), we use two types of graphical representation: **Histogram** plots[1] - created for a given subset of data with the end goal to determine the shape of the data distribution; and **Box & Whiskers** plots[2] - for given subsets of data, we create box & whisker plots as they show various data characteristics in a compact format (data distribution, quartiles (boxes) based on the median values, the median (horizontal line), the mean (black dot) and the outliers).

The research aims to find the relevant keys to split the data in subsets that have a normal distribution and their box & whisker representation shows little to no outliers, with compact quartiles - the subset is concentrated in a narrow range of values. With these conditions fulfilled, we can conclude that the duration of the jobs can be estimated using the same heuristics and the estimated TTL value can fit all jobs. For the plots presented in this section, we use the following notation: “Time spent” - the job execution time in seconds, ‘H-ON’/‘H-OFF’ - whether CPU hyperthreading is activated. The features we considered for this section are the **CPU**, **hyperthreading** and **site/host configuration**.

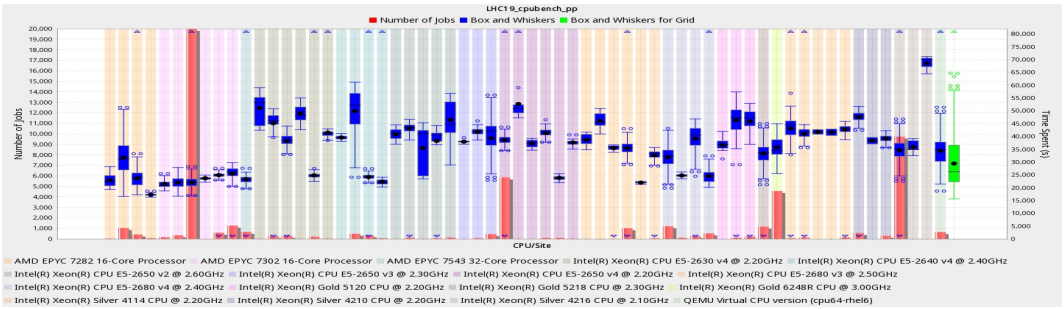
Figure 3.1 presents the box & whisker representations for the jobs split by *CPU* (3.1a) and by *site and CPU* (3.1b). All the CPUs from Figure 3.1a were split by site and presented in Figure 3.1b , while keeping the same colour scheme and order. Due to the large number of CPUs and

[1] https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/data-presentation/histograms.html, Online, Accessed 25th of July 2023

[2] https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/data-presentation/box-and-whisker-plots.html, Online, Accessed 25th of July 2023



(a) Production/CPU



(b) Production/Site/CPU

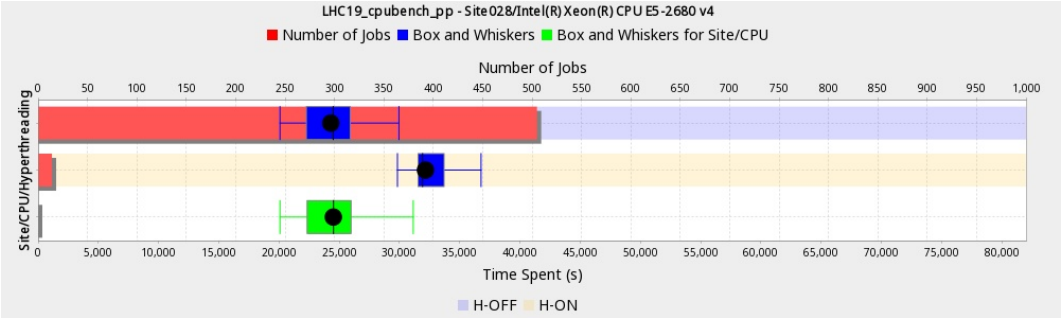
Figure 3.1: LHC19_cpubench_pp: Box & Whiskers for datasets where keys are *production/CPU* and, respectively, *production/site/CPU*. Data is in alphabetical order by CPU. The red bar charts represent the number of jobs of each subset (left y-axis), the blue boxes & whiskers are the job's time spent representation (right y-axis), while the green representation is the whole Grid. Each CPU model name has a different colour that is expanded per site in subfigure (b). The empty circles represent the outliers, while the triangles show that datasets have far out/distant outliers.

sites, we cannot show the full chart, but only a subset of the results. The rest of the charts are similar to the ones shown.

Each chart also contains the box for the whole Grid (green box) and bar charts for the number of jobs considered. The figures show that the boxes and the whiskers are tighter when taken into account the site, thus, we can conclude that the performance is influenced by the site configuration. Another important aspect that can be observed is that the CPUs do not have a uniform behaviour: while in some cases, the CPUs behave the same across sites, in others there is a large performance gap between sites.

Figure 3.2 shows the impact that hyperthreading has on the MonteCarlo job execution. We analysed the same site (Site28) and same CPU model, but different hyperthreading configuration. As presented in the charts, enabling hyperthreading impacts the MonteCarlo job execution. This behaviour is the same described in literature [44, 45, 46, 47]. It is worth mentioning that hyperthreading may benefit other job types, for example I/O intensive. Thus, we must model the proposed estimation algorithm to take into account hyperthreading.

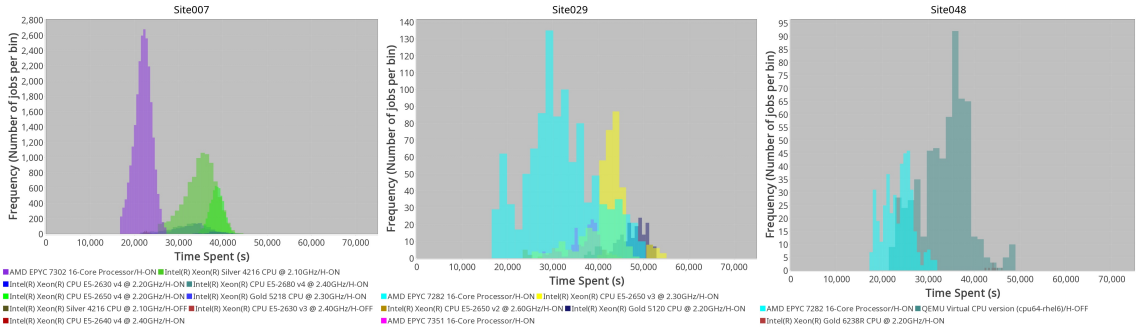
Figures 3.3 and 3.4 depict histograms of job duration for the two analysed productions. As observed, split by production, site, CPU model name and hyperthreading, the job datasets show normal distributions. As seen in Figures 3.1 and 3.2, the datasets have outliers, which we chose to remove on the histogram plots. Due to the large number of sites and CPU models, we chose



(a) LHC19_cpubench_pp

Figure 3.2: Box & Whiskers for Site28. The red bar charts represent the number of jobs of each subset (top x-axis), the blue boxes & whiskers are the job's time spent representation (bottom x-axis), while the green representation is the whole Grid. `H-ON` means that hyperthreading is enabled, while `H-OFF` means that hyperthreading is disabled.

to show only the most relevant charts illustrating the main behaviours and use-cases. The rest of the charts are similar to the shown.



(a) LHC19_cpubench_pp/Site007 (b) LHC19_cpubench_pp/Site029 (c) LHC19_cpubench_pp/Site048

Figure 3.3: Job duration distributions divided by CPU model name and hyperthreading and group by site name.

Figure 3.3 shows how the job duration is influenced by the CPU model on a given site. This is the expected behaviour since these jobs are CPU-intensive and each CPU model has its specific hardware configuration.

Figure 3.4 shows how the job duration on the same CPU model behaves on multiple sites. While on some sites, for the same production, the datasets have the same distributions for the same CPU model name (e.g., Figures 3.3c), other datasets have different distributions (e.g., more wide time range - Figure 3.3a, or two clearly separated distributions.

The abnormal cases encountered were analyzed separately to determine if there are other factors influencing the execution time of the jobs, other than those considered so far. Local site job management applications, virtualization, containerization solution, as well as other hardware aspects (hardware vendor, RAM, NUMA configuration of sites, etc.) were considered.

After the analysis, there is no other factor that alone influences the execution, but the execution is influenced by an accumulation of factors. However, these cases are usually isolated, and the number of possible existing combinations is too large to be considered for TTL estimation.

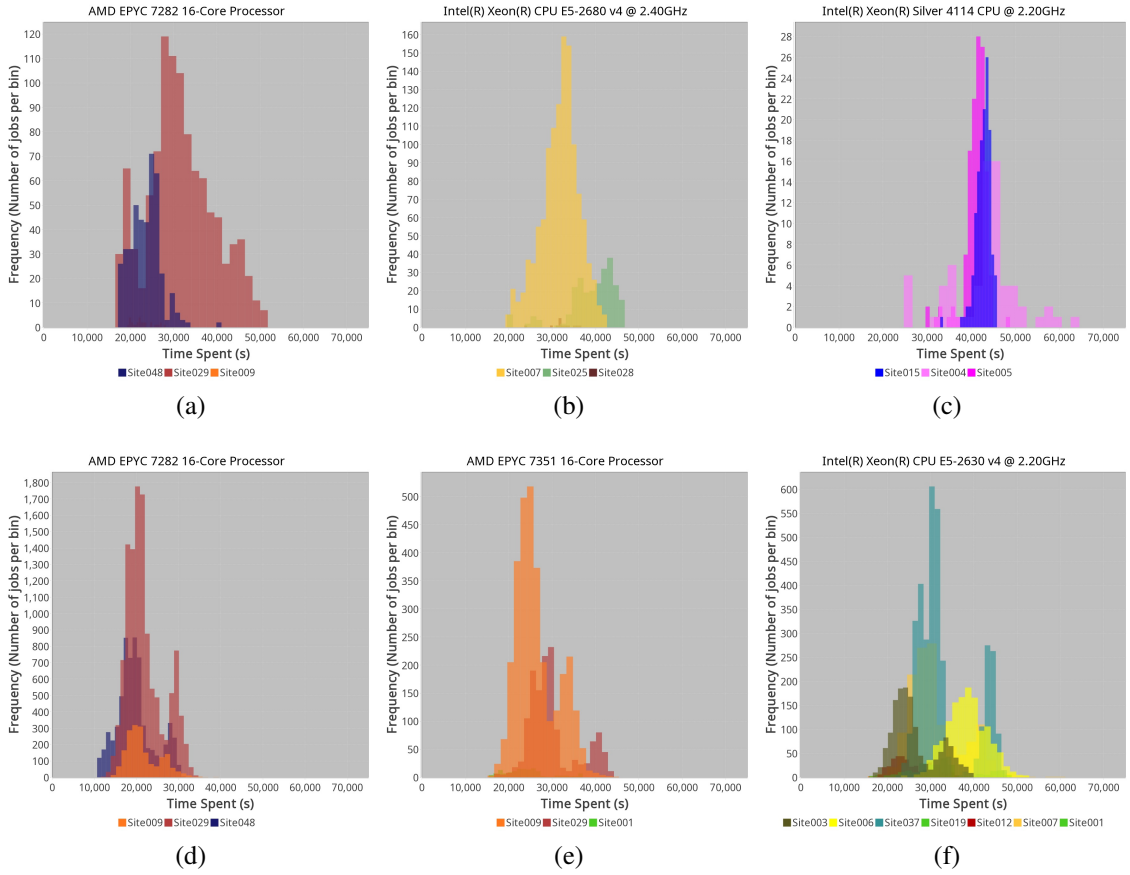


Figure 3.4: Job duration distributions divided by site name and group by CPU model name and hypethreading. Histograms 3.4a, 3.4b and 3.4c are for LHC19_cpubench_pp, while 3.4d, 3.4e and 3.4f are for LHC22e1_extra. For all presented CPU model names in this figure, the hyperthreading is on.

Execution nodes can also be considered to have similar configurations, achieved through configuration management applications such as Puppet and Ansible. Thus, the site name covers most of the configurations tracked in this analysis.

Another analysis performed on the data sets is related to the behavior of production over time. Based on the analysis, there are isolated cases when, after 5-6 months of running a Monte Carlo production, its behavior changes. In the analyzed case, the production parameters changed, doubling the number of generated events. As a result, the execution time of a job has doubled. These isolated cases indicate that productions are dynamic, and running the same production at large time intervals may also mean changing execution parameters.

3.1.1 Discussion regarding MonteCarlo productions activity

In this section, we present an advanced analysis of the MonteCarlo productions. We consider two large productions (`LHC19_cpubench_pp` and `LHC22e1_extra`) for a better understanding of the Grid behaviour. We draw the following conclusions regarding the execution of MonteCarlo jobs:

- sites are configured to have a good overall performance for both CPU-intensive and I/O-

intensive jobs.

- As expected by the benchmark tests[1], the CPU model and its hyperthreading configuration influences the job execution - Figures 3.1 and 3.2.
- the site configuration depends upon site administrator practices and no two sites are identically configured: vendors, inter-connectivity, CPU models and cores, RAM, operating system and fine-tuned system software parameters - Figures 3.4, 3.3.
- same CPU model may behave differently depending the site configuration - Figure 3.4.
- the production parameters impact the job behaviour - each production behaves differently; a production may have particularities.
- productions behaviour is dynamic - even though some stay the same over the time, others change their behaviour, depending on their configurations.
- modelling the job behaviour with large number of parameters leads to a large number of datasets with limited statistics, insufficient for robust estimators.

Taking the above into account, we consider that the most suitable key for splitting jobs in datasets are the production cycle, site, CPU Model and hyperthreading status.

It is worth mentioning that adding the host name into the key may give better results. We started from the assumption that the site hosts are uniformly configured. However, due to various reasons, it may happen that a host has a custom configuration (i.e., boot option, different RAM setup) which alters the distributions. During our tests, adding host name as a parameter results in a more robust result for the estimator. This additional parameter has to be weighted with the available statistics for a given dataset.

3.2 Monte Carlo Job Duration Prediction

Based on the analysis presented in Section 3.1, we consider that the production cycle, site, CPU model and the hyperthreading configuration are the primary parameters we can use to estimate the job duration. In this section, we refer to the aforementioned fields as key. Based on the histograms presented in Section 3.1, we consider that for a large number of entries in the datasets split by the key, the data has a normal distribution and we will use the normal distribution properties in our proposed algorithm.

When altering the job's TTL, we must take the following requirements into consideration:

- **R1**: the algorithm must estimate the maximum value of the job' duration to prevent a premature termination of the job. If the TTL we set is too low (e.g., the median), the jobs

[1]HEPiX, HEPScore23, Online: https://w3.hepix.org/benchmarking/scores_HS23.html, Accessed 17th November 2023

will be killed early, wasting the computing time invested up to that point, thus the Grid resource utilization efficiency will suffer.

- **R2**: the algorithm must work with irregular or bi-modal distributions. The majority of the datasets we have observed and that have a large number of jobs have a normal-shaped distribution. However, the proposed algorithm should accommodate abnormal cases as well.
- **R3**: the algorithm must be fast. On the ALICE Grid, tens of jobs are scheduled every second. Having an algorithm than spends too much time to find the best match for the site will affect the Grid performance.
- **R4**: the algorithm must not use a large number of resources. Thus, the algorithm must dynamically update the saved resources when a job finishes its execution.
- **R5**: the algorithm must keep its correctness when implemented (transposed to code instructions). When working with large datasets with values between 0 and 86000, computing errors may occur (e.g., catastrophic cancellation).
- **R6**: the algorithm must adapt to the Grid dynamism. Sites, hosts and productions configurations change over time. The proposed algorithm must take into account these changes.

The algorithm we propose works as follows: in the process of job matching, based on the key and the history of the previous jobs, we can determine a running maximum value of the job and, if matching the current requirements, schedule it on the host with a modified TTL. When a job finishes its execution successfully, based on the key and actual running time, the estimation metadata will be updated with the new values.

Starting from the assumption of a standard distribution, we can determine the interval of the most probable job duration..

Let D_k be the datasets formal definition (Equation 3.1) where each $x_i \in D_k$ represents the time spent by a successful MonteCarlo job on the Grid and k is the datasets key: for a given key k (e.g., "Production P/SiteXYZ/CPU Model T/hyperthreading is on"), we create a dataset with the values of the time spent by the MonteCarlo jobs that successfully finished their execution (their stats is `DONE`) and match on the key k (e.g., jobs that are part of Production P, run on hosts from SiteXYZ that have CPU Model T as CPU Model Name and for which the hyperthreading configuration is on).

$$D_k = \{timeSpent(job) \mid job \in MCJobs, status(job) = DONE, key(job) = k\} \tag{3.1}$$

Taking into account normal distribution's properties and the dataset's formal definition (Equation 3.1), we propose to estimate the maximum TTL of the jobs that will match this dataset is

given by the formula presented in Equation 3.2, where σ is the dataset's standard deviation.

$$eTTL_k = max(D_k) + 2 * \sigma \tag{3.2}$$

Since we consider two standard deviations from the maximum value, the estimation accounts for bi-modal and skewed distributions. Moreover, we add 2 standard deviations above the datasets maximum value to account for the model approximation. This allows jobs that might take longer to complete. When updating the dataset, i.e. new job matching the key and the jobs time spent is added to dataset, the estimation metadata (mean, standard deviation and maxim) must be updated as well.

The assumptions made work only for sufficiently large sample of jobs ($|D_k| = n$). If n is too small, the estimated TTL may be too low, resulting in a premature job termination. Thus, we need to weight the estimated TTL with the number of jobs with the confidence increasing as the number of jobs in a dataset increases.

We propose weighting the estimated TTL with the number of jobs and the initially requested TTL (reqTTL). This method allows us to have a smooth transition from the original TTL to the estimated one. The generic weighting formula is presented in Equation 3.3.

$$\begin{aligned} wTTL_k &= w_0 * reqTTL_k + w_1 * eTTL_k \\ w_0 &= \frac{f_0(n)}{f_0(n) + f_1(n)} \\ w_1 &= \frac{f_1(n)}{f_0(n) + f_1(n)}, \text{where } |D_k| = n \end{aligned} \tag{3.3}$$

Even though there are numerous functions that can be used for weighting, for simplicity, we choose the functions presented in Equation 3.4. The two functions allow that after 50 jobs, the modified TTL to be the mean between the estimated TTL and the original TTL. After 100 jobs, the original TTL weight decreases to one third of the modified, while after 1000 jobs, the estimated TTL will have a weight of 95%.

$$f_0(n) = 50, f_1(n) = n, n \in \mathbf{N}, n > 0 \tag{3.4}$$

We can generalize the previously presented equations if more than one key is considered (i.e., combining multiple keys for the final TTL). Let $key_i, i = 1..m]$ be the key functions taken into account for estimation, where m is the number of key functions. We consider $D_{i,k}$ the dataset generated for key k when applied key function key_i as defined in Equation 3.5. Each dataset contains the time spent by the MonteCarlo jobs that run on hosts that matches key k.

$$D_{i,k} = \{timeSpent(job) \mid job \in MCJobs, status(job) = DONE, key_i(job) = k\}, i = 1..m. \tag{3.5}$$

Equation 3.6 presents the general formula for weighing the TTL requested by a job when multiple keys are considered. The formula takes into account the number of jobs contained by each dataset.

$$\begin{aligned} wTTL_{key_{1..m}} &= w_0 * reqTTL + \sum_{i=1}^{m} w_i * eTTL_{i,k} \\ w_0 &= \frac{f_0(\sum_{i=1}^{m} |D_{i,k}|)}{f_0(\sum_{i=1}^{m} |D_{i,k}|) + \sum_{i=1}^{m} f_i(|D_{i,k}|)} \\ w_i &= \frac{f_i(|D_{i,k}|)}{f_0(\sum_{i=1}^{m} |D_{i,k}|) + \sum_{i=1}^{m} f_i(|D_{i,k}|)} \\ eTTL_{i,k} &= estimation(D_{i,k}) \end{aligned} \tag{3.6}$$

While implementing the algorithm, we must assure that the requirements are met. **R1** and **R2** are satisfied by computing the final modified TTL using the formulas presented in Equations 3.2 and 3.3.

To satisfy **R3**, **R4** and **R5** which require that the algorithm to be fast and to use a limited number of resources, we use the following implementation choices: use Welford's algorithm for computing mean and variance [48, 49]; use databases/tables and in-table operations; the estimated TTL will be computed in advance when a job successfully finishes its execution; use in-memory short-term caches to keep the last updated estimated TTLs.

To satisfy **R6**, an additional tool is implemented that checks the Grid activity and, if a production did not run for some time, its estimation metadata is removed from the database.

Algorithm 1 Modify job TTL based on estimation

```
1: k = key(j);
2: if isMonteCarlo(j) and existsEstimationMetadata(k) then
3:     eTTL = getETTL(k);
4:     n = getNum(k);
5:     reqTTL = getReqTTL(j);
6:     wTTL = getWeightedTTL(eTTL, reqTTL, n);
7:     setNewTTL(j, wTTL);
8: end if
```

Algorithm 2 Updating key metadata when a job finishes its execution

```
1: if isMonteCarlo(j) and jobStatus(j) == DONE then
2:     k = key(j);
3:     timeSpent = timeSpent(j);
4:     updateETTL(k, timeSpent);
5: end if
```

Algorithms 1 and 2 show the pseudocode used for implementing the proposed algorithm.

3.3 Job Scheduling Improvements for Monte Carlo simulation jobs in ALICE

For testing, we prepared a proof of concept application that simulates the Grid activity: the jobs extracted using the previously described method were chronological sorted. Each job that would have run on the Grid receives an estimated TTL that is saved in a database. Its run time value is used to compute the estimation metadata as described in Section 3.2. At the end of the simulation, the predicted TTL with the actual time the job spent is compared. Our goal is to reduce the required TTL and achieve as low as possible job termination rate.

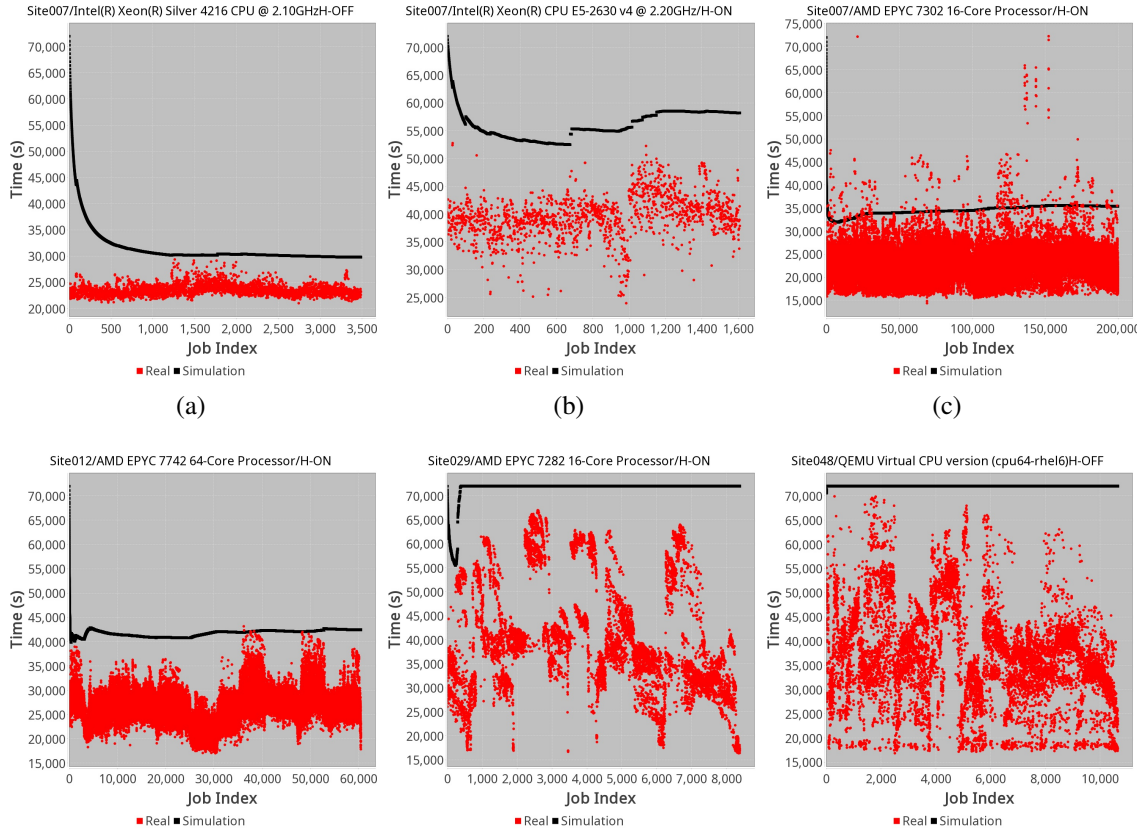


Figure 3.5: Simulation results for LHC19_cpubench_pp. The dataset used is from October 2022 (Table 3.1)

In this section, we present the results on the datasets we extracted in October 2022 (`LHC19_cpubench_pp`) and October-November 2022 (`LHC22e1_extra`). We choose the two datasets because they contain the largest number of jobs we have extracted for each production, resulting in multiple keys. We present the relevant plots we have obtained (different behaviours). We represented the time spent by the jobs with red and the modified TTL with different colours, depending on the key considered: black for (production, site, CPU and hyperthreading configuration) and blue for (production, site and hostname).

In an ideal case, all the red jobs should be below the estimation lines, as close as possible to it. Nevertheless, some datasets have outliers that go above the modified TTL. In this case, we

aim to have a small number of jobs that would have been killed if the TTL was modified as indicated.

Figure 3.5 presents the simulation results we have obtained for `LHC19_cpubench_pp`. Figures 3.5a, 3.5b, 3.5c and 3.5d show how the modified TTL decreases from 72000s (set in the job's JDL) to a smaller value, while Figures 3.5e and 3.5f depicts how the algorithm reacts when the data is too wide distributed (we keep the original TTL in this case). Figure 3.5f presents a generic virtualized CPU (`QEMU Virtual CPU`), thus various CPU models may run these jobs. Figure 3.5d shows that some jobs would have been killed by the modified TTL. However, the dataset we plotted contains around 200000 jobs, while the number of jobs that would have been killed is around 400 (2‰), of relatively low impact considering the optimized scheduling of the rest of the jobs.

Figure 3.6 show the simulation results we have obtained for `LHC22e1_extra`. While Figure 3.6a contains almost 100000 jobs (with almost 70 killed jobs, less than 1‰), Figure 3.6b contains almost 750 jobs, showing that the proposed algorithm works for smaller datasets as well. Furthermore, Figures 3.6b and 3.6c present the bi-modal behaviour we have observed from the histograms from Section 3.1 (the two red "bands" from the chart). Nevertheless, the proposed algorithm does not kill jobs and reduces the job's TTL from 72000s down to almost 36000s and, respectively, 44000s.

(a) (b) (c)

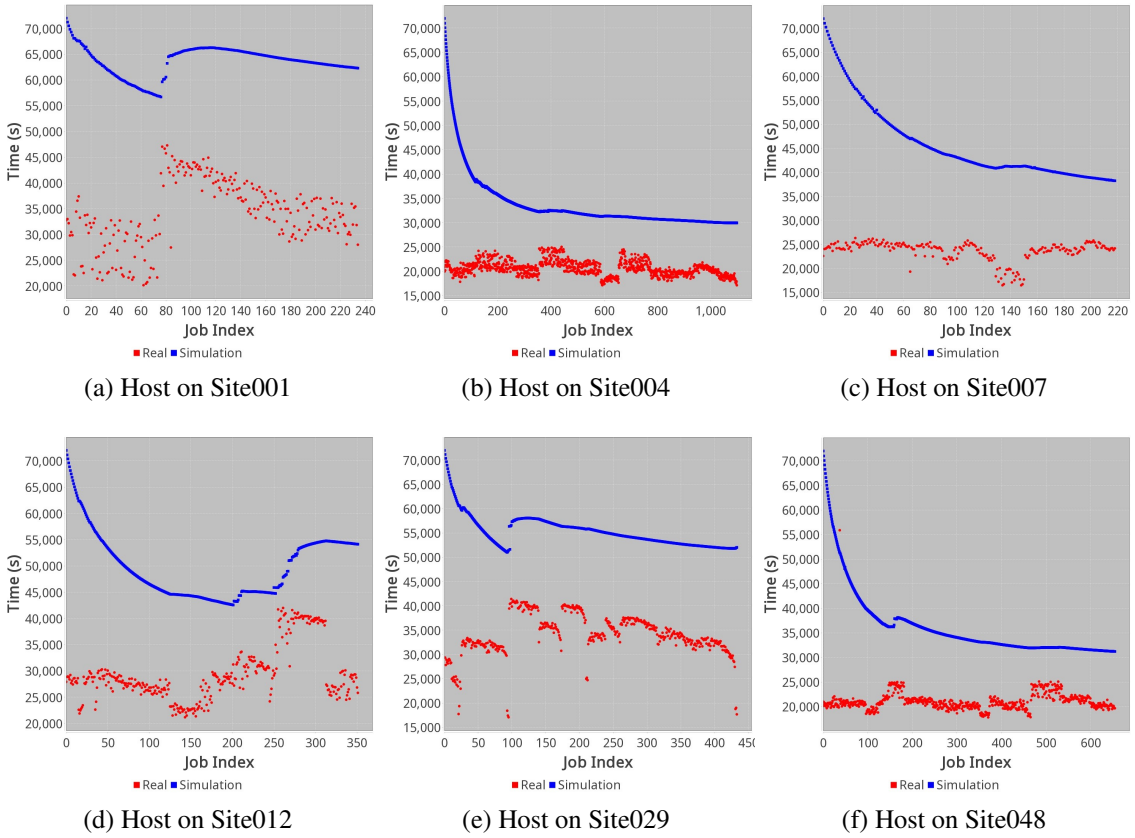
Figure 3.6: Simulation results for LHC22e1_extra. The dataset used is from October-November 2022 (Table 3.1)

Table 3.2: Overview on the number of jobs used for simulation and their results

Production	**Jobs**	**Killed by modified TTL**
LHC19_cpubench_pp	858532	735
LHC22e1_extra	634088	209

Figures 3.5 and 3.6 show that the overall results of the proposed algorithm are good. Using this algorithm, we can reduce the jobs TTL by up to 50%. This means that the MonteCarlo jobs that will have their TTL modified by the proposed algorithm may be scheduled on JobRunners

that would not have anything else to run otherwise. Furthermore, the number of jobs that would have been killed by the algorithm remains under 1‰(Table 3.2).



(a) Host on Site001 (b) Host on Site004 (c) Host on Site007

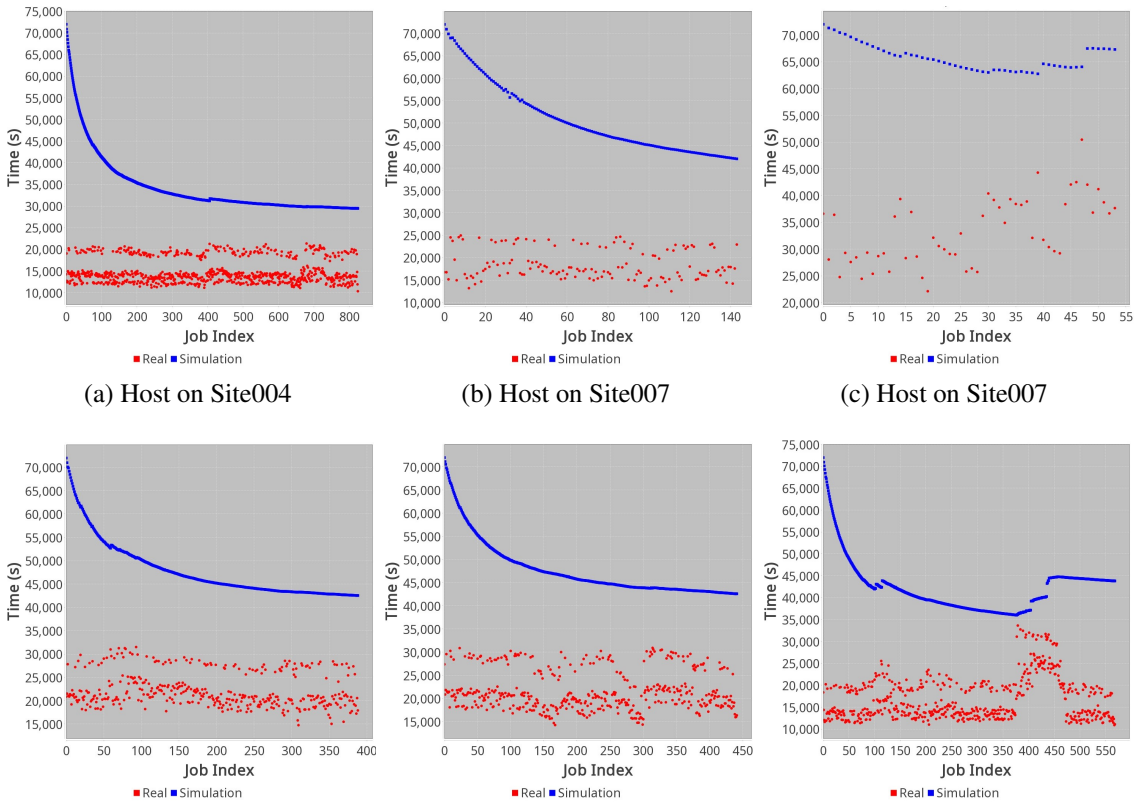
(d) Host on Site012 (e) Host on Site029 (f) Host on Site048

Figure 3.7: Simulation results for LHC19_cpubench_pp when only production, site and host are considered for estimation. The dataset used is from October 2022 (Table 3.1)

Figures 3.7 and 3.8 presents the results we obtained by applying the proposed algorithm when considering (production, site and host) as key, for productions `LHC19_cpubench_pp` and, respectively, for `LHC22e1_extra`.

Figures 3.7b, 3.7f, 3.8a, 3.8d, 3.8e shows that the proposed estimation minimizes the required TTL and converges almost to a constant value. Figures 3.7a, 3.7d, 3.8f shows that the algorithm reacts to changes in host's performance. Moreover, Figures 3.7f and 3.8f presents the behaviour for virtualized hosts on Site048. Compared to the results obtained for the same site when considering (production, site, CPU and hyperthreading configuration) (3.5f), we observed that the current considered key has better results.

Table 3.3 presents the number of jobs that would have been killed if the required TTL were modified with the proposed algorithm and key. Compared to the results obtained in Table 3.2 for (production, site, CPU, hyperthreading configuration), using (production, site, host) as key kills fewer jobs. Nevertheless, it should be mentioned that, in the current case, the estimation converges slower, thus the modified TTL will be closed to the required TTL.



(a) Host on Site004 (b) Host on Site007 (c) Host on Site007

(d) Host on Site012 (e) Host on Site029 (f) Host on Site048

Figure 3.8: Simulation results for LHC22e1_extra when only production, site and host are considered for estimation. The dataset used is from October-November 2022 (Table 3.1)

Table 3.3: Overview on the number of jobs used for simulation and their results for `production, site and host`

Production	**Jobs**	**Killed by modified TTL**
LHC19_cpubench_pp	858532	321
LHC22e1_extra	634088	154

During extensive testing (on more than 7 million jobs grouped in 180 productions from the ALICE Grid, from which only 40 were Monte Carlo), we observed that the algorithm has good results on I/O intensive productions as well (due to the uniform storage configuration on sites), showing that the proposed algorithm works as well as the similarity key is correctly determined.

While all proposed keys and keys combinations have good results, we must choose the most suitable approach to be implemented in the job scheduler. The first proposed estimation (production, site, CPU, and hyperthreading configuration) converges fast in the majority of the cases. The second proposed estimation (production, site and host) converges slow, but has a better accuracy for the sites with undefined behaviour for the first key (e.g., CPU Model Name is hidden by virtualization). Moreover, combining the two estimations has good results, as well.

Taking into account the previously presented analysis, we consider that the most suitable approach is to estimate jobs' TTL based on production, site, CPU, hyperthreading configuration).

Chapter 4

Improvements in Computing Infrastructures for Learning and Research Activities

Research and learning activities rely on computing infrastructure. Whether it is heterogeneous resources for complex operations [4, 10, 11, 33, 16, 25, 26] or for facilitating learning operations through e-learning platforms [4, 5, 50, 34, 28, 29, 30, 31, 32, 27, 35, 51], the working environment must be improved. Based on an appropriate infrastructure, the processes of education, research and development can be carried out more easily.

This chapter presents several improvements implemented to develop the education and research environments. The application of these contributions was carried out within the University Politehnica of Bucharest. Thus, a **structure for automatic course creation** was created in an e-learning platform (Moodle), based on the educational plans. On the basis of study contracts, an **automatic enrollment of students** in courses was carried out.

To improve the working environment of students and researchers developing projects based on FreeBSD, various changes have been applied so that FreeBSD can be run as a **compute node in OpenStack** (without the network component). This chapter presents tests performed in **integrating FreeBSD as an execution node in the cluster**. The thesis shows **improvements to the process of live migration of virtual machines using bhyve**, the hypervisor in FreeBSD.

4.1 Improvements for Automatic Structure Creation in e-learning platforms

Moodle is a versatile platform that allows its users, mainly teachers, to personize their content. Moreover, the system administrators can customize the LMS to be tailored to the institution's, students', and teachers' needs. In this section, we present how Moodle can be used in

the decision-making process in the education field though two types of views: from the web application side, through plugins and activities that can customize the way the students learn improving their performances, and from the server side, through the scalability of the server nodes.

To simplify the Moodle configuration process, we created and scheduled some tasks (that run on-demand or are scheduled through RunDeck). They synchronise the student databases with the Moodle platform and ensure that the courses are created according to the study plans. Then, they enrol students on their classes according to their corresponding contracts. Figure 4.1 depicts the automation process described in the following paragraphs.

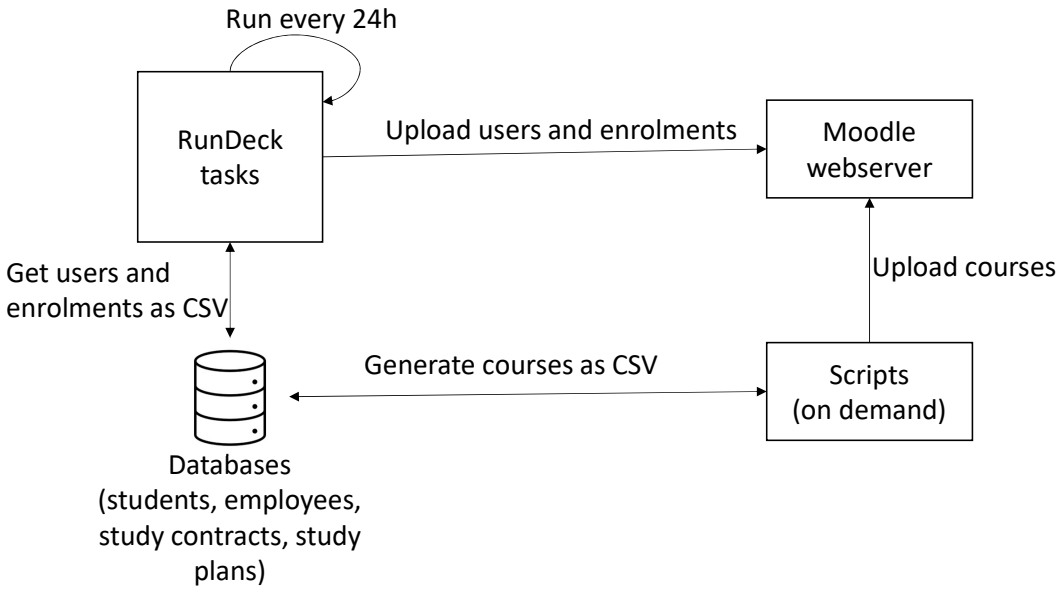


Figure 4.1: Moodle upload users and enrolments automation

Since there are more than 11,000 courses from all faculties within the university, it is not feasible to manually create them. Thus, we use some in-house MySQL scripts that extract all classes for the current year from the internal databases based on faculty, year of study, and semester. Additional courses enhancements are added while creating the Moodle course structure. Each course is derived from a template course that contains a basic setup (format, number of weeks in a semester, an anonymous feedback form with all the required configurations). When creating the courses, we also add the start and end dates according to the course' semester.

Similarly, it is impossible to manually enrol a very large number of students (around 30,000) on their specific courses (each student has around 5-7 courses each semester). Considering this, the enrolment process must also be automated. The first step in the automation process is to create a profile for each student. Furthermore, teaching staff members also need to have their profiles created. Using custom MySQL scripts, we extract information from the University's database with the profiles of the users that have an active study or teaching contract. Then, using Moodle's upload users functionality, we create or update each user's profile. This step

also updates extra fields that are required for each user: faculty, department, role, group, study year.

Having the users added to the platform, we then extract the course list for each student based on their study contract using SQL queries. Afterwards, we enrol the students on their specific courses using the same upload users functionality. A student's study contract does not change often. However, we run the students' user creation and enrolment tasks every night (using jobs in RunDeck) to ensure that the students can access the online e-Learning resources in the shortest time possible in case any changes to contracts are made.

4.2 FreeBSD as an Operating System in Cluster and Grid Nodes

Since 2014[1], multiple improvements and research projects have been developed in UPB: save/restore procedure for bhyve [52, 53, 54, 55], virtual machine migration procedures using bhyve [52, 56], porting bhyve to the ARM platform[57, 58, 59], improvements to "libvdsk" support in bhyve, instruction-level caching[60]. However, developing functionality within an operating system has the following drawbacks:

- compiling the code takes a long time [61], given that operating systems have millions of lines of code, and full compilation requires a lot of computational resources.
- programming errors introduced into the kernel can generate kernel panic, and fixing them is difficult, and can lead to the loss of the working environment. That is why copies of the data (backups) are needed.
- code should be tested using varied resources (e.g., virtual machines migrated between two identical nodes).
- the code review process takes a long time [62] and features that are large have to be broken into multiple parts. It is necessary to maintain the test infrastructure even as the project moves forward.
- updating the code with officially published changes must be done periodically.

Given the difficulties of students and researchers in the process of developing changes within the operating system, there is a need for a heterogeneous working environment that includes the FreeBSD operating system.

FreeBSD has many advantages and is used, especially as a server operating system, by companies for high data transfer speeds (Netflix [63], ScaleEngine[2], WhatsApp [64]), as the un-

[1]The FreeBSD Foundation's Wiki Page, Online: https://wiki.freebsd.org/SummerOfCode2014/InstructionCachingInBHyVe, Accessed: 2 November 2023

[2]ScaleEngine, Online: https://www.scaleengine.com/, Accessed 21 September 2023

derlying operating system for derivative products (AsyncOS[1], JunOS[2]), as firewall solutions (pfSense [65]), as network storage solutions (TrueNas[3]). However, bhyve, the hypervisor on FreeBSD does not have virtual machine migration functionality added to its code base. The procedures for implementing this functionality were carried out within the Politehnica University of Bucharest.

FreeBSD is one of the most widely used BSD distributions. This operating system has multiple advantages, such as the existence of stable software [66, 67, 68, 69], multiple security solutions (Capsicum [66, 70], the use of the bhyve hypervisor, which uses Capsicum [66] as well, jails [71] as a containerization solution).

This section presents how FreeBSD can be used as an operating systemfor development in cluster and cloud environments, as well as improvements to the procedure for live migration of virtual machines using bhyve. At the time of writing, FreeBSD has no added support for running as a cloud image in OpenStack, nor as a compute node in the OpenStack infrastructure. The chapter also presents the improvements made in this direction.

4.2.1 FreeBSD integration in a heterogeneous cluster and cloud architecture

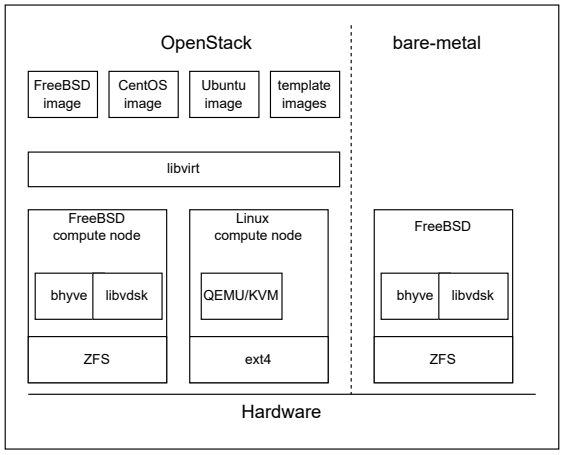


Figure 4.2: Integrarea FreeBSD într-un cluster eterogen

Figure 4.2 presents a proposal for integrating the FreeBSD operating system into a heterogeneous environment. FreeBSD can be integrated as a cluster compute node (running bare-metal). For data integrity and fast data backups, ZFS can be used as the file system. Moreover, bhyve can be used for virtualization. However, to ensure load balancing between multiple nodes using FreeBSD, the virtual machine migration procedure is required. Moreover, for using FreeBSD in the cloud, it can integrate both as a user-available image in OpenStack (along with other existing images) and as a compute node (for managing virtual machines). Through libvirt, access

[1]Cisco, Online: https://www.cisco.com/c/en/us/td/docs/security/wsa/wsa-15-0/release-notes/release-notes-for-wsa-15-0.html, Accessed 21 September 2023

[2]Juniper, Online: https://www.juniper.net/documentation/us/en/software/junos/junos-install-upgrade/topics/topic-map/junos-os-overview.html, Accessed 21 September 2023

[3]iXSystems, Online: https://www.truenas.com/, Accessed 21 September 2023

to the execution node can be abstracted, being able to have both Linux and FreeBSD nodes in parallel.

Considering the many bhyve development project, our end goal is to integrate the FreeBSD cloud images to run bhyve in our server architecture. However, bhyve does not implement nested virtualization yet, and developers must use a FreeBSD virtual machine in a Linux environment in OpenStack for testing changes in the bhyve code base. Thus, we also considered a Linux servers.

We considered for testing two servers with Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz (14 cores, hyperthreading enabled) with 380GB of RAM. At the time of these tests were executed, we used FreeBSD 13.0 amd64 with bhyve for one of the servers and various Linux distributions with KVM for the other one. On the FreeBSD server we tested bhyve, while on the CentOS server, we started a FreeBSD virtual machine, and then, tested bhyve. The two servers are part of a series of same type servers that are running Linux nodes in the UPB's OpenStack infrastructure. Since bhyve does not support, yet, nested virtualization, a Linux node needs to be used to run FreeBSD cloud images under KVM. From the KVM virtual machine, a nested virtual machine may be started using bhyve.

Table 4.1 summarizes the results we have obtained while testing FreeBSD using the before mentioned scenario. While bare-metal FreeBSD and bhyve worked fine, on this server, we could not manage to use bhyve in KVM, even though we tried various KVM configurations (disable `x2lapic`, tested both `host-model` and `host-passthrough` configurations in libvirt).

Table 4.1: FreeBSD and bhyve tests on bare-metal and nested virtualization

	FreeBSD bare-metal	**CentOS 8 (host)**		**Ubuntu (host)**
		FreeBSD12 (guest)	**FreeBSD 13 (guest)**	**FreeBSD13 (guest)**
bhyveload	ok	VM is blocked (no interraction)	kernel panic in bhyve guest	kernel panic in bhyve guest
uefi	ok	VM is blocked (no interraction)	VM is blocked (no interraction)	VM is blocked (no interraction)

As `cloud-init` support in FreeBSD is still under development[1], we could not properly test FreeBSD images in UPB OpenStack infrastructure. However, adding a private FreeBSD `qcow2` image with predefined user and password worked as intended. Nevertheless, due to the nested virtualization issues presented in Table 4.1, we cannot use bhyve under KVM.

Integrating FreeBSD as an execution node into OpenStack requires changes to the OpenStack infrastructure. In addition to the necessary changes added to `libvirt` and the OpenStack

[1]The FreeBSD Foundation, Online: https://freebsdfoundation.org/project/freebsd-as-a-tier-i-cloud-init-platform/, Last Accessed 6th of September 2023

Nova and Neutron services to recognize FreeBSD as an operating system, disabling the OpenStack service `oslo-privsep` is required. The latter uses the capabilities features from Linux to ensure security and isolation, but these features do not exist in FreeBSD. The results obtained are opening a virtual machine using bhyve, within OpenStack, but without the network component associated with the virtual machine.

4.2.2 Live Migration improvements for bhyve

Virtual machine migration is an important mechanism in cluster administration, typically used to ensure an even node load, but also to empty an execution node. Migrating a virtual machine means moving it from one node to another with as little downtime as possible. This section describes the improvements to the process of live migration of virtual machines using bhyve.

Virtual machine migration means to move a guest from one node to another while the guest is running. There are several methods for migration: cold migration (which means to move the guest image while the guest is powered off), warm migration (pause the guest's execution and move it to another node and, then, restore its execution) and live migration (move the guest while it is still running, with the smallest possible downtime).

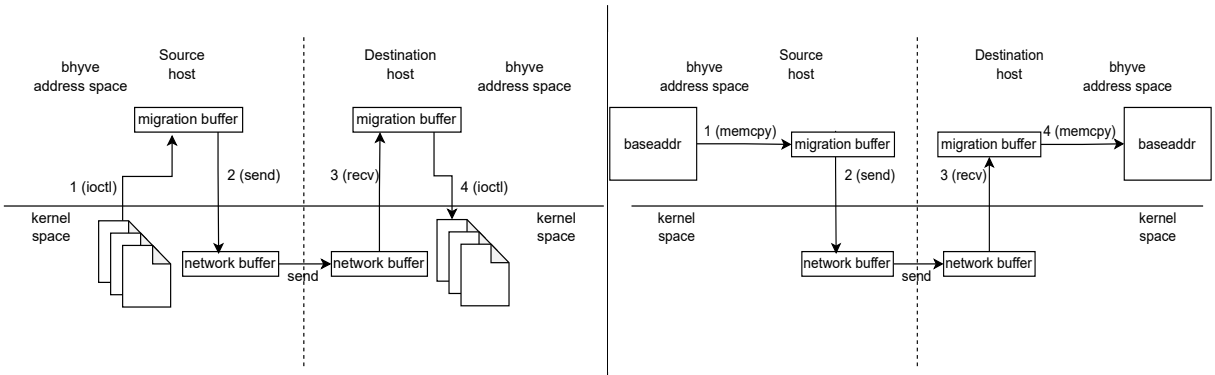
With this study, we aim to measure the impact of the current implementation in the migration process and to check the overhead brought by the swap in-swap out operations. Moreover, this paper aims to detail the scenarios we have used and their results to confirm that our implementation can be further taken into consideration for upstreaming.

As presented in [52][56], students from University Politehnica of Bucharest have implemented warm and live migration features for bhyve. The later works only for wired guests (i.e., their memory is allocated beforehand and cannot be swapped). However, a full live migration feature must work on non-wired guests as well.

Between the migration types, the most complex one, but with the smallest downtime (i.e., the time in which the guest is unavailable) is live migration. To allow the guest's memory movement from one host to another, the migration is done in rounds. While the guest runs on the source node, its memory is migrated to the destination node. In the first round, all the guest memory is moved across the network. In the next ones, only the pages that were modified during the previous rounds are migrated. The memory changes are tracked using a custom dirty bit [52].

The previous live migration implementation [52] needs the physical pages to be allocated and present in the main memory in order to traverse and retrieve the pages that were modified between rounds. Moreover, it copies the pages' content from kernel space to user space and, then, sends it over the network. However, this approach adds overhead through:

- context switches - the implementation switches from user space (bhyve process) to kernel space to inspect the pages that are dirty, then switches back to user space to report the results, then back to kernel space to start copying the modified pages.



(a) Memory migration in the previous implementation (b) Memory migration in the current implementation

Figure 4.3: Reducing the number of copies in the memory migration process

- buffers - the implementation duplicates the memory content: various buffers are allocated to retrieve the page content from kernel space while the guest's memory is already mapped in the user space bhyve process (indicated by `vmctx->baseaddr`.

One of the most important improvements added to the live migration feature is the extension to the non-wired guests. This addition offers bhyve a robust feature that works without memory allocation constrains.

For this improvement, we checked the page state. If it is not allocated, the page does not need to be migrated, thus being ignored. If the page is allocated, but swapped out, the page is brought into the memory and migrated, taking into consideration that the VMM dirty bit is also saved on disk.

The baseline implementation relayed on multiple copies of the same information between kernel space and user space. As depicted in Figure 4.3a, the baseline implementation [52] needed `ioctl` calls to copy the page content from the kernel space to the user space. Then, this data was sent through a socket to the destination where, using another `ioctl` call, the page is written in the kernel space.

Instead of copying the page's content from the kernel space, we are using the guest's memory-mapped area in the bhyve user space process. Based on the starting memory address, the page index, and the page size, we can determine the memory area corresponding to the desired page. The new process is depicted in Figure 4.3b.

Even though the proposed changes should have improved the migration time, during tests, we observed that Round 1 took as much time as Round 0, even if the guest's memory activity was almost nonexistent.

Further debugging showed that this behaviour is caused by the bhyve's dual memory view [52]: the same physical pages are once accessed by the guest through the nested page table [72] and once by the bhyve user space process that also maps the guest's memory (starting from `vmctx->baseaddr`).

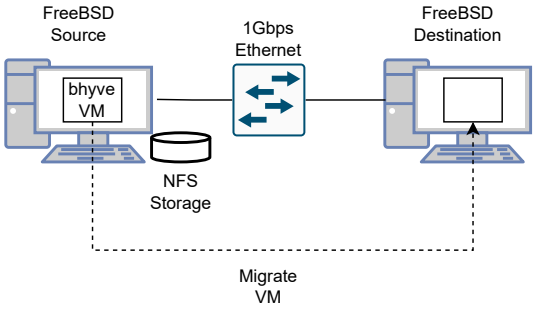
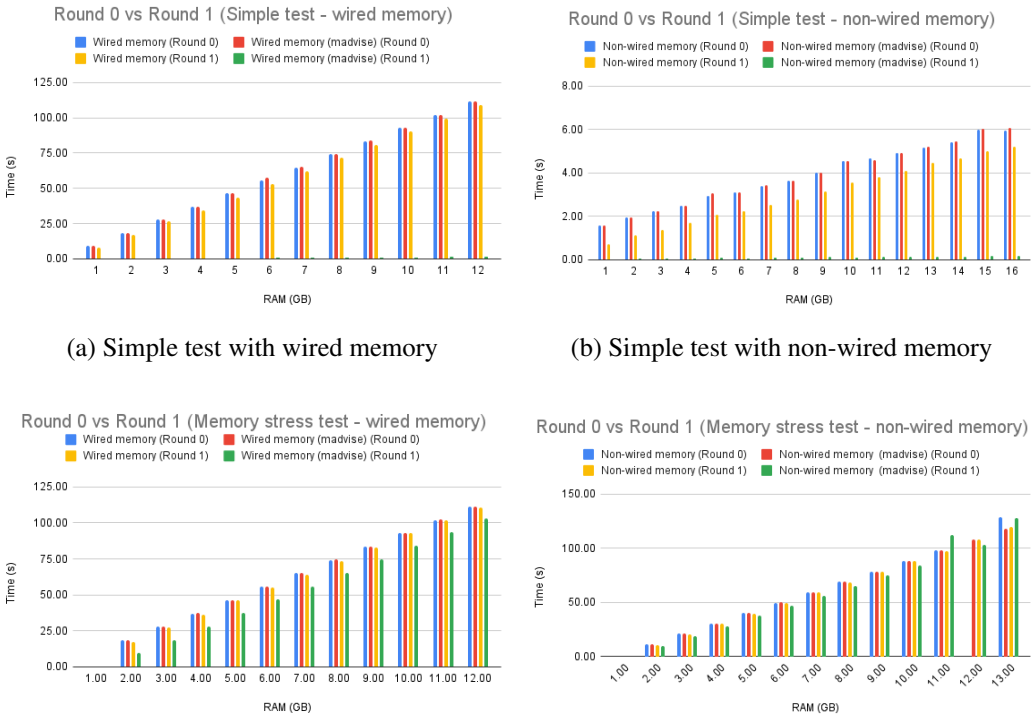


Figure 4.4: Testing setup

In our case, the round duration anomaly we observed was determined by this dual memory view: the first time the guest's memory was accessed from the bhyve user space process was during migration's Round 0. Thus, the dirty bit was set again for all the guest's pages. The solution was to hint the virtual memory subsystem, using the `madvise()` function with `MADV_WILLNEED` as behaviour, that those pages need to be immediately mapped, without page faulting since they are already allocated.



(a) Simple test with wired memory

(b) Simple test with non-wired memory

(c) Memory stress test with wired memory

(d) Memory stress test with non-wired memory

Figure 4.5: Rounds 0 and 1 comparison before and after using `madvise()`

The testing setup presented in Figure 4.4 is designed to mimic a cluster architecture: we have multiple same-type nodes that are connected in the same network. In addition, we have a distributed storage between them where the guest's disk image is kept. The setup depicted in Figure 4.4 uses two identical FreeBSD hosts with 4 cores and 16GB of RAM. As CPU, both run on an Intel(R) Core(TM) i7-4790 @ 3,60GHz. They run the same version of FreeBSD with the same migration code. Both systems use HDDs for storage, and both have configured up to

16GB of swap. Then, a distributed (NFS) storage that contains the guest's image file is added. The hosts communicate using a 1Gbps Ethernet network.

For testing, we used a FreeBSD guest that is started on the FreeBSD source with various parameters and, then, is migrated to the FreeBSD destination host. Since we aim to check our changes over the old test-bed, we run the same two types of tests: a simple test that starts the virtual machine, logs in, and, waits for commands; a memory stress test that starts the virtual machine, allocates a lot of memory and, then, continuously reads and writes one byte from each allocated memory page. While the former does not impact the guest's virtual memory, the latter heavily modifies almost all the available memory.

A comparison before and after this improvement with regarding the first two rounds can be seen in Figure 4.5. The time drops significantly in Round 1 for the simple test scenario after we added the improvements. For the memory stress test scenario, we can see that the memory that is migrated is the one that is modified by our testing executable.

Chapter 5

Network Security in Cluster and Grid Architectures

Ensuring the security of a complex infrastructure is a difficult task. Usually, several complementary solutions are used to protect the network. However, most of the new security applications are not tested in real environments and the machine learning models they use are not trained with enough data sets. This chapter presents a methodology for testing a suite of applications in a real environment, as well as the results obtained.

A typical network architecture [73, 74] is composed of multiple subnetworks that have different functionality and though which flows different type of data: **the backbone network** – the core network that interconnects all the subnetworks and through which flows all the traffic from the entire network architecture; **the DMZ (demilitarized zone) network** – the part of the network that connects and exposed the organization's network to a not trusted zone, usually the Internet; **the Datacenter network** – the network that contains the running services in the network architecture; **the User's network** – the network and systems that are accessible to the employees, students, researchers and so on.

There are several steps involved in testing a security application suite (toolkit). First, data types used for analysis are established and a series of test scenarios are created to cover all areas of a network. After that, a pilot infrastructure is created in which these scenarios are tested. In this chapter, the application of these steps in the SIMARGL project is presented. SIMARGL[2] [75, 76, 77], is a suite of security applications (toolkit) [78, 79].

5.1 Pilot network architecture for SIMARGL

The traffic that flows throughout the pilot network should use multiple direct links (e.g., a stellar topology) and should have a backup topology (e.g., a ring topology can be used). The traffic flowing through the pilot network should not be filtered anywhere like any ISP (Internet Service

[2]SIMARGL Project, Online: https://simargl.eu/, Accessed 14 September 2023

Provider) would while testing the security toolkit's functionality and each client must ensure its security of the service they are providing or using though the security toolkit.

The traffic that flows through the pilot network should be vast and diverse. For testing a security toolkit, we propose the following types of data to flow though the network: **research and educational data**, **e-payments related data: transactions, e-commerce, online banking application**, **server hosting services**, **electronic documents signing, file handling**, **streaming data**, **B2B (business-to-business) Enterprise traffic**.

The pilot environment of the SIMARGL project uses virtualization and virtual machines to deploy the tools used to analyze and detect threats in the end user's network. We use VMWare vSphere 6 Enterprise Plus as a hypervisor due to the facilities it offers such as high availability and live migration. The hypervisor virtualizes the physical infrastructure that, for the SIMARGL project, contains multiple servers with tens of gigabytes of RAM and cores and a large storage composed from Solid State Drives (SSD) disks. Then, the agent part of the security tools (the services that run on the client premises) are deployed in two networks (Network A and Network B)

5.2 Network malware detection using SIMARGL

This section presents the two environments (Figure 5.1) that generated the network traffic that was analysed and whose results are presented in this section. Network A is a research network and Network B is an education and research networks.

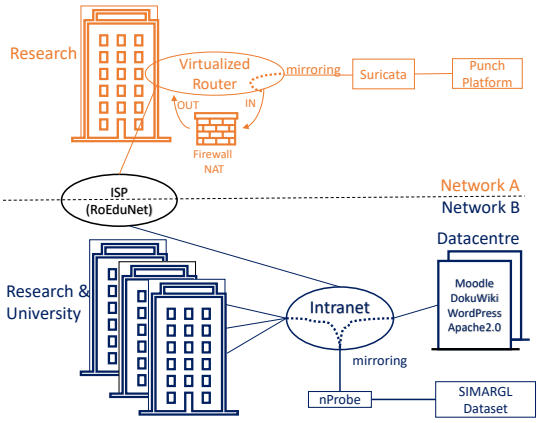


Figure 5.1: Network architectures and mirroring setups

Network A is a user-only network, meaning that it is used only by normal users, namely researchers. Network B is a complex network, having both normal users (researchers, students, teachers) and services (e-learning platforms). The two proposed networks both contain traffic from the researchers' labs. However, the research centres differ between network A and network B.

The two network architectures presented in Figure 5.2 are similar since they both mirror and then collect the network traffic that flows through the topology. Nevertheless, they differ in the aspects presented in Table 5.1. We considered the number of sub-networks, who routes the traffic (virtual or physical router), where we mirror the traffic to be analysed for security threats and what probing mechanism we use, the data direction and the traffic fluctuations.

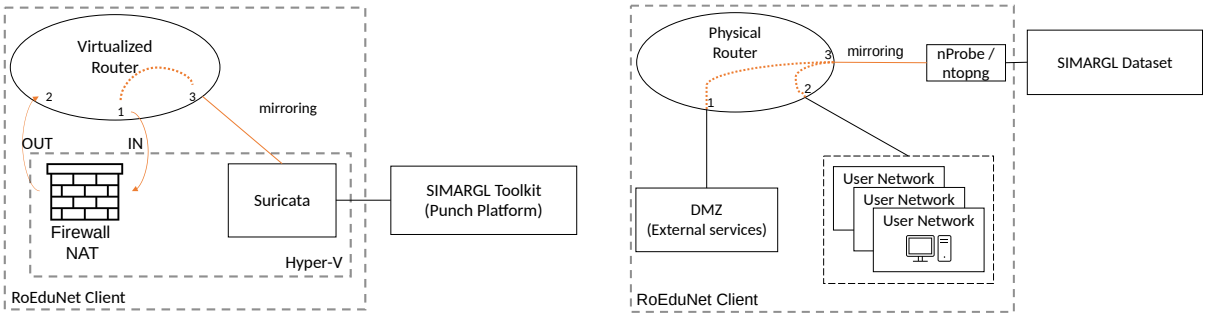
Table 5.1: Comparisons of Network A and Network B.

	Network A	**Network B**
complexity	one network	multiple networks
router type	virtual router (virtualised using Hyper-V)	physical dedicated router (more powerful and routes more traffic)
mirrored traffic	virtual switch level with Hyper-V mirroring	from the router using SPAN
data source	researchers	users (office staff, students, researchers) and services
data direction	the local network to the Internet	both directions
probing mechanism	Suricata deployment (VM)	nProbe (physical node)
traffic fluctuations	constant	fluctuating
flexibility	rigid (cannot be easily modified)	flexible (inject attacks in a controlled manner)
users	$>$ 250	200 - 2000
devices	$>$ 250	$>$ 1000
outbound traffic	$\sim$500mbps	$\sim$400mbps
services that run in the network	none	e-learning platforms (Moodle, learning VMs), Wordpress, Dokuwiki

Figure 5.2a shows the topology used for extracting the traffic from Network A that is analysed by the Punch Platform. For extracting data, we mirror the traffic that flows in the network (research traffic) using SPAN (Switched Port Analyzer). The traffic is parsed using Suricata and further analysed by one or more tools. Suricata probes were connected to the regular office users and researchers.

Figure 5.2b shows the topology (named Network B) used for extracting the traffic that composes the *SIMARGL2021 - Network Intrusion Detection Dataset*[1]. It is worth mentioning that the dataset only contains reconnaissance and denial of service attacks.

[1]SIMARGL 2021 - Network Intrusion Detection Dataset, Online: https://www.kaggle.com/h2020simargl/simargl2021-network-intrusion-detection-dataset, Last Accessed February, 2022



(a) Rețeaua A (b) Rețeaua B

Figure 5.2: Network architectures.

In Network B, two controlled networks were introduced and controlled attacks were conducted. Figure 5.3 shows the architecture used, consisting of: **Target network**, the network attacked in a controlled manner; **Legitimate traffic**, a sector with normal, non-malicious data traffic; **External attacker network**, a network from which external attacks were run; **Controlled attack environment**, a network from which internal attacks were run; the router through which all network traffic passes. The orchestration of controlled attacks was achieved through OpenStack. The resulting traffic (passing through the Router), contains both malicious and normal traffic. Attacks are both external (scanning, denial-of-service) and internal (scanning, botnet attacks). The e-learning platforms attacked are not production ones, but similar platforms on which user traffic was generated using Kubernetes and JMeter.

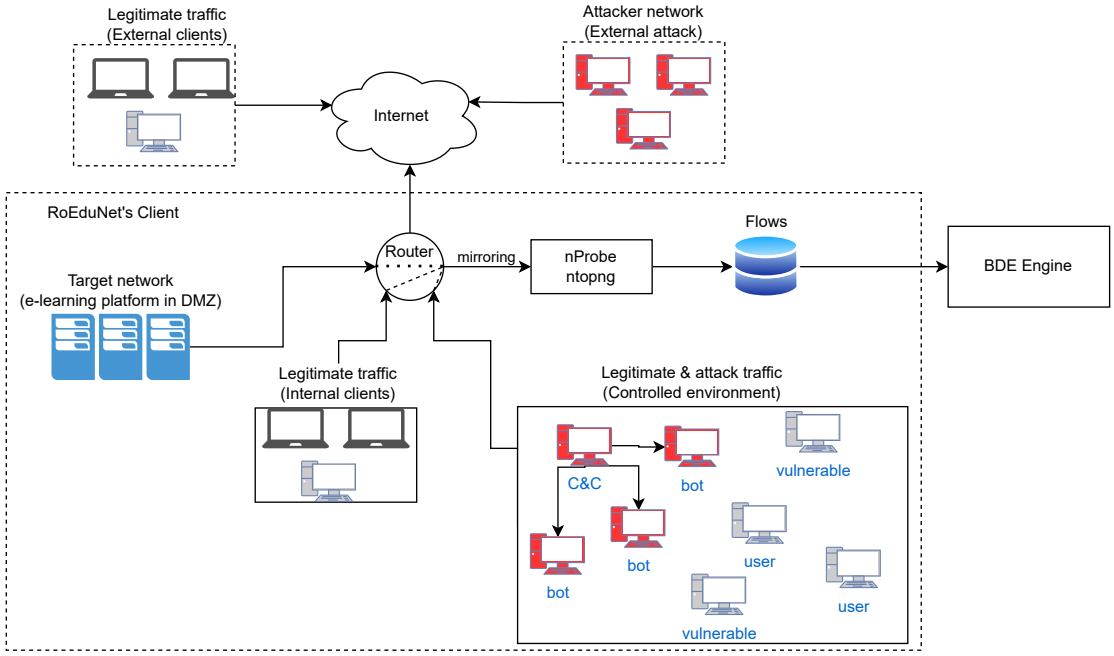


Figure 5.3: Network attack infrastructure

Network scanning (using port scanning) is the first step run by an attacker to detect network structure (operating systems, open ports, application version). Thus, scans using "SYN Scan" (most frequently used and fast), but also UDP, FIN, NULL and XMAS Scan were considered.

After the scanning step, an attacker usually tries to use the discovered information to continue the attack.

Another type of attack often used is denial-of-service, when a system is attacked to make it inaccessible. In this thesis, slow attacks (SlowLoris [80] and R-U-Dead-Yet (RUDY) [81]) were considered because they simulate slow clients. The sent requests are normal, but the response in the communication with the server is delayed (in the case of RUDY, small size HTTP Post messages are sent, usually 1 byte). Opening a large number of such slow connections prevents the server from serving other clients as well.

Denial-of-service attacks are more effective when distributed. Usually, distributed attacks are launched from botnet infrastructures (infected systems controlled by a central server). A botnet attack has two parts: in the first part, the malware multiplies in the network (each botnet scans the network and detects vulnerable systems that it infects); in the second part, a central command server (Command & Control server) notifies the bots it controls that they can launch a certain attack (eg, denial-of-service, cryptocurrency mining, etc.). Within SIMARGL, the attack was simulated **Mirai** [82]. This attack targets IoT devices by scanning the network and identifying systems with weak telnet passwords (prot 23). Within the architecture, we only simulated the spread of the malware, not the denial-of-service attack.

Based on the attacks, a data set resulted that can be used in the learning stages of the machine learning algorithms. The dataset contains network traffic in Netflow format, where IP addresses are anonymized. Based on internal information about the sources that generated the attack, the traffic in the data set was marked with a "LABEL" (identifier field) that contains information about the respective packet: whether it is normal or malicious traffic. For malicious traffic, the attack type was also specified. The resulting data set was used to train the machine learning model used by the BDE Engine.

After the training phase, the attacks were repeated within the network, but without specifying the type of traffic (normal or malicious). This step was used to test the BDE Engine, which detected all attacks.

Chapter 6

Conclusions

Managing many hardware resources is not an easy task. As a system administrator, one has to take into consideration multiple factors when sharing the institutional resources among users. From security design choices to performance, a wide range of requirements and restrictions needs to be met.

Research institutions gather all their computing resources (CPUs, memory, storage, network) to be managed as a single entity, usually in form of a cluster architecture. Nevertheless, complex problems may require more computing power than the institution can provide. Thus, multiple clusters can be united in a geographically distributed environment, named Grid. Moreover, to share resources with end-users that may or may not need powerful distributed computing architectures, cloud platforms are configured on top of cluster and Grid infrastructures. The software tools that are used for cluster and Grid management is often referred to as the middleware.

This thesis analyses the literature and defines the needs we have identified by analysing the domain: the need of heterogeneity, scheduling and resource management, cost and energy efficient infrastructure, security and privacy, support for research and learning activities, datasets variety and support for jobs and application diversity. Then, it studies the main performance requirements of a cluster, focusing on operating system choices and network security concerns.

To better understand how the needs of cluster, Grid and cloud infrastructure reflect in real-world architectures, the thesis presents the two case studies: UPB's cluster and cloud architectures (UPB provides network access to its users, e-learning and online services as well as computing resources for students, teachers and researchers, having a complex setup and multiple use-cases: from security, heterogeneous research environments to scalable and automatic e-learning platforms management) and CERN's ALICE Grid architecture (multiple research institutes provide access to their hardware resources to help processing and fasten data simulations and analysis).

This thesis shows **an improvement for the MonteCarlo job scheduling** in the ALICE Grid. MonteCarlo productions have thousands of CPU intensive jobs that are distributed into a highly heterogeneous Grid. Since the production is split into multiple tasks, each task inherits the TTL

(Time To Live - maximum time needed for execution) for the master job.

Due to the high heterogeneity of the resources, the TTL needed to be set to a very high value, even though it was observed that some sites finish the job execution faster. Keeping a high TTL (almost 24h) implies that the required slots from the Grid to be almost newly created (a Grid slot has 24h lifetime). This means that slots that have sufficient time left cannot have assigned these MonteCarlo jobs (i.e., based on the TTL, the Job Agent will be killed before the MonteCarlo job will finish its execution), leading to waste of resources when the Grid does not have any other jobs to be executed. In contrast, reducing the TTL means that more MonteCarlo jobs will be killed when landing on slower sites.

This thesis presents **a profiling methodology for the MonteCarlo productions in ALICE**. It analyses two large productions (`LHC19_cpubench_pp` and `LHC22e1_extra`) and determines the factors that impacts the job performance. Based on the plots we have generated, we concluded that, beside the production itself, the CPU, the hyperthreading option and the site configuration are the main keys that can be used to forecast the job's activity.

Based on the **MonteCarlo job analysis** and on the observation that datasets have a cvasi-normal distribution when splitting using (production, site, CPU model name, hyperthreading configuration), we propose a formula to be used when forecasting the TTL. Moreover, we observe that we must accumulate large datasets for a better prediction, thus, we propose **a weighting formula** between the predicted value and the original requested TTL. We also explore the possibility of weighting multiple estimations for a better fine-tuning. Since in some cases, the host activity influences the production performance, we also consider estimating the TTL using subsets of the proposed-key datasets, namely using (production, site, host) as key.

In addition, we implement **a validation mechanism** to check if the proposed algorithm works as expected. Based on extracted data for the jobs that already ran in the ALICE Grid, we check if our estimation algorithm would reduce the TTL without killing the jobs. As presented in Chapter 3.3, the algorithm works as intended when using (production, site, CPU, hyperthreading configuration) as key. Using (production, site, host) as a key has better results only when the number of jobs is large enough. It was observed that the first proposed key converges faster to a better TTL value. However, the latter key works better when external factors (e.g., virtualization of the CPU model name) affects the correctness of the datasets. Furthermore, the algorithm was also tested on 180 productions where it obtained good results. Moreover, even if it is designed for CPU-intensive jobs, it was observed that the proposed algorithm also works for IO intensive productions.

This thesis describes **the improvements added for sustaining the research and learning activities**. It first presents the automatic management operations we have setup for UPB's e-learning infrastructure, Moodle. UPB has almost 30000 students and 5000 teachers that are distributed to almost 11000 courses (a course has one or more teachers and up to 150 students; a student has between 5 and 9 courses each semester); manually creating and assigning students

and teachers to courses is not feasible. Thus, using the Moodle features and plugins, we **created an automatic pipeline** using RunDeck.

Based on the student's learning contracts, learning plans of each faculty, we **automate the process creation in Moodle**. We create two course templates that contain all the necessary setup for a course: start and end date, format, number of weeks, anonymous feedback forms). Then, based on the internal data, we create the course structure and using automated processes, we generate the Moodle courses and categories.

Furthermore, this thesis presents how FreeBSD is needed in research activities and the need of using FreeBSD in cluster and cloud environments. We observe that companies such as Netflix, WhatsApp, ScaleEngine or products like IronPort, JunOS, pfSense, TrueNAS are based or derived from FreeBSD due to its powerful network infrastructure, security enhancements or support for ZFS. Furthermore, during the last years, we observed an increasing number of FreeBSD-based projects that are developing in UPB (from adding support for various types of disk images to bhyve, FreeBSD's hypervisor, to implement bhyve to work on arm hardware). However, developing or improving operating systems components can be troublesome (i.e., need of nested virtualization, code compiling can take up to 3h, easy image or kernel corruption). Thus, the research community, especially the one in UPB, would benefit if FreeBSD were to run in UPB's cluster.

Based on the projects we were already working on, we **propose an architecture to add FreeBSD nodes** into UPB's cluster and cloud architecture. We encounter issues when running bhyve over FreeBSD on bare metal and observe the lack of support for two major essential features: running FreeBSD in OpenStack and virtual machine live migration in bhyve.

We also focused on **improving the live migration support for bhyve**. While a proof-of-concept migration solution was already implemented for bhyve, also as a project in UPB, the patches were not yet accepted into upstream. The initial migration support had a major downside: live migration worked only when the guest's memory was wired. In consequence, this thesis proposes improvements for the live migration feature in bhyve. First, it **adds support for live migrating non-wired guests**. Then, it **improves the migration time** by removing multiple copies of the same data and proposes a solution based on `madvise()` to map the pages in memory ahead of migration (so there will not be page fault penalties when migration). Furthermore, the thesis presents the testing procedures and the current results of the live migration implementation .

This thesis also presents how **real-life testing scenarios for security toolkits** were implemented for SIMARGL, a security toolkit. Maintaining the network security is a high-priority task for every system administrator. However, traditional security solutions (e.g., firewalls) does not protect against all types of security threats, especially internal ones (botnets, phishing, ransomware attacks). In consequence, modern machine-learning based solutions are being used. While developing such solutions, researchers face a difficult challenges: how can they

train their ML models using real data? Moreover, for enhancing the system's protection, system administrators add multiple security layers and tools into their network.

This thesis **proposes a methodology for testing a security toolkit** that was applied for verifying SIMARGL. Furthermore, it creates real-life testbeds for two security tools. We integrated Punch Platform, that was already trained in Network A and passively studied the network threats (scanning, crypto-mining, peer-to-peer communications, access to sites with bad reputation). Then, we deployed BDE Engine in Network B. We injected attack data (scanning, denial of service, Mirai-like botnet attacks) into the network, labeled it and used it in the BDE Engine training phase. After that, without labeling the traffic, we injected the same attacks to check if the tool correctly identifies the security threats, which it did.

Bibliography

[1] S. Singh, Y.-S. Jeong, and J. H. Park, "A survey on cloud computing security: Issues, threats, and solutions," *Journal of Network and Computer Applications*, vol. 75, pp. 200–222, 2016.

[2] R. Rajak, "A comparative study: Taxonomy of high performance computing (hpc)," *Int. J. Electr. Comput. Eng*, vol. 8, no. 5, pp. 3386–3391, 2018.

[3] J. Kolodziej, S. U. Khan, L. Wang, and A. Y. Zomaya, "Energy efficient genetic-based schedulers in computational grids," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 4, pp. 809–829, 2015.

[4] M. Carabas, A. Draghici, G. Lupescu, C.-G. Samoila, and E.-I. Slusanschi, "Integrating parallel computing in the curriculum of the university politehnica of bucharest," in *Euro-Par 2018: Parallel Processing Workshops: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers 24*, pp. 222–234, Springer, 2019.

[5] J. Fabra, S. Hernandez, E. Otero, J. C. Vidal, M. Lama, and P. Alvarez, "Integration of grid, cluster and cloud resources to semantically annotate a large-sized repository of learning objects," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4603–4629, 2015.

[6] S. Campa, M. Danelutto, M. Goli, H. Gonzalez-Velez, A. M. Popescu, and M. Torquati, "Parallel patterns for heterogeneous cpu/gpu architectures: Structured parallelism from cluster to cloud," *Future Generation Computer Systems*, vol. 37, pp. 354–366, 2014.

[7] Z. Wu, Y. Li, A. Plaza, J. Li, F. Xiao, and Z. Wei, "Parallel and distributed dimensionality reduction of hyperspectral data on cloud computing architectures," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 6, pp. 2270–2278, 2016.

[8] T. B. Rehman, "Cloud computing: A juxtapositioning with grid computing," in *2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)*, pp. 498–503, IEEE, 2017.

[9] M. Sharma and S. Husain, "Analyzing the difference of cluster, grid, utility & cloud computing," *IOSR Journal of Computer Engineering*, vol. 19, no. 1, pp. 55–60, 2017.

[10] A. Larsson, "The need for research infrastructures: a narrative review of large-scale research infrastructures in biobanking," *Biopreservation and Biobanking*, vol. 15, no. 4, pp. 375–383, 2017.

[11] B. K. Daniel, "Big data and data science: A critical review of issues for educational research," *British Journal of Educational Technology*, vol. 50, no. 1, pp. 101–113, 2019.

[12] S. Weisz and M. B. Ferrer, "Adding multi-core support to the alice grid middleware," in *Journal of Physics: Conference Series*, vol. 2438, p. 012009, IOP Publishing, 2023.

[13] M. Martinez Pedreira, C. Grigoras, and V. Yurchenko, "Jalien: the new alice high-performance and high-scalability grid framework. epj web conf 214: 03037," 2019.

[14] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi, "Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues," *Journal of Systems and Software*, vol. 113, pp. 1–26, 2016.

[15] M. Zakarya and L. Gillam, "Energy efficient computing, clusters, grids and clouds: A taxonomy and survey," *Sustainable Computing: Informatics and Systems*, vol. 14, pp. 13–33, 2017.

[16] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, and J. Kolodziej, "Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing," *Future Generation Computer Systems*, vol. 51, pp. 61–71, 2015.

[17] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 292–302, 2018.

[18] M. Carabas and P. G. Popescu, "Energy-efficient virtualized clusters," *Future Generation Computer Systems*, vol. 74, pp. 151–157, 2017.

[19] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "Ddos attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.

[20] T. Vissers, T. S. Somasundaram, L. Pieters, K. Govindarajan, and P. Hellinckx, "Ddos defense system for web services in a cloud environment," *Future Generation Computer Systems*, vol. 37, pp. 37–45, 2014.

[21] A. Carlin, M. Hammoudeh, and O. Aldabbas, "Defence for distributed denial of service attacks in cloud computing," *Procedia computer science*, vol. 73, pp. 490–497, 2015.

[22] P. Xiao, W. Qu, H. Qi, and Z. Li, "Detecting ddos attacks against data center with correlation analysis," *Computer Communications*, vol. 67, pp. 66–74, 2015.

[23] D. Crooks, L. Vlsan, K. Mohammad, M. Carabas, S. McKee, and J. Trinder, "Sissa: Harnessing the power of threat intelligence in grids and clouds: Wlcg soc working group," *PoS*, p. 012, 2018.

[24] J. A. González-Martínez, M. L. Bote-Lorenzo, E. Gómez-Sánchez, and R. Cano-Parra, "Cloud computing and education: A state-of-the-art survey," *Computers & Education*, vol. 80, pp. 132–151, 2015.

[25] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: moving across paradigms," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–39, 2016.

[26] E. C. Dragut, P. Baker, J. Xu, M. I. Sarfraz, E. Bertino, A. Madhkour, R. Agarwal, A. Mahmood, and S. Han, "Cris—computational research infrastructure for science," in *2013 IEEE 14th International Conference on Information Reuse & Integration (IRI)*, pp. 301–308, IEEE, 2013.

[27] I. Kureshi, C. Pulley, J. Brennan, V. Holmes, S. Bonner, and Y. James, "Advancing research infrastructure using openstack," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 4, pp. 64–70, 2013.

[28] M. Alier, M. J. Casany, A. Llorens, J. Alcober, and J. d. Prat, "Atenea exams, an ims lti application to solve scalability problems: A study case," *Applied Sciences*, vol. 11, no. 1, p. 80, 2020.

[29] A. Zaini, H. Santoso, and M. Sulistyanto, "Fault tolerance strategy to increase moodle service reliability," in *Journal of Physics: Conference Series*, vol. 1869, p. 012095, IOP Publishing, 2021.

[30] J. Prat, A. Llorens, F. Salvador, M. Alier, and D. Amo, "A methodology to study the university's online teaching activity from virtual platform indicators: The effect of the covid-19 pandemic at universitat politècnica de catalunya," *Sustainability*, vol. 13, no. 9, p. 5177, 2021.

[31] M. Sadikin, R. Yusuf, and A. D. Rifai, "Load balancing clustering on moodle lms to overcome performance issue of e-learning system," *Telkomnika*, vol. 17, no. 1, pp. 131–138, 2019.

[32] J. Prat, A. Llorens, M. Alier, F. Salvador, and D. Amo, "Impact of covid-19 on upc's moodle platform and ice's role," in *Eighth International Conference on Technological Ecosystems for Enhancing Multiculturality*, (New York, NY, USA), p. 765–769, Association for Computing Machinery, 2020.

[33] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "On the diversity of cluster workloads and its impact on research results," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 533–546, 2018.

[34] K. L. Dangwal *et al.*, "Blended learning: An innovative approach.," *Universal Journal of Educational Research*, vol. 5, no. 1, pp. 129–136, 2017.

[35] A. S. Chow and R. A. Croxton, "Designing a responsive e-learning infrastructure: Systemic change in higher education," *American Journal of Distance Education*, vol. 31, no. 1, pp. 20–42, 2017.

[36] A. Grigoras, C. Grigoras, M. Pedreira, P. Saiz, and S. Schreiner, "Jalien–a new interface between the alien jobs and the central services," in *Journal of Physics: Conference Series*, vol. 523, p. 012010, IOP Publishing, 2014.

[37] S. Schreiner, C. Grigoras, A. Grigoras, L. Betev, and J. Buchmann, "A security architecture for the alice grid services," in *The International Symposium on Grids and Clouds (ISGC)*, vol. 2012, 2012.

[38] M. Bertran Ferrer *et al.*, "Adapting heterogeneous high-performance computing infrastructures for data analysis of the alice experiment at the lhc grid," 2022.

[39] M. M. Pedreira, C. Grigoras, V. Yurchenko, and M. M. Storetvedt, "The security model of the alice next generation grid framework," in *EPJ Web of Conferences*, vol. 214, p. 03042, EDP Sciences, 2019.

[40] M. Storetvedt, L. Betev, H. Helstrup, K. F. Hetland, and B. Kileng, "Running alice grid jobs in containers a new approach to job execution for the next generation alice grid framework," in *EPJ Web of Conferences*, vol. 245, p. 07052, EDP Sciences, 2020.

[41] M. Storetvedt, M. Litmaath, L. Betev, H. Helstrup, K. F. Hetland, and B. Kileng, "Grid services in a box: container management in alice," in *EPJ Web of Conferences*, vol. 214, p. 07018, EDP Sciences, 2019.

[42] M. M. Storetvedt, "A new grid workflow for data analysis within the alice project using containers and modern cloud technologies," 2023.

[43] E. B. Sandvik, "Site sonar-a monitoring tool for alice's grid sites," Master's thesis, The University of Bergen, 2021.

[44] M. Bandieramonte, J. D. Chapman, J. Chiu, H. Gray, and M. Muskinja, "Multi-threaded simulation for atlas: challenges and validation strategy," in *EPJ Web of Conferences*, vol. 245, p. 02001, EDP Sciences, 2020.

[45] J. Scheins, M. Lenz, U. Pietrzyk, N. Shah, and C. Lerche, "High-throughput, accurate monte carlo simulation on cpu hardware for pet applications," *Physics in Medicine & Biology*, vol. 66, no. 18, p. 185001, 2021.

[46] P. Schweitzer, C. Mazel, D. R. Hill, and C. Cârloganu, "Performance analysis with a memory-bound monte carlo simulation on xeon phi," in *2015 International Conference on High Performance Computing & Simulation (HPCS)*, pp. 444–452, IEEE, 2015.

[47] R. A. Tau Leng, J. Hsieh, V. Mashayekhi, and R. Rooholamini, "An empirical study of hyper-threading in high performance computing clusters," *Linux HPC Revolution*, vol. 45, 2002.

[48] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[49] A. Chmielowiec, "Algorithm for error-free determination of the variance of all contiguous subsequences and fixed-length contiguous subsequences for a sequence of industrial measurement data," *Computational Statistics*, vol. 36, no. 4, pp. 2813–2840, 2021.

[50] A. Y. Alsabawy, A. Cater-Steel, and J. Soar, "It infrastructure services as a requirement for e-learning system success," *Computers & Education*, vol. 69, pp. 431–451, 2013.

[51] N. Cavus, H. Uzunboylu, and D. Ibrahim, "Assessing the success rate of students using a learning management system together with a collaborative tool in web-based teaching of programming languages," *Journal of educational computing research*, vol. 36, no. 3, pp. 301–321, 2007.

[52] M.-E. Mihailescu and M. Carabas, "Freebsd-live migration feature for bhyve," in *AsiaBSDCon*, 2019.

[53] E.-B. Postolache, D. Mihai, M.-E. Mihailescu, S. Weisz, M. Barbulescu, M. Carabas, and N. Tapus, "Suspend feature for multiple devices of same type in bhyve," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–5, IEEE, 2020.

[54] I. Mihalache, M.-E. Mihăilescu, D. Mihai, M. Carabaş, and N. Ţăpuş, "bhyve-json format and capsicum support for the snapshot feature," in *2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 25–29, IEEE, 2021.

[55] I. Mihalache, M.-E. Mihăilescu, D. Mihai, M. Carabas, and N. Ţăpuş, "bhyve-checkpoint functionality based on zfs," in *2022 IEEE 18th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 259–262, IEEE, 2022.

[56] M.-E. Mihailescu, D. Mihai, M. Carabas, and N. Tapus, "Improving and testing live migration for bhyve," in *2022 21st RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–5, IEEE, 2022.

[57] N.-A. Ivan and M. Carabas, "Finalizing booting requirements for a guest running under bhyvearm,"

[58] A. Elisei and M. Carabas, "bhyvearm64: Generic interrupt controller version 3 virtualization,"

[59] A.-C. Martin, D. Mihai, M.-E. Mihailescu, M. Carabas, and N. Tapus, "Symmetric multiprocessor support for bhyve on arm64," in *2022 21st RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pp. 1–4, IEEE, 2022.

[60] M. Carabas and C. Carabas, "Instruction caching for bhyve," in *Proceedings of the 6th Conference on the Engineering of Computer Based Systems*, pp. 1–5, 2019.

[61] G. A. Randrianaina, D. E. Khelladi, O. Zendra, and M. Acher, "Towards incremental build of software configurations," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pp. 101–105, 2022.

[62] G. Kudrjavets, J. Thomas, N. Nagappan, and A. Rastogi, "Is kernel code different from non-kernel code? a case study of bsd family operating systems," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 211–222, IEEE, 2022.

[63] R. Stewart and S. Long, "Improving high–bandwidth tls in the freebsd kernel," *FreeBSD Journal*, pp. 8–13, 2016.

[64] R. Reed, "That's billion with a b: Scaling to the next level at whatsapp," *Erlang Factory*, 2014.

[65] K. C. Patel and P. Sharma, "A review paper on pfsense-an open source firewall introducing with different capabilities & customization," *IJARIIE*, vol. 3, pp. 2395–4396, 2017.

[66] J. Anderson, "A comparison of unix sandboxing techniques," *FreeBSD Journal*, 2017.

[67] F.-X. Puig, J. J. Villalobos, I. Rodero, and M. Parashar, "Exploring the potential of freebsd virtualization in containerized environments," in *Proceedings of the10th International Conference on Utility and Cloud Computing*, pp. 191–192, 2017.

[68] J. Bouron, S. Chevalley, B. Lepers, W. Zwaenepoel, R. Gouicem, J. Lawall, G. Muller, and J. Sopena, "The battle of the schedulers:{FreeBSD}{ULE} vs. linux {CFS}," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 85–96, 2018.

[69] G. Lencse and S. Répás, "Performance analysis and comparison of different dns64 implementations for linux, openbsd and freebsd," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 877–884, IEEE, 2013.

[70] J. Anderson, S. Godfrey, and R. N. Watson, "Towards oblivious sandboxing with capsicum," *FreeBSD Journal*, 2017.

[71] Y. Takagawa and K. Matsubara, "Yet another container migration on freebsd," *AsiaBSDCon 2019 Proceedings*, pp. 97–102, 2019.

[72] N. Natu and P. Grehan, "Nested paging in bhyve," *The FreeBSD Project, http://people. freebsd. org/neel/bhyve/bhyvenestedpaging. pdf*, 2014.

[73] J. Kugathasan, "Network design report," 12 2017.

[74] J. Ren and T. Li, *Handbook of Technology Management*, ch. Enterprise Security Architecture. John Wiley & Sons, Inc., January 2010.

[75] M. Komisarek, M. Pawlicki, M. Kowalski, A. Marzecki, R. Kozik, and M. Choras, "Network intrusion detection in the wild-the orange use case in the simargl project," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pp. 1–7, 2021.

[76] L. Caviglione, M. Grabowski, K. Gutberlet, A. Marzecki, M. Zuppelli, A. Schaffhauser, and W. Mazurczyk, "Detection of malicious images in production-quality scenarios with the simargl toolkit," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pp. 1–7, 2022.

[77] D. Puchalski, L. Caviglione, R. Kozik, A. Marzecki, S. Krawczyk, and M. Choras, "Stego-malware detection through structural analysis of media files," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pp. 1–6, 2020.

[78] R. Neisse, G. Steri, I. N. Fovino, and G. Baldini, "Seckit: a model-based security toolkit for the internet of things," *computers & security*, vol. 54, pp. 60–76, 2015.

[79] B. Barak, A. Herzberg, D. Naor, and E. Shai, "The proactive security toolkit and applications," in *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pp. 18–27, 1999.

[80] Y. Tarasov, E. Pakulova, and O. Basov, "Modeling of low-rate ddos-attacks," in *Proceedings of the 12th International Conference on Security of Information and Networks*, SIN '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[81] M. M. Najafabadi, T. M. Khoshgoftaar, A. Napolitano, and C. Wheelus, "Rudy attack: Detection at the network level and its important features," in *The twenty-ninth international flairs conference*, 2016.

[82] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th USENIX security symposium (USENIX Security 17)*, pp. 1093–1110, 2017.