Universitatea Națională de Știință și Tehnologie POLITEHNICA București

Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare

TEZĂ DE DOCTORAT

Rezumat

Middleware în Sisteme Cluster și Grid

Conducător Științific:
Prof. Dr. Ing. Nicolae Țăpuș

Autor:
drd. Sergiu Weisz

București, 2024

National University of Science and Technology POLITEHNICA Bucharest

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

PHD THESIS

Summary

Middleware in Cluster and Grid Computing

Scientific Adviser:
Prof. Dr. Ing. Nicolae Țăpuș

Author:
drd. Sergiu Weisz

Bucharest, 2024

Abstract

The current world has been forever changed by the introduction of distributed computing as a development paradigm. Instead of making single-core applications faster, a process with diminishing returns, research has moved into transitioning computing to multi-process, multi-processor, and multi-node work. By scaling processing horizontally, we can increase processing capabilities for workloads at a smaller cost than increasing processing power per core.

Cluster computing comes to help in this regard, by introducing a model that organizes applications based on the use case, and offering management tools to help users implement workloads, customize environments, and add monitoring straightforwardly, reducing the overhead for each user to manage their processes. Cluster computing offers a cost-effective solution for groups of users working inside of the same institution to share resources, allowing for better collaborations and a larger available resource pool to take advantage of.

Grid computing comes as an evolution of the cluster computing model. A grid is made up of multiple heterogeneous clusters, running the same management software and coordinating on which workloads to run. Grid environments reduce the barrier to entry for high performance computing, as an institution can partake in a grid by dedicating their resources to the grid effort, and gaining access to more resources than otherwise available, especially in the context of Big Science requiring tens or hundreds of thousands of cores to participate in the research field. Grid environments enable a more collaborative model for research, allowing multiple institutions to establish a common processing framework that they can deploy and use no matter the local cluster setup.

In light of the increase in demand for computing power in both cluster and grid computing, the following thesis implements optimizations for resource usage in grid environments through improving data transfers by reducing transfer overhead. Grid operations managers must keep up with the resource demand from grid users and their workflows. In this thesis, we explore the possibility of increasing available resources in grid environments by integrating multi-core workloads into grid environments, decreasing memory usage and allowing for more processing to be done in parallel.

The distributed computing paradigm can be adapted in many fields, not only purely research or commercial. Education has received a boost by moving to digitalize resources and embracing e-learning. Applications such as Moodle, Draw.IO, or Microsoft Teams have allowed teachers and collaborators to move some of the burden of organizing classes off the shoulders of the educators. An avenue that hasn't been explored as much is that of automating class materials development, deployment, and assignment checking, activities that are difficult to do and require a lot of manual work from teachers. This thesis shows a framework built using distributed computing concepts such as containerization, continuous integration, and continuous distribution, giving educators the ability to create and deploy class materials automatically. To further reduce teachers' workload, we have developed an assignment-checking middleware that integrates with the Moodle e-learning platform, moving assignment-checking from teachers to automated scripts deployed in public or private infrastructures. The new assignment checker has been built based on the techniques used by grid management software, with it being able to integrate with multiple institutional infrastructures based on GitLab, and allowing students and teachers to schedule the verification process in queues shared between classes.

With the evolution of cluster computing, there has been developed a breadth of management tools for deploying, maintaining, and operating clusters for various use cases based on bare metal applications, containers, or virtual machines. This thesis develops a model for cluster management using virtual machines, deploying OpenStack together with its services to give users the freedom of setting up and managing their applications in a secure environment. To ensure low downtimes, measures have been taken to deploy highly available user virtual machines by taking advantage of current cloud computing functionalities such as continuous health checks and network-mounted storage devices to achieve fast virtual machine migration. Monitoring complex environments is a task that requires advanced tools that flexibly integrate with applications to check their status and alert administrators before issues appear, intending to use as much of the available resources as possible. We have developed a model for monitoring virtualization-oriented infrastructures based on currently available open-source tools which would allow administrators to be alerted in case of events, giving them proper information to prevent issues or remediate them.

Contents

Abstract **ii**

1 Introduction **1**
- 1.1 Thesis Objectives . . . 2
- 1.2 Thesis Contributions . . . 3
- 1.3 Thesis Structure . . . 4

2 Improving Grid Resources Usage **6**
- 2.1 Improving Transfers . . . 7
  - 2.1.1 Storage Configuration in the ALICE Experiment . . . 7
  - 2.1.2 The ALICE Transfer Middleware . . . 9
  - 2.1.3 Results of 2022 Transfer Campaign . . . 11
  - 2.1.4 Conclusion and Further Work . . . 15
- 2.2 Decreasing Memory Usage by Integrating Multi-core Jobs . . . 15
  - 2.2.1 The JAliEn Middleware Architecture . . . 16
  - 2.2.2 Integrating Multi-core Job Support in the JAliEn Middleware . . 18
  - 2.2.3 Observed Reduction in Memory Usage and Use of Scavenging Queues . . . 21
  - 2.2.4 Conclusion . . . 24

3 Adapting grid environments to educational environments **25**
- 3.1 Open Education Resources Builder . . . 26
  - 3.1.1 Educational Content Builder Goals . . . 26
  - 3.1.2 oer-builder Architecture . . . 27
  - 3.1.3 Deploying Content . . . 28
- 3.2 Assignment checker . . . 28
  - 3.2.1 Goals . . . 29
  - 3.2.2 Architecture . . . 29
  - 3.2.3 VMchecker Use Cases . . . 31
  - 3.2.4 Container Infrastructure . . . 31
- 3.3 Results . . . 31
  - 3.3.1 oer-builder . . . 32
  - 3.3.2 VMchecker . . . 34
- 3.4 Conclusion and Future Work . . . 35

4 Highly available clusters for grid computing purposes **36**

4.1 Creating Highly Available Environments 37
4.1.1 OpenStack Cluster Infrastructure 38
4.1.2 Automating OpenStack Install 40
4.1.3 Running High Availability Virtual Machines 41
4.1.4 Virtual Machine Recovery Performance 41
4.1.5 Conclusion and Future Work 43
4.2 Monitoring Cluster Infrastructures 44
4.2.1 Monitoring Infrastructure 45
4.2.2 Alerting Infrastructure 48
4.2.3 Conclusion and Further Work 49

5 Conclusion **50**
5.1 Summary 50
5.2 Contributions 51

Chapter 1

Introduction

As digitalization has been integrated into more facets of our lives, so have resource requirements increased. In a digitalized environment computing resources must be used efficiently and new types of computing resources have to be found in order to keep pace. As computer scientists and engineers, it is our task to increase resource usage by finding new means of integrating and using resources in innovative ways. With the increase in resource usage and large-scale computing requirements, new methods must be investigated for managing resources.

Computing has moved more and more from on-device processing, requiring powerful processors that can only be used in conjunction with unwieldy hardware, generating high amounts of heat and needing to be plugged into powerful circuits to an offloaded computing model, with requests being offloaded to remote machines that can be managed more efficiently. Offloading workloads to remote environments allows for them to be run at a larger scale, combining multiple devices to help in accomplishing the work. The distributed computing environment has allowed for large-scale usage of computers for scientific, educational, and business purposes by making it easier for institutions to use a digital environment and decreasing the barrier of entry to science, education, and hard computing tasks.

A computing cluster is a distributed environment based on a combination of hardware and software that runs specialized services responding to user requests. Clusters use purpose-built enterprise hardware guaranteeing longevity and serviceability while being built to allow better cooling and higher power draw than a regular home computer. Cluster management software can be deployed in many different ways, depending on the cluster use cases, some emphasizing security and availability, whilst others try to attain as small an overhead as possible so that the cluster services can take advantage of the available resources.

Grid infrastructures have appeared as a consequence of higher processing power needs. A computing grid is an association of clusters that communicate with each other over the network and have similar software environments. This allows for an application to be distributed not only inside of the same cluster, but in multiple clusters, distributed in locations, even different countries, taking advantage of available resources for computing a single output in parallel on multiple machines. Grid infrastructures run specialized software to report resource availability, schedule distributed file transfers and start workloads along the grid. By entering grid environments, institutions can have more processing power available at a time by guaranteeing that a certain amount of their own resources will be used by other institutions.

A middleware is specialized software that manages software in cluster and grid environments. A middleware can be an application that starts a script on a specific machine or makes sure that certain files are present on a storage solution inside the cluster. They are a critical component of distributed computing, as they are used as the backbone for the distributed software that is run by them.

1.1 Thesis Objectives

The main objective of the thesis is to increase resource usage efficiency in cloud, cluster and grid computing, improving procedures, decreasing management overhead and adding new types of resources to be used.

With the need for more processing power in for complex grid workloads, we need to make sure that the resources available today are used at their highest capacity. To this end, the thesis looks to implement improvements to data management, data transfers, and data location in order to decrease the time spent waiting for data transfers and data deletions.

Modern grid environments frameworks have introduced the need for multi-core jobs that can share memory between processes. Multi-core jobs have been proven to decrease memory usage in grid environments [22]. The thesis plans to implement an algorithm for scheduling multi-core jobs in a grid environment, specialized to the ALICE grid environment. Using the implemented multi-core job support, scavenging queues can been explored and exploited as a new avenue for running grid workloads at no additional cost.

This thesis explores the available configurations for managing clusters in research and educational environments, which require high uptime and resilience in order to serve user requests. The objective is to deploy a highly available private cloud infrastructure to host educational and institutional resources such as user-oriented services, research workloads and sandbox virtual machines for students.

Service uptime is not only a measure of deploying services that are self-sufficient, as even these may fail. A robust infrastructure must deploy monitoring and alerting services which can be used to gauge the status of clusters and services and can alert responsible parties to issues that might arise. By efficiently managing monitoring, workload scheduling, and storage management, a cluster can offer users a working environment that allows their projects to develop, flourish, and succeed. In order to ensure service availability a monitoring system must also be deployed inside a cluster.

Middleware development can be harnessed to allow, not only for efficient computing for research and business, but also for education. The thesis approaches middleware development from the lens of educational automatization, using middleware software to enhance learning outcomes for students by automating tasks regularly done by hand by teachers, enabling digitalization, and allowing educators to spend more time teaching and less time doing repetitive and menial tasks. The thesis aims to develop educational services which help automate educational processes like documentation building and assignment checking using the distributed computing paradigms through pipelining actions and parallelizing processing on multiple machines.

1.2 Thesis Contributions

An impactful issue in grid computing is ensuring that data is correctly distributed across the grid environment, to increase job parallelization by removing the storage component bottleneck. Data must be transferred from a central repository, or source of origin to different storage solutions, such as archives and staging areas. **We have optimized data transfers for the ALICE grid**, during a critical data transfer campaign by optimizing data transfer performance in in highly variable bandwidth environments over production network links and heterogeneous hardware and software solutions. By reducing overhead, we have been able to spend less time waiting for files to be transferred to the processing point, time which has been spent processing the data for the ALICE experiment.

The new ALICE data processing framework has been rewritten by the ALICE team with the purpose of running data processing on multiple cores. **In this thesis we show the JAliEn improvements we have made to account for multi-core jobs, enabling running multiple multi-core jobs in parallel**. Increasing the number of cores used by a job has been done to **decrease the memory usage per core**, which leads to being able to schedule more jobs in parallel, as computing centers are more limited by the amount of RAM they can provide per processor [39], than the number of CPU cores they can acquire. The new architecture required the re-engineering of the middleware, monitoring multiple jobs, and adding code to limit the resources allocated for jobs.

While integrating support for multi-core jobs, **we have implemented a mechanism for dynamically allocating jobs based on available resources** as such we are able to take the scheduling burden from the Local Resource Management System (LRMS) and take it upon the middleware to schedule processing. Integrating support for queues where scheduling is done only per node, not per process, has allowed the ALICE experiment to take advantage of clusters that otherwise would not have been available. As the grid middleware is more in tune with job scheduling parameters than the LRMS, we can make better scheduling choices by managing whole-node scheduling.

In the process of increasing the resource capacity of a grid environment, we have to take into account untapped resources, or types of resources that we can integrate with no additional costs. In batch job processing there exists a work scheduling mode in which processes can be scheduled, and if a higher priority job is scheduled, the current running job is stopped, called scavenging queues. Scavenging queues have been identified in this thesis as a possible avenue of increasing available resources, by scheduling jobs on them with the risk of the processing being interrupted by other kinds of jobs. **The thesis proves the viability of scavenging process time in clusters which allow for flexible scheduling, despite the risk of processing time being wasted when processing is preempted.**

In the context of educational digitalization middleware software plays a role in managing services and functionalities offered to students and teachers. Teachers and students go through repetitive tasks, solved by hand, which can be automated using middleware software built based on the lessons learned in software development for cluster and grid environments. **The thesis adapts middleware software, and software development solutions such as Git, job management software, Continuous Integration, and Continuous Deployment (CI/CD) to educational use cases** with the intent of helping teachers and students spend more time learning.

As part of the thesis, we tackle the task of managing institutional clusters and integrating them in both research and production environments for institutional services. **The thesis implements a cluster management strategy based on OpenStack virtual machines, including highly available services and control plane which are automatically deployed**. Using virtual machines allows systems administrators to enhance cluster security and service isolation while giving the users flexibility in deploying their own workloads and controlling the work environment, needing minimal involvement from administrators. Providing highly available resources is crucial for encouraging users to adopt a cluster platform, as they can trust it to host their workloads and services without disruption.

Monitoring cluster infrastructures is a task employed by most professional deployments, as it allows visibility into resource usage patterns which can be used for preempting or early detection of issues and notification for incident response. Being such a critical task, there exists a breadth of solutions to cover this use case such as Grafana [24], CheckMk [3], and many more, each covering distributed computing use cases. **This thesis proposes a reference architecture for monitoring clusters** using Prometheus for data point gathering and Grafana for visualizing data using custom-made dashboards.

1.3 Thesis Structure

The thesis follows the structure as such:

1. Chapter 1 defines the thesis scope, objectives and major contributions. The chapter highlights the need for distributed computing to scale not only by acquiring more hardware, but also by reducing current software overhead, integrating new resource models and adding support for new kinds of resources.

2. Chapter 2 delves into the subject of resource usage efficiency in grid environments, concentrating on avenues for improving the middleware to decrease resource usage per core, increase file transfer efficiency, and add new resource types to gain more available resources on the grid. Section 2.1 shows how the ALICE experiment at CERN is managing data transfers, studying transfers made during the 2022-2023 time period, when experimental data has been moved to archival or processing areas, discussing the optimizations done for increasing transfer speeds, which can apply to grid file transfers overall. Section 2.2 talks about the work we have done to adapt the ALICE experiment middleware to manage multi-core jobs on the grid, with the advantage of decreasing the memory usage per core, by having fewer duplicated memory areas.

3. Chapter 3 introduces the need for more digitalization adoption in education, con-

centrating on Computer Science education, which contains many elements that can be automated to decrease the number of repetitive tasks done by teachers and students. Section 3.1 explains the work done on the openedu-builder platform, a middleware build for automating class material management; automating material creation and deployment using CI/CD techniques and Git [35]. Section 3.2 proposes the vmchecker tool, an automatic assignment checker middleware, which handles student and teacher requests for running tests on student-written code. vmchecker has been built with a centralized checking manager and plugins to integrate with the Moodle e-learning platform [25].

4. Chapter 4 provides a template for organizing and deploying a cluster environment using virtualization for use in running a grid site and offering specialized services for an educational institution, using the UNSTPB cluster as a case study. Section 4.1 defines steps to be made in order to achieve a highly available infrastructure for hosting services and compute workloads in a cluster environment based on OpenStack virtual machines, which gives users the ability to host services and sensitive workloads that should run without interruption. Section 4.2 describes the current monitoring solutions used for maintaining cluster awareness and alerting administrators in case of events. Because of the breadth of monitoring solutions available, we provide descriptions for the currently available solutions and propose a template for monitoring clusters and storages used for grid workloads based on Grafana, Prometheus, and AlertManager.

Chapter 2

Improving Grid Resources Usage

The focus in grid computing is to run workloads as fast as possible, spreading processing around over multiple clusters to scale the processing vertically. Adding new resources to the grid environment is done through adding new sites, or through annual pledges [33], where sites plan new hardware acquisition and integration. As these processes are slow and incremental, the current systems must be improved to allow for less overhead and increased efficiency.

Middleware applications are the nervous system of a grid environment. They coordinate multiple sites, workloads, check various system availabilities, schedule jobs, manage data, apply access restrictions to users and more. As a vital resource, they are the first place where improvements can be applied to add features. As parts of this thesis two possible avenues have been identified for increasing resource usage efficiency in grid environments by modifying the grid middleware.

Jobs in the ALICE computing grid require input files which need to be located on a storage system nearby. Files have to be transferred after first hand processing to different sites for further processing or to archival storage to be stored for reprocessing. A method of increasing transfer speed for file transfers in the ALICE grid has been implemented and detailed in Section 2.1.

Memory usage is an issue in parallel processing, because it is a resource that is a hard limit in terms of scaling up processing. CPUs can be oversubscribed, especially in mix-use environments where not all processes are CPU-dependant, and IO can be saturated and communication can still be established, but the current bandwidth won't scale. Memory on the other hand can only be extended through swapping, incurring a large performance penalty. In this context the ALICE experiment has sought to reduce the memory usage per job by implementing multi-core jobs which use the memory sharing mechanism in Linux. Section 2.2 details how scheduling support has been integrated for multi-core jobs in the JAliEn middleware used by the ALICE experiment, capable of managing multi-core jobs in the grid environment.

The following chapter presents our inquiries intro ways to increase process efficiencies in grid environments and the measures implemented in the ALICE grid environment to increase resource usage.

2.1 Improving Transfers

The ALICE detector [17] has been upgraded for the LHC Run 3 (2022–2025) with a triggerless data acquisition system. In this mode, the data rate is several orders of magnitude larger than in the previous data taking periods. Through a real time compression (O2) performed close to the experimental setup, ALICE is able to reduce the size of the data stream 10 fold, to about 100 GB/s [18]. The efficient management of this data requires a profound changes in the various levels of data persistence and movement.

The ALICE O2 disk buffer has been designed and deployed as a storage element for the output of the O2 compression which is stored in Compressed Time Frame (CTF) [43] format. The CTFs can be reduced further to smaller file sizes by at least 95% through the process of skimming. The skimmed CTFs are archived after being converted into the second generation Analysis Object Data (AO2D) format, which is used for physics data analysis. The AO2D format is a table-based data file that represents the reconstructed experimental data in an easy to parse way. These are distributed across the Grid storage elements and are read by the analysis workflow.

Since the start of detector commissioning for Run 3 in late 2021, the management of the data on the O2 disk buffer presented an issue, as the methods to control the data flow and annotate it according to the type was not yet fully designed and deployed. This was especially the case for detector-specific calibration and test data, which is not converted to AO2D. One of the first tasks of the new data management system was to separate the different data types and to redirect these to the specific set of storage elements, for example archival type, and to remove the processed data from the O2 buffer, thus keeping it free for the incoming CTFs. It also consists of removing data with bad quality and data not suitable for physics analysis - this action is accomplished through flags set in the run logbook by the ALICE Run and Physics coordination.

2.1.1 Storage Configuration in the ALICE Experiment

The ALICE computing Grid is made up of a variety of software and hardware solutions for storage systems configuration and management, and each storage level of performance is reflected in a set of data transfer parameters.

Figure 2.1 shows the data flow out of the ALICE detector and from Monte Carlo production to the various storages and their bandwidth and capabilities. The diagram does not show the data read by the analysis tasks. Among these storages three main systems can be used for moving the data:

- EOSALICEO2 buffer, used for storing raw data and CTF files temporarily before processing, skimming, and archiving;
- EOSALICE, which is the main storage system dedicated for the ALICE experiment in the CERN data center. It stores the AO2D files produced by the data reconstruction and MC simulation, as well as the output of the physics analysis tasks.
- CERN CTA is the tape archive (custodial storage) used for long term storage of CTF files and AO2Ds.

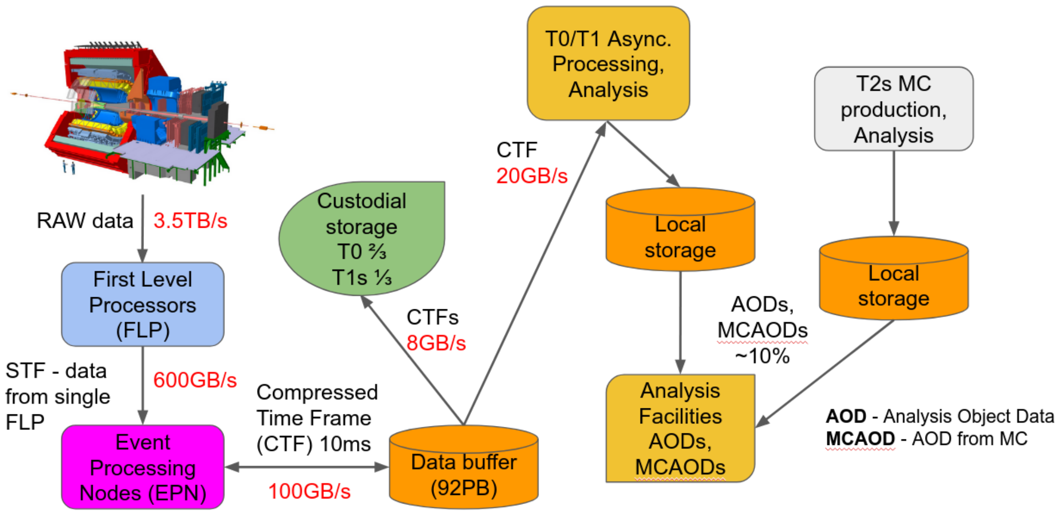

T0/T1 Async. Processing, Analysis
T2s MC production, Analysis
CTF
20GB/s
Custodial storage T0 ⅔ T1s ⅓
Local storage
Local storage
RAW data
3.5TB/s
First Level Processors (FLP)
CTFs
8GB/s
AODs, MCAODs ~10%
STF - data from single FLP
600GB/s
Compressed Time Frame (CTF) 10ms
Analysis Facilities AODs, MCAODs
AOD - Analysis Object Data
MCAOD - AOD from MC
Event Processing Nodes (EPN)
100GB/s
Data buffer (92PB)

Figure 2.1: The ALICE experiment data flow.

Across the ALICE Grid, there are about 60 storage nodes, encapsulated in the figure as 'Local storage' of the corresponding T1 and T2 centres and analysis facilities. These also play a key role in the data management picture.

The protocol used for transferring and managing files on storage systems, and the respective storage systems in use are described in the following.

XRootD

The ALICE experiment uses the XRootD [16] protocol to transfer files. This protocol is a standard for physics experiments, as it interfaces with the ROOT [23] program library and its I/O system. ROOT is a framework widely used in the HEP community to write reconstruction and analysis software. The framework provides as a container the `.root` files with internal structure resembling a UNIX file directory. A `.root` file can contain directories and objects organized in unlimited number of levels and stored in machine independent format. The XrootD protocol is aware of the internal structure of these files and uses this knowledge to perform optimal data transfer over short and long network distances (LAN and WAN) by adjusting dynamically the transfer buffer sizes and through read-ahead of entire objects from the `.root` files.

EOS

EOS [30] is a storage management solution built at CERN on top of the XRootD application and protocol, with the purpose of providing XRootD based endpoints for storing multi-PB file namespaces in an efficient way. An EOS instance is deployed for each of the four LHC experiments at CERN and at many Grid sites around the world.

EOSALICE

The main production storage for ALICE is an EOS based storage element hosted at the CERN IT data center. It is built from inexpensive high-density JBODs connected to front-end server nodes through high-speed SAS HBA cards. The data stored on this instance is used for running jobs located at CERN, which means that the jobs both read and write large amounts of data to this storage system[19]. This makes the EOS ALICE storage a uniquely important element of the ALICE computing infrastructure.

EOSALICEO2 buffer

The EOSALICEO2 buffer is the storage area where the results of the real-time data compression are stored by the O2 Event Processing Nodes (EPN). The buffer must provide sufficient space to store the CTFs for one Pb–Pb data taking period, of the order of 80PB with average input rate of approximately 100 GB/s. After the end of the Pb–Pb period, the data are read and processed to AO2Ds, which are stored on Grid SEs for further physics analysis. The entirety of the Pb–Pb CTFs are copied from EOSALICEO2 to custodial storage, at CERN (2/3 of the volume) and to 6 T1 centres (1/3 of the volume). The reconstruction is run on the CERN and T1 batch systems and on the EPN nodes outside the data taking periods.

Tape archive

Tape archives are a form of long term storage suitable for keeping infrequently used data. The cost, expressed in price per terabyte, is still considered [26] more advantageous than for hard disk based storage and the modern tape systems offer a high level of data redundancy, thus making it a very secure format. The disadvantages of using tape media is the lower egress speed and linear access, in addition to the inherent complexity and the required special support. Since the entry barrier for setting up and operating tape archives is rather high, these are deployed at large computing centres only.

2.1.2 The ALICE Transfer Middleware

The ALICE experiment produces, stores, and reads a large amount of data. Consequently it requires a robust framework that can handle the considerable number of file operations (copy, move, read, remove) that are done on a daily basis.

This subsection hilights the way files and file transfers are being handled by the ALICE Central Services.

File registration of CTFs

The CTF files that are generated by the EPNs are initially stored on the internal SSDs of the nodes. They are moved to the EOSALICEO2 buffer using a purpose-build system, EPN2EOS, which copies the data, registers the files in the ALICE catalogue, and upon successful registration, deletes them from the EPN SSD.

In the ALICE catalogue, each file is associated the following logical and physical attributes upon registration:

- An LFN, that is used to refer to a file in a POSIX compliant manner, using a path in a file system, for example
/alice/data/2023/LHC23a/539457/ctf_run00539457.root;
- A GUID, used to identify file uniquely, which is an equivalent to an inode number in a UNIX file system. The file example above has the following GUID:
cfa6adab-1ac0-11ee-a3ab-0242f0fa34f9;
- A Physical File name (PFN), which contains the physical location of a file, and the protocol to access it. A file can have multiple PFNs, if it has multiple replicas on different storage nodes. Example for a file with 2 replicas:
pfn = root://eosalice.cern.ch:1094/13/19787/cfa6adab
pfn = root://mgm.spacescience.ro:1094/13/19787/cfa6adab;
- ACL, which shows the file ownership and access permissions.

Transfer management

The entire ALICE Grid Central Services system is built in Java, and this includes the transfer management software. A user can request a file transfers either through a standalone Java client or using the `alimonitor` [1] transfers web page.

When a transfer is requested, a call is sent to a Central Services instance with a list of files to be transferred, a transfer message to identify it, a source and target SEs, and the nature of operation - move or replicate. The Central Services instance receives the request and adds it to the `transfer_requests` table of a PostgreSQL[13] database it manages. Each transfer request has an ID, a destination, a name and in the case of move operation, on file transfer success, the replica will be deleted on the storage element marked as source. The database entry will also contain entries for each file transfer, so that the state counters are stored in a persistent manner. This will determine the state of a transfer, which can be RUNNING, SUCCESSFUL, or ERROR, if some of the transfers could not be performed. A transfer was successful if there were no errors for the entire set of requested files.

Transfer optimizations

For some originating storage nodes, for example EOSALICEO2 buffer, the target SEs are limited to T1 custodial storages due to the type of data. In such cases there is a static optimization which limits the transfer bandwidth to each individual SE and sets a specific portion of the data which should be received by the target SE. For the other transfers, the optimization methods are described below.

Client transfers The first type of optimization is with regard to the number of concurrent transfers that are allowed per storage. If the number of files that are being transferred to a storage element is large, but the file sizes are small, the maximum bandwidth can be maintained only if a large number of transfers is started in parallel. This happens as the transfer time is dominated by the setup overhead rather than the copy operation itself.

Sending large files with many threads is also sub-optimal, as the speed will already be limited by the available network throughput or the bandwidth limitations of the receiving storage.

Thus the client transfer limit is a balance of the two cases and is set per target storage element through a configuration stored in a database based on the Lightweight Directory Access Protocol (LDAP). At present, this value must be changed manually, to reflect evolution of the target storage elements or network fabric. Although not ideal, this is a reasonable operational compromise as the storage elements and WAN changes are infrequent.

Third party copy A simple way to copy using `xrdcp` is to transfer the files from one endpoint to another through the host originating the transfer. This is the way the `xrdcp` command operates by default, as it is the predominant way of client interaction with the storage on a processing node. This method certainly poses a severe limitation on data transfers between storage nodes in the existing heterogeneous Grid structure, where the two elements from/to which the data should be copied can be at a very large distance, thus increasing considerably the transfer latency. It also presents a scaling issue, as the host will route all traffic and multiple host must be set to assure sufficient bandwidth for all requested transfers.

Third party copy (TPC) is a mechanism which allows the copy process to be initiated by a central machine with the actual data transfer done directly between the source and destination storage nodes. This naturally enables a more efficient and direct data flow, and allows the use of one or few central machine to manage all data transfer. TPC is a fully implemented option of `xrdcp` and all ALICE storage elements are configured with it. TPC is using the same authorization and authentication plugin and envelopes as the normal clients to enforce security.

2.1.3 Results of 2022 Transfer Campaign

This subsection presents the data life-cycle on the EOSALICEO2 buffer and the tuning of the various transfer operations to secondary storage elements.

EOSALICEO2 buffer data lifecycle

Figure 2.2 shows the data accumulation cycle on the EOSALICDEO2 buffer in the data taking periods of year 2022 (July to October), the end of year LHC accelerator stop (November to May), and data taking periods of year 2023 (May to July). A typical 10 PB/month data volume is collected during the data taking periods and throughout the year, some of it is removed from the buffer, indicated by the downward slope of the curve, as it is being processed to AO2D, copied to other storage elements or identified as low quality. In the time period in Fig 2.2, the buffer has received 146 PB of data from the EPNs, of which 17.7 PB has been transferred out and 54 PB has been deleted. It should be noted that the availability of the EOSALICEO2 buffer was above 99.8% throughout the entire period and 100% during the data taking. The remaining 0.2% are in scheduled downtime for planned upgrades of the EOS software. The bulk of the data on the buffer will be removed before the Pb–Pb period in November 2023 to make place for the large volume expected during the Pb–Pb data taking.

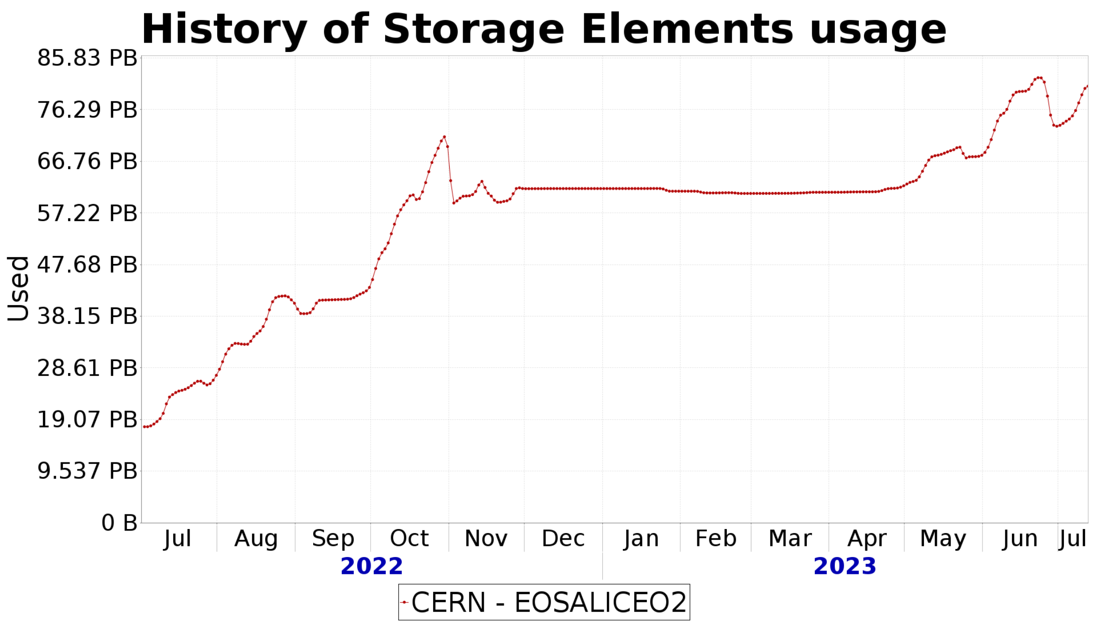


Figure 2.2: EOSALICEO2 buffer data accumulation.

Transfer of data to EOSALICE

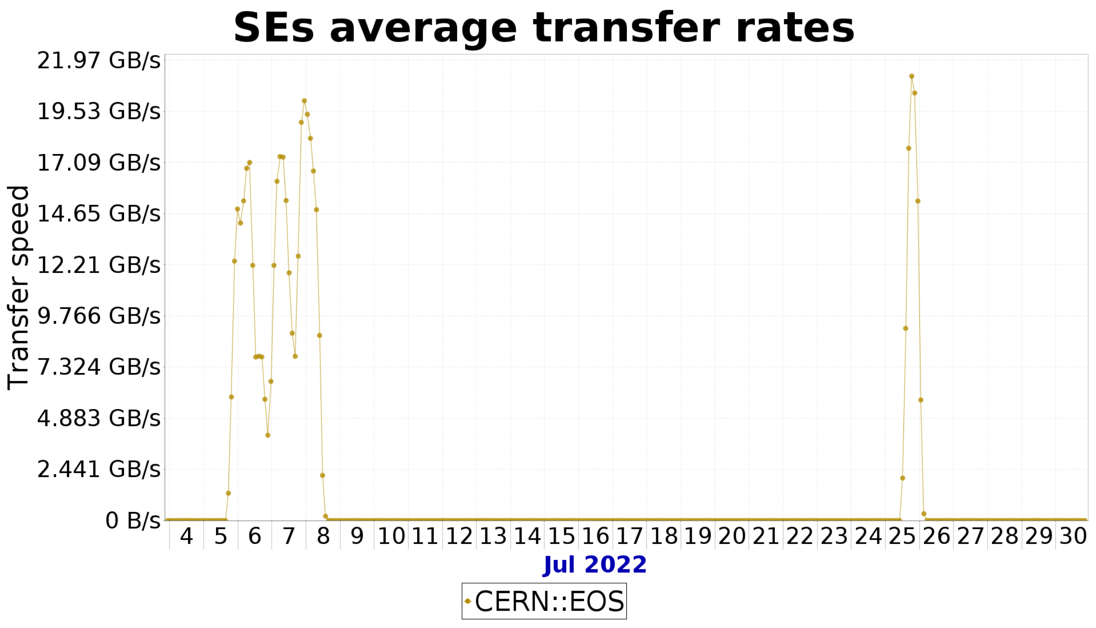


Figure 2.3: EOSALICEO2 to EOSALICE transfer speed.

The data transfers from EOSALICEO2 to EOSALICE are one of the standard data paths in the ALICE data management cycle. Figure 2.3 shows a typical rate and frequency of transfers to EOSALICE. It includes only data transferred from EOSALICEO2 and does not show the data written by the reconstruction process or data analysis, which is substantially higher.
The total transferred data volume during the period on Fig. 2.3 is 221 TB and the

speed is limited to about 20 GBps in the configuration of the transfer tools. Although higher speed is easily achievable, this value was chosen to minimize the effect on the usual high volume data reading and writing to this EOS instance.

Transfers to CERN custodial storage

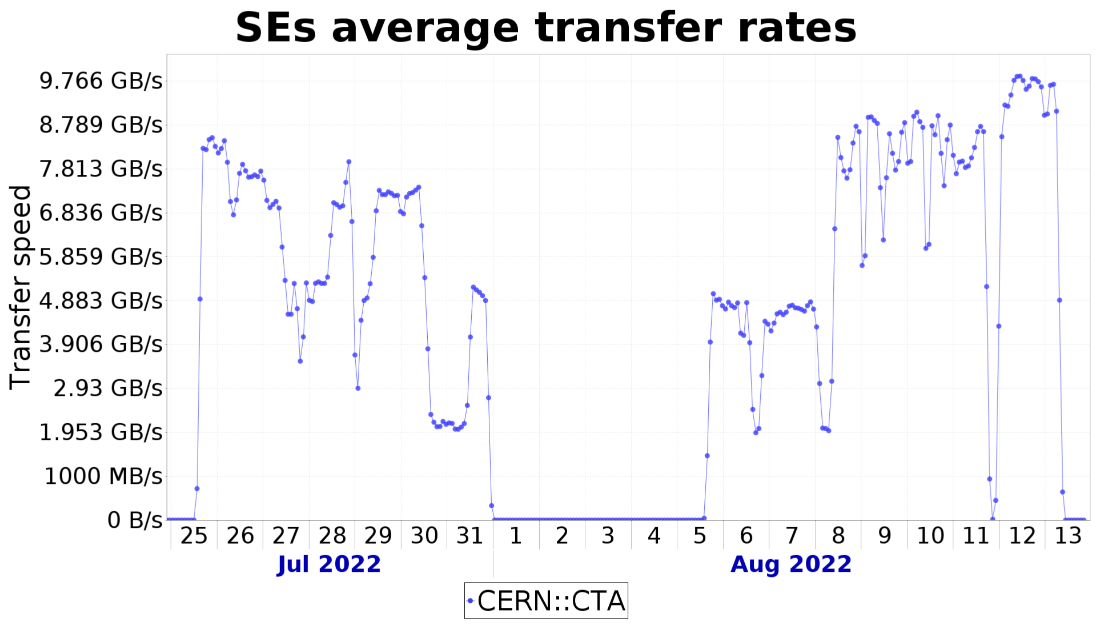


Figure 2.4: Transfer to CERN custodial storage in July-August 2022.

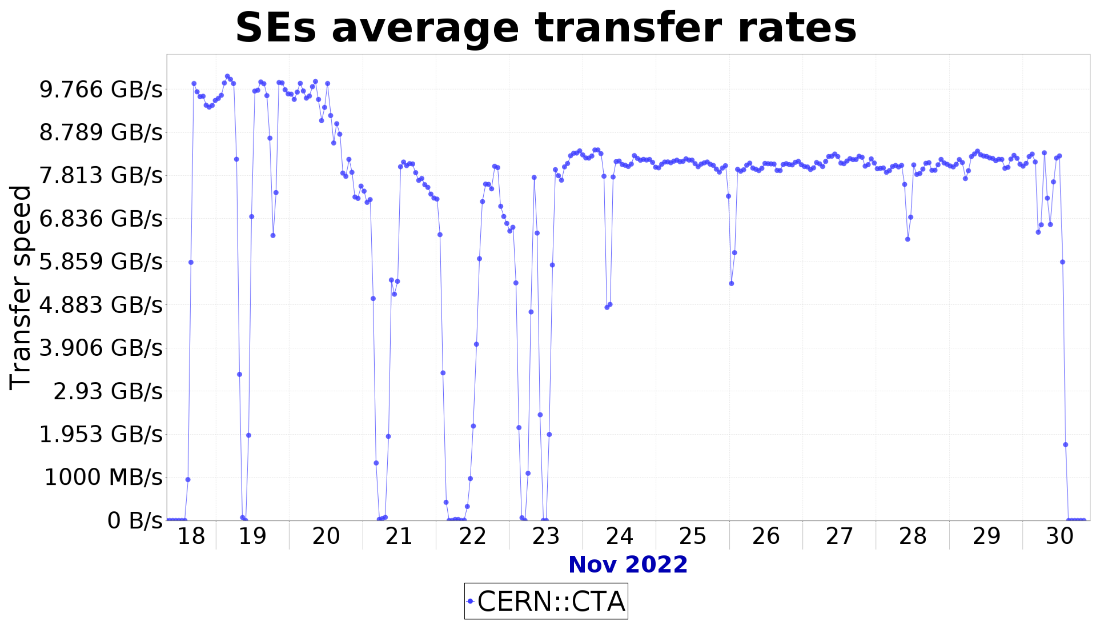


Figure 2.5: Transfer to CERN custodial storage in November 2022.

Two periods of data transfers are shown in Figs. 2.4 and 2.5. In the period July to August 2022, data accumulated between November 2021 to June 2022 was moved to

the tape archive. The files have 1 GB size with small variations and 1250 simultaneous threads were sufficient to saturate the set bandwidth of 10 GBps. During the initial period in July, the custodial storage was tuned for performance, indicated by the frequent variations of the transfer speed. On August 5th we started transferring runs from October 2021, with files of smaller sizes. This necessitated an increase of the number of parallel transfers to 3000, done on 8 August. With this final parameter in place, the 10 GBps nominal speed was achieved.
In November 2022 there was another set of large file transfers, as shown in Fig. 2.5 with stable speed to custodial storage. On 21 November, this speed was decreased to about 7.5 GBps due to high concurrency of tape operations from the other experiments and slight decrease in the ALICE tape share.

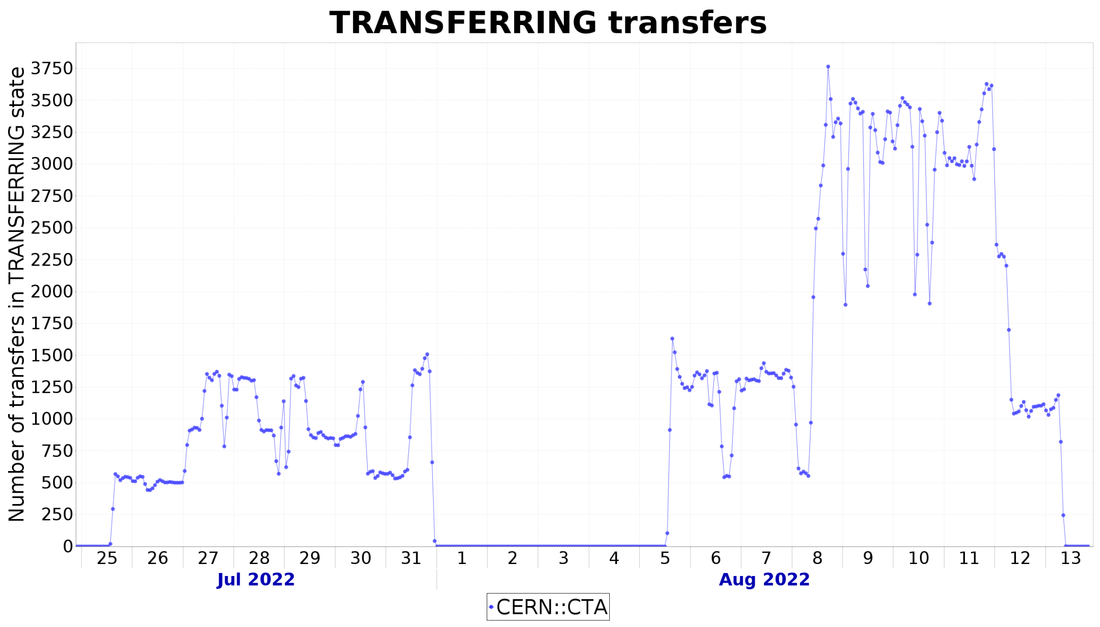


Figure 2.6: Tape transfer clients.

In total 17.7PB of data have been transferred.

Control of transfer bandwidth

One of the standard methods to control the transfer bandwidth to any storage is the number of parallel threads. Given a known average file size and if the available bandwidth is not a limiting factor, increasing or decreasing the number of threads results in almost linear increase/decrease of the transfer speed. This is a trivial, but very effective method of control and is the primary one used in the ALICE transfer system. The transfer speed is affected also by the overhead of setting each individual thread, however, it has a smaller weight for reasonable file sizes (of the order of 100 MB+). It only becomes dominant for very small files (of the order of tens KB).
The effect of number of threads on the transfer speed is illustrated in Fig. 2.6. In the period 5–13 August, the step at 8 August is due to the change from 1250 to 3500 parallel threads (factor 2.8) to accommodate for the small size of the files being trans-

ferred. The transfer speed increased from 4.3 GBps to 9.8 GBps, a factor 2.3. The approximately 20% difference between the two factors is due to the transfer overhead and some throttling in the receiving storage element.

2.1.4 Conclusion and Further Work

The ALICE experiment data management system has been upgrade for the LHC Run 3 to cope with the considerably higher volumes of detector and the more complex data paths. A new storage node, EOSALICEO2, was introduced in the system, able to accommodate more than 100PB of data and to support sustained data rates of the order of 100 GBps. At the same time, the data management of the more than 60 storage nodes around the world was upgraded with new transfer methods and tools, capable of controlling the data flows automatically and adapt to the variations of conditions of a distributed computing system. The new data management is fully in production and has shown its abilities to store and distribute data in excess of 150PB per year, and at the same time to maintain the working status of the storage elements, which are serving a large number of compute nodes preforming data reconstruction and analysis.

In the future we will continue to update the data management system with more automatic functions, one of these is the dynamic transfer triggers (DTT). DTT will be activated on completion of a previous actions, for example end of processing of set of files, and will enable the move, copy or deletion of the file set. It can also trigger an operation, for example start of analysis on a dataset upon its replication to a specific storage node.

2.2 Decreasing Memory Usage by Integrating Multi-core Jobs

The Grid is constantly evolving, and the available resources are increasing. In order to get the most out of them, the services and clients of the framework have also evolved and have been re-implemented in the new framework. This reimplementation has allowed deployments to be carried out more easily and with greater scalability. The authentication and authorisation mechanisms have also been improved, making use of the Token Certificates provided by the JAliEn Certificate Authority X.509.

The ALICE Grid pools together the computational, storage and network resources of multitude of distributed computing centres around the world. This allows for the massive amounts of collected data to be processed and analyzed by the physicits in relatively short amount of time and assures the necessary growth of the combined resources as more data is collected by the experiment in the ongoing LHC operation.

The software that enables researchers to run jobs on the ALICE Grid is the JAliEn middleware. It is made up of JAliEn Computing Elements services that run on dedicated nodes on each site (called VOBox-es), which connect to their infrastructure and submit generic jobs from the Grid to the cluster. Another component of the framework is the Central Services, located at CERN, that work as a centralized point of management which dispatches payloads to run on sites, manages the input and output files required by the jobs, catalogues them and steers the clients to the appropriate Grid storage nodes to write their output to.

For LHC Run3 the ALICE experiment has upgraded its detector and changed the data acquisition model from a triggered to a streaming mode with sharp increases in bandwidth and storage requirements. The paradigm shift from processing events to looking at 10ms long continuous data frames required a complete rewrite of the experiment software, from simulation and reconstruction to the analysis framework. For an efficient processing of the new data type the framework requires larger physical memory allocations per job, in the order of 10 to 20 GB. Swapping them out is not an option as the entire data file content is accessed and thus the CPU efficiency would be dramatically impacted.

The new experiment software is multi-threaded and allows for efficient use of multiple core slots on the sites while maintaining the existing 2 GB per core ratio demand from the resource providers. This thesis addresses the modifications that have been made to the experiment Grid middleware JAliEn for brokering and running multi-core jobs.

2.2.1 The JAliEn Middleware Architecture

The Java ALICE environment (JAliEn) is the Grid middleware used by the ALICE experiment to store, process and analyse data obtained from the physical phenomena [29]. It is an evolution of the legacy AliEn [20] middleware and has been developed taking into account the evolving needs of the upcoming LHC Run 3. The Worldwide LHC Computing Grid (WLCG) infrastructure [33], which is composed of more than 200 computing centres in 39 countries around the world, is used for its operation.

The JAliEn framework has two types of services, some of which run on each of the Grid sites and others which are run centrally on servers hosted at CERN. Figure 2.7 shows how the main components of the framework are organised and how they relate to each other.

Central Services

The framework has a centralised architecture, which means that job executed at different locations of the WLCG connect to the Central Services (CS) at CERN. The centralized management point dispatches payloads to run on sites, manages the input and output files required by the jobs, catalogues them and steers the clients to the appropriate Grid storage nodes to write their output to. Having central data and task management entities provides the system with load balancing capabilities between its multiple sites.

Grid site

The ALICE Grid is currently composed of 53 sites where jobs that study the physical phenomena are executed. Job scheduling takes into account the data locality (as per the central file catalogue) in order to minimize the IO latency and thus the resources consumed by the job.

Computing Element The Computing Element (CE) is the gateway to each of the sites on the Grid. It runs on a persistent point of presence on the site called VOBox

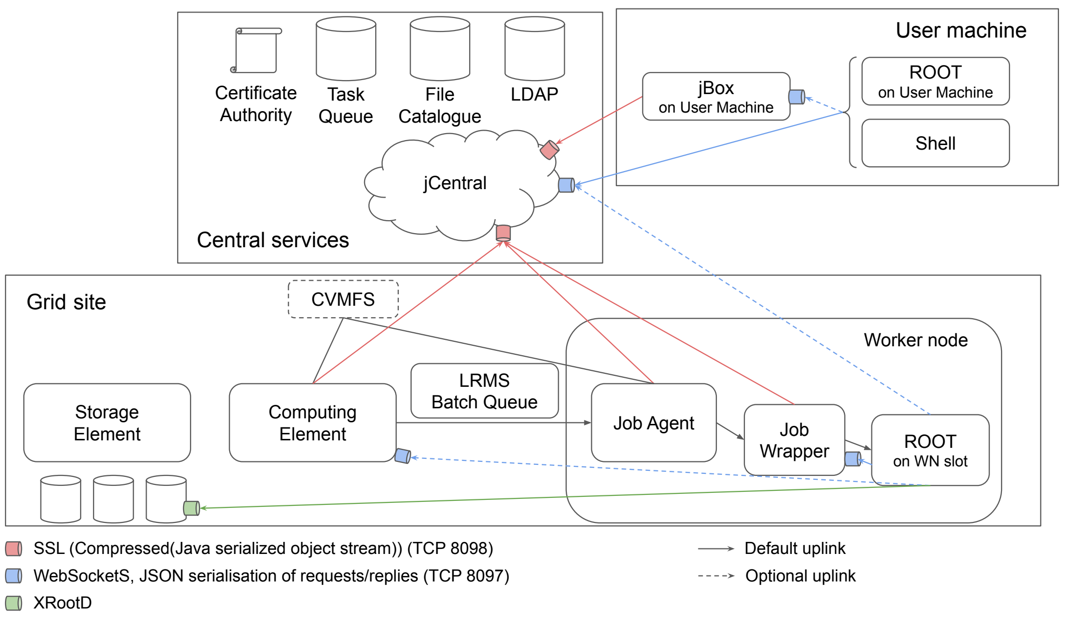


Figure 2.7: Diagram of the main components integrating the framework

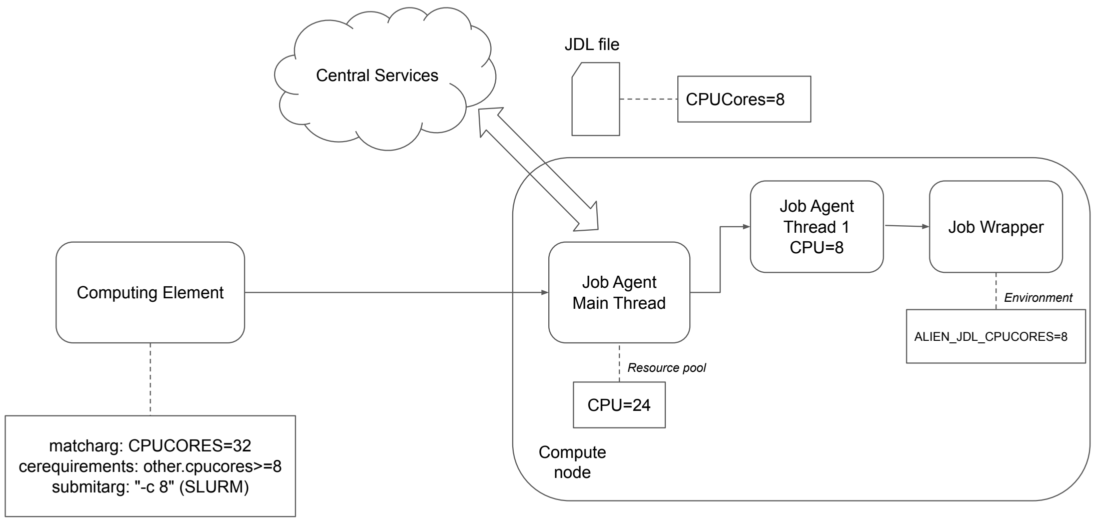


Figure 2.8: Diagram of the CPU Cores parameter definition and mechanism used for the execution of multi-core jobs.

(Virtual Organization Box) and from there it submits generic jobs to the site Batch Queue via a Local Resource Management System (LRMS), upon instructions from the Central Services. The CE also performs monitoring and resource accounting functions.

Worker Nodes The worker nodes are the machines where the jobs are executed, being configured with an environment containing all the needed tools. This environment can be set up either directly on the machine or in a container.

Job Agent and Job Wrapper The generic jobs started by the LRMS launch Job Agent (JA) instances on the worker nodes. They connect to the Central Services and advertise their available resources, receiving an actual job that matches the constraints. The JA then forks a Job Wrapper (JW) that executes the actual payload inside a sandboxed directory or a container where available. The JA supervises the correct functioning of the JW and interacts with the CS to guarantee continuous monitoring and supervision of the resources used.

2.2.2 Integrating Multi-core Job Support in the JAliEn Middleware

The nature of the new experiment's hight throughput requires for a shift in the analysis framework, because of the large abount of data this framework will work differently, and it will have to work in a multi-core fashion. We have to be able to provide middleware support to the software. Our goal is to find a way in which to manage this new type of jobs by refactoring the way we to job scheduling, resource management and job launching. A consequence of this refactoring is that now we can run our software in an efficient way on a whole-node scheduling queue because of the resource management components that we have added.

The framework had to be adapted to increase its throughput while using the available resources more efficiently. This thesis presents the implementation of the logic that enables the execution of multi-core jobs, besides continuing to offer support for single-core jobs. It has involved a refactoring in several domains, such as job scheduling, resource management and job launching.

The first of these structural changes is in the scope of job scheduling, involving modifications to the Local Resource Management System architecture used. To address that issue, the running LRMS-based implementation has been modified to allow multi-threading, letting the middleware spawn job running threads. This approach not only has benefits in terms of scheduling, but also in terms of consumption of RAM resources, as the new model does not make use of a JVM instance for each of the Job Agents launched.

The process of requesting new jobs to run, i.e. communicating with the Central Services, has also varied due to the non-homogeneous nature of the Grid. The involvement of the Job Agent threads in that process is needed to be able to configure the granularity of the jobs to be executed depending on the resources available at any given time.

To manage the new dimension of CPU cores resources we have introduced an intermediate entity called Job Runner (JR), replacing the direct calls to start JAs via LRMS with others that start the new component. JRs keep track of the available resources (CPU cores, memory and disk space) and spawn JA threads to pick up actual jobs from the queue (and subtract the respective amount of resources from the pool). This approach also reduces the amount of Java Virtual Machine processes on the worker node and consequently the memory footprint from the LRMS point of view.

A new parameter, *CPUCores*, has been added to the JDL syntax to enable the definition of multi-core jobs. With this new parameter, the number of cores to be used is defined and then taken into account by the Central Services when scheduling new jobs.

The task scheduling logic previously described has been adapted to accommodate this new parameter. In addition, the definition of the sites has been extended to include their constraints in terms of the available CPU cores and their distribution among the advertised job slots. In the definition of the Computing Elements in the LDAP system, three parameters have been added that refer to the requirements and possibilities of each of the worker nodes. Figure 2.8 illustrates the definition of the involved parameters and the steps followed for the execution of multi-core tasks.

Before adding the support for the execution of multi-core jobs, the framework was just capable of providing all the CPU cycles with only one CPU core per slot.

The Computing Element has been configured to retrieve the *matcharg:CPUCORES* parameter from the LDAP database when initialized. This parameter can have two behaviours: either be set to make use of a fixed amount of cores, which are going to be used as default allocation for the slot, or be stated to make use of the whole node, which means that it is going to auto-detect the number of cores that will be utilised and advertised to the Central Services. In this last case the parameter *matcharg:CPUCORES* is assigned a value of 0. In the example shown in figure 2.8, the Job Agent can make use of a total of 32 CPU cores for job execution.

As the Grid resources are not homogeneous, the Job Agent needs to advertise the available resources to the Central Services in order to receive jobs tailored to its available resources. Therefore, when the Job Agent communicates with the Central Services to request a new job, it must report the number of free CPU cores, as well as the amount of RAM memory and disk space available. Based on this information, the Central Services scheduler performs the matching process to find a job that matches the requirements, and adds it to the accounting information.

Listing 2.1 shows the scheduling steps taken by the JobAgent for saturating a slot, as described above.

```
retries = 0
TTL = getRunnerTimeToLive()

if cpuCores == 0 then
    availableCores = min(getCpuCores(), getRamCapacity()/4)
else
    availableCores = cpuCores;
end if

while availableCores > 0 and TTL > 0 and retries < 5 do
    jdl = getNextJob(availableCores, TTL)

    if jdl == null then
        retries = retries + 1
        sleep(300s)
        continue
    end if

    retries = 0
    availableCores = availableCores - jdl.JobCores
    runningJobs = runningJobs + 1

    startJob(jdl)
done
```

Listing 2.1: JobAgent scheduling algorithm

Resource isolation on a whole-node scheduling context

Because the new payload framework launches sub-process and coordinates them, instead of doing all the computation in a single process, we cannot be sure that a single payload would not launch as many processes as the system has free resources. We have implemented a mechanism which keeps the resource consumption within the bounds of the initial allocation.

CPU isolation In our research scenario, we want to make sure that the CPU cores are only used by the process to which they have been assigned. When a single job is submitted to a Grid site, the internal mechanisms of its LRMS that takes care of the scheduling of the tasks are also responsible for guaranteeing the isolation of resources. Although whole-node scheduling provides the framework with great flexibility and has the potential for efficient job execution, it does not in itself provide resource isolation for each of the processes. For this reason, it is necessary to provide the system with additional tools to perform this function. The task of finding tools for this purpose is not as easy as it may seem, since there are many limiting constraints, mostly caused by the few privileges granted on the Grid sites.

There is a variety of ways in which a job's CPU usage can be isolated such as *cgroups* versions 1 and 2 [31], *isolcpus* [21] and *taskset* [15]. Each of these options handles CPU isolation in different ways, and some sites have already implemented some of them independently of the JAliEn framework.

Proposed solution We propose to use the *taskset* command to pin each job to a set of CPU threads of the same size as the number of requested CPU cores, in order to ensure that jobs cannot overrun their allocation. We have implemented a mechanism that checks for the CPU threads that have already been pinned and, avoiding those, selects a set of cores on which to run a new job.

Although each situation has its own peculiarities, we can distinguish three main scenarios:

- Sites where no constraint is applied on the CPU cores to be used. In this case, we are free to make use of *taskset* and pin the payload processes to explicit cores.
- Sites where our processes are already constrained to run on specific cores. The task affinity that will be assigned to them may have been set either via its *cgroup* (*cpuset*) or via *taskset*. Although the configuration can be done using different tools, the CPU affinity masks of the processes can be seen using the *taskset* command.

- Sites where containers are being used. In these sites, *cpuset* or *taskset* should be configured by the system administrators because it is not possible to inspect external processes from inside the container, as they are isolated.

2.2.3 Observed Reduction in Memory Usage and Use of Scavenging Queues

The JR mechanism has enabled the execution of new workflows on new system types. In order to evaluate the updated job execution architecture, we have studied the number of jobs that have run so far using the new mechanism, how many of these jobs are multi-core jobs, and if the goal of reducing the amount of memory used per core has been accomplished.

Running multi-core jobs

We have chosen to analyse two different types of jobs, as they give us an overview of how multi-core jobs will run on the Grid when their usage increases popularity, and because they have run in a large enough number of nodes to get meaningful results.

These two types of jobs are:

- Reconstruction jobs which convert raw experiment data into AOD format. These jobs load into memory the compressed files in order to parse them, so they run on sites with a large amount of memory that can accommodate the raw files parsed;
- Organized analysis jobs, which use pipelines and start multiple processes that read events resulted from the aforementioned conversion or from other productions. These jobs access many files in parallel and do a large amount of I/O operations, so they require a fast connection between the computing nodes and the storage connected to the site.

These two types of jobs have different requirements, so they run on different Grid sites. Because of the fluid situation of the payload software, and the fact that it has not been run on a large scale, we will limit the observations to two queues: CERN-CORONA, that runs reconstruction jobs, and Wigner-KFKI-8core, that runs analysis jobs.

We noticed that while both CORONA and Wigner run 8 core jobs, these jobs do not use all the resources that the sites provide. The sites also have to run single core jobs in order to not waste CPU resources. This reveals that the Grid hasn't fully moved to the multi-core job infrastructure.

Memory usage per core As expected, the amount of RAM used per job has increased from an average of 3.17 GB, to an average memory usage of 9.04 GB. That being the case, if we divide by the number of cores, we notice that the memory footprint per core has decreased to 1.13 GB. These results are in line with what we expected, meaning that we will be able to fit more computational power per batch slot, because the resource that is lacking Grid-wide is mainly memory. By lowering the memory usage, we will be able to run fewer jobs that do more computational work.

Isolating CPU usage

For testing CPU isolation we have chosen to run a prototype of simulation jobs that haven't yet been fully deployed on the Grid. We use these jobs for testing because they are the most likely to run on more CPUs than required, as they use a forking mechanism that is prone to launching more than eight processes that could be eventually scheduled at the same time.

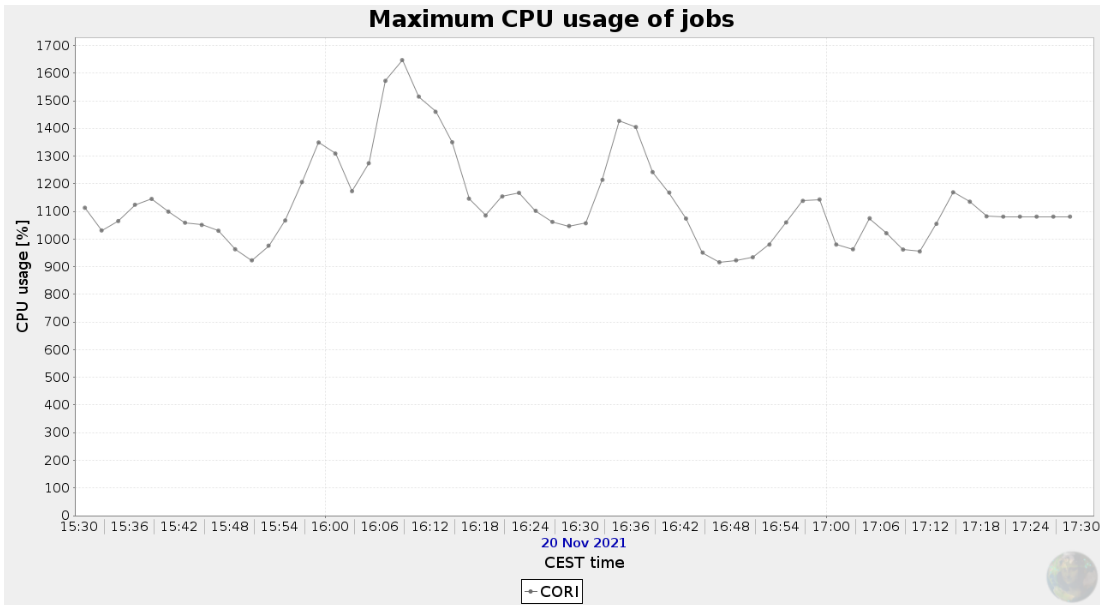


Figure 2.9: Per job CPU usage before isolating tasks

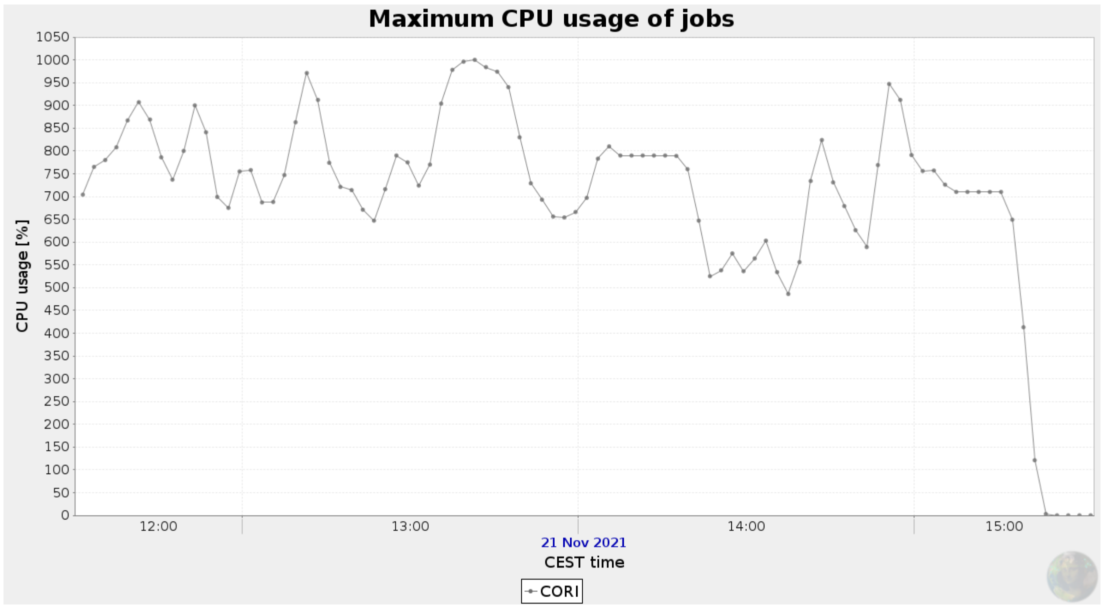


Figure 2.10: Per job CPU usage after isolating tasks

We notice from figure 2.9 that although multi-core jobs request eight cores, they actually use more than this amount of CPUs on average. There are jobs whose CPU usage goes up to 1600%, meaning that they fully use 16 cores. We expect this to be an issue that will escalate in the future, as more physicists start using the new framework.

Figure 2.10 represents the CPU usage of jobs after implementing the task isolation system. As expected, the CPU usage has decreased to close to 800%, equating to 8 CPUs working at 100%. In the graphic we can notice a peak of 1000%, but this is caused by the accounting method used by the monitoring software.

Opportunistic job deployment so far

In order to test the integration with new queue types, we have installed the JAliEn Computing Element on two new resource types:

- Cori supercomputer [28], run by the National Energy Research Scientific Computing Center and hosted at the Lawrance Berkeley National Lab, which has been deployed as a test site;
- The HPCS (High Performance Computing Service) cluster hosted at LBNL, which uses scavenging queues.

The two selected systems represent new resources that the ALICE Grid can make use of because the sites already have access to them, but they had not being used for job execution yet.

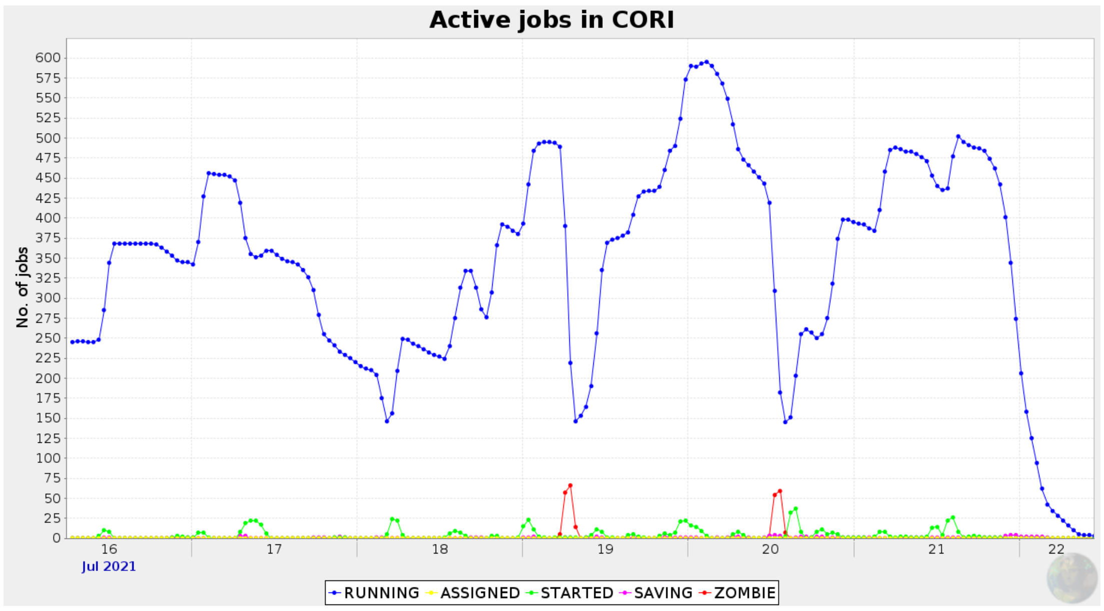


Figure 2.11: CORI job distribution

HPC jobs After deploying the new job framework on the Cori supercomputer, we have managed to find the correct TTL (Time To Live) and job submission rate so that there would be a constant amount of jobs running on the supercomputer. This constant flow of jobs can be seen in figure 2.11.

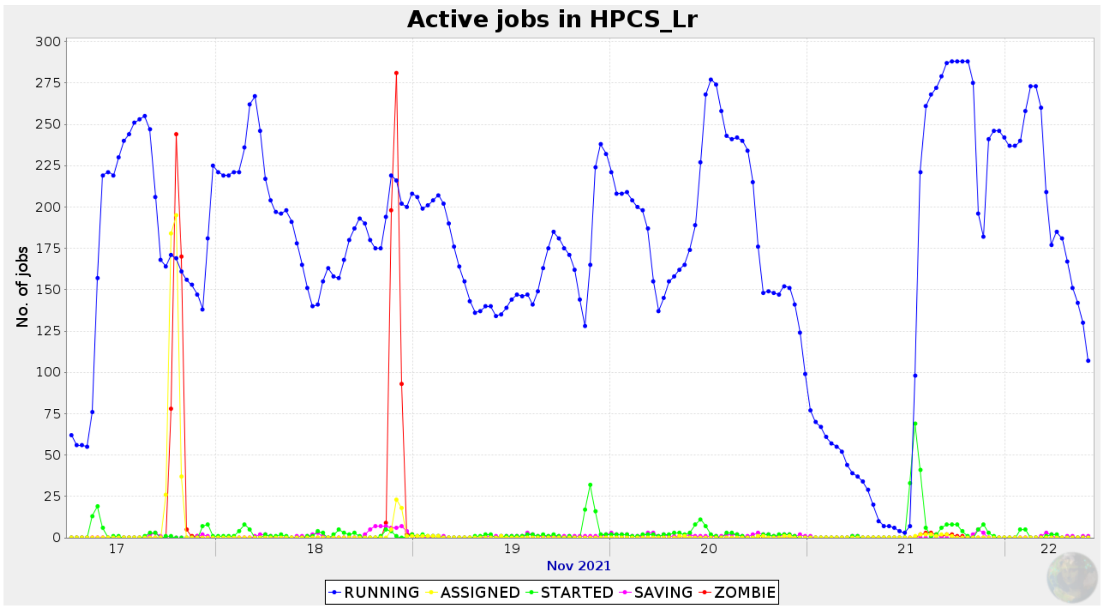


Figure 2.12: Lawrencium job distribution

Scavenging queues When deploying on the HPCS cluster, we see that while jobs run, we get allocations that are equivalent to a small Grid site. This happens because we have chosen to limit the maximum number of jobs that can run at a time on the site, as jobs on scavenging queues will be preempted and shutdown. If we chose to run a larger number of jobs on this cluster, more jobs will be killed because there will be a larger number of jobs that will be preempted. Figure 2.12 illustrates the way jobs are being cut off by jobs with higher priority.

2.2.4 Conclusion

We implemented a mechanism capable of managing multi-core slots and even whole nodes allocated by ALICE Grid sites and using the given resources to launch single and multi-core jobs from the Grid with a user-defined granularity. This new feature allows the experiment to run the new generation of jobs, but it also allows the software to be deployed on new sites, such as whole-node scheduling sites and supercomputers. Furthermore, this thesis proposes a solution for slot fragmentation on the nodes and for CPU resources isolation.

Whole-node queus have been integrated into the ALICE grid, seen by using the example of the Lawrencium site, hosted at the Lawrance Berkeley National Lab, which runs as a scavenging queue, using resources on an opportunistic basis, without the experiment being charged for them.

Further research questions that arise from this thesis are as follows:

- What is the impact of job containerization in regards to the job efficiency once the new framework has been broadly deployed on the Grid;
- How could Computing Element deployment automation be done on supercomputers;

Chapter 3

Adapting grid environments to educational environments

Education has long lagged in implementing the advancements that software development has enabled in other fields such as automation, cloud resource hosting, and outside sourcing for contributions. Following the 5Rs principles for Open Educational Resources (**r**etain, **r**euse, **r**evise, **r**emix, **r**edistribute), we have identified a set of goals for class organization in a digital age:

- resources must be easily available online or offline, so more users can have access to them, and later contribute;
- classes have to use as much automation as possible; teachers are tasked with guiding and mentoring students, having them do administrative tasks defeats this purpose;
- the resources have to be easy to contribute to, as this allows interested parties to contribute without hassle.

The focus of our work is on creating automation mechanisms to **reduce time spent building and maintaining materials**, allowing teachers to focus on the act of teaching. We intend to **increase resource availability** by making hosting resources online for teachers using our Open Education builder framework. By using a Git-centered workflow, we make it **easy for parties to contribute** by creating change requests through Pull Requests or raising issues through the community features Git repositories have. The building framework has been implemented for three classes and other classes have expressed interest in migrating to it.

Building class materials is just one aspect of class preparation. Evaluating students is another facet of teaching that requires the attention of teachers. In the context of Computer Science education, student evaluation can be done through quizzes, exams, or practical assignments. Open Education Hub presents a way in which quiz and exam evaluation can also be automated.

While there exist platforms for managing and automating classroom quizzes and exams, the problem of grading assignments at scale is not yet solved. The current tools allow for limited flexibility in evaluation, and they do not offer integration with existing learning management software that has been adopted as part of the learning digitalization effort.

We propose the VMchecker assignment checker to solve the issue of time waste during assignment evaluation. VMchecker has been integrated with success as part of 11 classes at the National University for Sciences and Technology POLITEHNICA BUCUREȘTI, having run 40,000 homeworks and covering use cases that could not be using other existing software solutions such as virtualization and multi-core assignments. It has increased the availability of homework checking, removing the overhead of assignment setup by using template and CI/CD techniques to easily deploy assignment checkers.

This thesis makes the following contributions:

- a framework to automate the workload of building and maintaining class materials;
- processes, tools, and materials that can be used by educators to create their class materials;
- an educational resource management platform and class-building framework that builds and deploys classes, permitting others to contribute to them;
- a tool to automate the task of assignment checking and grading.

Section 3.1 presents the goals, architecture, and implementation of the content building platform, called the OER-builder. Section 3.2 discusses the challenges of automating homework checking and our approach to solving this issue by implementing a new tool called VMchecker. Section 3.3 debates the results obtained when deploying the implementation, the classes that we have migrated, and the achievements of the VMchecker tools during the school year. Section 3.4 concludes the thesis by highlighting the work done, and the growth opportunities for the Open Education Project moving forward.

3.1 Open Education Resources Builder

Educational content has been hosted online since the dawn of the internet [34] in the form of wikis, tutorials, blog posts, and others. The internet has, by its nature, increased access to educational resources, and it has increased the amount of information that is proliferated. As part of the Open Education Hub project, we have taken to creating tools that make building, distributing, and using educational materials across the internet easier.

The issue with current educational resources is that in many cases they are not built in an OER fashion. Resources cannot be remixed, or reused without hassle, or they have restrictive licensing, especially if they come from higher education institutions, which use them to generate more revenue. As part of OER resources, they should also be easy to contribute to and remix. To achieve this, resources should be organized in an easy-to-change format to the maximum extent.

3.1.1 Educational Content Builder Goals

Open Education Hub wishes to use a framework with which students and teachers can build and contribute to educational resources. Using existing software solutions for enabling OER contributions is a priority because this means less time spent on maintaining code, and adding new features. An existing solution can evolve without the need to change the framework.

An OER builder should allow for an automated setup for the required resources, to spend less time managing the framework and more time creating content. The framework should have options for automated checking for various aspects, such as spelling errors, or bad formatting. While documentation builders exist, they are focused more on generating content such as manuals and tutorials. They do not cover the breadth of materials such as quizzes, tutorials, slides, and interactive work. This limits the amount of resources a class can include in them if used as an out-of-the-box experience. A content builder has to be easy to use and configure. A minimum number of steps should be taken to start creating and deploying content. The content that is deployed has to be easy to copy and re-build by other interested parties. Text must be de primary content storage method because it can be easily edited without requiring special tools. It should be used whenever possible, even for diagrams, because changes can be easily tracked using a versioning tool.

3.1.2 oer-builder Architecture

The oer-builder project has been built as a modular project, that can be flexible in integrating different kinds of materials in a class. The builder also needed to allow for integrating different kinds of use cases.

Git was chosen as a text management solution for class materials. Since we require that all class materials be in text format, so we can edit them with basic applications, using Git was a viable solution.

Classes need to be built in a rich text format that can be converted into deployable content. The rich text needs to support links, formulas, itemized lists, enumeration, images, GIFs, and more. We have chosen to create content in the Markdown (MD) format because it is used by many existing documentation builders. Markdown is extendable and text can be converted to it from other formats such as RST or LaTeX using already existing tools, allowing easier migration from other content forms.

Tutorial Content

We have concluded that the Minimum Viable Product for a class would be an interactive class where students would follow tutorials available on a page online.

This requirement would map directly to the use of a documentation builder. Docusaurus was chosen as a documentation builder because it is fully featured and provides plugins for added functionality. Docusaurus supports the Markdown text format, which supports the use of rich formatting, and admonitions and can even include in-page iframes, allowing us to add more advanced features to pages.

A user has to create a configuration file in which they have to specify the paths to the markdown files they wish to include and the chapter name. Mode advanced options can also be included, to customize the course.

Slide Content

In academia, slides are regularly stored and displayed in PDF or PPT format. We chose not to use this because while the slides contain text, they are stored in binary format PDF and PPT slides make integrating feedback a manual process that cannot be tracked and automated easily.

reveal-md [14] is a tool that builds markdown files into JavaScript content to be viewed inside a browser window. The JavaScript content can be integrated into other forms of content, or it can be viewed by itself.

Non-graded Quiz

oer-builder integrates support for quizzes in the tutorial material. Quizzes are a way of testing students on the knowledge gained during the tutorials. A quiz is also a tool to keep them engaged with the content when it skews towards being more theoretical. They can be graded or non-graded. As we do not link the oer-builder to any institutional infrastructure, there is no way to authenticate students to store their grades.

A common quiz template has been created to represent a question and a set of answers, with the correct answer being marked.

3.1.3 Deploying Content

Deploying content is the action of making content built locally by creators and contributors publicly accessible. This requires having a platform where the content can be hosted The content also needs to be in a format that can be accessed using commodity software.

The deployment actions allow for code to be deployed automatically on changes, not requiring any input from the user after the code is uploaded to Git. Both GitHub and GitLab also provide DNS entries, so the pages can be easily accessible if the user knows the project and repository names. oer-builder outputs HTML code, which is viewable in an internet browser. Because the code is HTML, it can be hosted as static content on many types of platforms.

3.2 Assignment checker

One of the greatest challenges of teaching is evaluating the students' work. For a teacher, the action of grading work is repetitive and requires a large amount of attention.

In Computer Science there is a breadth of tools that do code evaluation. This is a process that happens both in a commercial context, but also in research and education. Code evaluation is used to check the functionality and the expected behavior of a tool or a project.

Assignments in IT education take the form of code assignments or projects, where students have to implement a set of tasks inside of a template or framework. The grading system for these assignments is based on a score for the code functionality, and good coding practices, such as the use of certain design patterns, or efficient use of memory allocation.

Teachers use code evaluation tools to test students' assignments, projects, or lab work based on a set of reference tests written by the teachers. The output of the evaluation tool should be a score for the functionality of the homework, and code feedback.

The current landscape of assignment evaluation tools is made up of software that has to be licensed from companies, or that doesn't automate enough of the work. The existing tools are not integrated with popular e-learning platforms, such as Moodle. This increases the overhead of integrating an evaluation tool into the institutional infrastructure.

VMchecker proposes a free and open source software solution for homework evaluation that uses integrations with other tools such as Moodle for front-end user and homework lifetime management, and GitLab for running the code in a safe environment.

3.2.1 Goals

VMchecker is built to satisfy the goals of the Open Education Hub project. They are the same goals as the ones set for the oer-builder, but they have been particularized for assignment checking as follows.

VMchecker has to be easy to use and set up by infrastructure managers, teachers, and students. This can be done through automation, providing setup scripts to allow for automatic setup.

The goal of VMchecker is to be a software solution that requires as little coding as possible. A low-code solution is a low-maintenance solution, requiring fewer interventions from programmers. A smaller codebase leaves less room for bugs to be added by software developers.

VMchecker aims to teach students how to use code versioning tools such as Git, so they will be able to manage a production code environment while working on projects.

As resource usage increases, and demand is increased for VMchecker, it should be able to scale the amount of assignments that it checks in parallel. A parallel approach means students don't have to wait in long queues for assignment checking.

Seeing as VMchecker is supposed to be a free and open source solution, it should be built on free and open source solutions. VMchecker must be compatible with the OER 5R requirement. Using open source solutions allows us to patch the solution that we are using if issues are encountered.

Many educational institutions use e-learning platforms to manage classrooms. These tools allow users to upload materials, test students, and grade assignments using a single application. Integrating VMchecker with e-learning platforms makes it more appealing to integrate for institutions that already rely on these systems. Educational institutions need a seamless process from assignment submission to grading in a shared grade book.

3.2.2 Architecture

VMchecker has been built with a modular architecture so that it fits the diverse needs of educational institutions.

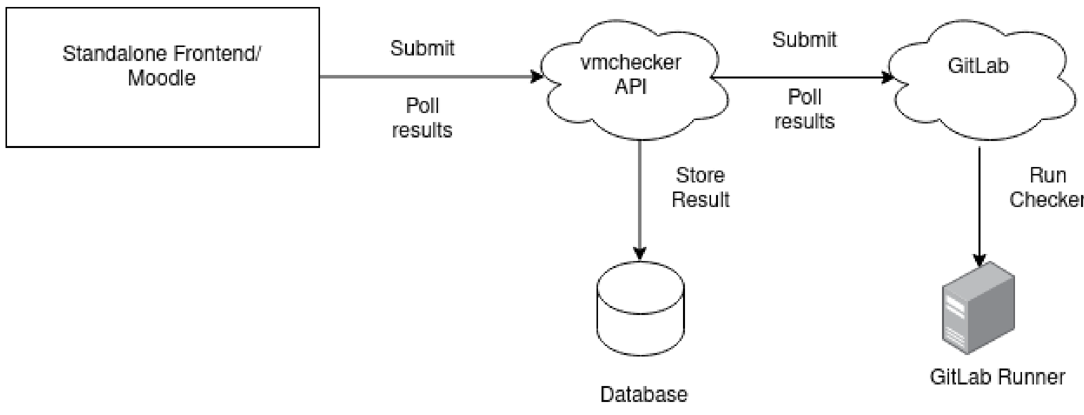


Figure 3.1: VMchecker architecture

Figure 3.1 displays the minimal components needed to have a working VMchecker service.

- front-end - receives assignments from students, uploads them to the VMchecker middleware, and receives the assignment run information after it has finished;
- middleware - receives requests from the front-end and uploads the homework to the assignment checking infrastructure, which runs the assignment, waits for the result, and then uploads it to the front-end;
- assignment runner - receives homework checking requests from the middleware service, and runs the homework using a specific recipe defined by the assignment.

Front-end

The first component required is a front-end that presents the students with a homework upload area and gives teachers a place to manage assignments and download and upload assignments.

For the first implemented example at UNSTPB, we have used a purpose-built Moodle plugin that connects to the already existing assignment module built into Moodle. By using Moodle, we have been able to reduce the amount of code needed to manage assignments, as this is already handled by Moodle.

Middleware

The middleware is the application that handles requests from the front-end. It offers a REST API so that multiple front-ends can be used, even in parallel, as they have to create HTTP requests to assignment management operations in VMchecker. The VMchecker middleware manages the lifetime of homework checking.

GitLab Integration

One can run code in the GitLab CI/CD infrastructure by creating a commit that triggers a pipeline that runs a configured script. VMchecker needs to be configured through the front-end with a project and access to the project's GitLab instance. This allows the middleware to create a commit through the middleware that triggers assignment checking.

GitLab Runner

A GitLab CI/CD pipeline needs physical resources to start the checker script. The CI/CD resources are managed by GitLab runners, which can be freely available on the platform, or they can be configured per repository if specific conditions are needed.

3.2.3 VMchecker Use Cases

VMchecker has been built with scenarios in mind for three different users:

- the institutional system administrator who manages VMchecker and has to maintain it and ensure a service level agreement, especially during high load situations, such as homework deadlines;
- the teaching staff whose role is to create an assignment and to grade the student assignments once the deadline has passed;
- the student who uploads the assignment, checks the output and receives the grade once it is marked by the teacher.

Each of these users must interact with VMchecker through a user-friendly interface. The task of building a front-end interface for the users takes a large amount of programming work to manage, keep up to date, and implement new features. This is the reason why we chose to use a front-end that was already built, maintained, and used at large educational institutions.

3.2.4 Container Infrastructure

The issue with using the same assignment checking script for many students is that they can have different libraries and applications installed on their systems. Different environments may lead to certain optimization inconsistencies. To mitigate this, we recommend teachers set up the checking infrastructure in VMchecker so that the scripts run inside of a container. The VMchecker team provides a minimal template that can be expanded based on individual class needs.

The container infrastructure presents an easier approach than the one seen in current solutions, which either require an internet connection for remote checking, or a virtual machine, which uses more resources. Using containers allows for easier patches to the environment, for example installing a new library, because the changes will be sent incrementally, the student not needing to download a whole new image.

3.3 Results

The Open Education Hub GitHub organization has been created to automate the operation of the different components that take part in class management and class creation. A methodology has been created to guide teachers in building and maintaining OER classes.

3.3.1 oer-builder

The GitHub organization has been created to maintain classes that have been built using the methodology and whose maintenance is passed to the Open Education Project. This frees content creators from the burden of managing pull requests and issues with the class.

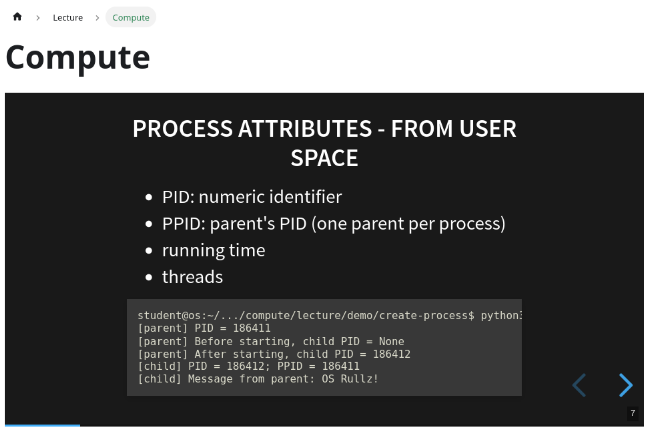
Classes can be freely forked and remixed by other projects or institutions; automation having been put in place to allow classes to have automatic deployment and linting using the GitHub API.

Three classes have been integrated into the Open Education Hub GitHub organization. These courses have been rewritten to take advantage of the oer-builder. They include quizzes, slides, figures, tutorials, and assignments in the repositories.

The Operating Systems Class

The Operating Systems class was the first one integrated. The repository is based on the class that is taught at the National University of Science and Technology POLITEHNICA Bucharest (UNSTPB). It was migrated from a wiki-based format where the content was hosted on institutional servers. While the wiki was easy to access for the students, including student feedback was difficult without giving students access to the whole class workspace which included internal documentation for the OS team.

The class is focused on tutorial work done by students as part of interactive sessions with TAs. Consequently, the largest part of the repository is taken up by written content and assignments.



Compute

PROCESS ATTRIBUTES - FROM USER SPACE

- PID: numeric identifier
- PPID: parent's PID (one parent per process)
- running time
- threads

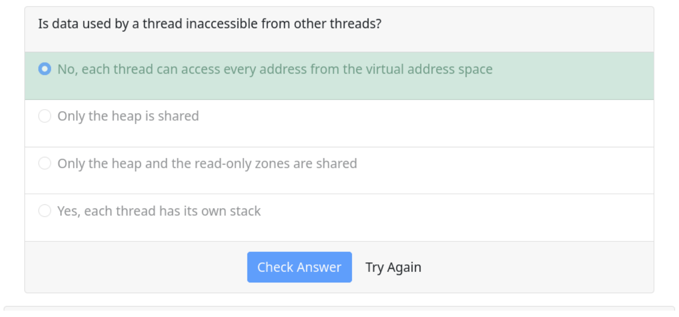
```
student@os:~/.../compute/lecture/demo/create-process$ python3
[parent] PID = 186411
[parent] Before starting, child PID = None
[parent] After starting, child PID = 186412
[child] PID = 186412; PPID = 186411
[child] Message from parent: OS Rullz!
```

Figure 3.2: Operating Systems Slides

Figure 3.2 displays slides built using the builder integrated into the operating systems class web page.

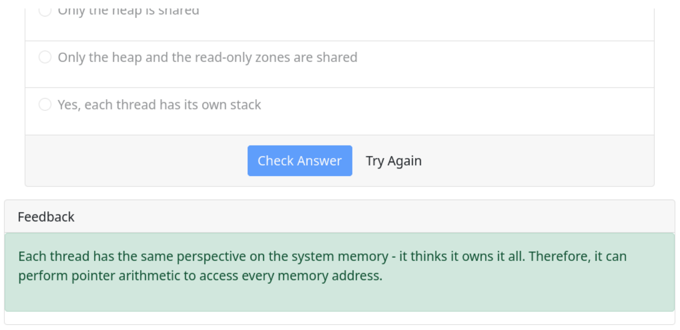
The quiz format we have deployed is displayed in Figures 3.3 and 3.4. We can see they display the right answer and an explanation for why it is correct.

The class has gone through two semesters of teaching based on the new materials. The idea of providing feedback and raising issues in GitHub has been promoted among the


Is data used by a thread inaccessible from other threads?

- [✓ No, each thread can access every address from the virtual address space]
- Only the heap is shared
- Only the heap and the read-only zones are shared
- Yes, each thread has its own stack

Figure 3.3: oer-builder quiz correct answer


- Only the heap is shared
- Only the heap and the read-only zones are shared
- Yes, each thread has its own stack

Feedback

Each thread has the same perspective on the system memory - it thinks it owns it all. Therefore, it can perform pointer arithmetic to access every memory address.

Figure 3.4: oer-builder quiz feedback

students, with them being rewarded with a physical Operating Systems pin for service to the community.

Students have opened 32 pull requests which have been integrated into the class materials.

The new content building process has allowed the team to scale up the amount of people who can work in parallel on the content. By using GitHub, each new content chapter had its pull request. This helped organize the feedback process, as the changes could be discussed on the page, and suggestions were made in-line. The content was automatically checked for format and spelling errors, allowing users to concentrate on the content quality. When merging the new content, it is automatically deployed, removing the need for a systems administrator to do this and coordinate the different merges.

CCAS

The CCAS class is a statistics class taught at the University of Iceland. It is a course focused on lab work based on a class written for the R programming language.

The class has been migrated from a lab book that was built in LaTeX and it was distributed as a PDF document. This approach is not easy to deploy, as you need a shared space with the students or a public website, it is not easy to search for, and it is not easy to contribute to, because the source code is not shared. Another issue is that LaTeX requires more expertise than a rich text format to write in, and the packages needed to use it are large and not ubiquitous on all platforms.

A script based on the `pandoc` tool was used to migrate the LaTeX content to Markdown. Because the classes were already in a text format, stored centrally, the process of converting them to another format was easier than having the resources dispersed in multiple classes or multiple PDFs.

The fact that we were able to use scripts to automatically convert content proves that other classes could convert their content to work with the oer-builder, and take advantage of its features.

Security Summer School

Security Summer School (SSS) is a workshop focused on teaching security principles. It is taught online, with resources hosted online. The resources were already built using Docsy [4], which is a documentation builder that converts MD files to HTML. It was already using GitHub Actions to automate resource deployment and pull request checking.

Although the setup was done following OER principles, it was difficult to replicate by other institutions and make changes locally because it was using Docsy-specific configuration files. A user would have to learn the Docsy configuration parameters to learn how to contribute to the project. This presented a hurdle in encouraging other users and institutions to contribute content.

The SSS workshop has been migrated to the Open Education Hub. It has been configured to run the oer-builder and integrate with Docusaurus. The migration was easy to do, as it only involved moving the Markdown files to a new repository which was configured using the builder.

3.3.2 VMchecker

VMchecker has been put into the production environment at UNSTPB as of September 2022. The middleware is connected to the UNSTPB Moodle instance through the Moodle VMchecker plugin. The middleware has been configured to connect to the GitLab instance deployed at UPB because we wanted to take advantage of the large storage space provided by the institution and the integration with institutional IDs. The integration allows students to add their TAs to the assignments to receive feedback and reviews.

VMchecker has been used by eleven different classes, each with different assignment structures and requirements. The following are noteworthy requirements and observations made from certain classes:

- The Operating Systems Internals class required starting virtual machines for code checking, a task that was supported.
- The Parallel and Distributed Algorithms class required multi-core support and resource isolation so the run time for different users will be computed in the same way for all users, as the assignments were graded based on solve time.
- The Computer Architecture assignments were GPU-dependant, this required a specific runner that had access to the UNSTPB cluster, where workloads could be

submitted to the institutional grid; the solution to this requirement has increased the potential size of VMchecker checking infrastructure from a GitLab runner running on a virtual machine inside the cluster to the whole institutional grid infrastructure, measuring at 600 cores and more than 2.5 TB of memory available.

- The Communication Protocols class required the use of raw networking capabilities, which were not enabled by default inside containers.

More than 40,000 assignments have been checked during the school year. The assignments used the container infrastructure to run the same assignment environment

3.4 Conclusion and Future Work

The Open Education Hub project has developed and deployed tools that help teachers reduce the administration load that is set on teachers. We have done this by creating workflows, templates, and frameworks that automate the tasks of integrating and deploying new materials and checking assignments. These actions have been done while also increasing educational resources availability by posting all materials online in an open source fashion, allowing third parties to both consume the content and reuse it.

We have built an Open Source tool that improves the process of creating materials and publishing them. Creating the tool, we have migrated three classes to the OER standard improving the process of maintaining the classes.

The task of automated assignment checking has been enhanced at UNSTPB by integrating VMchecker. A large number of classes have already picked up the new assignment checking tool, as it scales up better than other tools, and requires fewer setup steps. Using the existing institutional infrastructure, we have been able to increase the performance of homework checking, while making the process of grading, creating, and managing assignments easier.

The conversion process has already begun for other classes at the UNSTPB. We are targeting easy wins, classes that are already text-based, that can be more easily converted than writing them from scratch.

We aim to increase student and third-party engagement with the public resources available in Open Education Hub. To achieve this, we want to implement digital rewards using Smiley Coins, an educational cryptocurrency. Integrating it in a production environment will be a priority, as we have already implemented a Proof of Concept for the Operating Systems class.

Chapter 4

Highly available clusters for grid computing purposes

Cluster architectures have become a crutch on which institutions and companies lean to offer services to users and clients which would otherwise have not been available. Services such as Single Sign On, web hosting, shared storage spaces, learning platforms all need a backing infrastructure that manages their life time, assigns hardware resources and offers security assurances.

Virtual machines can be used as such a backing infrastructure, as they offer an isolated environment, running their own independent operating system. Users can take advantage of permissions in cluster environments that would otherwise not be granted, such as root access or network configuration privileges without putting at risk the underlying cluster infrastructure.

Since virtualized environments are now used to host mission-critical services, they have to be always available to serve their users. Clusters have to be designed considering fault-tolerance, to maximize server and virtual machine uptime. Measures have to be implemented so that the cluster state is continuously monitored so that faults can be prevented before they appear, as to not lead to service downtime. In the case that such a fault appears, it has to be detected and remedied as soon as possible. Monitoring systems fulfill this need, allowing for data points to be gathered from host systems, and be used to trigger alerts on events such as node failures, disk degradation or network bottlenecking.

The following chapter investigates deployment methods for highly available clusters, concentrating on using the OpenStack system, testing them in the UNSTPB cluster. As a result, mechanism for deploying highly available has been deployed, allowing for rapid response times in the case of host server failures. Monitoring systems have been designed and deployed in the UNSTPB cluster, gathering data from services and hardware to manage the status of running applications.

4.1 Creating Highly Available Environments

As the need for computing resources grows, so does the overhead to manage these resources. In our times, the computing resources that most people use are not hosted on our own machines, laptops, phones or workstations; they are hosted in cloud infrastructures, where they can be managed by system administrators using specialized software to wrangle the resources. The advantage of doing this is that you will move all the heat generation, power consumption and maintenance costs away from the users and towards the data center managers, while also benefiting from economies of scale when it comes to resource aquisition, power draw and cooling efficiency.

Most people use the cloud - from regular people who watch movies to programmers, researchers, and non-technical personnel who offload their workloads from local computers to remotely accessed systems.

Institutions can either choose to buy resources from large infrastructure or services providers, such as Amazon Web Service, Google Cloud, Oracle Cloud, Digital Ocean, and others, or they can buy resources and deploy them in a local cluster. The advantage of running services in an already deployed cloud platform is the fact that, while in the long run it might cost the institution more, the upfront cost of deploying a smaller service will be smaller, because you will not have to buy the hardware to run it, and the pricing will be dynamically scaled, based on the resource usage [27]. The main disadvantage is that for larger scale institutions, private clouds may become more economical, as the difference between private clouds and public clouds can lead up to being equivalent to multiple Full Time Employment salaries.

Our use case would be a large research institution, with more than 100 physical nodes, which expects to run more than 1000 medium or large scale virtual machines. For this kind of employment, it is more cost effective to deploy a private cloud solution.

As OpenStack [32] is the largest private cloud manager, and one of the largest open-source projects, we have chosen to deploy it on the nodes hosted at University Politehnica of Bucharest. Because OpenStack is such a large application, it has been developed to have many components that manage different subsystems, such as the image management service, the authentication service, the network management service and many more.

Our intent for the OpenStack cloud cluster is to host both student virtual machines, which are used for running different test and laboratories workload, but also for hosting mission critical services, such as homework checkers, data repositories, authentication platforms and others. Because of this, we need a mechanism that will guarantee high availability without increasing the resource usage of the nodes.

This thesis aims to explore the ways in which an operator can deploy a large OpenStack cluster and what are the options available in which one can increase the availability of resources in case of node or process failure without increasing the hardware costs of the cluster.

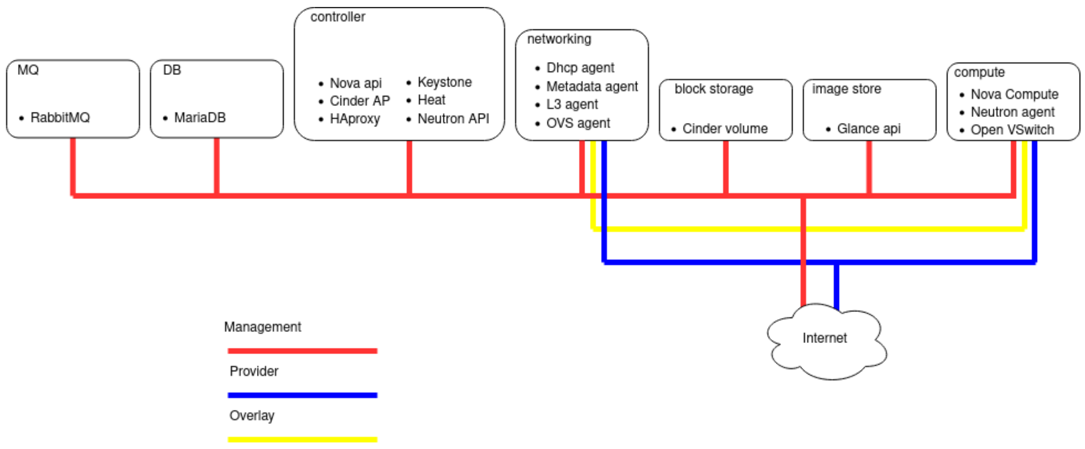


Figure 4.1: Openstack Node architecture

4.1.1 OpenStack Cluster Infrastructure

OpenStack Components needed for running a large cluster

Because OpenStack is a large and modular system, a site administrator can pick and choose what services are needed in order to run a cluster that is tailor made for the particularities of their workloads.

The cluster infrastructure at UPB has a mixed workload: The research teams, run different services for their outward facing purposes, such as project dissemination, external user access and others, and computing workloads, such as Machine Learning, image processing and High Energy Physics. The second type of usage is for students who run some of their homework and lab work on the cloud infrastructure, which puts a load on the control plane because they build and delete many machines.

The services we have chosen to deploy at UPB are represented in Figure 4.1.

In this subsection we will take a look at all the services deployed, and we will explain their use inside of the cluster, and also the changes that we have made to them in order to run.

Keystone Keystone [10] is the identity management service, which handles user and service authentication and authorization. By default it stores all the users in a database. Because we are part of a large institution, the cluster has to support authentication using institutional IDs. In order to accomplish this, we configured Keystone to connect to the LDAP instance where user data is stored. Keystone is configured on the controller node.

Database and Message Queues The MariaDB[5] database and the RabbitMQ[36] instances were configured by Kolla-ansible on separate nodes, because they are critical services that must be able to communicate, even if the control plane is offline, because the compute and networking services also communicate through these nodes.

Glance image service The Glance [8] image service stores the base images for the machines, from which they will be copied on the hypervisors. This service is configured with the default settings on a separate node from the other control plane services because it needs to be connected to a large storage device, since it will store many large files, and it needs a good storage connection.

Nova The OpenStack Nova [12] service is the core component which manages the Virtual Machine resource scheduling, lifetime management. This service has many components that are managed by Kolla-ansible such as:

- Nova API, which receives requests from the user about VM operations such as boot up, shut down, creation, deletion and others;
- Nova Conductor, also deployed on the controller node, which manages communication with the systems hosting VMs;
- Nova Scheduler, which picks the host systems where the VMs will be running;
- Placement, which is a separate service but which is tightly integrated with Nova, as it manages the resource inventory, knowing which systems have what resources available;
- Nova compute, which is a daemon deployed on the compute machines which actually starts up the virtual machines, prepares the environment, downloads the image files and makes sure that the VMs are running correctly.

Neutron Neutron [11] is the OpenStack service which manages the network. OpenStack provides support for running virtual machines on already existing networks, or it can provide Software Defined Networks (SDNs) that can be created at will by the users. This advantage lets us dynamically allocate public IPs, and also isolate communication between critical VMs and regular ones by creating virtual networks that run over the overlay network, as seen in Figure 4.1.

Cinder Cinder [7] is the block storage solution used by OpenStack. This service allows a Virtual Machine to use a remote storage, connected through remote storage protocols. The advantage of removing the VM storage from the node is the fact that if there is a node failure the data will not be lost, because the storage is on another medium. The storage volumes are mainly used for critical services, which cannot afford the downtime. We cannot use Cinder for all Virtual Machines, because it will be too much of a resource drain.

Heat The Heat [9] service is the OpenStack orchestration service. This in itself is not a core service, but it is widely used because it can ease the user's job when repeatedly creating similar virtual machines. It takes a resource specification as input, and it generates the requested virtual machines, the request API being the same as the popular AWS CloudFormation [2] API.

4.1.2 Automating OpenStack Install

Since the entire infrastructure involves a large number of nodes that need to be configured, we want to have a solution where system administrators put as little effort as possible into integrating machines into the system. For the software stack required to run OpenStack, we need to ensure that the same versions of applications are installed on all systems, and we want to have the right configuration files for the role of all stations in the system. In order to ensure that we meet all these conditions, we want to automate as much of the application installation and configuration system as possible, thus ensuring the homogeneity of software solutions and the configuration of systems with as little intervention by administrators as possible.

Automating core system deployment

The first step in preparing a system to run on the OpenStack infrastructure is to install the operating system and core applications that OpenStack needs. To ensure that all systems use the same applications, we use the installation automation solution provided by the CentOS operating system, as it provides an easy scripting interface through which we can specify which applications to install. Another advantage of the solution offered by CentOS, called Kickstart, is that in addition to installing applications, we can configure disk partitioning and set options to installed applications.

Since we want to automate the process of installing operating systems as much as possible, we have chosen to use the PXE boot system for remote installation of the operating system. PXE boot is a mechanism that instructs a system to boot a basic operating system over the network. Once the operating system is loaded onto the network, it will perform a set of operations. In the case of CentOS, the server will download the previously described Kickstart script and interpret each line in it. Once the script is finished running, the system will have a fresh operating system installed with all the necessary applications for the subsequent installation and running of the OpenStack system.

Automating the installation of OpenStack services using Kolla Ansible

Depending on the OpenStack service in question, there is a list of applications that it depends on, from the OpenVSwitch application for the network management service, to the libvirt library for the virtual machine management service on the host system. In addition to dependencies, each service needs a service-specific configuration file. Thus, it becomes a burden for the administrator to manually configure each OpenStack service on each host system and ensure that all applications remain in the same operational state.

For this reason, we chose to use the service installation automation solution based on Kolla Ansible. Kolla Ansible is a set of recipes written in the format specific to the Ansible software suite, which allows us to install and configure the entire OpenStack installation using only two major configuration files: a configuration file where we specify system options, `/etc/kolla/globals.yaml` and a configuration file where we configure the association between the names of the services and the nodes on which they must be installed. `/etc/kolla/multinode`.

Authentication System Installation Automation To install the authentication system, it was necessary to modify the source files of the Kolla Ansible service, because it did not allow configuring predefined usernames, especially for admin users, who manage each service individually. We needed to do this because admin users are predefined in the university's already existing LDAP infrastructure.

Automating Compute Service Installation Because each physical server we run virtual machines on has a different configuration, depending on the server model used, the network card used, and other hardware-related aspects that we cannot control, it was necessary to modify the configuration file where we defined associations between roles and nodes.

To make configuring nodes easier and reduce the complexity of the configuration file, we have assigned each node family a name, and then mapped the node family to the type of service that will run on the node.

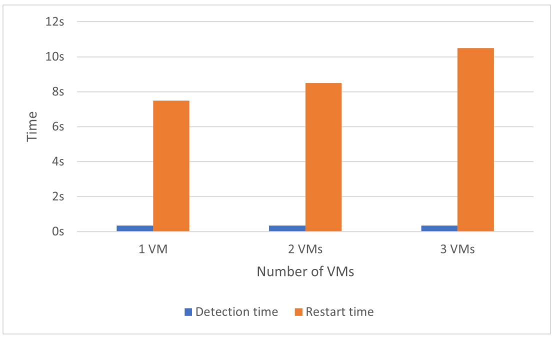
4.1.3 Running High Availability Virtual Machines

High-availability not only helps with user satisfaction by assuring higher uptimes without the need for human intervention, but provides a safety net for important or critical services deployed inside OpenStack. The high-availability of resources in OpenStack can be achieved using the Masakari [6] service. Masakari provides a recovery mechanism for VM instances in the case of failures at multiple levels across the deployment. Its architecture makes use of independent monitors for processes, instances and hosts, generating notifications for the Masakari engine to process and trigger a recovery sequence. Since our deployment is based on the Kolla Ansible tool, which automatically deploys OpenStack services in containers, the process monitoring function of Masakari will not be included due to the limitations imposed by process namespace separation in containerized environments. The process monitor can be deployed manually on the host and configured to monitor the processes that launch containers in execution, restarting said containers if needed. Since this complicates the deployment process we choose to only focus on instance and host monitoring in our testing.

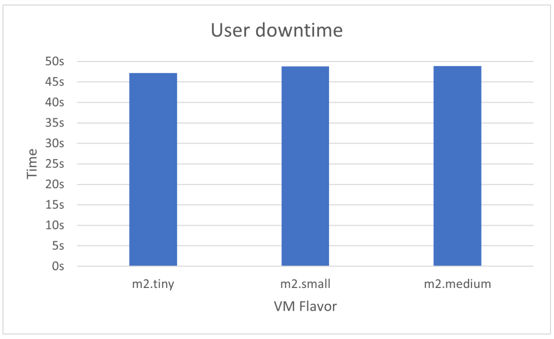
Virtual machine monitoring tracks hypervisor processes started on compute nodes (in our case `qemu`); in case of sudden termination of such a process, the virtual machine is launched back into execution using the initial command. Monitoring of compute nodes uses already existing software (Corosync and Pacemaker [41]); when a computing node shutdown is detected, the evacuation process is triggered: the virtual machines on that node will be launched again in execution on available nodes. In both cases it is possible to specify, by using metadata, the restart or evacuation of a subset of virtual machines.

4.1.4 Virtual Machine Recovery Performance

The tests were split in different scenarios, varying parameters such as VM flavor, disk size, operating system and number of VMs. The flavors used in our tests can be seen in Table 4.1. The results and a summary of each scenario will be presented in the next paragraphs.



(a) Detection and restart time for varying number of VMs



(b) User downtime for varying VM resources

Figure 4.2: Instance monitoring performance

Instance monitoring For instance monitoring, we first measured detection time and time to restart, varying the number of VMs in the process. We deployed up to three Ubuntu Server 22.04 VM, m2.tiny flavor with a 10 GB disk. Results are shown in Figure 4.2a. Detection time remains constant at 350ms, with the total time to restart all the VMs increasing linearly with the number of VMs.

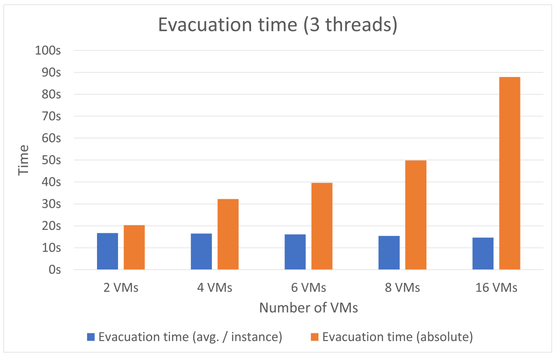
Second test was done with the intent of measuring user downtime. For this, we deployed a single Ubuntu Server 22.04 VM with a 10 GB disk, varying the flavor of the VM from m2.tiny to m2.medium. From Figure 4.2b we can observe that the resources of a VM do not influence the user experienced downtime. This is in part due to our web server serving static content; a more complex application might benefit from more resources.

Host monitoring For host monitoring, we measured evacuation time, both absolute and average per instance, and user downtime. In the case of host monitoring, Masakari allows the user to configure the number of threads used for the evacuation process. This first scenario varies the number of VMs from 2 to 16, in steps, with 3 (the default) and 8 evacuation threads. The controller processing these notifications had 16 CPU cores. The VMs were Ubuntu Server 22.04 with a 10 GB disk and the m1.small flavor.

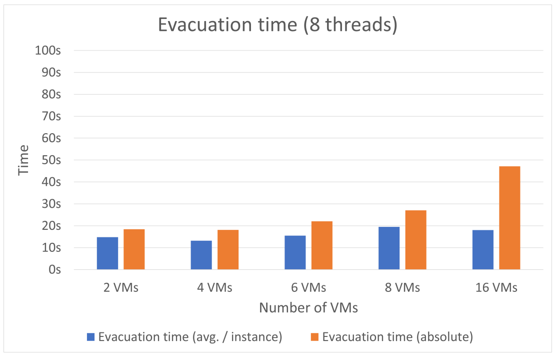
From Figure 4.3a we can see that the absolute evacuation time is similar to the average evacuation time when the number of VMs is smaller than or equal to the number of threads, increasing abruptly when this threshold is passed. Figure 4.3b shows the results for 8 threads. Since the number of threads causes a higher load on the controller node, this increases the average evacuation time, but the absolute evacuation time is overall

Flavor	vCPUs	RAM
m1.small	1	1 GB
m2.tiny	1	512 MB
m2.small	1	2 GB
m2.medium	2	4 GB

Table 4.1: Virtual machine flavors



(a) 3 evacuation threads



(b) 8 evacuation threads

Figure 4.3: Evacuation time for varying number of VMs and evacuation threads

better than with less threads, following the trend seen before.

User downtime is not influenced in an absolute way (reduced or increased). Instead, with more evacuation threads than CPU cores, the user downtime starts to vary more than before, as seen in Figure 4.4.

Variations such as operating system choice and VM flavor did not have a perceivable impact on evacuation time or overall user downtime.

4.1.5 Conclusion and Future Work

In the previous subsections we have looked at ways in which an institution can deploy a private cloud infrastructure in order to save costs by not running virtual machines in a public cloud infrastructure. We have explored the different deployment strategies, and what they add to installing OpenStack, and we have made an argument for why UPB has chosen Kolla-ansible as a deployment system for OpenStack.

Kolla-ansible provides us with a good balance between the configurability of Ansible roles and the ease of use of deploying pre-build Docker container on our own cluster, and we have showcased how this has improved the deployment strategy for the UPB private cloud while highlighting the different OpenStack components deployed using OpenStack.

We have also explored the performance of instance and host high-availability in OpenStack, using Masakari. While instance high-availability works as expected, we encountered a few issues with host monitoring. One such issue was the stability, marked by random occurrences of service disabling on healthy nodes. We plan on investigating the root cause of this.

An issue highlighted by our research was the performance, with up to 7 minutes of user downtime in the case of a host failure. Analysis showed us that half of this time is spent in bugs regarding to notification processing or in the default values for configurable timeouts. Our plan is to investigate the necessity of the timeouts, since the need for them is not documented or clearly stated.

We will also keep on improving the security of the private cloud offering, taking into

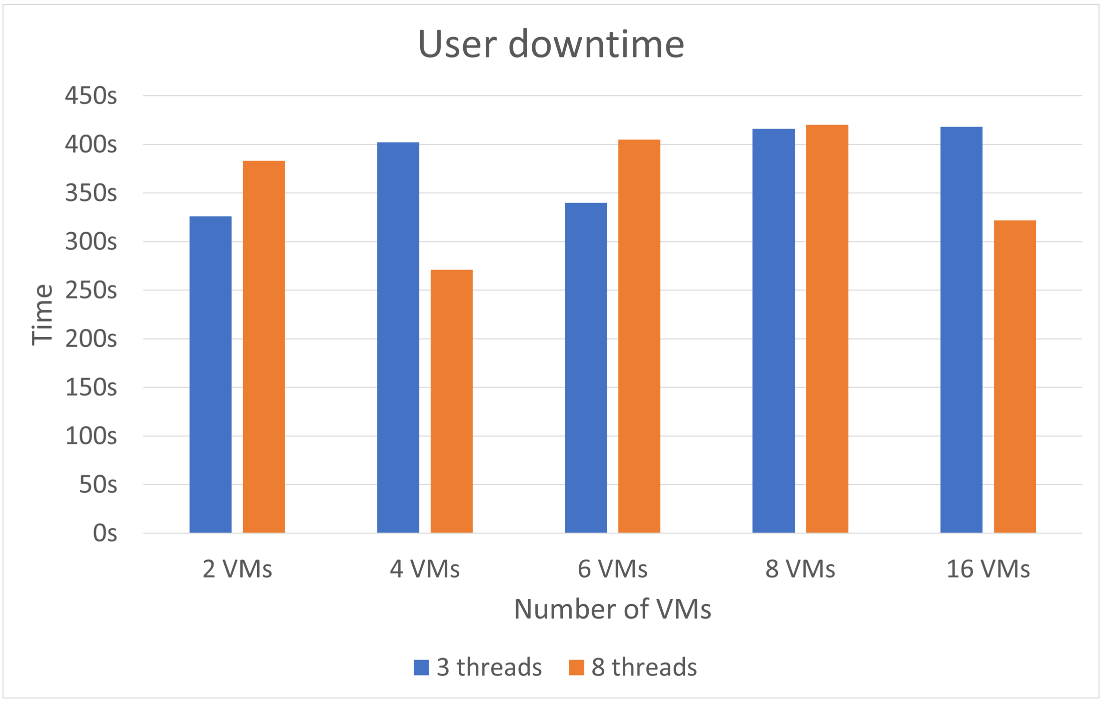


Figure 4.4: User downtime for varying number of VMs and evacuation threads

consideration the OpenStack security guidelines and building upon them.

4.2 Monitoring Cluster Infrastructures

As the size of institutional IT clusters has increased, the likelihood of systems failure has turned from a rare occurrence into an expected cost of doing business. In order for the mission critical services to not be affected system issues have to be detected or prevented from appearing.

System administrators use monitoring tools to extract information about the state of nodes and applications. The status can be represented by metrics, such as request rate, memory usage, response time for applications, or resource usage for hosts. Logs can also be an information source that can indicate issues with nodes or services.

There exists a breadth of monitoring and observability tools that can be deployed and configured. Each solution has advantages, such as optimized information lookup, visualization support, easy scalability and setup.

In this thesis we will present the current state of the art of free and open source monitoring solutions. We will focus on their advantages, deployment use cases and disadvantages, taking into account scalability, visualization support, alerting support.

Based on the state of the art we will be presenting a use case for deploying monitoring solutions in a HPC cluster environment situated at the National University of Sciences and Technology Politehnica Bucharest (UNSTPB). The infrastructure description will include the services and types of nodes that we integrate and issues that we have had in deploying the monitoring tools.

4.2.1 Monitoring Infrastructure

Based on the state of the art for monitoring solutions we have had to make a choice of what system to use in the institutional deployment. The monitoring solution selected should be easy to set up, and configure. It should provide a flexible node configuration system that allows for splitting hosts into groups. We should be able to visualize the state of the different systems deployed, with further detailed panels available for deep dives. An alerting system should be setup which checks the node states and notifies administrators of outages and system failures.

The following chapter presents the infrastructure deployed at UNSTPB, which satisfies the above mentioned requirements, using free and open source software. We will be detailing the configurations made, the software solutions chosen and we will display the resulting visualizations.

Cluster infrastructure overview

The UNSTPB cluster is made up of an OpenStack [32] private cloud infrastructure that servers the compute needs of the teaching and research staff. The private cloud runs on physical machines distributed in three data centers which serve as both compute and service nodes collocated, installed through the `kolla-ansible` [40] deployment tool. The cluster also hosts a SLURM [42] grid that runs compute-intensive workloads for users who want to run multi-machine OpenMPI [37] jobs or GPU accelerated applications. Both the grid and cloud software use a CEPH [38] storage cluster which is deployed in a single data center location.

Managing OpenStack logs

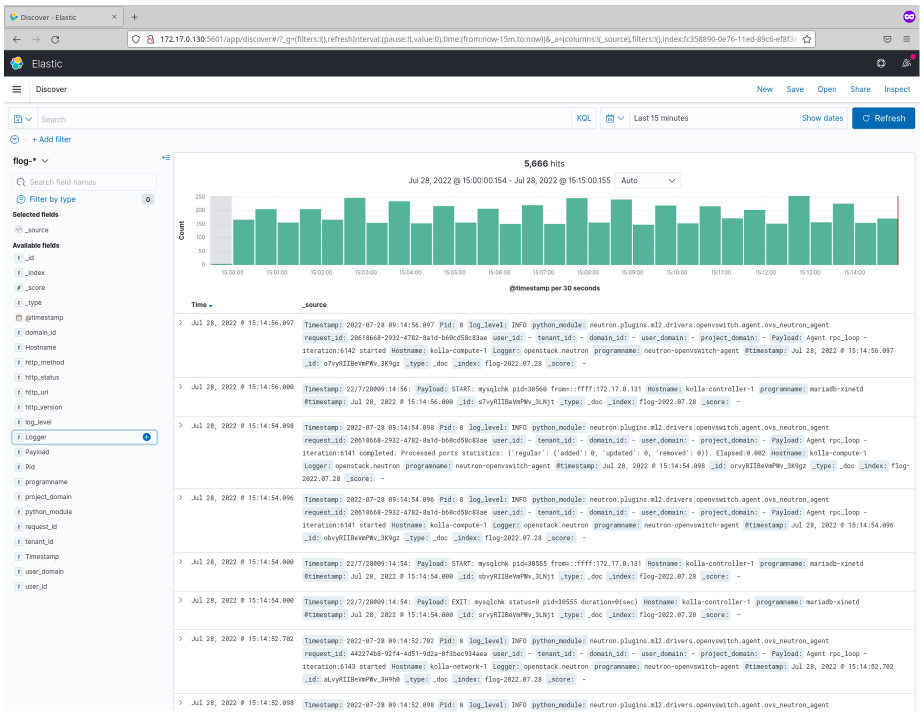
To achieve this we need to take preventive measures to ensure that the system has as much uptime as possible and issues can be solved before they impede system functions. Log messages can be ingested into a centralized log management system so that we can easily inspect the health of services and investigate service logs easily.

We chose to use industry standard solutions such as the ELK Stack, consisting of Logstash, Elasticsearch and Kibana to reduce the amount of effort to maintain log ingestion, parsing and display information. We use syslog to aggregate service logs and send them to a central log aggregation systems.

We have installed Kibana to view the logs once they are uploaded. Figure 4.5 displays a log filter used in Kibana to display OpenStack logs. The filter detects `WARNING` type messages launched by the `nova-compute` service within the Nova service.

Implementing the runtime data collection service

The OpenStack infrastructure hosts a large number of services and the nodes, which requires a scalable method to regularly monitor host and service health. Logs are not enough to inspect the health of a service, because they are a way to display certain errors, but they do not display a simple to parse message. Metrics can be used to parse at a glance the general state of services and systems. They can be used to determine both the operational parameters in which services and systems are expected to run, and



```
Time          _source
Jul 28, 2022 @ 15:14:56.097  Timestamp: 2022-07-28 09:14:56.097 Pid: 8 log_level: INFO python_module: neutron.plugins.ml2.drivers.openvswitch.agent.ovs_neutron_agent
  request_id: 20618668-2932-4782-8a1d-b60cd58c83ae user_id: - tenant_id: - domain_id: - user_domain: - project_domain: - Payload: Agent rpc_loop -
  iteration:6142 started Hostname: kolla-compute-1 Logger: openstack.neutron programname: neutron-openvswitch-agent @timestamp: Jul 28, 2022 @ 15:14:56.097
  _id: o7vyRIIBeVmPWv_3K9gz _type: _doc _index: flog-2022.07.28 _score: -
Jul 28, 2022 @ 15:14:56.000  Timestamp: 22/7/28@09:14:56: Payload: START: mysqlchk pid=30560 from=::ffff:172.17.0.131 Hostname: kolla-controller-1 programname: mariadb-xinetd
  @timestamp: Jul 28, 2022 @ 15:14:56.000 _id: s7vyRIIBeVmPWv_3LNjt _type: _doc _index: flog-2022.07.28 _score: -
Jul 28, 2022 @ 15:14:54.098  Timestamp: 2022-07-28 09:14:54.098 Pid: 8 log_level: INFO python_module: neutron.plugins.ml2.drivers.openvswitch.agent.ovs_neutron_agent
  request_id: 20618668-2932-4782-8a1d-b60cd58c83ae user_id: - tenant_id: - domain_id: - user_domain: - project_domain: - Payload: Agent rpc_loop -
  iteration:6141 completed. Processed ports statistics: {'regular': {'added': 0, 'updated': 0, 'removed': 0}}. Elapsed:0.002 Hostname: kolla-compute-1
  Logger: openstack.neutron programname: neutron-openvswitch-agent @timestamp: Jul 28, 2022 @ 15:14:54.098 _id: orvyRIIBeVmPWv_3K9gz _type: _doc _index: flog-
  2022.07.28 _score: -
Jul 28, 2022 @ 15:14:54.096  Timestamp: 2022-07-28 09:14:54.096 Pid: 8 log_level: INFO python_module: neutron.plugins.ml2.drivers.openvswitch.agent.ovs_neutron_agent
  request_id: 20618668-2932-4782-8a1d-b60cd58c83ae user_id: - tenant_id: - domain_id: - user_domain: - project_domain: - Payload: Agent rpc_loop -
  iteration:6141 started Hostname: kolla-compute-1 Logger: openstack.neutron programname: neutron-openvswitch-agent @timestamp: Jul 28, 2022 @ 15:14:54.096
  _id: obvyRIIBeVmPWv_3K9gz _type: _doc _index: flog-2022.07.28 _score: -
Jul 28, 2022 @ 15:14:54.000  Timestamp: 22/7/28@09:14:54: Payload: START: mysqlchk pid=30555 from=::ffff:172.17.0.131 Hostname: kolla-controller-1 programname: mariadb-xinetd
  @timestamp: Jul 28, 2022 @ 15:14:54.000 _id: sbvyRIIBeVmPWv_3LNjt _type: _doc _index: flog-2022.07.28 _score: -
Jul 28, 2022 @ 15:14:54.000  Timestamp: 22/7/28@09:14:54: Payload: EXIT: mysqlchk status=0 pid=30555 duration=0(sec) Hostname: kolla-controller-1 programname: mariadb-xinetd
  @timestamp: Jul 28, 2022 @ 15:14:54.000 _id: srvyRIIBeVmPWv_3LNjt _type: _doc _index: flog-2022.07.28 _score: -
Jul 28, 2022 @ 15:14:52.702  Timestamp: 2022-07-28 09:14:52.702 Pid: 8 log_level: INFO python_module: neutron.plugins.ml2.drivers.openvswitch.agent.ovs_neutron_agent
  request_id: 442274b8-92f4-4d51-9d2a-0f3bec934aea user_id: - tenant_id: - domain_id: - user_domain: - project_domain: - Payload: Agent rpc_loop -
  iteration:6143 started Hostname: kolla-network-1 Logger: openstack.neutron programname: neutron-openvswitch-agent @timestamp: Jul 28, 2022 @ 15:14:52.702
  _id: aLvyRIIBeVmPWv_3H9h0 _type: _doc _index: flog-2022.07.28 _score: -
Jul 28, 2022 @ 15:14:52.098  Timestamp: 2022-07-28 09:14:52.098 Pid: 8 log_level: INFO python_module: neutron.plugins.ml2.drivers.openvswitch.agent.ovs_neutron_agent
```

Figure 4.5: Logs lookup in Kibana

also deviations from the expected behaviour. Logs would only allow us to see when a system actually fails, not the iterative process of its failure.

For the collection of statistical data, we chose to use the Prometheus service. Prometheus works by connecting to a list of endpoints configured to export runtime parameters.

We chose to use Prometheus because it runs using the pull model, where it connects to services, instead of the push model, where services connect directly to a central node.

The disadvantage of using Prometheus is that we have to ensure connectivity between the system on which Prometheus is running and all the stations from which we want to collect information.

Prometheus provides a graphical interface in the form of a web page where we can run queries and view the results.

Since it was necessary to configure the Prometheus service for more than 100 physical nodes, each hosting several services, we developed a set of scripts that generate Prometheus configuration files based on a set of services and node groups that are associated with the.

We have integrated the following information using different exporters:

- OpenStack service status and statistics;
- Compute node resource usage;
- Web service status, such as GitLab, login services, e-learning platforms and storage systems;

- SSL certificate lifetime for managed web services;
- Data center generator power and fuel status;
- ICMP prober for checking latency between systems;
- SLURM grid metrics;
- Storage statistics such as bandwidth usage and IO time;
- GPU usage metrics.

Investigating the health of services using dashboard

For advanced investigation of systems health, we chose to use the Grafana service to generate graphs and dashboards from which we can visualize complex images based on time series. We have chosen to use Grafana because of its extensive visualization support, because different services have different visualization needs. Grafana offers us the query functionality of the Prometheus service, so we can use the Prometheus Query Language queries that we have used so far directly from the Prometheus dashboard to easily view the information.

Grafana is configured to access information based on the time frame we request access to. Since there have been cases where there are too many systems for which we collect the information, we have chosen to add to each metric in Prometheus a tag that defines the type of service provided, so that we can filter the metrics in queries based on the type of service we want to investigate.

The advantage of using Grafana for graphs is that it has a very flexible suite of visualization modes, which allows us to present information in many different ways, from regular graphs and numerical summaries to heatmaps and histogram graphs. We can integrate other dashboards in Grafana by creating our own or using the existing dashboards published by the community.

We have integrated into Grafana dashboards for viewing the following elements:

- OpenStack infrastructure status;
- Connection latency between nodes;
- Web services status;
- Power generator status;
- Resource consumption of host nodes;
- CEPH storage status;
- Grid services status and usage;
- GPU usage per node.

Monitoring disk health

UNSTPB runs a large amount of hard disks for different use cases. Hard disks wear out over time, which requires monitoring for errors that might appear due to hard disk failure.

We needed to have a method to check the disk health and prevent a situation where they might fail during high demand loads. To inspect the disk state at a given time, we can use the S.M.A.R.T. metrics exposed by the disk, which gives us insight into disk wear. The S.M.A.R.T. metrics that we are interested in are the read error rate, the running temperature of the disk, and how many errors occur when transmitting data from the disk to the system. They can indicate the first signs that a disk might fail in the near future.

To interpret the S.M.A.R.T. characteristics we use the values recommended by the manufacturers for the condition of the discs.

To expose the metrics in an easy-to-read environment, we used a script that collects the S.M.A.R.T. and exposes them using the already installed `node_exporter` service.

The Prometheus service, installed on the Grafana instance automatically, pulls the metrics. They are used to output generate Grafana dashboard through which we can view the state of the disks, where disks suspect of failure can be highlighted through the S.M.A.R.T. metrics.

4.2.2 Alerting Infrastructure

In this subsection we will be discussing the choices made in order to implement an automated alerting system which can be configured using the existing metrics to send messages to the system administrators if issues are detected with the systems.

Because we have access to a large amount of metrics by accessing the Prometheus service, we can easily determine the state of the systems at any given time.

Infrastructure issues must be detected as early as possible, and administrators must be notified in advance when a service is not operating within appropriate parameters. For this reason we want to have the fastest possible method of being alerted when problems arise.

We have chosen to use AlertManager as an alert system. The advantage of using AlertManager is that it can connect directly to Prometheus and query the metrics database. Thus, we can use AlertManager to query the state of the system constantly and send messages to responsible administrators in case a service stops working, or technical failures have occurred at a node.

We have enabled the AlertManager service to send mail directly to system administrators for any critical event that occurs, to allow early notification of administrators. If successive events occur, administrators will be notified for each. If alerts are false alarms, or are an acknowledged issue, administrators can disable rules using expressions to granularly disable alerting rules.

We have created AlertManager notifications for the following situations:

- Network services that cannot be contacted;
- Too much disk space used on a node;
- Computing services that cannot be contacted;

- Web services that cannot be contacted;
- Too much memory used on nodes;
- A web service's certificate will expire in the next two weeks;
- A system cannot be contacted.

We use the metrics exposed through S.M.A.R.T. to detect a disk failure before it occurs. We use the `WORST` and `THRESH` to identify if a drive has already shown signs of failure. If the `WORST` value approaches the `THRESH` value, it means that the drive will fail soon.

4.2.3 Conclusion and Further Work

We have researched the optimal monitoring system for managing metrics for a private cloud infrastructure which also host institutional and e-learning systems.

Each viable monitoring solution has been analyzed and we have deployed Prometheus to download metrics from nodes, storage systems and services. Grafana was used to display the metrics in a new dashboards that allow administrators to see at a glance the cluster state.

We have deployed an alert system for notifying administrator when services fail, SSL certificates expire, or when disks show signs of degradation. The new system has allowed us to pre-empt disk failure incidents by replacing disks and rebuilding storage arrays in time and notice anomalous events in service behaviours leading to bug fixes.

In the future we aim to move the monitoring setup from virtual machines to a container-based environment with automated setup, so that it can be easier to install and replicate for further use.

Chapter 5

Conclusion

Distributed computing has allowed more processing to be done in parallel, accelerating both research and education by making it easier for users to run complex workloads and automation in a remote environment. The demand for more parallel processing has led to the adoption of cluster computing and grid computing.

In this new and dynamic environment, we must discover new ways to include resources in cluster and grid computing and adapt them to our workloads, so that we can take advantage of their power. To take advantage of these newly introduced resources we must also be able to make use of them efficiently, spending less time in non-computing tasks such as environment preparation, such as transferring files.

5.1 Summary

This thesis has proposed new avenues for using middleware to enhance cluster and grid environments by increasing resource usage efficiency through improving parallelism during file transfers and tapping into new possible resource types such as scavenging queues. By implementing support for multi-core jobs we have enabled the ALICE grid environment to workloads with a lower memory-per-core footprint, allowing for more jobs tu run in a memory restricted environment.

We have also deployed the same middleware techniques used to manage grid infrastructures to automate educational activities such as assignment checking and class materials building. We have used existing developer-centric solutions such as GitLab and CI/CD documentation builders to achieve our goals of automating class activities for teachers.

A high availability cloud computing infrastructure has been deployed in the UNSTPB cluster, using modern day deployment techniques, offering fault resistant virtual environments for research and education-oriented users. The thesis has defined through the case study of the UNSTPB cluster reference deployment models for monitoring infrastructures and high availability cluster deployments used for virtualizing educational services and grid workloads.

5.2 Contributions

The thesis analyzes possible optimizations that can be done to grid transfers to increase transfer bandwidth and decrease transfer times. In the context of the ALICE experiment transfer campaign, we have been able to **decrease the transfer times** for data generated by the experiment by increasing the rate of transfers handled by the grid services based on the number of files transferred and their size. During the transfer tests, we have been able to achieve the advertised rate of 100 GBps, which would allow the experiment to continuously transfer data to storage sites without its main buffer being filled. By increasing transfer rates we have made data processing faster, as it can be accessed sooner by interested parties.

We have decreased the memory usage for the ALICE grid by implementing support for multi-core job management, enabling lower RAM usage jobs to run on the grid, thus giving memory-deficient sites the ability to run more jobs in parallel without increasing the available RAM. In implementing multi-core job support we have taken into account the issue of workloads running on more cores than allocated, **demonstrating the ability to limit used CPU cores by using the `taskset` mechanism**.

We have identified in this thesis a class of clusters that could be integrated into the ALICE grid environment if support for managing multiple jobs in parallel through a single pilot job were added. In the ALICE experiment, we have implemented support for running multiple payloads from the same pilot jobs and managing a multi-core slot or a whole node by accounting for job resource usage and currently available resources. **By implementing support for multi-job slots we have been able to integrate more computing resources in the grid.**

As part of this thesis software has been **designed and developed to help educators in automating documentation generation**. The framework has been developed based on DevOps techniques for generating class materials, creating a pluggable framework that now implements Docusaurus to output HTML content based on markdown files and publish the files to the Internet using GitLab CI/CD. With classes being automatically built and published online, other teachers can pick up the material, edit it using Git, improve it, and publish it automatically, without the need to build their own.

An assignment-checking solution, called vmchecker, has been developed, based on the job managing principles behind grid management software, using workloads to represent student assignments and CI/CD pipelines for sites. vmchecker allows both users and teachers to check assignments using the same environment and scripts, and it automatically uploads the checking results to a front-end viewer, such as the e-learning platform Moodle. The platform having been deployed in the UNSTPB cluster and used for more than 10 subjects, it has checked more than 40.000 assignments for students and decreased the time teachers spend on grading and checking students' assignments by hand.

The thesis proposes an architecture for deploying institutional cluster environments emphasizing reliability, flexibility, and ease of use for the end-user. **An example architecture has been deployed in the UNSTPB cluster**, running OpenStack and hosting research projects and institutional services in a highly available manner, on virtual machines-based systems.

The field of cluster monitoring has been explored in this thesis, focusing on tools that can help administrators maintain a view of the cluster state at a given moment. Monitoring tools have been investigated which can allow users to troubleshoot complex systems and enable the use of preemptive measures such as alerts to prevent service issues. **The thesis proposes a reference architecture deployed at UNSTPB based on Grafana and AlertManager**, configured to pull node data, service data, and storage health and alert concerned parties in case of a fault such as a disk failure or a service not responding to requests.

Bibliography

[1] alimonitor transfers web page. https://alimonitor.cern.ch/transfers. Accessed: 2023-02-12.

[2] Amazon Web Services Cloudformation. https://docs.aws.amazon.com/AWSCloudFormation/latest/A Accessed: 2023-02-12.

[3] Checkmk website. https://www.checkmk.com. Accessed: 2023-10-04.

[4] Docsy documentation builder. https://www.docsy.dev/. Accessed: 2023-09-30.

[5] MariaDB. mariadb.org.

[6] Masakari. https://docs.openstack.org/masakari/latest/. Accessed: 2023-02-12.

[7] OpenStack Cinder. https://docs.openstack.org/cinder/latest/. Accessed: 2023-02-12.

[8] OpenStack Glance. https://docs.openstack.org/glance/latest/. Accessed: 2023-02-12.

[9] OpenStack Heat. https://docs.openstack.org/heat/latest/. Accessed: 2023-02-12.

[10] OpenStack Keystone. https://docs.openstack.org/keystone/latest/. Accessed: 2023-02-12.

[11] OpenStack Neutron. https://docs.openstack.org/neutron/latest/. Accessed: 2023-02-12.

[12] OpenStack Nova. https://docs.openstack.org/nova/latest/. Accessed: 2023-02-12.

[13] PostgreSQL. https://www.postgresql.org/. Accessed: 2023-02-12.

[14] reveal-md git repository. https://github.com/webpro/reveal-md. Accessed: 2023-09-30.

[15] taskset(1) — linux manual page. https://man7.org/linux/man-pages/man1/taskset.1.html. Accessed: 2023-10-04.

[16] XRootD documentation. https://www.xrootd.org. Accessed: 2023-02-12.

[17] K. Aamodt et al. The ALICE experiment at the CERN LHC. *JINST*, 3:S08002, 2008.

[18] B Abelev et al. Upgrade of the ALICE Experiment: Letter Of Intent. *J. Phys. G*, 41:087001, 2014.

[19] Maria Arsuaga-Rios, Vladimír Bahýl, Manuel Batalha, Cédric Caffy, Eric Cano, Niccolo Capitoni, Cristian Contescu, Michael Davis, David Alvarez, Jaroslav Guenther, Edouardos Karavakis, Oliver Keeble, Julien Leduc, Fabio Luchetti, Luca Mascetti, Steven Murray, Mihai Patrascoiu, Andreas Peters, Michal Simon, and Rainer Toebbicke. Lhc data storage: Preparing for the challenges of run-3. *EPJ Web of Conferences*, 251:02023, 01 2021.

[20] Stefano Bagnasco, L Betev, P Buncic, F Carminati, C Cirstoiu, C Grigoras, A Hayrapetyan, A Harutyunyan, AJ Peters, and P Saiz. Alien: Alice environment on the grid. In *Journal of Physics: Conference Series*, volume 119, page 062012. IOP Publishing, 2008.

[21] Emiliano Betti, Daniel Pierre Bovet, Marco Cesati, and Roberto Gioiosa. Hard real-time performances in multiprocessor-embedded systems using asmp-linux. *EURASIP Journal on Embedded Systems*, 2008:1–16, 2007.

[22] Sebastien Binet, P Calafiura, MK Jha, W Lavrijsen, C Leggett, D Lesny, H Severini, D Smith, S Snyder, M Tatarkhanov, et al. Multicore in production: Advantages and limits of the multiprocess approach in the atlas experiment. In *Journal of Physics: Conference Series*, volume 368, page 012018. IOP Publishing, 2012.

[23] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth. A*, 389:81–86, 1997.

[24] Mainak Chakraborty and Ajit Pratap Kundan. Grafana. In *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*, pages 187–240. Springer, 2021.

[25] Jason Cole and Helen Foster. *Using Moodle: Teaching with the popular open source course management system.* " O'Reilly Media, Inc.", 2007.

[26] Michael C. Davis, Vladímir Bahyl, Germán Cancio, Eric Cano, Julien Leduc, and Steven Murray. CERN Tape Archive - from development to production deployment. *EPJ Web Conf.*, 214:04015, 2019.

[27] Kris Jamsa. *Cloud computing.* Jones & Bartlett Learning, 2022.

[28] Kathy Kincade. Cori supercomputer now fully installed at berkeley lab. https://www.nersc.gov/news-publications/nersc-news/nersc-center-news/2016/cori-supercomputer-now-fully-installed-at-berkeley-lab/. Accessed: 2023-10-04.

[29] M Martinez Pedreira, C Grigoras, and V Yurchenko. Jalien: the new alice high-performance and high-scalability grid framework. In *EPJ Web of Conferences*, volume 214, page 03037. EDP Sciences, 2019.

[30] Andreas Peters and Lukasz Janyst. Exabyte scale storage at cern. *Journal of Physics: Conference Series*, 331, 12 2011.

[31] Rami Rosen. Resource management: Linux kernel namespaces and cgroups. *Haifux, May*, 186:70, 2013.

[32] Omar Sefraoui, Mohammed Aissaoui, Mohsine Eleuldj, et al. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, 2012.

[33] Jamie Shiers. The worldwide lhc computing grid (worldwide lcg). *Computer physics communications*, 177(1-2):219–223, 2007.

[34] Vandana Singh and Alexander Thurman. How many ways can we define online learning? a systematic literature review of definitions of online learning (1988-2018). *American Journal of Distance Education*, 33(4):289–306, 2019.

[35] Diomidis Spinellis. Git. *IEEE software*, 29(3):100–101, 2012.

[36] Alvaro Videla and Jason JW Williams. *RabbitMQ in action: distributed messaging for everyone*. Manning, 2012.

[37] David W Walker and Jack J Dongarra. Mpi: a standard message passing interface. *Supercomputer*, 12:56–68, 1996.

[38] Sage Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*, pages 307–320, 2006.

[39] Sergiu Weisz and Marta Bertran Ferrer. Adding multi-core support to the alice grid middleware. In *Journal of Physics: Conference Series*, volume 2438, page 012009. IOP Publishing, 2023.

[40] Sergiu Weisz, Vlad-Iulius Năstase, Maria-Elena Mihăilescu, Darius Mihai, and Nicolae Ţăpuş. Deploying large private clouds for high availatibily services. In *2023 24th International Conference on Control Systems and Computer Science (CSCS)*, pages 67–74. IEEE, 2023.

[41] Yoji Yamato, Yukihisa Nishizawa, Shinji Nagao, and Kenichi Sato. Fast and reliable restoration method of virtual resources on openstack. *IEEE Transactions on Cloud Computing*, 6(2):572–583, 2015.

[42] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003.

[43] Chiara Zampolli. Alice data processing for run 3 and run 4 at the lhc, 2020.