FLEXIBLE AND EXTENSIBLE INFRASTRUCTURE MONITORING ARCHITECTURE FOR COMPUTING GRIDS WITH INFRASTRUCTURE AWARE JOB MATCHING

R.M.K.D Wijethunga

228045C

Dissertation submitted in partial fulfillment of the requirements for the degree
Degree of Master of Science (Research)

Department of Computer Science and Engineering
University of Moratuwa

University of Moratuwa
Sri Lanka

October 2023

DECLARATION

I declare that this is my own work and this Dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: UOM Verified Signature Date: 2023.10.05

The supervisors should certify the Dissertation with the following declaration.

The above candidate has carried out research for the Degree of Master of Science (Research) Dissertation under our supervision. We confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Prof. Indika Perera

Signature of the Supervisor: UOM Verified Signature Date: 05.10.2023

Name of Supervisor: Dr. Gayashan Amarasinghe

Signature of the Supervisor: UOM Verified Signature Date: 05/10/2023

ACKNOWLEDGEMENT

I would like to take this opportunity to express my heartfelt gratitude and extend my sincere appreciation to my esteemed supervisors, Prof. Indika Perera and Dr. Gayashan Amarasinghe. Their unwavering support and guidance throughout the entire duration of this project work have been invaluable. Their insightful suggestions and constructive comments have played a pivotal role in resolving numerous challenges encountered during the project's development.

I am also immensely grateful to Dr. Adeesha Wijayasiri and Dr. A. Shehan Perera, who served as progress review panel members, for their unwavering dedication and valuable input. Their comments and suggestions have significantly contributed to the refinement of my work.

Furthermore, I would like to express my gratitude to Dr. Kutila Gunasekara, our MSc research degree coordinator, for assisting me in navigating the formalities associated with obtaining the degree. The guidance provided by Dr. Gunasekara has been immensely helpful throughout this process.

I would also like to extend my appreciation to all the lecturers at the Faculty of Computer Science and Engineering, University of Moratuwa. Their valuable advice and expertise have played a crucial role in shaping my academic journey.

Lastly, I am indebted to my parents, my brother, and my friends for their unwavering support and continuous motivation throughout my career. Their encouragement has been a constant source of inspiration, and I am truly grateful for their presence in my life.

ABSTRACT

Many research experiments with large data processing requirements rely on massive, distributed Computing Grids for their computational requirements. A Computing Grid is built by combining a large number of individual computing sites distributed globally. These Grid sites are maintained by different institutions across the world and contribute thousands of worker nodes possessing different capabilities and configurations. Developing software for Grid operations that works on all nodes while harnessing the maximum capabilities offered by any given Grid site is challenging without knowing what capabilities each site offers in advance. This research focuses on developing an architecture-independent Grid infrastructure monitoring design to monitor the infrastructure capabilities and configurations of worker nodes at sites across a Computing Grid without the need to contact local site administrators. The design presents a highly flexible and extensible architecture that offers infrastructure metric collection without local agent installations at Grid sites. The resulting design is used to implement a Grid infrastructure monitoring framework called "Site Sonar v2.0" that is currently being used to monitor the infrastructure of 7,000+ worker nodes across 60+ Grid sites in the ALICE Computing Grid. The proposed design is then used to introduce an improved Job matching architecture for Computing Grids that allows job matching based on any infrastructure property of the worker nodes. This dissertation introduces the proposed architecture for a highly flexible and extensible Grid infrastructure monitoring design and an improved job design for Computing Grids and the implementation of those designs to derive important findings about the infrastructure of ALICE Computing Grid while improving its job matching capabilities. This work provides a significant contribution to the development of distributed Computing Grids, particularly in terms of providing a more efficient and effective way to monitor infrastructure and match jobs to worker nodes.

Keywords: Grid computing, Grid monitoring, Grid infrastructure, infrastructure monitoring, Site Sonar, Job Matching, Infrastructure aware

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
ALICE	A Large Ion Collider Experiment
API	Application Programming Interface
ATLAS	A Toroidal LHC Apparatus
CE	Computing Element
CGroups	Control Groups
CMS	The Compact Muon Solenoid
CVMFS	CernVM File System
GUI	Graphical User Interface
I/O	Input/Output
JAliEn	Java ALICE Environment
JDL	Job Description Language
JSP	Java Server Pages
LHC	Large Hadron Collider
MDS	Globus Monitoring and Discovery Service
OS	Operating System
PanDA	Production and Distributed Analysis
TTL	Time To Live

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

Computing Grids are used widely in large research organizations to cater the massive data processing requirements of the experiments. A Computing Grid is a virtual collection of multiple Grid sites distributed globally and managed by a team of Grid administrators. Even though a Computing Grid is composed and maintained by Grid administrators in a single organization, Grid sites are built and controlled by site administrators in different organizations. Grid sites pledge their computing power to different Computing Grids allowing different Grids to use their data processing capabilities when required. They also provide storage facilities to store data relevant to the experiment or the output data from analyzing experiment data. Since Grid sites are owned and managed by different parties, each site is built and configured depending on the requirement of the owning party. This makes a Computing Grid highly heterogeneous as it contains a large number of computing nodes that have different capabilities and are configured different to each other. For example, one site could have compute-optimized clusters with Ubuntu nodes whereas another site could have GPU-optimized clusters with CentOS nodes. Since each Grid site is different from the other, jobs running on one site could fail on the other due to the infrastructure limitations or differences on that site. Therefore, it is important to consider the infrastructure available on one site when matching jobs to that site. Collecting the infrastructure details of each site is a hard task because the sites are not under the control of Grid administrators and the site administrators are reluctant to provide access or allow Grid administrators to install different software in their nodes for monitoring purposes. While different Grid infrastructure monitoring tools are available in the literature (described in chapter 2), those are very restricted with limited capabilities and they are completely isolated from the job matching process making it impossible to allow job matching based on infrastructure capabilities or limitations in Grid sites.

This research intends to address this problem by proposing a new flexible and extensible Grid monitoring architecture to collect Grid infrastructure metrics that can ultimately be integrated with the Job matching process to introduce Infrastructure Aware Job Matching for Computing Grids. The architecture has then been used to develop a software called “Site Sonar” that is currently monitoring 7,000+ worker nodes across 60+ Grid sites in ALICE Computing Grid in CERN, the European Organization for Nuclear Research, which is one of the world’s largest and most respected centers for scientific research. It has been used to provide Infrastructure Aware Job Matching capabilities for the ALICE Grid and the software is being used in production at the

moment.

1.2 Background

1.2.1 Grid Site

A Grid site is a computing site that consists of a large number of worker nodes. Different organizations build and maintain computing sites for their computational or storage requirements. Since it is a waste to allow this amount of resources to stay idle even for a small time, these computing sites share their resources with multiple research organizations or experiments so that the resources are fully utilized. Grid sites often pledge their resources to large Computing Grids which in turn allow their users to access a massive amount of resources than what is provided by the individual Grid site.

1.2.2 Computing Grid

A Computing Grid is a virtual combination of a large number of Grid sites. A Grid middleware is run on all the Grid sites that presents the combination of individual Grid sites as a massive computer to the end user allowing the end user to run large workloads that require thousands of core hours to complete. Computing Grids are widely used in large scale experiments which have large data processing requirements that cannot be catered by a few computing sites.

1.2.3 CERN

The European Organization for Nuclear Research, famously known as CERN, is an intergovernmental organization that operates the largest particle physics laboratory in the world. CERN houses the largest particle accelerator in the world called the Large Hadron Collider (LHC) [1] and more than 2660 staff members and hosts more than 12,000 users from more than 70 countries in the world[2]. CERN is largely famous in the computing domain for the origination of the "World Wide Web" at CERN in 1989[3]. CERN generates more than 49 petabytes of data annually which are used to do complex calculations and simulations related to physics[4].

1.2.4 ALICE experiment

A Large Ion Collider Experiment (ALICE) is one of the eight detector experiments at the Large Hadron Collider at CERN. It is dedicated to studying heavy-ion physics at LHC and intends to study the physics of strongly interacting matter at the highest energy densities reached so far in a laboratory. The experiment results in outputting

events building bandwidth of up to 3.5 TB/s and providing compressed data for permanent storage at a rate of 100 GB/s[5] leading to the requirement of having the most advanced computing techniques for handling the experiment data.

1.2.5 ALICE Computing Grid

ALICE Computing Grid has been formulated to cater to the ever-growing data processing and storage requirements of the ALICE experiment. It consists of over 0 Grid sites that collectively contribute over 7,000 worker nodes to the Grid. JAliEn[6] middleware developed by the ALICE Grid team presents all the Grid sites as a single Computing Grid to the ALICE users to satisfy their data processing requirements. The study of this research was done on the ALICE Computing Grid and the research outcomes are now deployed in production as a new upgrade to JAliEn middleware.

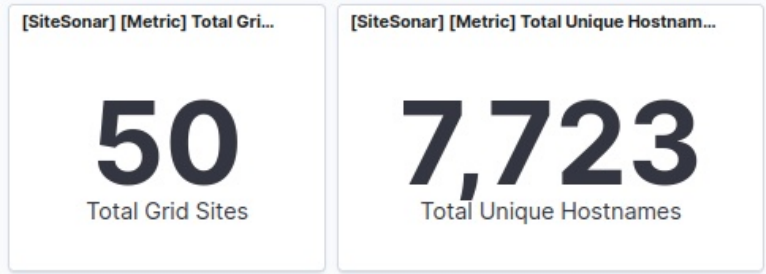

Fig. 1.1: ALICE Grid in numbers

1.2.6 Grid Infrastructure Monitoring

As explained in the previous sections 1.2.1 and 1.2.2, a Computing Grid is a collection of individual Grid sites operated by different parties. Given that these parties are often independent and have their sites configured to cater to their own needs, each site is different from one another leading to a highly heterogeneous network of computers.

To further intensify the problem, Grid admins do not have access to individual nodes on the system and hence they do not have a way to check the configuration of nodes in the site. The current process followed to address this is opening a ticket in a portal that is maintained by the Grid site and asking the site administrators about the configuration of the site if the Grid admins suspect that there is an issue with a specific site. This could cause a lot of redundant work since the site configurations could change from time to time.

In addition to that, while this is doable for a single site or two, it would be a tedious task if there is a need to know the configuration of nodes across the Grid. For example, as a part of a major update to the Grid middleware in ALICE computing Grid, support to run jobs inside containers was introduced. However, for this functionality to

work properly on a computing node, nodes were required to have some prerequisites already setup and there was no easy way to collect this information which led to the requirement of having a proper Grid infrastructure monitoring system.

A Grid infrastructure monitoring system possesses the capability to monitor the infrastructure properties of all the individual nodes in the Computing Grid and provide reports to the Grid administrators that eliminate the need to contact individual Site admins to troubleshoot an issue. Operating System, CPU information, memory information, available packages etc. are some of the infrastructure information monitored by a Grid infrastructure monitoring system.

1.2.7 Jobs

Computing Grids allow a user to run a task that accomplishes a certain computational requirement of that user. This kind of tasks are referred to as "Jobs" in Grid terminology. Users can submit jobs with specific requirements like CPU, RAM, storage etc., and with a specific goal like doing a complex calculation or running an analysis on a data set etc. Once the user defines their job and submits the job, the Grid middleware will ensure that the job is executed in a worker node in the Grid and the output of the job is made available to the user.

1.2.8 Pilot Jobs

Pilot Jobs are a kind of placeholder jobs that are submitted by Grid middleware to different computing sites to reserve a job execution slot in a worker node. They hold the place in the worker node until a suitable job is assigned to it which can execute in that job execution slot. Pilot jobs are the most widely used Grid middleware approach at the moment of this study as it allow lazy binding of the jobs to the worker node. The use of Pilot jobs allows the Grid middleware to reserve slots in many computing sites across the Grid and use those slots to develop a virtual centralized queue in the perspective of the Grid middleware which facilitates advanced job scheduling mechanisms.

1.2.9 Job Matching

User jobs are executed on a selected computing node based on the decision taken by the Grid middleware. Matching a user job to a Grid node is referred as "Job Matching" in Grid terminology.

The Grid contains computing nodes of different capabilities and configurations and the jobs submitted to the Grid by the users have different requirements. To ensure an optimum use of the computing resources, it is essential to match the submitted jobs to the nodes that fulfill the requirement with minimum idle resources. For example, if a user job requires 6 CPU cores to process, and we have only a computing node of

8 cores and 16 cores, we should assign the job to the computing node with 8 cores to reduce resource wastage. This process is usually undertaken by a component called "Job Matcher" or "Job Broker" in a Grid middleware.

Most Job Brokers are capable of matching jobs on a set of requirements like preferred grid site to run on, TTL (Time To Live) of the job, user type etc. However, they have very limited capability to see and use the configuration/capabilities(and limitations) of individual computing nodes to do the job matching. They have the visibility of a limited set of hard-coded infrastructure parameters of the computing nodes that are available for job matching. For example, Job Broker in JAliEn has visibility of the number of CPU cores and the installed packages in the candidate computing nodes to be considered in the job matching process. While the jobs are matched to individual nodes based on this information, jobs could fail due to other issues like lack of memory, incompatible operating system, lack of Operating System (OS) libraries etc. and the job matching could be considerably improved if the Job Broker has access to such extra information about the node.

1.3 Motivation

A Computing Grid contains a massive amount of resources and it is the responsibility of Grid owners/administrators to ensure that the available resources are optimally utilized. On one hand, this can be done by matching jobs to the node with the minimum amount of resources above the required amount in a job. On the other hand, this can be improved by reducing the job failures on the Grid that utilize resources and ultimately fail without giving a useful output. Jobs can fail due to multiple reasons like:

- Job is incompatible with the assigned computing node. (eg: Absence of GPUs to run a GPU-bound job)
- Libraries missing in the computing node. (eg: Incompatible operating system
- Grid middleware expects functionality from the node that the nodes do not possess. eg: Ability to run singularity in the node

In addition to reducing the job failures due to incompatibilities, we can optimize our middleware, job payload, and workflows to make maximum use of the Grid and identify current bottlenecks in the Grid if it was possible to have a better idea about the infrastructure capabilities and limitation of worker nodes in the Grid. This research is motivated by this factor and intends to find a better way to provide a complete idea about the infrastructure capabilities and limitations of a Computing Grid and to find a way to improve the job matching process to account for such limitation and capabilities when assigning jobs to worker nodes that could ultimately lead to reduced job failures and increased efficiency in resource utilization.

1.4 Problem Statement

"Introduce a flexible and extensible way to collect infrastructure parameters of a Computing Grid and use that information to improve the job matching process in the Grid"

1.5 Research Objectives

1. Identify a methodology to collect infrastructure metrics of individual computing nodes in a distributed Computing Grid
2. Introduce a new Grid infrastructure monitoring design
3. Design a new job matching architecture to account for infrastructure properties in the Job matching process

1.6 Research Outcomes

After achieving each research objective, the new concepts were used to deliver the following outcomes.

1. Develop a new Grid infrastructure metric collection framework
2. Implement an Analysis and visualization tool for collected metrics
3. Integrate infrastructure metrics to JAliEn Job Broker

1.7 Publications

K. Wijethunga, L. Betev, C. Grigoras, M. Stortvedt, I. Perera, G. Amarasinghe, and M. Litmaath, "Site Sonar - A Flexible and Extensible Infrastructure Monitoring Tool for ALICE Grid" 26th International Conference on Computing in High Energy & Nuclear Physics (CHEP 2023)

1.8 Organization of Thesis

This thesis studies how the infrastructure properties of worker nodes in a Computing Grid can be collected and how the job matching process in Computing Grids can be improved to include these properties in the job matching process. The thesis is organized as follows.

- Chapter 2 of the thesis discusses the literature associated with the research describing the existing Grid infrastructure monitoring systems and Grid job matching processes by existing Grid middleware. The literature review also discusses

the issues and limitations with the existing systems that led to the requirement to develop system to address the existing problems

- Chapter 3 describes the methodology employed to achieve the research objective. It details out each aspects of the research and how the proposed methodology helps to achieve the goals of this research
- Chapter 4 explains the implementation of the system that was developed by implementing the proposed methodology to achieve the research outcomes
- Chapter 5 discusses the experimental results, analysis and findings of the research and evaluates the new system to prove that the research goals were achieved successfully
- Chapter 6 concludes the research presenting the achieved outcomes and conclusions of the study
- Chapter 7 lays out the plan for future work that could be accomplished as an extension of this research

CHAPTER 2

LITERATURE REVIEW

2.1 Grid Infrastructure Monitoring

2.1.1 Grid Computing

Grid Computing is a concept that is closely related with High high-throughput computing. While Grid computing does not always guarantee to provide high throughput, it focuses on completing a large amount of work over a period of time[7]. A Computing Grid is created by seamlessly integrating geographically distributed and heterogeneous computing resources into a single unit using virtualization. From a user's view, the Grid can be seen as a single entity of massive computing power when in reality it is a number of distributed computing resources in different computing sites working in conjunction. These computing sites are owned and operated by individual organizations with their own administrative and configuration policies. These organizations come into an agreement to share their resources with each other forming a "Virtual Organization"[8]. This concept allows each organization to receive computing power that is much larger than any individual site can contain.

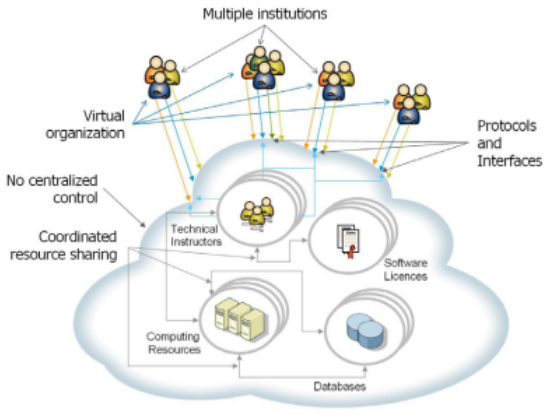

Fig. 1. Foster's model of grid computing.

Fig. 2.1: Foster's model of Grid Computing

Figure 2.1 presents the Grid computing model presented by Ian Foster et al. [9] who are pioneers in introducing the concept of Grid computing. He proposes the Grid to be a system that [9]:

- coordinates resources that are not subject to centralized control
- using standard, open, general-purpose protocols and interfaces
- to deliver nontrivial qualities of service

2.1.2 Grid Monitoring

As a Computing Grid consists of a large number of resources and there is a critical requirement to use them efficiently, closely monitoring the system is an essential task. Reducing the idle time of computing resources leading to an efficient use of resources is a widely discussed research topic in academia. A lot of techniques like task clustering[10], intelligent resource matching[11], advance resource reserving[12], performance predictions[13] have been discussed in this regard.

This research focuses on collecting infrastructure details of the underlying nodes in the computing grid and using that information to provide enhanced intelligence to the resource matching process to use the node infrastructure properties in matching jobs to the most suitable nodes. In that regard, the literature review can be broken into 2 main sections: Studying the existing Grid monitoring systems and the possibility of using them to collect infrastructure metrics of underlying computing nodes, and studying the existing job matching processes and the possibility of using the collected infrastructure metrics at the job matching stage.

While there are a lot of famous Grid monitoring tools, most of them are focused on collecting information at the application level. Although these tools provide information about the application layer and performance usage, they lack the capability to provide an understanding of the infrastructure of the underlying nodes. Even though the basic functionalities of a Computing Grid can be performed by only monitoring the application layer, much more powerful functionalities can be offered if the Grid middleware has access to the infrastructure details of the underlying nodes.

Given that a Computing Grid is setup by a virtual organization that comprises independent individual entities, they have their own ways to setup the computing cluster, their own administrative policies, different security policies etc. Developing a Grid middleware that is generic to all these sites and their configurations while providing a good performance is a hard task. For example, ALICE Computing Grid[14] had a requirement to support containerized jobs for its next stage of operation called “LHC Run 3”[15], but developing JAliEn which is the ALICE Grid middleware to allow this depended on all the computing sites running CentOS 6 or above to provide necessary operating system features. An understanding of the infrastructure details of each computing site and its nodes were essential for this which led to the requirement of developing a new Grid Infrastructure Monitoring system.

2.1.3 Existing Tools

Zanikolas et al[16] classifies the Grid monitoring systems into four levels in their proposed taxonomy as shown in Fig. 2.2. Their taxonomy includes a set of components: “sensor”, “producer”, “consumer”, “republishers” which are introduced by the Global Grid Forum[17].

- Sensor(S) : A process that monitors an entity and generates events.
- Producer(P) : A process that implements at least one producer Application Programming Interface (API) for providing events.
- Consumer(C) : A process that receives events by implementing at least one consumer API
- Republisher(R) : Any single component implementing both producer and consumer interfaces for functions like filtering, aggregating, broadcasting etc. There can also be a hierarchy of republishers(H) with each republisher responsible for one functionality.

In simple terms, a sensor can be identified as a data collecting probe that makes the data available through a non programmable interface like a web page, producer can be identified as a data provider, consumer as a data receiver and a republisher as a component that consumes data, process data and output them in an improved format.

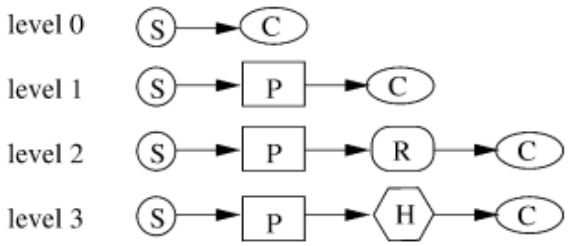

Fig. 2.2: Taxonomy of Grid monitoring systems

The Grid monitoring systems in which data flows directly from sensors to consumers are categorized as Level 0. These are the most trivial systems that collect data by running a periodic probe and make them available through a basic mechanism like a log file or a web page.

In level 1 machines, either sensors are implemented in the same machine, or their functionality is also provided by the producers, and the collected data is exposed to consumers via an API. Level 2 monitoring systems consist of at least one republisher with fixed functionality in addition to the producers. Highly flexible monitoring systems fall under level 3 featuring a hierarchy of configurable republishers when data flows from producers to the consumers. All the Grid monitoring systems seem to fall under one of these 4 categories and the following section discusses a few of the existing systems that provide close functionality to the context of this research.

2.1.3.1 GridICE

GridICE[18] is a level 1 legacy Grid infrastructure monitoring system that was developed to monitor the INFN Grid. The system consist of individual agents installed at

each Grid site which collect the infrastructure data and report to the central services daily. With GridICE, local site administrators can decide what should be exposed to the outside as monitoring data. Making use of GridICE then implies that the ALICE Grid team would need to negotiate with each ALICE Grid site on the desired information to be exposed, and yet again each time a change is desired, making for an unnecessarily cumbersome operational model.

GridICE was designed by Andreozzi et al.[18] to monitor the INFN Computing Grid in Italy. It consists of a layered architecture with multiple services as described below.

- Measurement service: Responsible for collecting metrics and storing them in a local repository (Resembling a sensor)
- Publisher service: Publish the collected information to consumers
- Data Collector service: Collect data from the publishers (Resembling to a consumer)
- Detection/Notification and Data Analyzer services: These services are responsible for detecting issues in the Grid and send alerts to the relevant parties. Data analysis layer provides performance analysis, usage levels and statistics to the users.
- Presentation service : Present collected data in a web-based Graphical User Interface (GUI)

GridICE follows a Level 2 Grid monitoring system architecture with the presence of sensors, producers, consumers, and a set of republishers with a fixed functionality. It provides many infrastructure details of the nodes like Operating System, Processor, Memory information etc. which are defined in the GLUE schema[19] and provide many interesting information that we plan to gather in this research. However, the GLUE schema is a legacy schema and GridICE uses old technologies like XML[20], Globus Monitoring and Discovery Service (MDS) . In addition to that it uses the pull model for collecting data which is covered in detail in section 3.1.4 which has been shown as a major issue in monitoring systems. In order to collect data from nodes in the Grid, GridICE has configured monitoring agents on the edge nodes. While this is possible when each organization agrees, in larger Grids this approach is not feasible.

Using of GLUE schema to collect data reduces the flexibility of GridICE considerably. While it gives a lot of information it is not possible to collect custom information which makes the system less desirable. GridICE has an important feature of being able to detect and notify abnormal behaviours in the Grid, however, it does not have the capability to act on that information to provide a resolution.

2.1.3.2 Paryavekshanam

Paryavekshanam[21] is a Grid infrastructure monitoring system developed by Prasad et al to monitor the GARUDA[22] computing Grid. At the time of the paper, GARUDA Grid consisted of 45 organizations distributed across 17 cities in India. It is capable of providing both application level and infrastructure level monitoring including monitoring of:

- Computing resources
- Network
- Grid Middleware
- Storage
- Jobs tracking
- Software Installed

Its architecture resembles a simple level 1 Grid monitoring system without the presence of republishers and it consists of the following components.

1. Information Generator - Provides the functionality of a sensor/ producer by running as a daemon process in the Head node of every cluster collecting data from all the nodes in regular intervals.
2. Information Receiver - A daemon residing on the monitoring server that periodically requests information from the Information Generator
3. Information Repository - Provides data persistence capability to the system. Information Receiver and repository can be considered as the single monitoring server
4. Paryavekshanama Visualizer - A service that takes data from the Information Repository and displays them in the form of graphs and tables

The system queries job information every 20 minutes, middleware information each hour, and other software information daily. The authors have properly defined the data collection period with the frequency of the collected data type being proportional to the probability of changing the considered data over time. But the data collection in Paryavekshanam is done using the "Pull model" where a single process tries to crawl the Grid and collect information. While this could be feasible for a small computing Grid, it would become considerably harder when the Grid size increases leading to scalability issues. As we have explained in detail in section 3.1.4 such a

system had to run for a large number of hours to complete a single crawl in the ALICE Computing Grid. Additionally, this could lead to different resource efficiency issues like collecting data from idle sources, unnecessary load on nodes etc. However, this eliminates the need to install agents on computing nodes which is a highly desirable feature when developing a monitoring system to a Computing Grid.

In terms of infrastructure information collected, Paryavekshanam is more flexible than GridICE and monitors some important information regarding the computing nodes which are close to what we plan to collect in this research. However, it is not flexible enough to allow changing the structure of collected data dynamically. This essentially leads to the problem of having to know the exact data that you need to collect before you start collecting data. This is an infeasible task in a highly heterogeneous Computing Grid like this because a Grid admin does not know what data to expect if he is defining a new monitoring configuration as discussed in section 3.2.2.

Paryavekshanam uses a pull model to collect Grid information. The pull model usually requires a large resource-intensive process to run the data collection continuously, whereas most of the Grid monitoring system use a push model to send data to the central servers only when necessary.

2.1.3.3 MonALISA

MonALISA[23] is a widely used level 2 Grid monitoring system that has been used to monitor the ALICE Grid for more than 15 years. It specializes in collecting real-time performance data rather than infrastructure data which doesn't change that often.

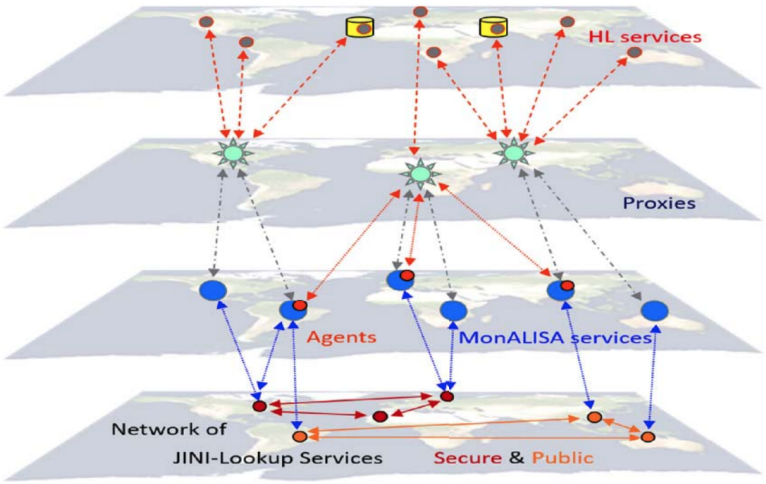

Fig. 2.3: High level architecture of MonALISA

Fig. 2.3 presents the high-level architecture of MonALISA presented by Balcas et al[24]. The bottom layer consists of lookup services that are used to discover other

MonALISA agents. This layer is not related to the domain of this research and hence it won't be covered in this section. The second layer from the bottom shows how MonALISA is created by installing agents in different computing sites. Since it is not allowed to install different monitoring agents on computing nodes, MonALISA is usually installed at edge nodes in each computing Site. These nodes collect highly frequent metrics from computing nodes and provide a high-level idea of how the Computing site is performing. Due to the reason that the Grid admins do not prefer to allow external monitoring agents like MonALISA to be installed in their nodes, it is not possible to collect granular metrics regarding the individual nodes using MonALISA. MonALISA focuses on fast-changing parameters like CPU usage, latency, network round trip time etc. Each monitored parameter typically is reported once per minute. For each job, about 200 parameters are reported for an aggregated update rate of approximately 500 kHz(considering 2500 jobs which is the average number of jobs that are running on ALICE Grid at any given time). The MonALISA agents can be seen as the data producers.

The next layer of MonALISA consists of proxies distributed around the globe. The proxies are central services that are placed between the agents and the high-level services to aggregate the data from different computing sites. The main intention of the proxies is to collect the data from individual Grid sites and make them available to other services using publicly accessible endpoints. For accounting purposes, aggregated higher-level metrics are collected centrally, while detailed information for each parameter is available in real-time locally for a short recent history. These proxies only consume and aggregate the data and make them available to high-level services making them republishers with fixed functionality.

The top level of the MonALISA architecture consists of high-level services that consume the collected data. These are the consumers of the data. An important feature of MonALISA is that it uses a push model to collect the data where the lower-level data producers push their data to central servers to be used for monitoring purposes. This allows MonALISA to be highly scalable because there is low overhead on the central servers and new agents can be added easily without any changes to the central servers. However, the functionality of MonALISA is focused on providing the required metrics to measure the health and performance of Grid jobs which eliminates infrastructure monitoring out of its monitoring scope.

It uses a PostgreSQL database which allows built-in support for JSON[25]. This provides some flexibility to the data collection process, however, it does not provide much flexibility as the database is still SQL-based. It can be seen that it is very hard to extend MonALISA as adding a new data collection parameter requires changing multiple components and deploying those changes across the Grid. It does not have any built-in options to detect or report anomalies as well. A main concern regarding the suitability of MonALISA arises from the fact that it analyzes the variation of a

specific parameter across time whereas we are looking to analyze the variation of a specific parameter across worker nodes.

2.1.3.4 MONIT

MONIT[26] is a monitoring framework based on a suite of popular open-source systems. It was developed by CERN IT to replace the LEMON[27] framework used for CERN Data Center infrastructure monitoring as well as the WLCG Dashboards[28] which were used to monitor activities on the Worldwide LHC Computing Grid (WLCG) until a few years ago.

MONIT collects high-frequency data about the health of CERN Data Center hosts and their services. It completely removes the use of in-house data monitoring tools that have been used so far and makes use of a technology stack that has become a global standard for monitoring activities. This is an important feature in a monitoring system because the monitoring domain has been standardized in the last couple of years with the huge rise in popularity of tools like Elasticsearch[29], Kibana[30], Apache Kafka[31] etc. Unlike the systems discussed above, usage of these tools provides a large number of advantages and powerful features like no-code visualizations, schema-free data ingestion, built-in data indexing leading to fast query performance etc. MONIT architecture consists of 4 major components described below.

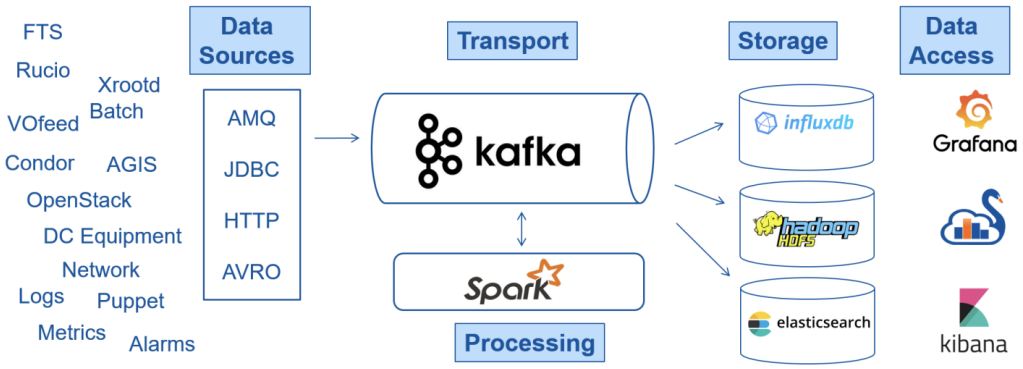

Fig. 2.4: MONIT Architecture

1. Data Sources - Data sources resemble Producers in the discussed taxonomy. Unlike the traditional system, MONIT offers a range of data sources due to the use of new an improved data transport and processing technologies. CollectD local agents are used for sending metrics that replace LEMON agents on every node[32]. HTTP and ActiveMQ protocols are exposed for the external data producers to inject data into the system. It also uses JDBC Query to periodically pull external data for existing monitoring databases. Data is sent in a schema-free JSON format delivering a high flexibility to the system allowing ad-hoc

changes to data collection parameters and allows post data filtering which is an important feature that we seek in a suitable monitoring system.

2. Transport and Processing - All the received data from the Data sources are transported to an Apache Kafka cluster. This introduces a separate layer(That can be identified as a hierarchy of republishers that can be repurposed) between data producer and consumer thus eliminating a major bottleneck that is available in many existing systems. It also allows data handling via Apache Spark[33] with Kafka data in real-time or HDFS data in batch processing. It provides important features like enhancing the data to include additional fields, aggregating data across multiple fields and time, data correlation etc. which are not available in traditional monitoring systems.

3. Storage - Around 3 TB of data(per day in compressed mode) received by MONIT are stored in different storage systems like HDFS[34] for long-term archival, Elasticsearch for short-term indexing and visualizing and Influx DB for medium storage of time series data.

4. Data Access - Data Access component resembles the Sensors in the discussed taxonomy. MONIT offers data access via popular data visualization tools like Kibana, Grafana[35], SWAN[36]. This approach provides much needed features to the users for easy data visualization like simple query language, fast data retrieval, advanced plot types, powerful visualizations etc. The power of abandoning of the traditional in-house monitoring systems yields as this stage because the traditional system are not very well maintained(as they are not widely adopted) and hence for example to add a new visualization type, a user has to write code and design the type. Such needs are eliminated and now the users can visualize their data in powerful visualizations with zero code additions

With the presence of Producers, Consumers, and a hierarchy of configurable republishers, we can see that MONIT is a level 3 Monitoring system. We can see that MONIT consists of a number of desirable features, however, MONIT is aimed at monitoring a large data center(because it requires monitoring agent installation in nodes) and not a computing Grid. Due to this reason, MONIT does not provide a suitable solution to the research problem although it is closely related.

2.1.3.5 Site Sonar v1.0

Site Sonar v1.0 is a simple monitoring system developed for collecting infrastructure information from the worker nodes[37] in the ALICE Computing Grid. The idea of the project is to collect infrastructure information of the Grid worker nodes by submitting

an actual job whose purpose is to collect infrastructure metrics instead of running an actual payload.

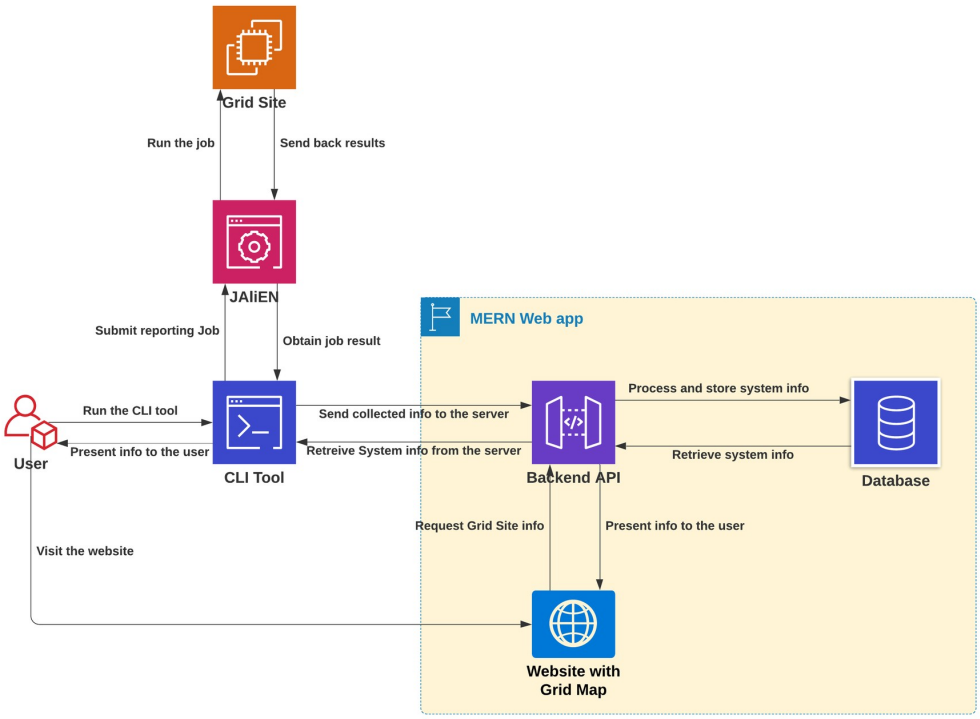

Fig. 2.5: Site Sonar v1.0 architecture

It has a simple architecture as shown in Fig. 2.5. Site Sonar Tool is a simple CLI utility that possesses the capability to submit jobs to the ALICE computing Grid. A trivial job that executes a set of scripts that collect worker node information like operating system, RAM, space left etc. is being continuously submitted by this tool to each Grid site in the Computing Grid. The job executes the monitoring scripts and sends back the result to a web server that is accompanied by the CLI tool which persists the data for long-term usage. A separate web service uses this database to expose a frontend[38] to query these data by the users to be used for their analysis which is shown in 2.6.

Given its simple architecture, Site Sonar v1.0 possesses a set of major drawbacks like flooding the Grid with jobs to collect data from nodes, high resource wastage, requirement of manual intervention for data monitoring etc. These issues are discussed in detail in section 3.1.3. It is important to note that, while Site Sonar v1.0 has its drawbacks in terms of resource utilization, flexibility and extensibility etc., it provides the basic functionality that is intended to be used in this research: It proves that it is possible to collect infrastructure information from worker nodes in ALICE Computing Grid without installing agents on any Grid sites. The steps taken to address the drawbacks and the suggested solutions for these drawbacks are discussed under the section

3.1.4.

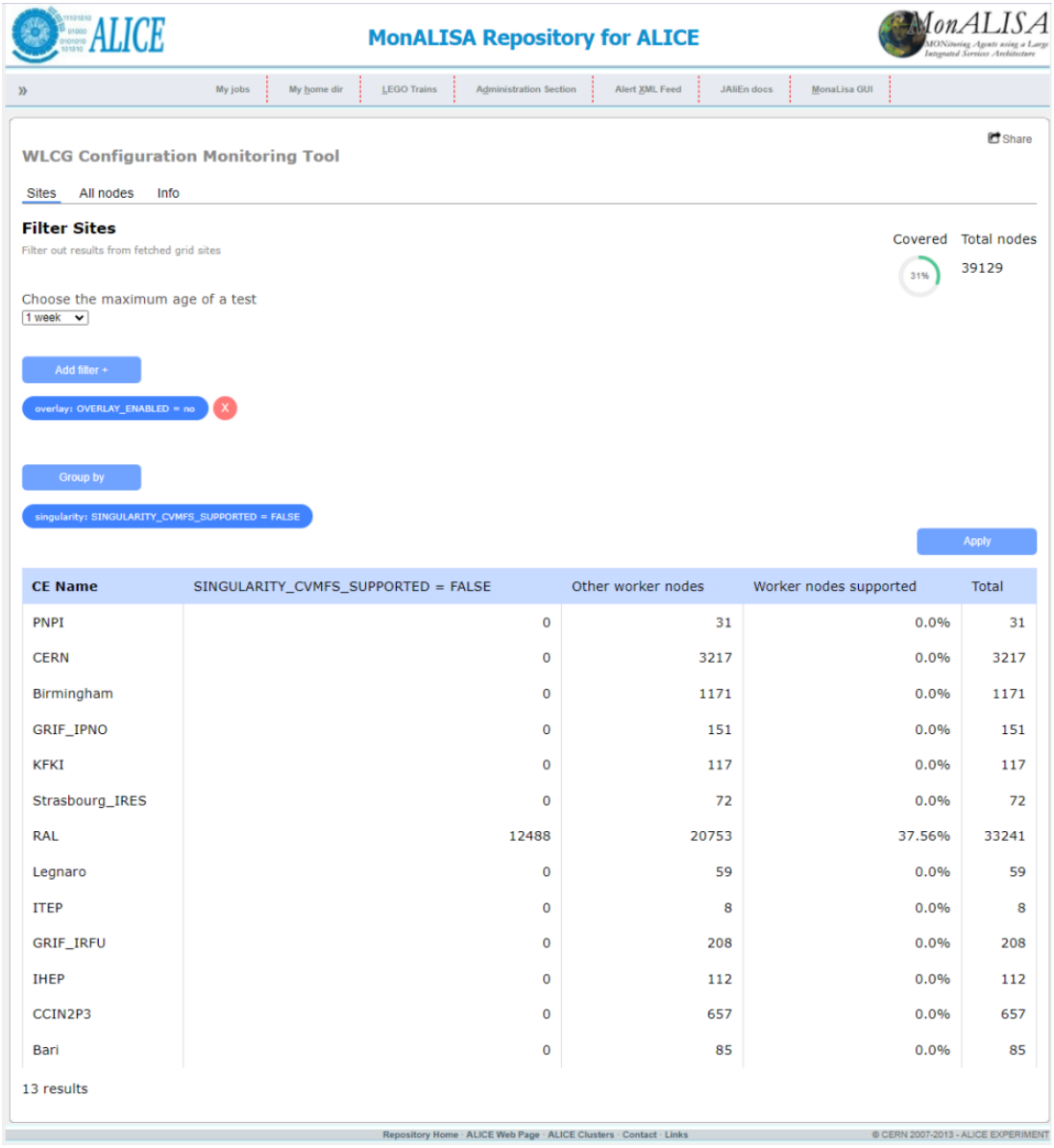

Fig. 2.6: Site Sonar v1.0 Interface

2.1.4 Issues with Existing Tools

In the previous section, we have discussed several Grid monitoring tools that are closely related to achieving the goals of this research. However, each of them has its own pros and cons and none of the existing systems provide the final outcome expected in this research. Some of the essential requirements that we have identified to support the final outcome are as follows:

1. Focus on infrastructure monitoring - Grid monitoring can be categorized into 2

sections: Application level monitoring and infrastructure monitoring. Famous application-level monitoring systems like MonALISA are already available and widely used. Since they cannot be used for monitoring the infrastructure, it is important to identify if the discussed tools focus on infrastructure or application-level monitoring

2. Push model - We have identified the pull model in data collection poses major scalability issues and performance bottlenecks in the monitoring process. This is proved with experimental results in section 5.3.1.1 under the evaluation of the study. Hence the new system must include a push model for collecting data from nodes.

3. Absence of agent installation - Grid sites are owned and operated by individual organizations and they often support multiple Virtual Organizations. Installing different monitoring tools to support each Virtual Organizations' requirement is infeasible and Grid site administrators do not prefer this. Already established and proven systems like Nagios[39], Ganglia[40] or SaaS platforms like Data-Dog could have been easily used for our case if this was allowed.

4. Flexibility in data collection - It can be seen that there are number of infrastructure monitoring tools already for Grids. However, they are not flexible in terms of data collection. Most of the tools are SQL backed leading to the requirement of having a strict schema. This raises issues like the lack of Post Data Filtering capabilities described in section 3.2.2 which are not ideal.

5. High extensibility - Most of the existing systems tend to be quite cumbersome when adding a new data collection metric or a data visualization plot. The agent-based tools need updating the agents across all deployment and the use of in-house tools for data visualization lead to less developed data visualization tools that require coding to add new visualizations and analysis.

6. Act upon alarming information - While all the above features are desirable in an infrastructure monitoring system, there seem to be very few actions taken based on the collected data. Each system has its own notification and alerting mechanisms, but most of them do not seem to be able to use that information to prevent future failures or increase resource utilization.

These features are evaluated with respect to the studied monitoring tools and Table 2.1 summarizes the presence of these features in each tool.

2.1.4.1 Issues with cloud monitoring systems

The development of monitoring software for Grid applications has been limited in recent years, as most of the communities that might have an interest in Grid computing

TABLE 2.1: ISSUES WITH EXISTING GRID MONITORING TOOLS

Feature \ Tool	GridIce	Paryavek -shanam	MonALISA	MONIT	Site Sonar v1.0
Focus on infrastructure monitoring	✓	✓	✓	✓	✓
Push Model	×	×	✓	✓	×
No agent installations on sites	✓	✓	×	×	✓
Flexibility in data collection	×	×	×	✓	×
High Extensibility	×	×	×	×	×
Can act upon alarming information	×	×	×	×	×

have rather invested their efforts in cloud computing solutions instead. Although the two fields are closely related, tools that are developed for monitoring cloud computing centers cannot be used to monitor a computing Grid for the following reasons:

- A cloud computing center is managed by a single organization that by construction already knows the properties of the resources it makes available to customers.
- Similarly by construction, customers already know what kinds of resources (hardware and software configurations) they requested and presumably received, deeming the collection of such information unnecessary.
- The monitoring software that does get used, whether by resource providers or consumers, is rather concerned with performance data instead of infrastructure data.

2.2 Job Matching

Job Matching is the process of assigning the jobs submitted by users to the most suitable nodes in the computing Grid. One of the critical functionalities of the Grid middleware is to undertake the job matching for the Grid. Different requirements related to the job are defined by users in the job submission request. Job submission requests are written in different formats as decided by the Grid administrators. For example, ALICE Computing Grid uses a language called Job Description Language (JDL) [41] to define the Job requirements.

```
User = "alidaq";
Packages = {
```

```
    "VO_ALICE@O2PDPSuite::gpu-nightly-20230320-1"
};
Executable = "/alice/cern.ch/user/a/xxxx/alien_mergeCurrents.sh";
InputFile = {
    "LF:/alice/data/2022/xxxx/tpcCurrents/tpcSPCalibration.xml"
};
JDLPath = "/alice/cern.ch/user/a/xxxx/mergeCurrents.jdl";
OutputDir = "/alice/data/2022/xxxx/tpcCurrents/";
Requirements = ( member(other.GridPartitions, "merger") )
                && ( other.TTL > 72000 )
                && ( other.Price <= 11 );
MemorySize = "128GB";
CPUCores = "16";
Type = "Job";
```

Above code section[1] shows an extract of a real job that has run on the ALICE Computing Grid. The fields InputFile, Executable, OutputDir are self-explanatory. The most important part to note is the MemorySize and CPUCores fields defined in the JDL. These fields correspond to the minimum amount of RAM and number of CPU Cores necessary to run the job. It is important to match the job to a node that has more than the requested amount of these components as otherwise, the job will fail to execute or take days to execute up to completion.

2.2.1 Issues with Existing Tools

The suitability of a node for a given job is decided by using multiple parameters. A few of them are discussed below.

1. CPU Cores - No. of CPU cores in a worker node is one of the main parameters considered in job matching. In order to reduce resource wastage, a node that provides the exact number of cores as required by the job. If such a node is not available, a node that has the least number of cores that is more than the required amount is assigned so as to reduce the number of idle cores in the job execution period.

2. Memory - Memory(RAM) is used in a similar manner to CPU cores in job matching processes, however not all middleware is capable of matching jobs based on RAM.

[1]Some fields are redacted to preserve the privacy of the data

3. Disk space - Many jobs require a certain amount of space in the node that executes the job to ensure that the job does not run out space to write the intermediary results while processing the data. Therefore the matched node should have more space than required by the job.

4. Geographical distance to data - Jobs submitted to Grids are large scale jobs that cannot be executed using normal computers. Some of them analyze terabytes of data that is stored in the Grid. In such cases, it is important to execute these jobs in sites that are closer to the node that is executing the job in order to reduce the network latency on reading Input/Output (I/O) and to reduce the extra stress on network bandwidth due to having to transfer data from one place to another. Therefore, it is desirable to match jobs to nodes that exist closer to the data.

While these are some parameters that are often used in the job matching process, Grid administrators desire to have more parameters available so that they can do a more optimized job matching. Given the heterogeneity of the users in the Grid, Grid resources are used to execute jobs related to a number of domains and use cases. Each of these has its own specifics and sometimes poses requirements that are not covered by the parameters used in the job matching process.

Some jobs need specific libraries installed in the worker node like language interpreters, utility functions etc. Some jobs can run only on some operating systems. Since the Grid contains nodes with different Operating systems, these jobs can easily fail execution on some Grid sites. For example, the new "hyperloop" jobs in ALICE Computing Grid can only execute in computing nodes that support AVX architecture as they make use of newer CPU instructions for providing improved performance[42]. If these requirements are not met, those jobs cannot be executed successfully. However, the existing job matching systems can support only a limited set of hardcoded parameters like CPU cores, Disk space etc., and cannot collect granular information like software versions, libraries installed, CPU architecture etc. of the worker node. The jobs will be matched to unsuitable nodes that do not support these requirements owing to the fact that such granular information is not included in the job matching process. Such jobs will start executing on the matched node and ultimately fail wasting the Grid resources due to the absence of said requirements. Even if the Grid administrators decide to introduce a new infrastructure parameter to their job matching process, this requires the following lengthy, multi-step process that makes it cumbersome to add or remove parameters which leads the administrators to decide to only have mandatory parameters in the job matching process.

1. Initially, Grid middleware job matchers should be updated to accept the parameter if it is sent from the worker nodes. These job matchers are mission-critical to the functionality of Grid middleware and they are updated only if its absolutely necessary.

2. Once the job matchers are updated, the change should be propagated to other central servers which is a careful process.

3. Then the job pilots(Grid middleware clients) should be updated to read these parameters and send this information to the job matchers. Since the existing systems do not have any integration with infrastructure monitoring tools, they are unable to obtain this information.

4. If the monitoring work is done in the pilot job, the code for the pilot should be updated and propagated across the Grid. Since any change to the pilots also breaks the functionality of the Grid, such updates are rolled out gradually across the Grid over a span of a couple of days or weeks.

As it is evident from the above process, updating Grid middleware causes code changes to critical code bases, large manual effort and a multi-week upgrade process and often it is deemed unnecessary to follow this process just to add a new matching parameter to the job matching process leading the Grid middleware developers to consider only a set of mandatory hard coded parameters in job matching.

This research focuses on addressing this problem by having a flexible way to use any of the infrastructure parameters in a worker node in the job matching process and providing an extensible way to add or remove new parameters to the job matching process without any changes to the codebases or updates to Grid middleware by using the information collected through the newly designed Grid monitoring system.

In the following section, we discuss a few of the famous Grid middleware that undertake job matching and how they are limited in terms of having extended job matching capabilities as discussed above.

2.2.2 Existing systems

2.2.2.1 PanDA

A Toroidal LHC Apparatus (ATLAS) is the largest experiment in the Large Hadron Collider (LHC) which is responsible for discovering the Higgs-Boson particle in 2012[43].

The Production and Distributed Analysis (PanDA) workflow management system(also known as Grid middleware) is developed for managing workloads in ATLAS experiment[44]. PanDA typically runs around 600,000 jobs concurrently while processing more than 5 million jobs per week.

When the modern Grid WMSs are considered, we can identify that each of the WMSs follows a common high-level architecture which has become the standard architecture for Grid WMSs. Each WMS has a central server that acts as the central point of the system. They also have a service running at Grid sites that submits pilot

jobs to batch queues in Grid sites. These pilot jobs are assigned to execute on worker nodes by the batch queue and once started, they communicate with the central server to complete job execution. The Fig. 2.7 shows the high-level overview of the system.

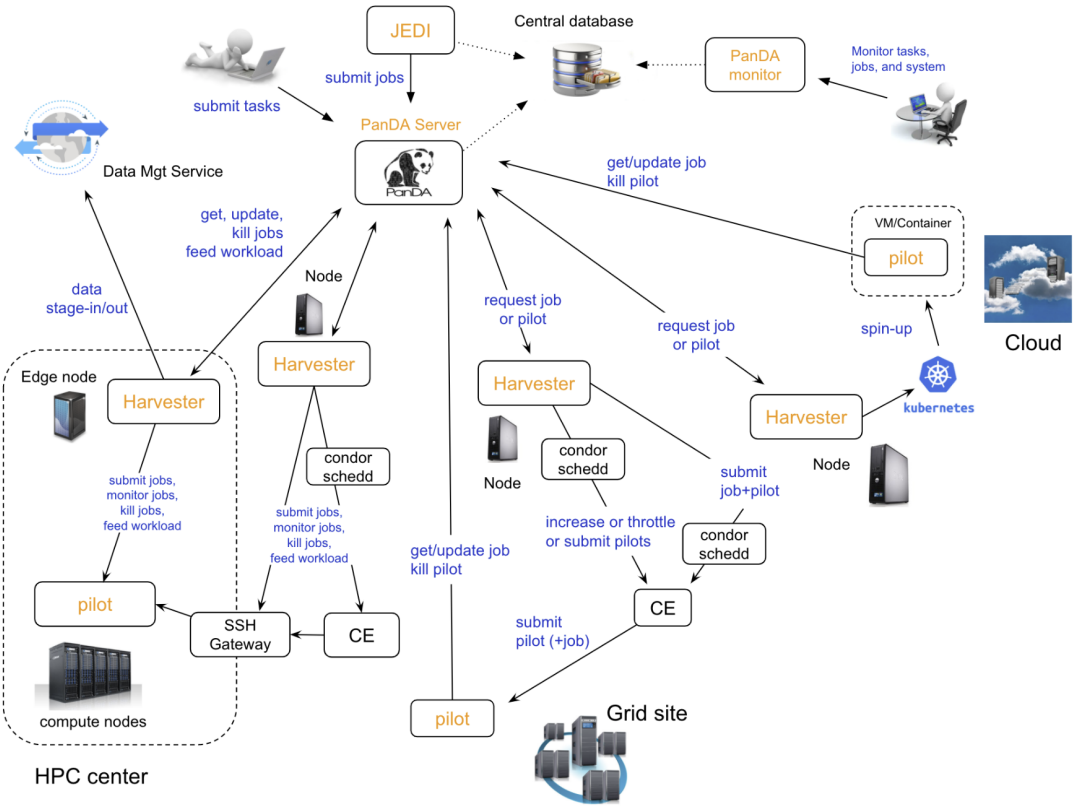

Fig. 2.7: PanDA WMS Overview

The central server of PanDA is called the PanDA server which is responsible for centralized management of the WMS. It also has a component called pilot scheduler which submits pilots as discussed above. PanDA has an additional component called Harvester[45] which has been introduced recently with the intention of providing a common machinery for pilot provisioning on ATLAS computing resources. Harvester sits between the PanDA server and pilot scheduler and is designed to run on edge nodes in HPC centers. The harvester handles the job submissions to worker nodes via the pilot scheduler.

The harvester is capable of collecting a set of worker node information and transferring it to the central PanDA server. Following is a sample of worker node data collected by the Harvester[46].

```
"AGLT2": {
  "cmtconfigs": [
    "x86_64-centos7-gcc62-opt",
    "x86_64-centos7-gcc8-opt",
```

```
    "x86_64-slc6-gcc49-opt",
    "x86_64-slc6-gcc62-opt",
    "x86_64-slc6-gcc8-opt"
  ],
  "containers": [
    "any",
    "/cvmfs"
  ],
  "cvmfs": [
    "atlas",
    "nightlies"
  ],
  "architectures": [
    {
      "arch": ["x86_64"],
      "instr": ["avx2"],
      "type": "cpu",
      "vendor": ["intel","excl"]
    },
    {
      "type": "gpu",
      "vendor": ["nvidia","excl"],
      "model":["kt100"]
    }
  ]
}
```


As we can see from the data, Harvester can report information like the type of containers supported, CPU architectures, Available GPUs etc. of the worker node. However as it is evident from the data, Harvester does not provide granular information like Operating systems, installed libraries etc. and it is not easy to add or remove monitoring parameters due to the long process needed to roll out upgrades to site services like Harvester. Therefore, PanDA cannot do brokering based on granular information on parameters like the Operating system.

2.2.2.2 GlideIns

GlideIns[47] is a workflow management system that is used for job execution in multiple organizations like The Compact Muon Solenoid (CMS) experiment at CERN, Fermilab in the United States etc. It is developed as an extension to the HTCondor[48] which is the most famous batch queue system worldwide. HTCondor undertakes the task of managing the resources of an individual computing site and functions as a local

resource manager that assigns tasks submitted by a pilot scheduler to nodes. GlideIns WMS extends its functionality to include the submission of pilot jobs to worker nodes and providing support for managing jobs in a Computing Grid instead of a single Grid site.

GlideIns WMS manages jobs of a Grid by submitting a process called GlideIn to worker nodes of the Grid. GlideIn intends to reserve a job execution slot in a worker node similar to a pilot job thus creating a Global job queue for the relevant Computing Grid.

In GlideIns, User jobs are submitted to Condor Pools. In parallel GlideIn factory keeps submitting Glidins to Grid sites that reserve the slots for job execution. Once the GlideIns land on sites, they choose the jobs that are most suitable to them and start execution.

When the infrastructure parameters of worker nodes are considered GlideIns WMS supports a limited parameter set like CPU Cores, request memory etc. and there is no evidence to show it supports granular parameters like Operating System or installed libraries which seems to be lacking in all the major Grid WMSs.

2.2.2.3 JAliEn

Java ALICE Environment (JAliEn) [49] is a new Grid middleware developed to support the operations of the ALICE Computing Grid. It handles Grid operations across 60+ Grid sites contributing a total of 70,000+ cores of computing power. It was developed since few years ago to replace the existing Grid middleware AliEn[41] and has been deployed in production across all Grid sites by the end of the first quarter of 2023. Owing to its novelty, JAliEn introduces new features like containerized job[50], resource oversubscription[51] etc. that are known to be some of the latest updates on the Grid computing domain.

JAliEn consists of a few separate components that constitute the complete middleware.

- jCentral - jCentral is the interface that opens up the central services of the WMS to the outside. Clients and other services communicate to jCentral for various purposes like job matching, reporting monitoring information, retrieving database information etc. It is the only component that communicates directly with the database to provide necessary information to clients

- Computing Element - Computing Element (CE) is a service that runs on edge nodes in individual Grid sites. It is the only node that is exposed to outside from the Grid site and the last point of access to ALICE Grid administrators. It keeps submitting JobAgents to the local batch queue similar to the functionality provided by Harvester in PanDA WMS.

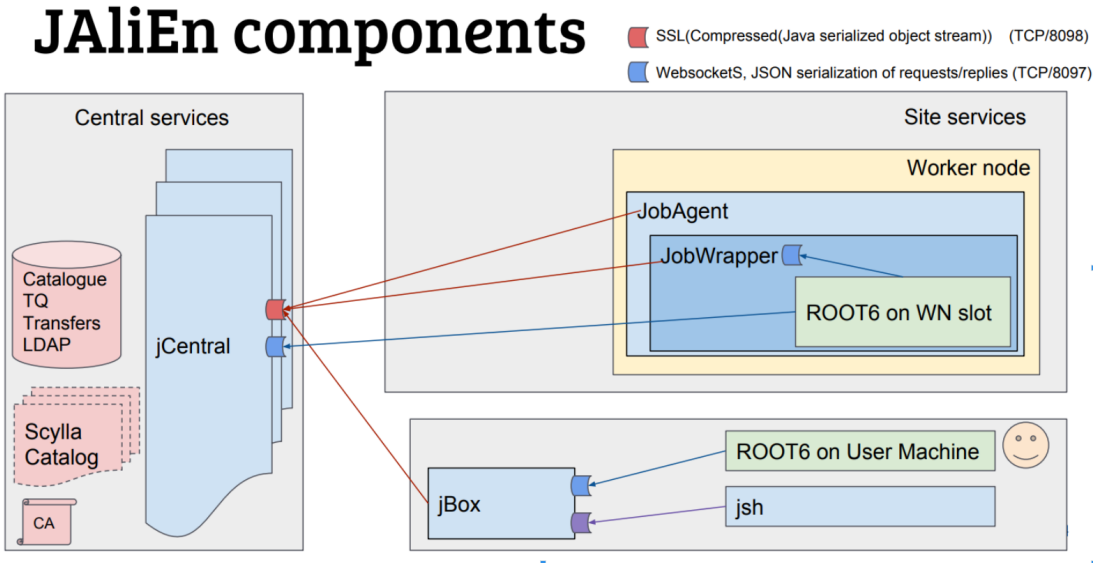

Fig. 2.8: Components of JAliEn WMS

- JobAgent - JobAgents are the pilot jobs that are submitted to Grid sites in JAliEn with the intention of reserving job slots in advance in the worker nodes. Job Agent is responsible for reserving the worker node for future job execution, requesting a matching job from Central, and spinning up a Job wrapper once it receives a job to execute.

- JobWrapper - JobWrapper is the last unit of component in the JAliEn WMS and it wraps the actual job payload that is required to be executed. Multiple JobWrappers can be spun up by the Job Agents leading to parallel execution of multiple jobs.

The job Matching process in JAliEn is undertaken by a component called “Job Broker” that sits inside the jCentral. When JobAgents land on a worker node, it collects some preliminary information about those worker nodes like CPU cores, space left on the node, available memory etc. and reports them to the JobBroker via jCentral. JobBroker queries the database for the pending job and picks the most suitable job for that node considering multiple factors like matching of the said infrastructure parameters, priority of the job, Time To Live (TTL) of the job etc.

In terms of the Job matching processes, JAliEn Job Broker is not very mature at the moment and it is only able to do some basic matchings like checking if enough resources are available to run the job or if the requested criteria are available in the node. The parameters sent from the JobAgent in the form of a JSON document named “SiteMap“ are used for the Job matching process. Site Map contains the said parameters like CPU Cores of the node. This parameter collection logic is hard coded in the Job Agent and thus it can only collect details about a limited set of parameters. Sim-

ilarly in the Job Broker, how to use these parameters of matching is hard coded. This leads to a highly limited job matching system in terms of the parameters that can be used for the successful matching of jobs to the most suitable nodes. Adding, removing, or updating a parameter could require updating Job Agents used across all Grid sites and pushing a change to jCentral which is a mission-critical component for the WMS. Therefore, the JAliEn Grid administrators were looking for a flexible and extensible way to add new parameters to the Job Brokering process without affecting the existing components. One of the focuses of this research is to provide a solution to this gap using the newly developed Grid infrastructure monitoring design.

CHAPTER 3

METHODOLOGY

3.1 Data Collection

The initial part of the research was focused on identifying the most ideal way to collect infrastructure information from the Grid nodes without reducing the job performance. Several approaches were tried and the following section describes each of the approaches along with their pros and cons.

3.1.1 Initial data collection with a Job

The approach that has been tried out in the Site Sonar v1.0 model was tested initially to get an understanding of the data collection process. To try out if it is possible to collect infrastructure metrics from a worker node as done by Site Sonar v1.0, a simple job that tries to read the operating system from the worker node it gets landed on was submitted to the Grid. The goal of this task was to ensure that all sites allow reading their system information as an external entity that does not have root access and understanding the overheads and complications associated with the current method of collecting data.

The sample job description file is shown below.

```
//job.jdl

Executable = "/localdomain/user/j/jalien/testscript.sh";
OutputDir = "/localdomain/user/j/jalien/output_dir_new";
Output = {stdout@disk=1};
```

The executable used in the JDL is given below.

```
//testscript.sh

#!/bin/bash

echo "Starting execution"
echo $(lsb_release -a)
sleep 10
```

Once its functionality was verified, a large number of jobs of the same type were submitted to the Grid without any Grid site restrictions to check the output from different sites. Analyzing the output from a number of sites showed that reading information

is unrestricted across the Grid. Therefore this approach was considered as a viable solution as a proof-of-concept implementation of the data collection part of the research.

3.1.2 Site Sonar v1.0 Implementation

This model has been built with Python as a Grid client which is capable of repeatedly submitting jobs to the Grid. However, Grid job submission allows only binding jobs to specific sites, not to specific nodes. Therefore, there is no direct way to collect infrastructure metrics of all the nodes in the Grid. Hence, the client has been programmed to submit jobs in the scale of twice the presumed number of nodes in each Grid site with the assumption that it will hopefully land on all the nodes in each Grid site providing a full Grid coverage. The collected information was pushed to a PostgreSQL repository which stored the results as plain text.

3.1.3 Site Sonar v1.0 Drawbacks

Site Sonar v1.0 is the existing system used by ALICE Computing Grid to monitor their infrastructure. This system for data collection has major drawbacks in its implementation which are discussed below.

3.1.3.1 Grid Flooding

Site Sonar v1.0 floods the whole Grid with jobs in the scale of twice the number of nodes in each Grid site in a single round of data collection. This causes a severe bottleneck in Grid usage, as when the tool is running it not possible for other users to easily obtain a slot to run their job because job slots in multiple Grid sites will be occupied by the Site Sonar at the same time.

3.1.3.2 Low Grid coverage

When a worker node starts accepting jobs, it tends to keep running for some time even after it finishes running its jobs to keep accepting future jobs. This is a measure implemented in the Job Agents(Controller task that handles the job execution in worker nodes) to keep utilizing the same node instead of reserving multiple nodes to run jobs with short life spans. Site Sonar v1.0 jobs fall into this exact category as it sends bursts of very short jobs that do a simple task and exit. Therefore Site Sonar v1.0 jobs easily tend to land on the same node in a specific site, when a large number of jobs are submitted to that site which reduces the number of worker nodes covered in a single data collection run.

3.1.3.3 Resource Utilization

To increase the Grid coverage, the solution in place was to increase the lifespan of the submitted jobs. Each job was appended with 5 minutes of sleep time to ensure they do not keep landing on the same node. This resulted in a huge waste of core hours of the Grid due to the intentional idle time in Site Sonar job causing low efficiency in Grid usage at the time of running Site Sonar. The calculated lost core hours amounts without this 5 minute delay are presented in section 5.3.1.2 under the evaluation. The lost core hour count would spike much higher when this delay is also considered.

3.1.3.4 Manual Intervention

Site Sonar v1.0 was based on pull model where it tries to pull infrastructure metrics from each of the nodes. Since the Grid consists of a large number of worker nodes, it takes around 24 hours to complete a single data collection run of Site Sonar v1.0. As this is a heavy resource-intensive process and it causes Grid flooding and Resource utilization issues discussed above, it was essential to manually execute and monitor this run to ensure minimum effect on usual Grid workloads.

3.1.3.5 Partial Overview

Given that Site Sonar yields a low Grid coverage as discussed above, most of the time we will see configuration information of only a part of the Grid site. It would be incorrect to use this data to represent the complete site as the nodes that were not covered could have different infrastructure metrics that would provide an incorrect idea about the Grid site. Therefore, Site Sonar v1.0 can be used to only have a partial overview of the Grid site.

3.1.3.6 Data Querying

Site Sonar v1.0 uses a PostgreSQL database to store and manage the data as explained in the section 3.1.2. Although PostgreSQL supports JSON format, the tool uses plain text for storing test results which is not ideal for storing this kind of data. It was not interoperable with any new NoSQL storage engines like Elasticsearch. Even though PostgreSQL supports querying data with JSON, it does not offer strong JSON indexing capabilities leading to very low data querying performance. The issue is intensified by the fact that it has to search more than 300,000 to provide the result for a data retrieval query. Although this was improved by the work of Evan Sandvik[52] in his project to develop an improved visualization interface for Site Sonar v1.0, the system still had a latency between 600 milliseconds to 20 seconds depending on the complexity of the query. One of the main aspects of this study was to optimize and improve the query

performance of the system to an acceptable level. Fig. 3.1 shows the execution analysis of one of the most basic queries in Site Sonar v1.0.

```
mon_data=# EXPLAIN ANALYZE SELECT sitesonar_tests.host_id,
    ↪ test_message_json -> 'SINGULARITY_LOCAL_SUPPORTED', ce_name,
    ↪ test_name FROM sitesonar_tests INNER JOIN sitesonar_hosts ON
    ↪ sitesonar_hosts.host_id = sitesonar_tests.host_id WHERE
    ↪ last_updated > '1635860606' AND test_name='singularity';

                          QUERY PLAN
-----------------------------------------------------------------------
Gather  (cost=53508.91..62708.26 rows=18071 width=52) (actual
   ↪ time=427.450..612.211 rows=17349 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Hash Join  (cost=52508.91..59901.16 rows=7530 width=52)
     ↪ (actual time=421.370..577.678 rows=5783 loops=3)
        Hash Cond: (sitesonar_hosts.host_id = sitesonar_tests.host_id)
        ->  Parallel Seq Scan on sitesonar_hosts  (cost=0.00..6882.43
           ↪ rows=187043 width=8) (actual time=0.109..58.125
           ↪ rows=149635 loops=3)
        ->  Parallel Hash  (cost=52414.78..52414.78 rows=7530 width=206)
           ↪ (actual time=417.045..417.046 rows=5783 loops=3)
              Buckets: 32768  Batches: 1  Memory Usage: 2592kB
              ->  Parallel Index Scan using updated_and_name_idx on
                 ↪ sitesonar_tests  (cost=0.56..52414.78 rows=7530
                 ↪ width=206) (actual time=1.590..408.852 rows=5783
                 ↪ loops=3)
                    Index Cond: ((last_updated > '1635860606'::bigint)
                       ↪ AND (test_name = 'singularity'::text))
Planning Time: 0.549 ms
Execution Time: 613.851 ms
(12 rows)
```

Fig. 3.1: Query Analysis of Site Sonar v1.0

3.1.4 Proposed Solution

Site Sonar v1.0 follows a Pull model where it tries to pull data from the nodes in the computing Grid. Most of the Site Sonar v1.0 drawbacks discussed in section 3.1.3 are a result of limitations introduced to the software due to the use of Pull model. It causes the software to run a largely resource intensive process for a very long time(around 24 hours) to collect data which requires a considerable manual intervention and causes resource utilization issues in the central machine where the process runs as well. The lack of a central way to communicate with the Grid nodes causes low Grid coverage and Partial Overview issues as described in the previous section.

3.1.4.1 New version based on Push Model

Since the pull model of data collection was identified as a major limitation of the existing system, the initial research goal was focused on devising a new data collection framework based on the push model for collecting data from Grid nodes. In this model, we plan to push data from individual nodes to a set of central servers rather than pulling data from the individual nodes using a central process.

The following concerns were considered when researching the feasibility of the push model.

1. Outbound network connectivity - A major blocker in pushing data from Grid nodes would be pushing data from Grid nodes to a central domain as firewall restrictions are enforced in any Grid node. However, each Grid site already allows outbound connection to a central domain for uploading job monitoring data which is essential for a Grid system. The same domain/server can be reused in our scenario without requiring new network whitelisting for site admins.

2. Installing agents - As we have discussed in the Literature review in section 2.1.4, the existing infrastructure monitoring systems require installing agents on the nodes to collect and push data. This is not accepted by Site administrators as they do not prefer to install optional software on their Grid sites to support experiments. On the other hand installing monitoring/data reporting agents on all the Grid nodes (around 7,000 nodes) with different site configurations and maintaining them will be a task that will require a massive effort and manpower. Reusing the existing Job Agents was identified as the ideal solution as they do not require any additional installations and can do the data collection without an issue.

3. Grid performance - Since Job Agent runs for each Job submitted to the Grid, adding long running tasks to run before the execution of the job would cause a detrimental impact on the efficiency of the Grid usage. As we discussed in section 3.1.3.3, this would essentially cause large resource wastages if not properly managed and the impact would be much worse in this approach as the wastage would occur each time a job runs. To avoid this, only short running data collection probes are used and the delays added in the section 3.1.2 were removed. Additionally, the data is collected only once within a day from a node which minimizes the impact of this.

4. Grid coverage - A major concern of the push model was the lack of Grid coverage. This problem is automatically avoided in the new approach because Job Agent is inherently developed to run on all nodes in the Grid and have a distributed workload on all nodes. Additionally, unlike the Site Sonar v1.0 job submission process(section 3.1.2), Job Agents continuously get spun up across every minute every day causing each node in the Grid ultimately to run a Job Agent and report infrastructure data.

In addition to this, it answers the issues discussed in 3.1.3 as follows.

1. Grid flooding - Since the new model pushes data from the nodes to the central

servers, there is no job submission required and hence the Grid won't receive any job extra submissions for collecting data.

2. Low Grid coverage - The old model had a concept called "Run" which refers to a single round of data collection by submitting jobs to flood the Grid. The new model has no concept of a Run and it runs on demand instead. Since the data collection task always runs before the job execution, if a node has accepted a job, it will definitely return the infrastructure metrics which means that if a node is up and running it will eventually report its metrics at some point yielding a higher Grid coverage.

3. Resource Utilization - The requirement to have idle time in data collection is removed with the new model. Therefore, there is no resource idling and the full process will take only a few tasks to execute leading to negligible resource wastage.

4. Manual Intervention - As the data collection is integrated with the Job Agent, there is no requirement to run this task manually, instead it will run on demand.

5. Partial Overview - The higher Grid coverage leads to providing the results from almost all nodes across the Grid sites leading to a full overview of each Grid site and allowing to derive conclusions about the Grid site out of the collected data without much uncertainty.

3.2 Data Storage

For the purpose of this research, we have evaluated both SQL and No SQL data storage options to check their suitability for the needs of the project. The existing data collection tool on Site Sonar v1.0 uses a SQL database engine and its implementation was used to evaluate the introduction of a new NoSQL database in the research to identify the pros and cons of the two database types.

3.2.1 SQL Storage

The existing implementation of Site Sonar v1.0 uses a PostgreSQL database and stores the data inside multiple tables. Existing system support only direct key-value pairs in the following form.

```
key1 : value1
key2 : value2
```

eg:-

```
OS : Ubuntu 22.04
GCC_VERSION : 11.3.0
```

This limitation has been introduced because native SQL does not support objects and expects the data to be in a format similar to above which can be directly mapped to a table row. One of the major requirements in designing the monitoring architecture in this research is making the system highly flexible allowing the monitoring probes to change at any time in an ad-hoc manner without having to change the underlying architecture. The general SQL table format does not support this requirement because we are unable to add multiple values for the same key without adding a new column in the table or using additional tables. The issue intensifies when it is required to store complex objects like nested maps which is hard to manage with SQL databases. Changing the database structure in an ad-hoc manner will also create major issues and hence the existing approach is not suitable for the purposes of this research.

3.2.2 Post Data Filtering

When it comes to the computing resources in a Grid computer, it can be considered as a black box from a Grid administrator's point of view. Although the Grid sites pledge their computing power to the Grid, there is no guarantee on how these computing nodes would be configured or managed. Therefore, often when there is a probable configuration issue to be debugged in the Grid, Grid administrators do not have an exact idea of what exactly should be checked and what values to expect. This makes working with existing infrastructure monitoring systems much harder because there is no straightforward way to collect unknown parameters with SQL-based systems as they follow a rigid schema.

When we need to monitor a new parameter in a computing Grid, there is no way to assume what values we see at the end. To increase the gravity of the problem, sometimes the administrators are not sure of what to monitor as well. To have a good idea about the Grid, they would initially like to collect a bunch of data although only some of that data will be useful in the end. None of the existing systems provide the capability to collect a considerable amount of data at first and then narrow down on the interesting data which means the administrator must know what he is looking for in advance which is not the case usually. Also when the administrator assumes what should be monitored, he might be loosing some important information in the process if his assumption is not correct.

When working with a highly heterogeneous system like a Computing Grid with a large number of nodes, each monitored configuration leads to a number of possible values. For example, if we look at the number of cores per node across the Grid there are about 15 possible values(Fig. 3.2 and if we look at CPU models across the Grid it

show more than 50 values(Fig. 3.3). It is not possible to anticipate what needs to be monitored in a large system like this.

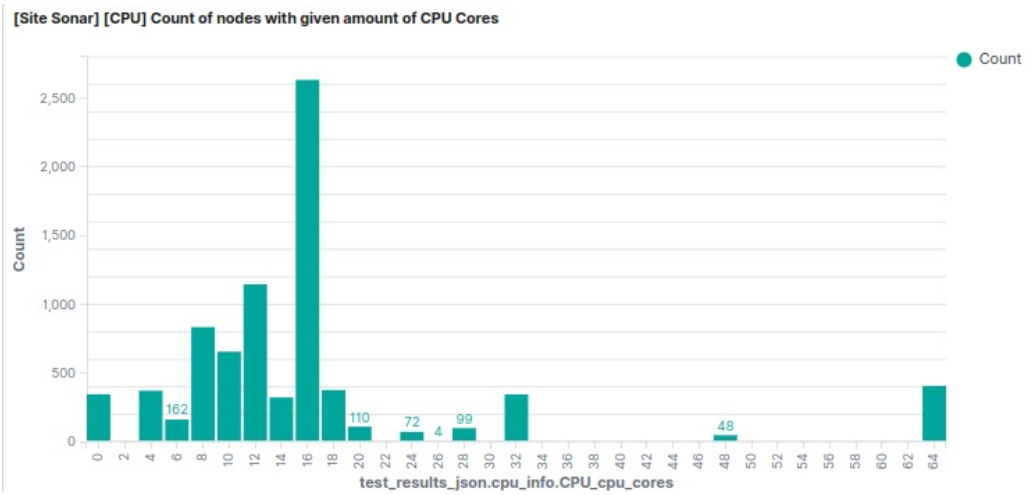

Fig. 3.2: No. of nodes with given CPU Core count

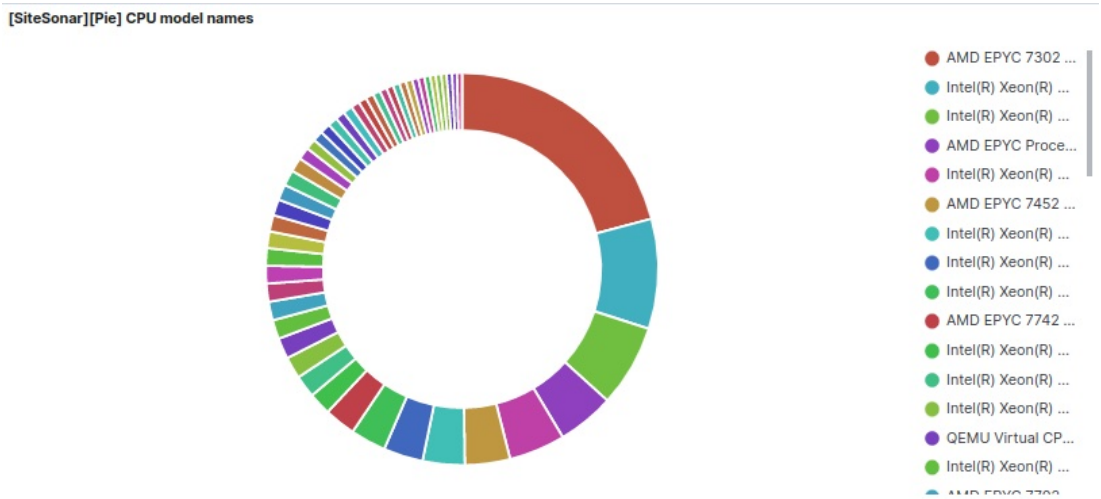

Fig. 3.3: CPU model distribution in the ALICE Grid

This research proposes storing objects using JSON to address this problem. This would allow collecting any number of parameters as the number of key-value pairs in the JSON object can be changed without an issue. Although the native SQL does not support complex objects, PostgreSQL offers support for JSON objects. All the monitoring probes in Site Sonar v1.0 were converted to output results in JSON format to utilize this feature of PostgreSQL allowing the monitoring probes to be changed in an ad-hoc manner that results in a highly flexible data collecting tool.

This will ultimately introduce a much-needed Post Data Filtering feature to the proposed system allowing the Grid administrator to monitor some suspicious configurations at the beginning, see what values they report, and then narrow down on what

seems to be interesting. This process is explained in detail in section 5.2.3 where this option is used to change monitoring probes on demand to identify issues in the Grid.

3.2.3 NoSQL Storage

Although storing data as JSON in PostgreSQL provides the required flexibility to the system, it takes a considerable time to query and retrieve JSON data via the PostgreSQL database. While the query times are much better than storing data as text fields and aggregating them manually, the retrieval speed becomes considerably slow when querying a large data set. This is explained and proven with experimental results in section 3.1.3.6 and Fig. 3.1.

Since Elasticsearch provides a much better data retrieval speed, this research proposes the use of Elasticsearch as the backend for storing data which provides some additional features as well. Some of the main features that we compared when moving the storage to Elasticsearch are summarized in table 3.2.3

Feature	MySQL with Plain Text	PostgreSQL with JSON	Elasticsearch
JSON support	×	×	✓
Fast query time	×	×	✓
Visualization support	×	×	✓
Flexible schema changes	×	✓	✓
Inject data without schema	×	×	✓

3.3 Data Visualization

Data visualization plays a crucial role in any data collection system. Since it is hard to analyze the collected data in text formats, they are often built into charts and tables that can be easily analyzed visually. Data visualization technologies had a major improvement in recent years with the introduction of new tools like Kibana, Grafana etc. that completely replaced the existing data visualization systems used by the organizations.

One of the critical issues with existing systems can be seen as the lack of data visualization capabilities. Most of the existing systems use in-house data visualization tools that lack a lot of features or are not being actively developed anymore. These tools can be seen as not up to date with the latest visualization technologies because advent of tools like Kibana, Grafana etc. recently has become the accepted standard for data visualization. We identified visualization of infrastructure monitoring data as an important feature that can be considerably improved in the course of this research in comparison to the existing literature.

Fig. 3.4 and Fig. 3.5 show the visualizations generated by systems that exist in the literature using old technologies and new technologies respectively. It is evident

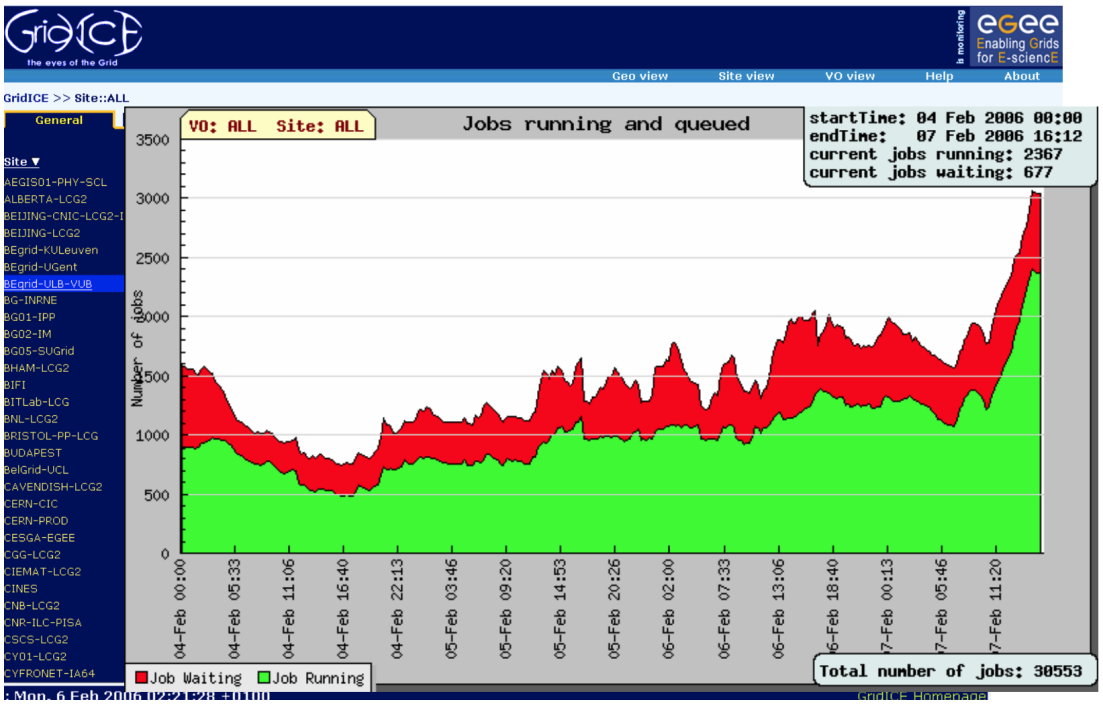

Fig. 3.4: GridICE Interface

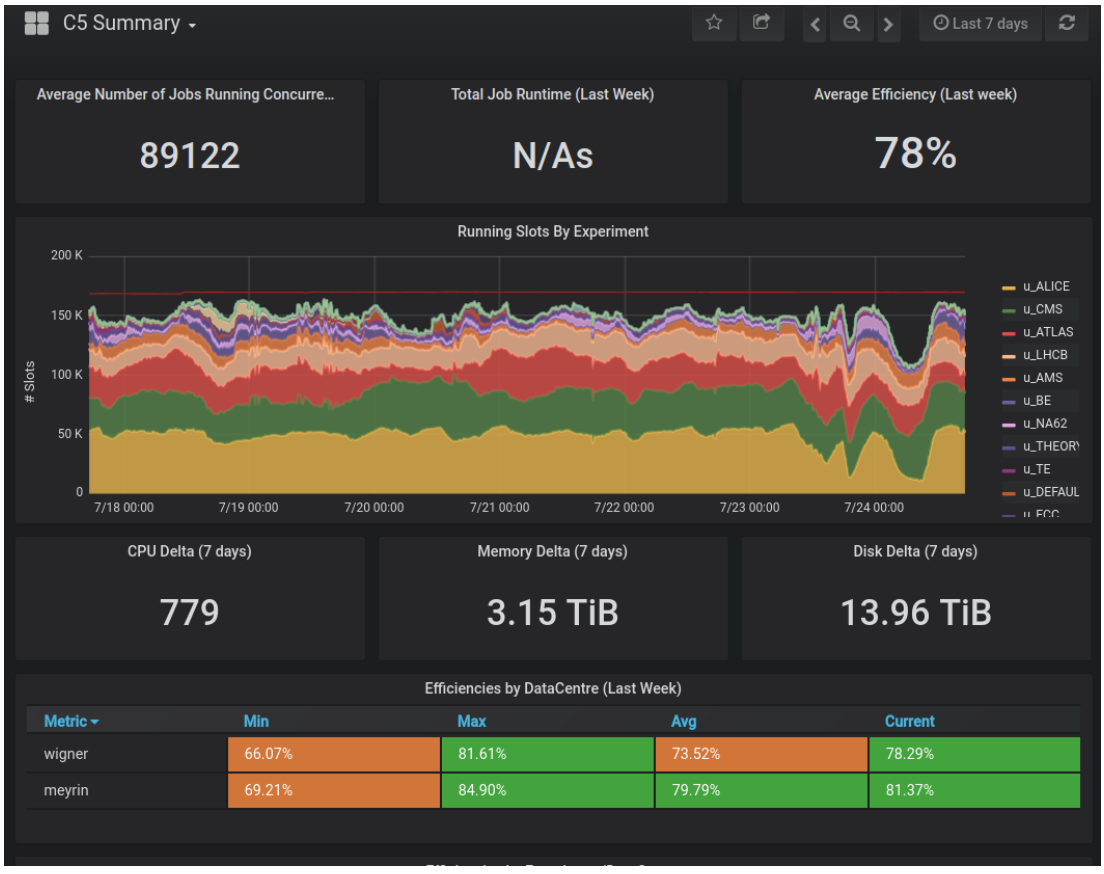

Fig. 3.5: MONIT Interface

from the images that compared to modern visualization tools like Kibana, the existing visualizations based on old technologies seem to be considerably less intuitive and user-friendly.

3.3.1 No Code Visualizations

In addition to the lack of features in the existing visualization systems, having to create visualizations using code was identified as a major drawback in them. Given that each system is developed in-house, they have their own visualization libraries and the user has to learn them add a new visualization to the existing dashboards. It is also necessary to deploy this code in the production system as a new release of the monitoring system often.

It can be clearly seen that this is a very hectic process especially when it is required to develop dashboards that consist of a number of visualizations. Additionally, most systems like MonAlisa and GridICE do not have the capability to create dashboards and there is no easy way to compare one plot with another because each one resides in its own window. A detailed breakdown of new features that can be easily provided by using modern visualization technologies are listed in table 3.3.1.

Feature	Old Visualization Systems(In house)	Modern Visualization Systems(Eg: Kibana)
Flexible Schema	×	✓
Quick data retrieval	×	✓
Combine multiple visualizations	×	✓
No-code visualizations	×	✓
Complex filters	×	✓

No code visualizations can be seen as the most prominent feature out of the above list because one of the major requirements in this course of research is providing Post Data Filtering as described in 3.2.2. In order to effectively use post-data filtering, there should be a possibility to quickly and frequently change the visualization on demand which is explained in section 5.2.3. This is very cumbersome with existing tools because updating the code frequently to change a visualization and deploying in production often requires a lot of effort and redundant efforts while causing human errors and deployment failures often.

Considering the above facts, we decided that the proposed Grid architecture must use a standard visualization engine that would reduce a lot of work and provide advanced capabilities out of the box. Kibana and Grafana were evaluated as the best options because they are considered standard and are the most widely used visualization solutions in the computing domain at the moment. Given that the 2 systems provide

very similar features, Kibana was chosen as the most suitable solution for this approach as it is directly related to Elasticsearch and directly bundled with Elasticsearch as well.

3.4 Proposed Monitoring Architecture

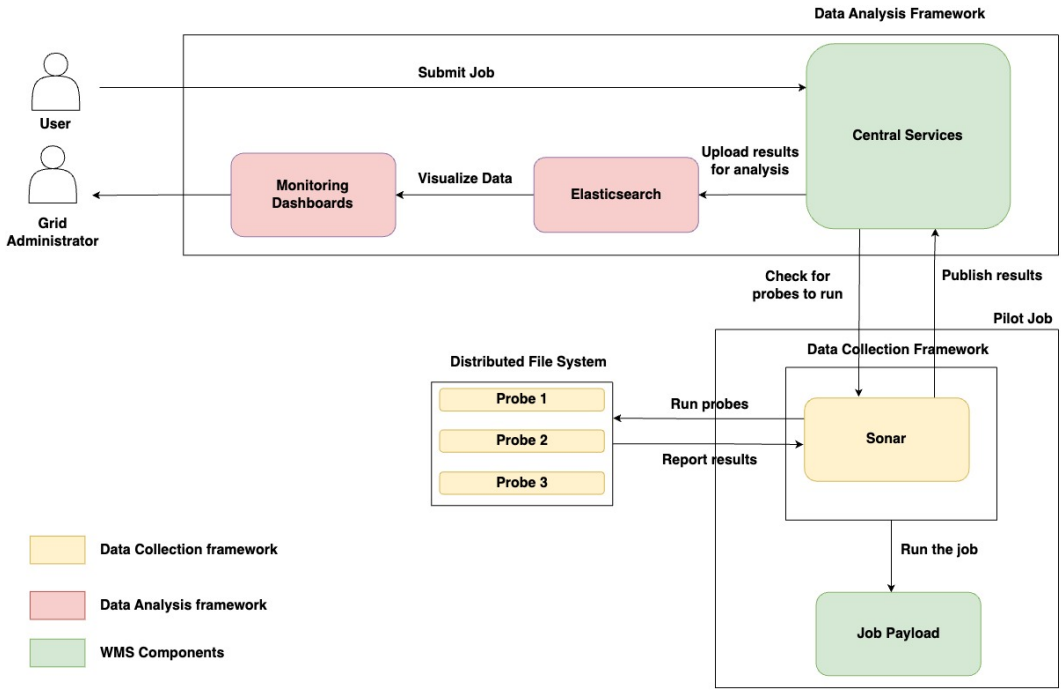

Fig. 3.6: Proposed Architecture for Grid Infrastructure Monitoring Tool

Figure 3.6 shows the full architecture proposed for the Grid infrastructure monitoring system that is discussed in this research.

The monitoring tool consists of 2 main components: The data collection framework and the Data Analysis framework which are discussed below.

3.4.1 Data Collection Framework

The Data Collection framework is responsible for collecting and persisting data. It defines how the monitoring probes should run and report data to match the requirements of the system. A number of factors are considered when designing the data collection framework.

1. Easy Extensibility
2. High Flexibility
3. Easy integration with Job Pilots

3.4.1.1 Easy Extensibility

A major requirement in a monitoring system is the ability to add or remove monitoring probes (i.e. new components of the existing types) to the system with minimum architectural impact. All the SQL-backed systems will require updating their databases in advance of adding a new monitoring probe. Other than that, a new version of the software that includes the new probe should be rolled out across the Grid which makes it nearly impossible to add or remove new probes easily without having an architectural impact on the system.

The proposed system follows a novel approach to address this issue which completely removes the architectural impact on the system. Each Computing Grid uses a distributed file system. The distributed file system is mounted to each computing node when they become a part of the Grid and this partition is used to provide the common dependencies and libraries that are required for the execution of jobs. We propose to have a separate directory on this shared file system that hosts the individual monitoring probes and requires the job pilot to fetch the monitoring probes directly from this directory. This approach has several advantages over the conventional approach.

- Since the monitoring probes are not packaged as a part of the data collection framework, there is no requirement to update or release a new version of the monitoring tool to include the new probes which results in zero architectural impact on the system
- Distributed file systems have their own mechanism to distribute a new file added to it. Therefore no additional steps are needed to add a new monitoring probe rather than copying the probe file to the monitoring directory.
- Distributed file systems update their content periodically(within a few minutes) to ensure that their content is up-to-date and they are in sync with their peers. Therefore the new probe is shared across the Grid within a few minutes removing the extra manual effort needed to share the probe in the conventional approach.

For edge cases like limiting the monitoring probes that should be run on a specific site, a database table that provides a whitelist of probe names are used. However, this table is not required to be used and all probes run for each site by default. This feature is added only as a precautionary measure for edge cases.

3.4.1.2 High Flexibility

As explained in the section 3.2.2, one of the major requirements that comes with the system is the ability to change the collected data in an ad-hoc manner which allows the Grid administrators to identify areas of interest and narrow down to the details in those

areas. This is achieved by using the NoSQL storage described in section 3.2.3. The non-rigid JSON schema will allow the collection of unstructured data and changing the keys of the collected data without any issues.

In addition to that, we propose using shell scripts to define the monitoring probes. This would provide a massive degree of freedom in data collection as it allows to read any file that can be accessed by a user of the system. Another reason for this is to ensure that minimal tools are used in data collection as the worker nodes might not have additional libraries like Python or Java installed and using them in probes could cause data collection failures in some sites.

We have verified that this does not pose a threat to the host system(worker node) as we can read only the files that are accessible by a general user in the system. Since most of the required configurations can be read by a user, there is no hindrance to the functionality of the software due to this. Additionally, only a very limited number of Grid administrators have the permission to update the distributed file system which further minimizes the risk of a potential security breach.

3.4.1.3 Easy Integration

As we have identified through the course of this research, the incapability of integration with the Grid WMSs seems to be a major drawback of the existing Grid monitoring systems. Each Grid monitoring system so far has required the Grid administrators to deploy a new layer of software relevant to monitoring to provide infrastructure monitoring to the system. This is heavily unwelcome by Grid administrators because this adds another layer of software that they have to manage across thousands of computing nodes that are already too complex to manage. This can be identified as the major reason why most of the infrastructure systems have not been adopted by the community leading to their end of life sooner than anticipated. Easy integration with existing Grid middleware without deploying a new control layer across the Grid has been heavily studied in this research and the following section describes the associated concerns and how each of them were resolved.

- Since we are proposing a general architecture, that can be used in any Grid middleware, it was critical to identify an integration point that is common to all Grid middleware, and the submission of pilot jobs was identified as the most common feature among all the Grid WMS[46].

- Given that the Grid middleware is carefully developed to match the specific requirements of the relevant Computing Grid, middleware developers are not keen on introducing additional code that is irrelevant to its functionality. To address this, we have developed a controller function that the middleware can call when starting to execute the job ensuring that their usual flow is not broken. Calling

this function will ensure that the monitoring probes are executed, and data is collected and reported back to the central servers without any impact on the existing functionality. The controller is available both as a Java function and a bash script to be used at the integration according to the preference of the developers.

- As the data collection controller will run at the beginning of a job, it was essential to ensure that the data collection will not have a performance impact on the job. Several measures were taken to ensure this.

    1. Data would be collected only once per day for each node as the infrastructure metrics do not change frequently
    2. Data is collected from very short and fast monitoring probes
    3. Provide the ability to enable/disable running any probe on any site depending upon the requirement

  The experimental results presented in the section 5.3.1.1 show that the overhead of the new design is minimal amounting to around 3.5 seconds of overhead per node per day which is negligibly small.

3.4.2 Data Analysis Framework

The Data Analysis framework is responsible for analyzing and visualizing the collected results in the proposed monitoring system. This process consists of 3 major sections: Data Collection(section 3.1), Data Storage(section 3.2), and Data Visualization (section: 3.3). We have selected Elasticsearch, Logstash, and Kibana as the technologies to develop our data analysis framework. This would require a self-hosted ELK cluster running in the environment of the monitoring system maintainer or a cloud subscription to an ELK stack. However each technology can be replaced with a similar technology like Grafana, Splunk etc. because they provide roughly the same functionality and can switch from one to another easily, It is foreseen that with the wide use of these technologies, each organization usually has either of those setups already and those instances can be reused to implement the monitoring system.

Usually, each Grid WMS consists of a central server or a server farm to accept the incoming details from the job pilots. We recommend introducing a new HTTP service in the server to accept the new monitoring data sent by the pilots. This approach provides two layers of security.

- Usually job pilots are allowed to communicate with a few whitelisted domains. Rather than whitelisting an additional domain for the purpose of monitoring, we would recommend reusing an existing domain and accepting the traffic from there to reduce the surface vector in case of a security breach.

- Usually the ELK stack of an organization is hosted within the private network(or private cloud) and contains sensitive information. Therefore, it is not usually exposed to the public internet. But if it is necessary to accept the traffic directly to the Logstash instance, it requires the Logstash instance to be public which increases the risk of the potential security breach. Therefore, making the Logstash instance private increases data security.

Considering the above reasons, we recommend accepting the traffic through a new service in the Central server(s) and routing that traffic to the Logstash instance via an internal network which will inject the data to Elasticsearch as shown in Fig. 3.7.

Fig. 3.7: Data flow from Job pilot to Elasticsearch cluster

3.5 Infrastructure Metrics Integration

The main goal of this research is to introduce infrastructure aware job matching to reduce job failures in Computing Grids. The previous sections described how we can collect, analyze, and store the infrastructure metrics from a Computing Grid in an optimized way to be used in the Job matching process. While the collected data will also be useful in many other aspects like understanding the Grid limitations, Optimizing Grid middleware etc., improving job matching was one of the usages.

At the end of the successful implementation of our Grid monitoring architecture, the Grid administrators will have a complete overview of infrastructure of the each node in the Computing Grid. A small sample of such collected data will look as follows. This is only a highly summarized sample of the actual data and the details of the sample will be explained in detail in section 4.1.3.1 and a complete data sample is available in Annex A.


```
"_source": {
  "@timestamp": "2023-04-03T00:00:00.000Z",
  "host_id": 2083748958,
  "addr": "192.108.46.230",
  "port": 55036,
  "last_updated": 1680480000,
  "ce_name": "FZK",
  "hostname": "c01-011-183.gridka.de",
```

```
  "test_results_json": {
    "home": {
      "HOME": "/tmp/condor_execute/dir_52414"
    },
    "max_namespaces": {
      "MAX_NAMESPACES": 15000
    },
    "ram_info": {
      "RAM_kB_MemTotal": 594138448
    },
    "gcc_version": {
      "GCC_VERSION": "gcc (GCC) 7.3.0"
    },
    "os": {
      "OS_REDHAT_SUPPORT_PRODUCT_VERSION": "7.9",
      "OS_REDHAT_SUPPORT_PRODUCT": "Scientific Linux",
      "OS_PRETTY_NAME": "Scientific Linux 7.9 (Nitrogen)",
      "OS_NAME": "Scientific Linux"
    },
    "cpu_info": {
      "CPU_model_name": "AMD EPYC 7742 64-Core Processor",
      "CPU_vendor_id": "AuthenticAMD",
      "CPU_cpu_cores": 64,
      "CPU_cache_size": "512 KB"
    }
  }
}
```


In summary, the above data sample provides information about CPU, Operating system, RAM and the GCC Compiler installed information in the computing node with hostname "c01-011-183.gridka.de". However, these are only available in the Data analysis framework and this section focuses on identifying the best way to integrate these data into the job matching process.

3.5.1 Job Matching

Job matching is the process of assigning a job created by a user to a computing node for execution. As a Computing Grid consists of a large number of worker nodes with different processing capabilities and configurations, it is important to identify the most suitable node for a given job so that it matches the minimum requirement of the job, while not having additional resources that stay idle in the course of the job execution.

This is a widely discussed topic in the research community and a lot of techniques have been presented that try to make optimal usage of resources as discussed under chapter 2.

However, those researchers focus only on matching jobs to the closest node that satisfies the minimum requirement of the user job. We can identify that most of the research focuses only on a very few infrastructure properties like the number of CPU cores, RAM and storage of a computing node. Given that this is only a couple of parameters, all WMSs like PanDA,JAliEn are sending these parameters as a set of hardcoded key-value pairs from a node. Although these are enough to provide a basic job matching based just on those parameters, we see that a much more intelligent job matching can be done if there is a flexible way to collect any worker node parameters.

The prominent concern we have identified is that although we can try to have optimal resource usage using just those hardcoded parameters, a considerable resource wastage could occur due to a job that tries to execute halfway and fail in the Grid due to multiple reasons:

- Lack of dependencies (eg: Missing libraries)
- Incompatibilities with infrastructure (eg: Some jobs can run only on specific operating systems)
- Maximum resource usage limits in worker node (eg: Exceeding maximum open files on the node could cause the job to get killed by the operating system)

Therefore, we identify that it is important to consider a large number of parameters in the job matching process depending on the use cases of the considered Grid. Given that these parameters are hardcoded and sent from the node, there is no way to achieve this using conventional approaches and we propose a new job matching architecture in section 3.6 section based on the new infrastructure monitoring architecture presented in section 3.4 we presented on this research. Because the monitoring architecture is designed in a highly flexible and extensible way, we can use that to collect any number of infrastructure parameters of interest from the computing node(that can also be changed easily on demand) and use that in the job matching process.

3.5.2 Unlimited Infrastructure Constraints

Each Computing Grid uses its own syntax for defining a job. Users can define jobs in this syntax and submit them to the Grid WMS for execution. The below example shows a simple job definition that requests the execution of a script called “sample-job.sh” in the ALICE computing Grid. The important field to note here is “Requirements”. This field allows the user to impose some constraints on the job like where it should be

Constraint \ Tool	GlideIns	PanDA	JAliEn	Proposed Tool
CPU Cores	✓	✓	✓	✓
Disk space	✓	✓	✓	✓
Memory	✓	✓	✓	✓
GPU	✓	×	×	✓
CPU Architecture	✓	✓	✓	✓
Container Support	×	×	×	✓
Software versions	×	×	×	✓

TABLE 3.1: Constraints supported by different WMSs

executed, how many cores it requires etc. At the moment this is very limited set of parameters as discussed in section 3.5.1.

```
Executable = "/alice/user/k/kwijethu/sample-job.sh";

Requirements = other.CE == "ALICE::RAL::LCG";
Jobtag = {
   "Site:ALICE::RAL::LCG"
};
OutputFile = {
  "std*", "*log"
};
OutputDir = "/alice/user/k/kwijethu/sample_job";
TTL="1200";
OutputErrorE = {"std*@disk=1"};
```

For example, Table 3.1 shows constraints that are supported by different Grid middleware in comparison to the proposed tool. It is important to note that the proposed tool will allow any infrastructure constraint in addition to the summarized list due to its flexible job matching architecture presented in section 3.6.

This set of constraints very rarely changes in a Grid middleware because introducing a new constraint requires a major change to the middleware. The process of introducing a new such constraint to ALICE Computing Grid and the time taken for each step are listed below.

1. Update the central server code to accept values for the new parameter

2. Test the functionality of the updated code in a central server (Duration: ~1 day)

3. Deploy and test the update code across the central server farm after testing (Duration: ~3 days)

4. Update the pilot job code to collect the values from the computing node before starting job execution

5. Deploy the new pilot job code in a few sites and test (Duration: 1-3 days)

6. Rollout the new pilot job code across all Grid sites (Duration: 1-2 weeks)

Other Computing Grids also follow the same process with different durations depending upon the amount of testing required. As it is evident from the list, this is a major change that takes a couple of weeks to be deployed in production even though it provides functionality similar to other parameters that are already collected. The reason for this is that Grid middleware runs 24x7 on a large number of computers and a single breaking change could cause thousands of jobs to fail and weeks of work to fix, test, and deploy a new release. Hence Grid middleware (especially central server code) is changed very rarely and updating constraint lists is not possible. The proposed Job Matching Architecture in section 3.6 provides a generalized way to define constraints so that the system is flexible add, remove, or update constraints at any time. That ultimately provides the capability to introduce unlimited infrastructure constraints to jobs that can be used to reduce job failures in the Grid due to infrastructure limitations.

3.6 Proposed Job Matching Architecture

Fig. 3.8 shows an overview of the proposed job matching architecture. In the proposed architecture, we suggest to introduce another web service in the data collection component called "Constraint Generation Service" to generate the constraints. Initially, the pilot job will run the infrastructure monitoring probes, collect the results, and report them to the data collection service as explained previously. This will provide us with a massive amount of information regarding the current worker node. The constraint generation service can make use of this data to identify what values would be of interest to the job matching process. Making a GET call to the constraint generation service from a worker node will return a set of key-value pairs in the form of (constraint name: value relevant to that constraint in the node that made the GET request). To achieve this, the constraint generation service queries the collected data and filters out only the interesting key-value pairs when it receives a request from a worker node. This service can be used to add, remove, or update new constraints that are to be used in job matching process.

Once the worker node receives the constraint pairs, it can update the job matching request to include additional constraints. The Job matcher will then use these additional constraints to match the job to the most suitable node considering not only a few

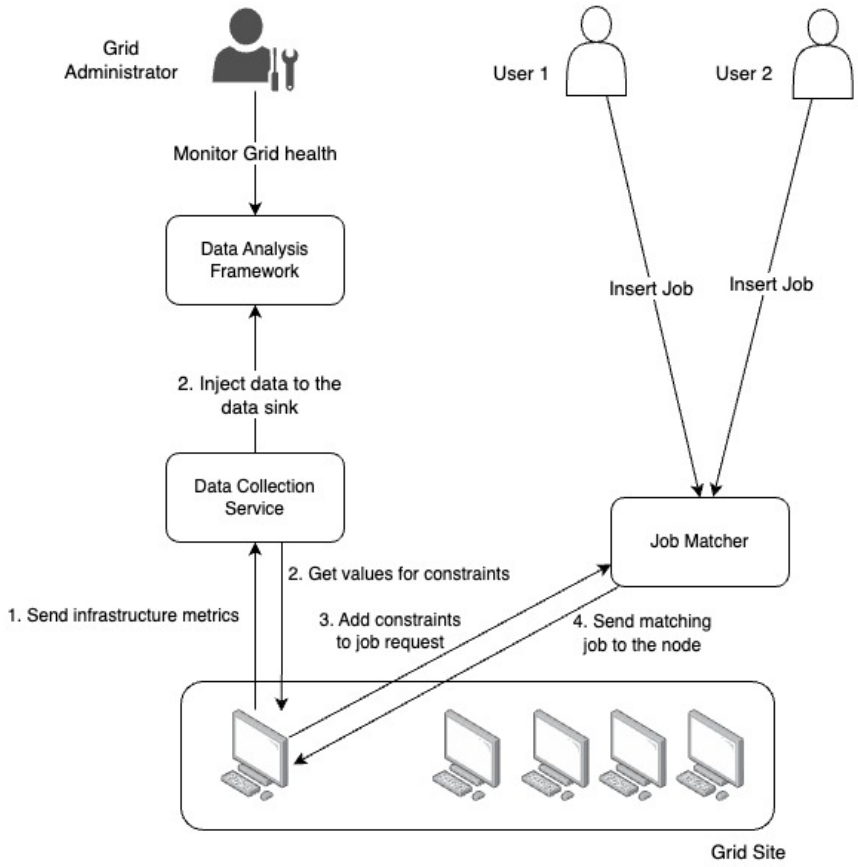

Fig. 3.8: Proposed Job Matching Architecture

parameters like CPU cores and storage but also many other parameters like Operating system, cache size, container support etc. Job matcher code would have to be improved just once to include additional constraints in the job matching process. The complete flow of this proposed architecture is shown in Fig. 3.9

It is important to note that the additional constraints are added to the job matching request with zero changes or code additions/removals to the Grid middleware. Therefore, unlike adding a new constraint using the previous approach which takes a couple of weeks, we can add a new constraint with the proposed architecture within hours. The complete process for searching for a new infrastructure parameter and adding a new constraint based on that under the new job matching process is described below.

1. Add the new monitoring probe to the distributed file system (1 hour)

2. Add the new constraint filtering code to the constraint generation service (1 - 15 min: Time taken to update the server code)

With just these 2 steps, users will be able to use the new constraint in their job matching process to impose additional requirements on how the job should be matched.

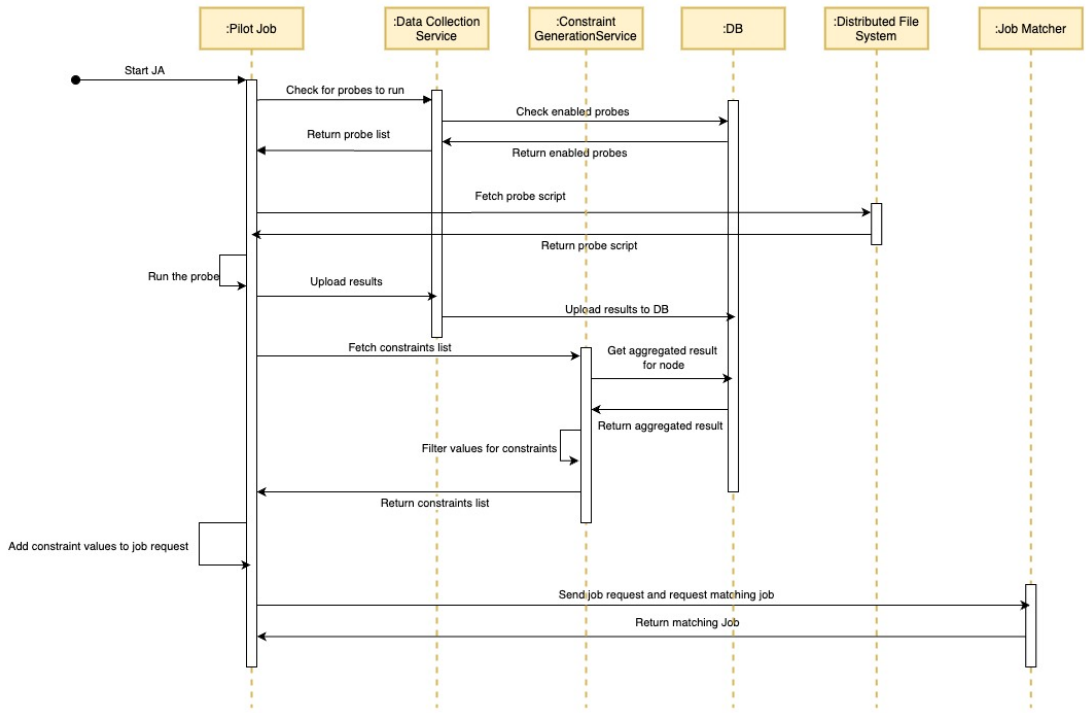

Fig. 3.9: Proposed Architecture Flow Diagram

With the proposed architecture we can define any number of constraints and use that in the job matching process within a few hours with zero code changes to the Grid middleware.

CHAPTER 4

IMPLEMENTATION

In the Methodology (Section 3) we have discussed the concerns associated and design decisions that led to the development of a new Grid Infrastructure monitoring architecture to monitor distributed Computing Grids and a new Job Matching architecture to account for infrastructure constraints in Grid job matching. The implementation and evaluation of the proposed architecture was carried out in the ALICE Computing Grid at CERN which contains more than 7,000 worker nodes distributed globally across 70 Grid sites. This chapter describes the complete implementation of the new tool called "Site Sonar v2.0" which is developed using the proposed architecture, that provides enhanced Infrastructure monitoring capabilities to ALICE Computing Grid while providing improved job matching architecture to its Grid middleware, Java ALICE Environment (JAliEn) .

4.1 Site Sonar Architecture

Site Sonar v2.0 (here onwards referred to as "Site Sonar") is a new Grid infrastructure monitoring tool developed to monitor the ALICE Grid. Site Sonar collects information about infrastructure and configuration of the worker nodes in ALICE Grid, aggregates and visualizes the results in monitoring dashboards with flexible filtering options. Site Sonar is set apart from the existing systems mainly by its high flexibility and extensibility. Site Sonar allows collecting any type of information from the worker nodes, from text strings up to complex JSON objects, and provides the ability to change or add any monitoring probes without having to make any changes to the Site Sonar framework. Additionally, it facilitates creating no-code visualization of the collected data with the help of the ELK stack. The architecture of Site Sonar is shown in Fig. 4.1 which has been explained in detail in the following sections.

4.1.1 Probe

The basic unit of Site Sonar is called a "Probe". A probe is a program that queries the worker node it runs on for some specific information and outputs the result as a JSON object. In practice, each probe is a shell script. Following is one such simple data collection probe deployed in the ALICE Grid.

```
#!/bin/bash

# Print the OS description
# and return the exit code of lsb_release
```

```
# Tested on lxplus (CentOS 7) and Ubuntu 20.04

LSB_RELEASE=$(lsb_release -s -d | cut -d\" -f2 | xargs)

if [[ $? == 0 ]]
then
    echo "{ \"LSB_RELEASE\" : \"$LSB_RELEASE\" }"
else
    exit_code=$?
    echo "{ \"LSB_RELEASE\" : \"\" }"
    exit $exit_code
fi
```

This probe is responsible for collecting the operating system information of the worker node and the output of the probe will look as follows.

```
{ "LSB_RELEASE" : "CentOS Linux release 7.9.2009 (Core)" }
```

Site Sonar consists of 30+ such probes as shown in Fig. 4.2 with more probes being added daily without any code changes to the Grid middleware. To add a new probe, it is simply required to copy the script to '/cvmfs/alice.cern.ch/sitesonar' directory in

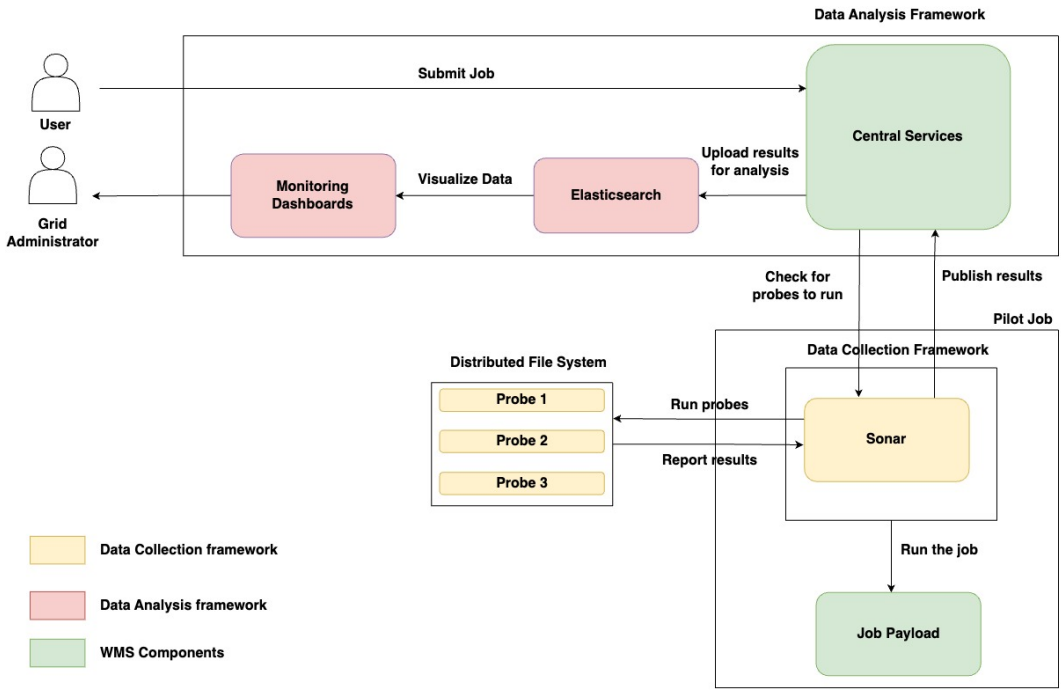

Fig. 4.1: Site Sonar v2.0 Architecture

CernVM File System (CVMFS) . An additional configuration is added in Site Sonar v2.0 implementation where each probe can be configured with a TTL after which its results are considered obsolete and need to be collected again. This is done to provide the ability to collect different metrics at different intervals allowing more granular control over how the data should be collected. Usually, the time interval is set to 24 hours because the infrastructure metrics do not change that often, reducing the performance impact on the Job Agent in JAliEn which is the equivalent of a pilot job in a Grid.

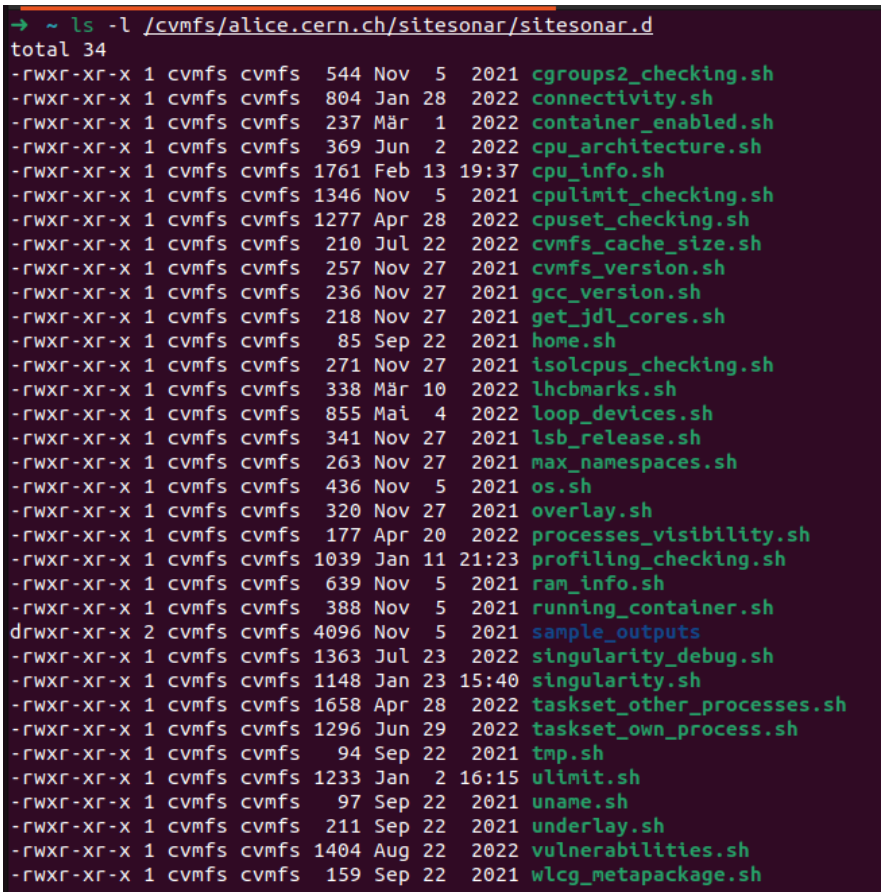

Fig. 4.2: List of Site Sonar probes

4.1.2 Sonar

The “Sonar” is the main component used for data collection in Site Sonar. Sonar is a controller function that is written in Java which controls the data collection and reporting process of the worker node. When an empty slot opens up on a worker node, JAliEn submits a pilot job which invokes a process called “Job Runner” in that slot. Depending on the number of cores in that computing node and the cores required by the jobs waiting for execution, Job Runner spins up multiple “Job Agents” on the node.

Each “Job Agent” is responsible for preparing the environment for executing job(s) on the node. When a Job Agent starts, it will contact the central services to check if there are waiting jobs and execute if any. We have identified this as the most suitable integration point in JAliEn to integrate the Sonar component in Site Sonar v2.0. This is shown in Fig. 4.3

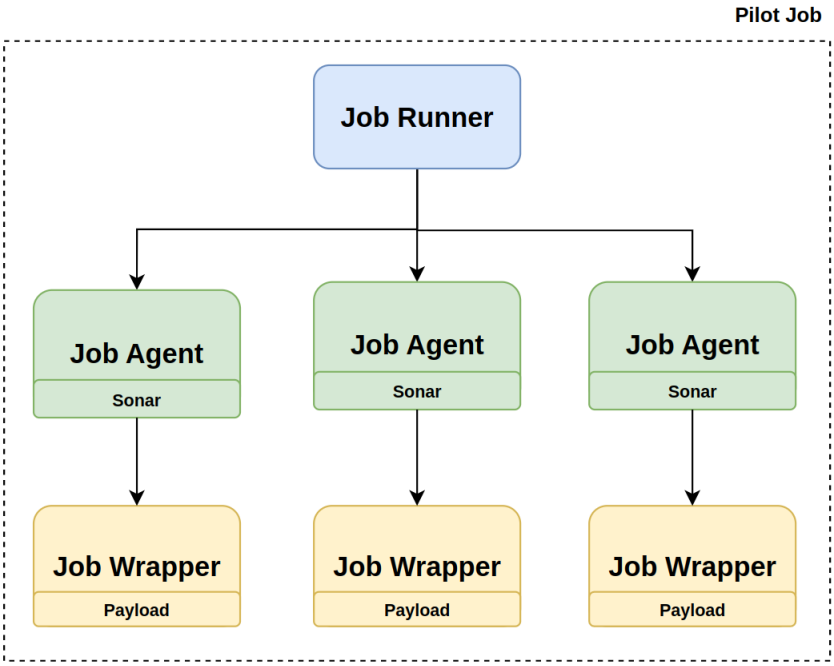

Fig. 4.3: Site Sonar Integration with JAliEn

Before a Job Agent starts the execution of the payload, it runs the Sonar which calls the Central Services to identify which probes need to be run on the current node. This is done to avoid sending results from probes that already had results reported for the given worker node within the TTL values of those probes. The probes whose results are absent or outdated are fetched from CVMFS, executed, and the newly obtained infrastructure metrics of that node are then reported back to the Central Services by the Sonar.

More than 300,000 such metrics are reported every day to the Central Services by Sonars running across the ALICE Grid. This data is used for monitoring and providing improved job matching to worker nodes.

After execution of the Sonar, Job Agent moves on to create the necessary environment for the Job and ultimately spins up a “Job Wrapper” which is responsible for executing the actual job payload.

4.1.3 Central Services

In JAliEn, web services are hosted and handled by a set of services in a central server farm referred as "Central Services". The Site Sonar data collection service and constraint generation service are hosted as a part of Central Services in the form of Java Server Pages (JSP) . These are 2 basic services that are capable of handling incoming Sonar requests and can be easily written in any other language as well.

4.1.3.1 Data Collection Service

The Data Collection Service act as the controller for data inflow from Sonar. It controls which probes should run, what their TTLs should be, and which parts of the collected data should be visible to the users. The data collection service hosts 2 endpoints.

```
GET https://alimonitor.cern.ch/sitesonar/queryProbes.jsp?
hostname=<name-of-the-host>&ce_name=<site-name>

Response:
{
  cgroups2_checking,
  connectivity,
  container_enabled ...
}
```

The initial "queryProbes" endpoint is used to decide which probes should run on the worker node. When the Sonar calls the queryProbes endpoint with the hostname of the node and CE Name(Name of the Grid Site), the service checks which probes have not been run in this node for some time (by checking the TTL of its previous results) and returns the names of the probes that should run in that worker node.

The Sonar will then fetch each script from CVMFS, execute it on the worker node, and upload the results to the "uploadResults" endpoint in the Data Collection Service as shown below.

```
POST https://alimonitor.cern.ch/sitesonar/uploadResults.jsp
{
  hostname: <name-of-the-host>
  ce_name: <site-name>
  test_name: "lsb-release"
  test_code: 0
  test_message: "CentOS Linux release 7.9.2009 (Core)"
}
```

Central Services stores the data reported by Sonars directly in a database for long-term persistence. We run a daily job that aggregates the collected metrics per node and prepares JSON documents of metrics reported per node. These JSON documents are then uploaded to Elasticsearch through Logstash.

Although the data can be routed directly to the analysis framework by injecting to Logstash as we have presented in our architecture in section 3.4, this extra step is taken considering deployment concerns like reducing traffic on Logstash instance and longer duration for data persistence. A sample of collected data for a node is shown below and a complete data object for a node is attached in Annex A


```
"_source": {
  "@timestamp": "2023-04-03T00:00:00.000Z",
  "host_id": 2083748958,
  "addr": "192.108.46.230",
  "port": 55036,
  "last_updated": 1680480000,
  "ce_name": "FZK",
  "hostname": "c01-011-183.gridka.de",
  "test_results_json": {
    "home": {
      "HOME": "/tmp/condor_execute/dir_52414"
      "EXITCODE": 0,
      "EXEC_TIME": 0
    },
    "cpu_info": {
      "CPU_model_name": "AMD EPYC 7742 64-Core Processor",
      "CPU_vendor_id": "AuthenticAMD",
      "CPU_cpu_cores": 64,
      "CPU_cache_size": "512 KB",
      "EXITCODE": 0,
      "EXEC_TIME": 2
    }
  }
}
```


The data is structured in the following format where the node metadata contains important information like hostname, ip address, site name etc. of the node. The "test_result_json" section includes the results of all monitoring probes. Each probe returns a JSON document including the parameters, exit code of the probe, and execution time of the probe. A collection of such documents representing each node is uploaded to Logstash every day.

```
{
  node_metadata,
  test_results_json : {
    test1 : {
      parameter1: value1,
      parameter2: value2,
      EXITCODE: <code>,
      EXEC_TIME: <execution-time>
    },
    test2 {
      ...
    }
  }
}
```

4.1.3.2 Constraint Generation Service

Constraints Generation Service is used to populate key-value pairs that contain the value for a given constraint for a computing node. Constraint Generation Service hosts a single endpoint that generates the relevant key-value pairs for a given hostname. After the results from the monitoring probes are executed, Sonar makes a call to the Constraint Generation endpoint which returns the constraint value pairs. This conforms to the design proposed in Fig. 3.9.

```
GET https://alimonitor.cern.ch/sitesonar/constraints.jsp?
hostname=<name-of-the-host>&ce_name=<site-name>

Response:
{
  "container_support" : true,
  "cpu_cores" : 64,
  "OS" : "CentOS 9",
  "CGROUPSv2_AVAILABLE" : true,
  "containsAVX" : 1
  ....
}
```

In JAliEn, a job matching request to the Job broker is accompanied by a JSON object called "SiteMap" which provides some basic information about the node like CPU cores, disk space, memory etc. The following section shows a real SiteMap extracted from the ALICE Computing Grid before the Site Sonar integration.

```
INFO: {Site=CERN, CE=ALICE::CERN::Yerevan,
Platform=Linux-x86_64, Host=voboxalice3.cern.ch,
TTL=87000, alienCm=voboxalice3.cern.ch:10000,
workdir=/pool/condor/dir_97385,
Localhost=b9s14p2116.cern.ch, CEhost=voboxalice3.cern.ch,
Disk=3208648912, CPUCores=1, CVMFS=1}
```

With the introduction of Site Sonar integration, all the constraint values returned from the Constraint Generation service are injected into the SiteMap. This allows JAliEn to collect any information from a worker node and include them in the job matching request to be used in the job matching process at the central services. The following sections show a real SiteMap of the same computing node after the Site Sonar integration.

```
INFO: {Site=CERN, CE=ALICE::CERN::Yerevan,
Platform=Linux-x86_64, Host=voboxalice3.cern.ch,
TTL=87000, alienCm=voboxalice3.cern.ch:10000,
workdir=/pool/condor/dir_97385,
Localhost=b9s14p2116.cern.ch, CEhost=voboxalice3.cern.ch,
Disk=3208648912, CPUCores=1, CVMFS=1,
CGROUPSv2_AVAILABLE=true, containsAVX=1}
```

Note that two new fields named “CGROUPSv2_AVAILABLE” and “containsAVX” have been added at the end which depicts the availability of Control Groups v2.0 which is used for resource isolation and the availability of AVX CPU Instruction set in the worker node respectively. These fields can be used in the job matching process after that. For example the ALICE Hyperloop jobs require the AVX instruction set as discussed in the section 2.2.1 to execute, and the updated SiteMap fields returned from Site Sonar are to be used in the future to ensure that the Hyperloop jobs are matched only to nodes with AVX instruction set. Any other feature of the worker node can also be added to the job matching process easily following the same approach as discussed in 3.5.2

4.2 Improved Job Broker

“Job Broker” is the JAliEn component that is responsible for handling the process of matching jobs to the most suitable node. It accepts Job matching requests that are sent by Job Agents running on different nodes and provides the most suitable job that matches the node. As explained in section 4.1.3.2, the job matching request will be accompanied by a SiteMap which contains information regarding the node. The SiteMap

contains a set hardcoded information collected from the worker node and the Job Broker provides some hard coded methods to do the job matching which considerably reduces the flexibility of the system and the ability to define a new constraint in JAliEn as explained in section 3.5.2

```
final ArrayList<Object> bindValues = new ArrayList<>();

if (matchRequest.containsKey("TTL")) {
    where += "and ttl < ? ";
    bindValues.add(matchRequest.get("TTL"));
}

if (matchRequest.containsKey("Disk")) {
    where += "and disk < ? ";
    bindValues.add(matchRequest.get("Disk"));
}

// Checks that if whole node scheduling, no constraint on CPUCores is added from the CE
if (matchRequest.containsKey("CPUCores") && Integer.parseInt(matchRequest.get("CPUCores").toString()) != 0) {
    where += "and cpucores <= ? ";
    bindValues.add(matchRequest.get("CPUCores"));
}

if (matchRequest.containsKey("Site")) {
    where += "and (site='' or site like concat('%,',?,',%')";
    bindValues.add(matchRequest.get("Site"));
}

if (matchRequest.containsKey("Extrasites")) {
    final ArrayList<String> extrasites = (ArrayList<String>) matchRequest.get("Extrasites");
    for (final String site : extrasites) {
        where += " or site like concat('%,',?,',%') ";
        bindValues.add(site);
    }
    where += ") ";
}
else if (matchRequest.containsKey("Site"))
    where += ") ";

// skipping extrasites: used ?

if (!matchRequest.containsKey("CVMFS"))
    if (matchRequest.containsKey("InstalledPackages")) {
        where += "and ? like packages ";
        bindValues.add(matchRequest.get("InstalledPackages"));
    }
    else {
        where += "and ? like packages ";
        bindValues.add(matchRequest.get("Packages"));
    }
```

Fig. 4.4: JAliEn Job Broker Constraint Matching Logic

Fig. 4.4 shows the existing logic that is being used by JAliEn to evaluate the parameters sent from the Job Agent in the job matching process. A new code section is added for each parameter leading to hardcoded set of constraints.

The current approach to adding new constraints is not flexible as they are hardcoded and it cannot be easily extended to add new constraints to the job matching process. Each new parameter requires adding a new code block on the job broker which is at the core of the system that could lead to critical system failures if not tested properly. This also requires rolling out a new software update across central servers to include the new code. This issue is explained in detail in section 3.5.2. To address this, we have improved the JAliEn Job Broker to be highly flexible to accept any number of parameters allowing to create unlimited constraints as shown in Fig. 4.5.

```
// Add Site Sonar Constraints
// Initialize or refresh constraint cache
HashMap<String, String> constraintCache = TaskQueueUtils.getConstraintCache();

if (constraintCache != null && constraintCache.size() > 0) {
    // Constraint name is the constraint key
    for ( Map.Entry<String, String> entry : constraintCache.entrySet()) {
        String constraintName = entry.getKey();
        String constraintType = entry.getValue();

        // If the site map has a value for the constraint, add the constraint check
        if (matchRequest.containsKey(constraintName)) {
            Object constraintValue = matchRequest.get(constraintName);
            logger.log(Level.FINE, msg: "Constraint value : " + constraintValue);
            if (constraintValue != null) {
                if (constraintType.equals("equality")) {
                    where += " and (( ? = " + constraintName + ") or (" + constraintName + " is null))";
                } else if (constraintType.equals("regex")) {
                    where += " and (( ? LIKE " + constraintName + ") or (" + constraintName + " is null))";
                } else {
                    // stop matching because the constraint type is not supported
                    logger.log(Level.SEVERE, msg: "Incorrect expression type provided: " + constraintType);
                    return matchAnswer;
                }
                bindValues.add(constraintValue);
            }
        } else {
            logger.log(Level.FINE, msg: "Site map does not contain a value for : " + constraintName +
                    ". Setting the constraint value to null");
            where += " and (" + constraintName + " is null)";
        }
    }
}
```

Fig. 4.5: Updated Job Broker Constraint Matching Logic

The improved logic allows any number of constraints to be added in the job matching process without any change to the code avoiding the introduction of bugs and the need to do new releases with complex deployment rollouts. The new constraints can be easily defined by adding the entry to the database and the logic in Fig. 4.5 will fetch the constraints from the database and impose them using the collected parameters.

4.2.1 Site Sonar ELK Stack

ELK is a collection of three widely used open-source log analysis products - Elasticsearch, Logstash and Kibana[53]. It is widely used for analyzing and visualizing large collections of data to provide insights of interest.

Elasticsearch is used for storing large amounts of documents and indexing those documents for fast and efficient querying. Logstash is used for processing and transferring data. Kibana is used as a flexible and user-friendly data visualization tool.

Due to the reasons discussed in 3.2.3 section and since the ALICE team already has an ELK cluster that is being used for other projects, we have decided to create the

visualization framework with the use of an ELK stack.

Site Sonar v2.0 injects more than 7,000 documents into the Site Sonar ELK stack every day, with each document containing more than 30 metrics collected on a worker node in the ALICE Grid. The data is visualized through preconfigured Kibana dashboards for the scrutiny of ALICE Grid team.

Using an ELK stack in Site Sonar provides the following advantages:

- Fast response times for queries running over large amounts of data.
- A query language with complex functionality that is easy to use.
- Availability of complex analysis and visualization features out of the box.
- Flexibility for the user to create desired visualizations without writing code.
- No need to maintain an in-house data visualization system.

4.2.1.1 Logstash

The Logstash instance is set up using Docker in one of the central machines. It exposes an endpoint to accept Site Sonar Central services and inject them to Elasticsearch for indexing. The following pipeline configuration is used to set up the instance.

```
input {
    tcp {
        port => 5402
    }
}

filter {
    # this is done in order for ES to interpret the
    # message as a JSON object
    json {
        source => "message"
    }

    # this is used to import logs in retrospective
    # (the @timestamp field will be set to the
    # date the log was created)
    date {
        match => ["last_updated", "UNIX"]
    }
```

```
    mutate {
        remove_field => [ "message" ],
    }
}

output {
    elasticsearch {
        hosts => [ "https://es-alicecs1.cern.ch:443/es/" ]
        index => "sitesonar-%{+YYYY.MM.dd}"
        user => "logstash"
        password => "${ES_LOGSTASH_PASSWORD}"
        ssl => "true"
        cacert => "/config-dir/ca/cern_grid_cert.pem"
    }
}
```

We use Logstash to convert the JSON documents to an Elasticsearch index. In addition to that, data transformations like deriving new fields by combining multiple existing fields, mutating existing fields etc. are done at the Logstash level before being indexed at Elasticsearch.

4.2.1.2 Elasticsearch

A standard Elasticsearch cluster is used in Site Sonar to store the data. However, an additional step is taken to index the data in a more meaningful way rather than storing the plain data so that it can be used in visualizations. A component template that describes how each field should be indexed in Site Sonar is defined in Elasticsearch for this. A sample component template is shown below and the complete template is available in Annex B.

```
"properties" : {
    "cpu_model" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "last_updated" : {
```

```
      "format" : "epoch_second",
      "type" : "date"
    },
    "cpu_cores" : {
      "type" : "integer"
    }
}
```

Component templates allow indexing each key of the data map with its own data type allowing the much better performance and much complex analysis. For example, it allows storing cpu model as a string, and cpu cores as a number etc. This allows us to provide different analysis and filtering options based on the data type, like range filters and histograms for cpu core counts, date ranges for 'last_updated' field etc. An SQL database would restrict us from storing all these different data values as strings(because we don't add a column per each key with the correct data type) which would lead to very rigid schema with only basic functionalities.

4.2.1.3 Kibana

Kibana is used to create visualizations based on the collected data. Site Sonar injects nearly 7,000 documents daily, containing over 300,000 key-value pairs to Elasticsearch every day as shown in Fig. 4.6 and it is nearly impossible to analyze them manually. Hence, this is one of the 2 most important components for a Grid administrator because it allows aggreagating and visualizing the collected data with many capabilities like filtering and sorting. Additionally, Kibana allows creating data visualization without having to write any code that eliminates the requirement to create releases or to update the production deployments.

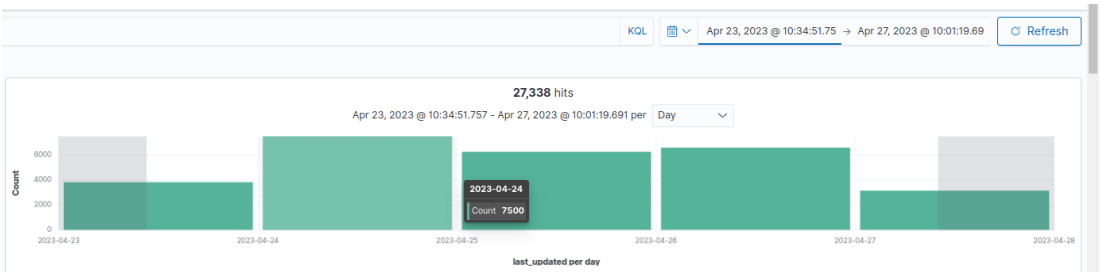

Fig. 4.6: No. of documents injected per day

Depending on the requirements of the ALICE Grid team, we have created a number of visualizations and combined them to create dashboards that are used to monitor the status of the Grid which are explained in the following chapter.

4.2.2 Summary

In summary, Site Sonar consists of 2 main components: The data collection framework and the Data analysis framework. The data collection framework runs inside the Job Agent which is responsible for executing the job on the worker node. Before the Job Agent runs the payload, it calls the Sonar which checks the probes that are needed to run on the given node and invokes the probes one by one. The resulting output is sent to Central Services by the Sonar for long term persistence.

Data analysis framework is a separate component that is built based on an ELK stack that ingests the data reported from the Data collection framework, and allows users to create analysis and visualization on that data without the need to write code.

These 2 components are used to improve the functionality of JAliEn Job Broker providing infrastructure aware job matching capabilities to the ALICE Computing Grid.

CHAPTER 5

RESULTS

This chapter focuses on analysing the results obtained in the course of this research and evaluating them to identify the success of this study. This chapter is divided into 3 sections. Section 5.1 analyzes the information collected throughout the study, presents them, and describes the importance of this information. Section 5.2 describes the findings of this research based on the collected data that resulted in discovering important flaws in the ALICE Grid. Section 5.3 presents the evaluation of the tool in comparison to its predecessor Site Sonar v1.0 and related tools and discusses the results to conclude that the study has achieved the initial outcomes that were set out as research outcomes

5.1 Analysis

Currently Site Sonar monitors a large number of infrastructure metrics in ALICE Computing Grid starting from basic information like the operating system up to more complex information like benchmarks or in which ways containers are supported. It has the capability to monitor any parameter of the worker node: both software and hardware, as long as it can be read by a user with non-root access. The collected monitoring information is used to develop visualizations that provide useful information to Grid Administrators about the infrastructure of the Grid. Few of those dashboards are presented below.

5.1.1 Operating System Distribution

The development of Grid software often involve having a good understanding of the features offered by the Operating Systems as they can be used to provide useful features like job isolation, job optimization, resource limitation etc.

For example, Control Groups (CGroups) are a famous tool for implementing resource isolation in Linux environments[54]. However, implementing resource limitation solutions for Grid jobs using CGroups required the base worker nodes to have compatible operating systems that support CGroups v2. Site Sonar analysis and visualizations were directly used in this study to check the Operating Systems across the Grid and conclude that the Grid is not ready to have CGroups v2.0-based features yet.

Fig.5.1 shows one of the monitoring dashboards of Site Sonar presenting the operating system version of worker nodes in the ALICE Grid as of 2023-04-21. The top charts of the dashboard show meta information regarding the data collection. The first chart on the first row shows how many unique Grid sites have reported data in total

within the given period of time and the chart next to it shows how many unique worker nodes have reported the data in the same period. The right side chart on the second row shows the distribution of worker nodes across the Grid. Until the origin of Site Sonar, there was no way to know how many nodes the actual Grid contained and how big each site was. This chart provides this information visualizing the scale of each Grid site. All the dashboards follow a similar structure with the ability to customize the view and add new visualizations on demand without any code changes. Dashboards can be used to easily filter out data by sites and/or attributes and create visualizations as well as summaries.

The left side chart on the second row lists all the unique operating systems that are available in the Grid nodes. The actual distribution of the Operating system is depicted in the pie chart in the third row. This is important to get an idea about the latest trends related to operating system usage and site wide transitions from one operating system to another.

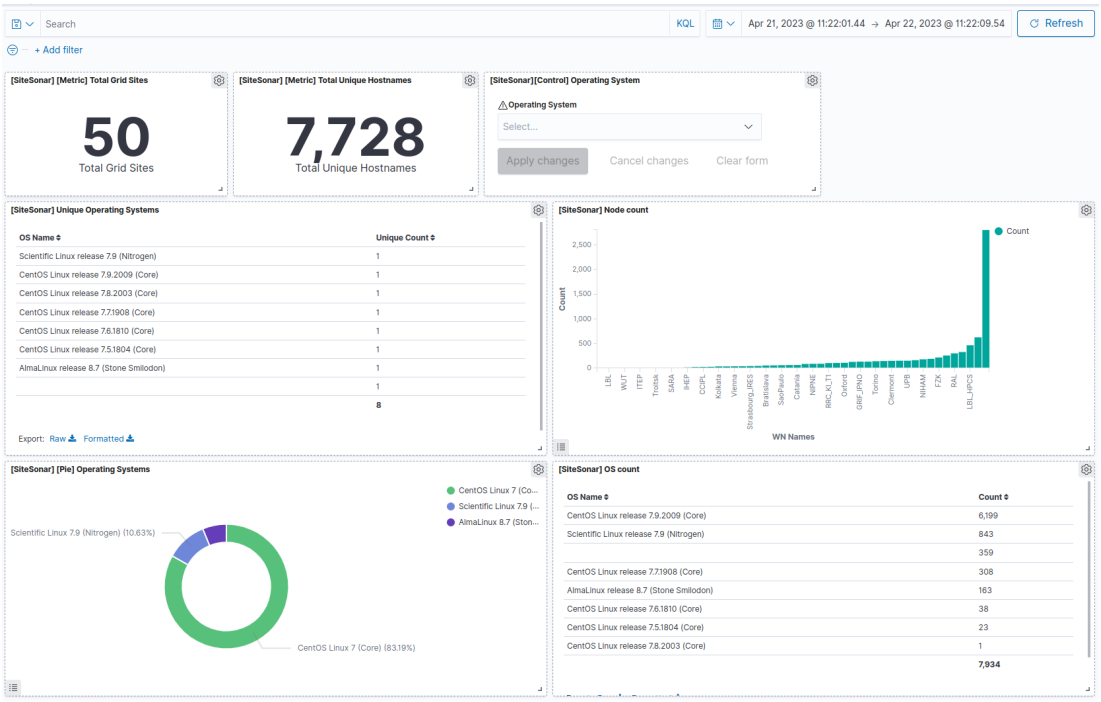

Fig. 5.1: Operating System Monitoring Dashboard

The most widely used operating systems and their versions in the ALICE Grid at two different points of time that are about 8 months apart as reported by Site Sonar are summarized in Table 5.1 and Table 5.2. We can clearly see the movement of Grid sites away from CentOS to other alternatives like AlmaLinux following the sudden discontinuation of the widely used CentOS by Red Hat Linux in 2020. It is important for a Grid administrators to have an idea about this kind of movement as the Grid middleware often uses features offered by operating systems and this helps them to

plan ahead and avoid issues that arise due to changes in operating systems.

Operating System	coverage%
CentOS Linux release 7.9.2009 (Core)	90.74%
Scientific Linux release 7.9 (Nitrogen)	6.78%
CentOS Linux release 7.7.1908 (Core)	1.13%

TABLE 5.1: Operating system distribution of ALICE Grid as of 2022-08-30

Operating System	coverage%
CentOS Linux release 7.9.2009 (Core)	83.18%
Scientific Linux release 7.9 (Nitrogen)	10.63%
AlmaLinux 8.7	6.19%

TABLE 5.2: Operating system distribution of ALICE Grid as of 2023-04-22

5.1.2 Singularity Support

All the latest Grid middleware aims to run jobs in containerized environments like Singularity / Apptainer[1] to provide job isolation along with a uniform environment for the jobs to run in. The new JAliEn[49] Grid middleware also had the requirement to adapt the standard approach of running jobs in containers to serve this purpose. However, there was no way to check if the sites(worker nodes in sites to be exact) possessed the necessary infrastructure to support the use of Singularity. One of the initial and critical usages of Site Sonar was to check which sites do not support Singularity on their worker nodes. This was tested using a probe that attempts to start a Singularity container in the two ways that are supported by JAliEn[50]: 1) via a local, privileged installation; 2) via the unprivileged installation provided on CVMFS by the ALICE Grid team. On a number of sites, neither method worked. Contacting the respective site admins led to Singularity being supported at almost all sites.

Fig. 5.2 extracted from Site Sonar's "Singularity Support Dashboard" presents the percentage of nodes supporting Singularity across the Grid. The percentage of Singularity support either through local installation or CVMFS stood at 96.6% as of 2023-04-23.

Given the critical importance of this feature, it was important to identify the sites that do not offer singularity support at the moment. Fig. 5.3 shows the information about these sites and nodes which helped the ALICE Grid team to easily communicate with the site admins to configure the necessary parameters required for the proper singularity functionality.

[1] "Singularity" project was renamed as "Apptainer" by the relevant organizations during the course of this research. Therefore the 2 words are used interchangeably.

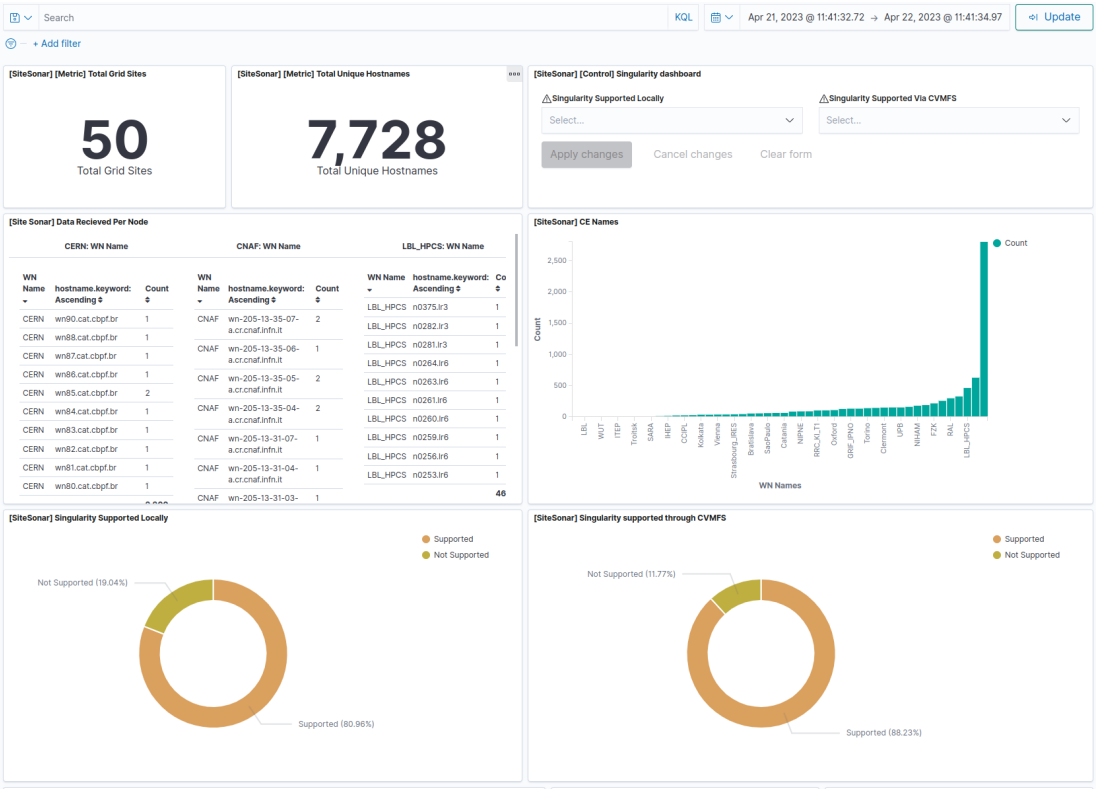

Fig. 5.2: Singularity Support Dashboard of ALICE Grid on 2023-04-21

5.1.3 Grid Overview

Fig. 5.4 shows the main dashboard of Site Sonar that provides an overview of the infrastructure of the whole Grid in one page. Other than the usual metadata that is available in all dashboards, this dashboard contains the most important information from each dashboard. The left side chart on the second row shows the Operating System distribution of the ALICE Grid that has been discussed in detail in the 5.1.1 section. The right side chart of the same row shows the number of cores available in each worker node. This is a piece of vital information to the ALICE Grid administrators as JAliEn is rolling out multi-core support and 8-core queues and this dashboard can be used to see how many worker nodes can support it. The charts on the third row show the different CPU models available in the ALICE Grid and their CPU vendor. A separate study is already on the way to analyze the impact of different CPU models on the job performance in the Grid.

5.2 Findings

Since its introduction, Site Sonar has been heavily used by the ALICE Grid administrators to understand the structure of the ALICE computing Grid. So far, the use of Site Sonar has led to multiple interesting discoveries about the ALICE Grid. The following

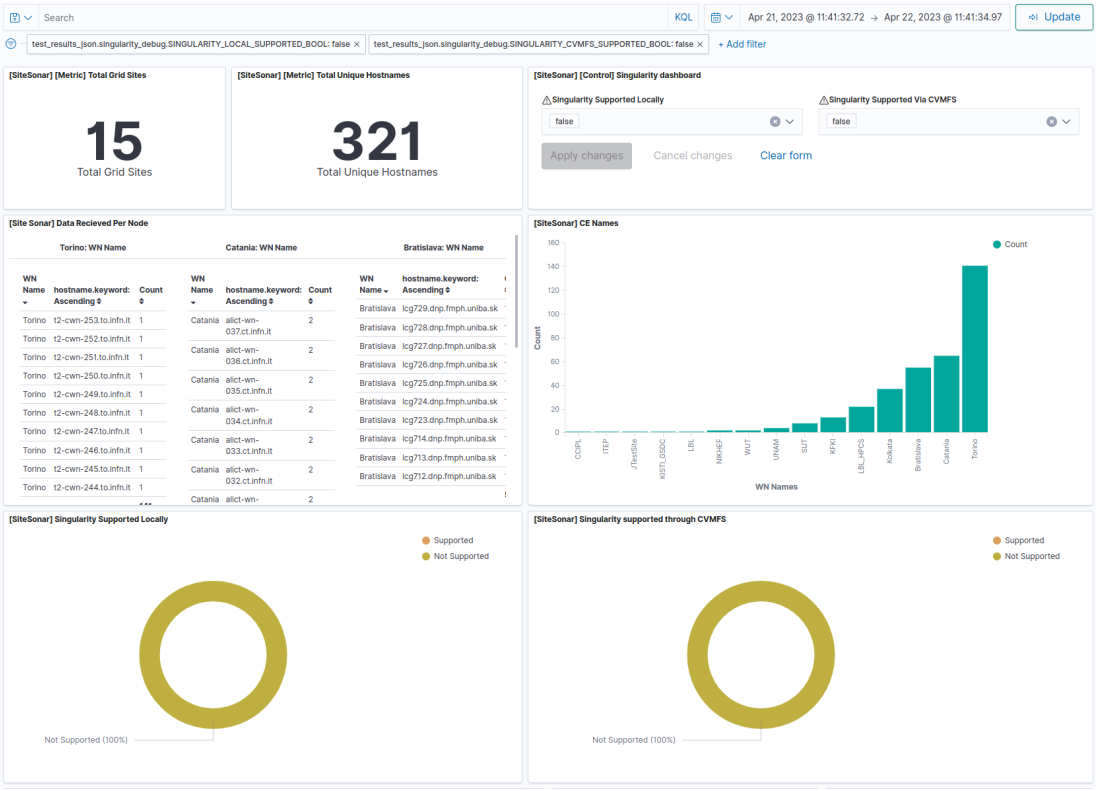

Fig. 5.3: Sites not supporting Singularity in ALICE Grid on 2023-04-21

section describes a few such discoveries.

5.2.1 Sites running CentOS 6

CentOS 6 was the most widely used to operating system across worker Grid worker nodes until recently. CentOS 6 does not by default come with sufficient support for running Singularity. As explained in section 5.1.2, this imposed a major limitation in rolling out the containerized job feature for ALICE Grid. Hence, when checking whether the ALICE Grid is ready for containerized jobs, one of the essential requirements was to check if all sites are running CentOS 7 or higher.

While the relevant packages can be installed and configured by talking to local Grid administrators, communicating with around 70 Grid sites first to check their OS version and then co-ordinating with them to do the upgrades is a daunting task. With Site Sonar, we easily narrowed down which sites are running CentOS 6 and their configurations that allowed the ALICE Grid administrators to easily get them upgraded to CentOS 6 or higher versions. At the end of this effort, the Grid admins were able to achieve 100% singularity support across the Grid as shown in Fig. 5.1 allowing the JAliEn to introduce the much needed feature of containerized jobs.

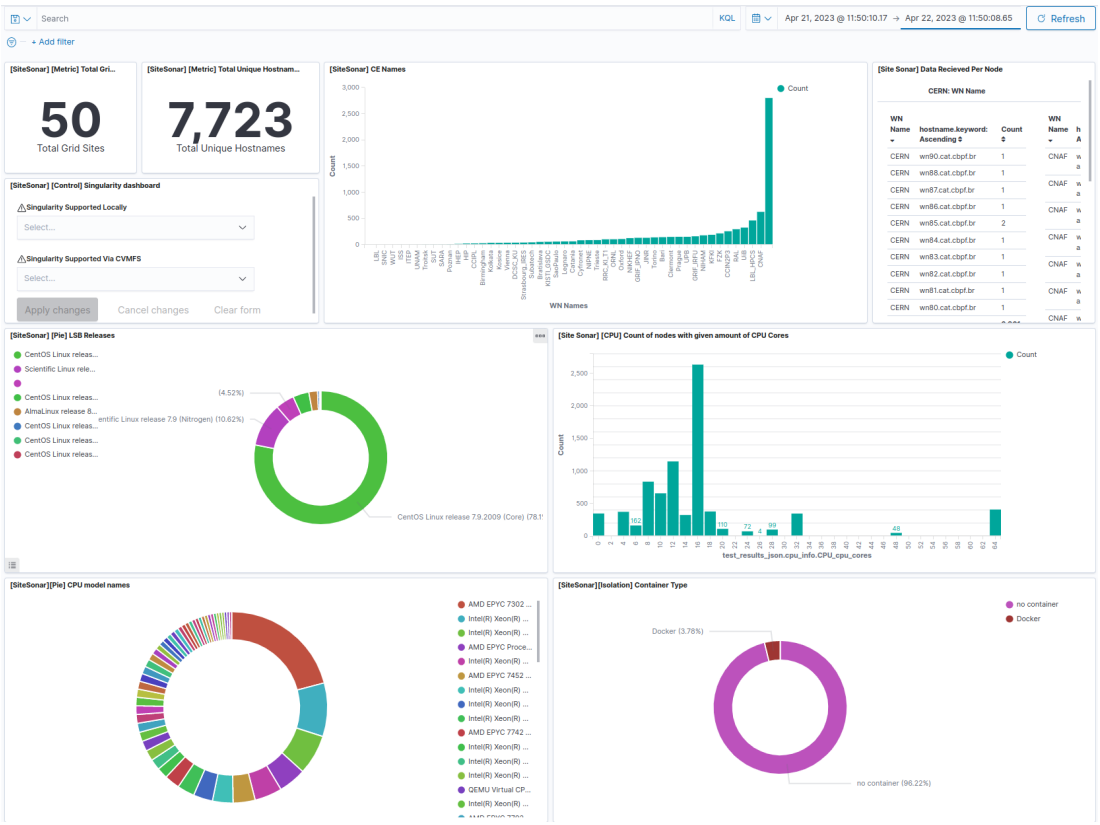

Fig. 5.4: ALICE Grid Overview on 2023-04-21

5.2.2 Reusing hostnames on different nodes

For logging and monitoring purposes, each worker node on the Grid should have a unique identifier. The worker node hostname usually is sufficient to play that role, but through Site Sonar it was found that some sites have such high turnover rates of hostnames that Site Sonar results for those sites then eclipsed the results of much bigger sites, at the same time suggesting that the ALICE Grid was bigger than it actually is. At one such site, each job runs in a Docker container which is given a hostname that is unique per job. At another site, jobs run in VMs with lifetimes of a few days and hostnames that are unique per instantiation. Such special cases had gone unnoticed until the introduction of Site Sonar because there was no way to track such information.

After identifying the issue, a special environment variable was added in sites with special cases to ensure that always provides a unique identifier only per host, which Site Sonar then uses instead. On the first site, hostname of the underlying hosts was setup in this environment variable whereas the hypervisor hostnames were set up in the second site as those VM hostnames happen to be extensions of the ones of the underlying hypervisors.

5.2.3 Use of Site Sonar as a Grid debugging tool

Although almost all sites ended up supporting Singularity, Site Sonar showed that many of the worker nodes often failed to start Singularity. This was observed as an intermittent issue on worker nodes across the Grid. Even when a worker node appeared to support Singularity, it sometimes failed due to some unknown cause. Traditional debugging in the ALICE Grid is done through reading job logs and traces, but in this case none of the jobs were showing this issue. Due to the high flexibility of Site Sonar which allows adding and changing monitoring probes on the fly, it was used as a debugging tool to track down the root cause of this issue by monitoring additional parameters relevant to the invocation of Singularity. In the end, the issue was caused by a specific payload script that originally had been used to test an early version of Site Sonar through a specific class of jobs and had not been adjusted when the Site Sonar functionality was moved up into the Job Agent to have it run by all jobs instead. This was an important finding which highlighted the potential of Site Sonar to be a Grid debugging tool that is lacking in the Grid Computing domain at the moment, which would be very useful in resolving Grid issues.

5.3 Evaluation

This section compares and contrasts the proposed design against the existing designs and evaluates the improvements gained by the new design.

5.3.1 Quantitative Evaluation

Since the proposed architecture cannot be directly evaluated quantitatively, the Site Sonar v2.0 solution implemented using the proposed design is evaluated quantitatively against the existing system Site Sonar v1.0. The qualitative evaluation of the architecture is available in section 5.3.2.

5.3.1.1 Execution Time evaluation

This section intends to compare the time taken for data collection between the new design and the old design based on the 2 versions of Site Sonar implementations. The experiment setup uses a setup called “JAliEn replica” which emulates a distributed Computing Grid in a local computer. It uses the actual components of the Grid and deploys them in the form of minimal containers that can provide the basic functionality of an actual Grid. The setup was run on an HP Elitebook x360-1040-GT Notebook PC equipped with 11th Gen Intel Core i5 (2.40GHz) 8 core processors running Ubuntu 20.04.6 LTS Operating system.

As explained in the previous chapters, both Site Sonar v1.0 and Site Sonar v2.0 collects the infrastructure information of the node that its Job Agent lands on by executing a set of scripts that query the infrastructure data and report to the central services. However, the time taken for the collected information to be available is different based on the approach that is used to collect data. Site Sonar v1.0 submits its own jobs to collect data which means that in order for the data to be available its job should run up to the completion. This requires a considerable time as initiating a job, running a job, finishing the job, cleaning up the environment, and uploading the results to the relevant locations take a considerable amount of time.

Site Sonar v2.0 takes a different approach to solve this problem. In Site Sonar v2.0 the data is collected and reported back even before the job is started. Since it is located in the JobAgent, it does not have to face any delays or overheads associated with running a job. Therefore Site Sonar v2.0 reports the results back within a few seconds after the job agent lands on a computing node which provides a large improvement in the time taken for reporting back the data from a computing node.

Batch name	Site Sonar v1.0(seconds)	Site Sonar v2.0(seconds)
Batch 1	140	3
Batch 2	146	4
Batch 3	527	4
Batch 4	605	3
Batch 5	636	3
Batch 6	152	4
Batch 7	146	4
Batch 8	639	3
Batch 9	769	3
Batch 10	646	4

TABLE 5.3: Time taken for Data Collection in Site Sonar

Time taken for collecting and reporting back the data in a single computing node has been collected in the two systems by running 10 batches of jobs in JAliEn replica. Each batch contains 10 jobs and the average time to report back the data in each batch is present in table 5.3 and the results are plotted in the chart 5.5.

It can be seen that the time taken for the data collection varies considerably in Site Sonar v1.0. This is mainly because it requires a complete job execution for the data to be available which will depend on many factors like downloading input files, transferring data to storage elements etc. which can also vary on multiple factors like I/O waitings, delay for processes to be completed etc. On average it takes 440 seconds for data collection in Site Sonar v1.0 which amounts to 7.3 minutes.

In Site Sonar v2.0 the overhead of other processes is completely removed because it is detached from the actual job. It runs as the initial task at the beginning of a

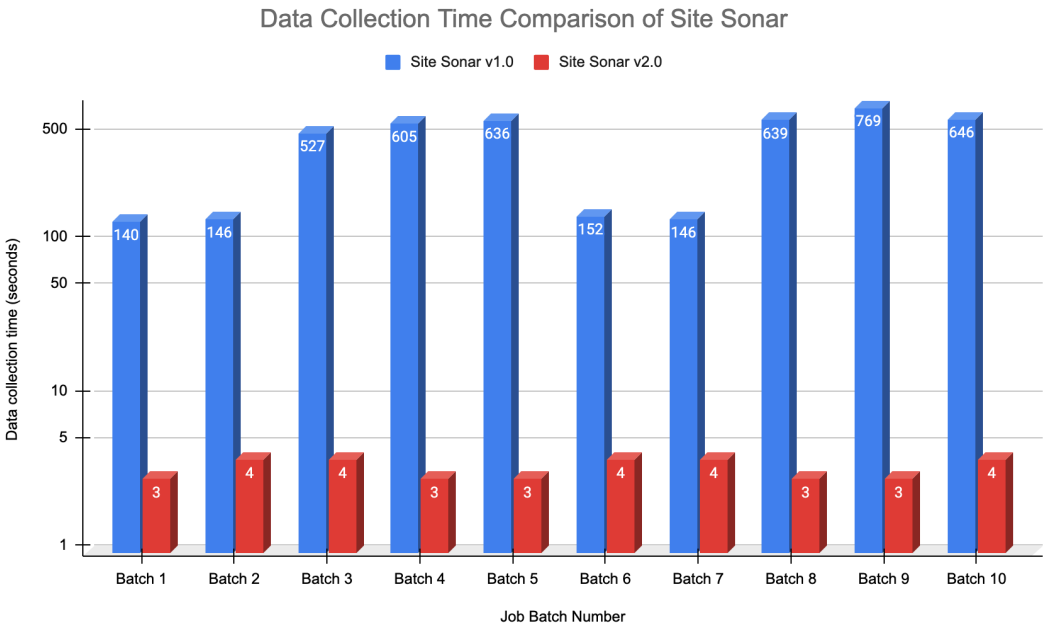

Fig. 5.5: Time taken for Data Collection in Site Sonar

JobAgent execution which does not have any dependencies. As a result of that, we can see that Site Sonar v2.0 always provides results within an identical range of time providing an average of 3.5 seconds which is a massive improvement compared to the existing system.

The results show that Site Sonar v2.0 possesses a much better data collection speed compared to Site Sonar v1.0. However, it is to be noted that the the considerable improvement is not due to improving the existing design, but rather due to using an entirely new design to address the problem.

5.3.1.2 Core hour wastage evaluation

In Grid domain number of work done is usually referred to as "core hours" where 1 core hour refers to allocating 1 core of a CPU for a specific job for one hour. The allocated core could be doing computation or not in this period. One of the most critical goals of a Grid environment is to use the core hours in an optimum manner. The efficiency of a job can be calculated in terms of core hours as follows.

$$\text{Efficiency of the job} = \frac{\text{No. of core hours spent doing useful computations}}{\text{Total core hours spent by the job}}$$

This highlights the fact that it is critical to reduce the no. of core hours that are spent on other work than doing the actual computation thus resulting in wastage of core hours in a Computing Grid. In this context, collecting infrastructure information

is considered a wastage of core hours because it is not executing an actual job payload when collecting information.

Site Sonar v1.0 causes a considerable wastage of Grid resources because it submits a job that is only targeted for the data collection of nodes. Executing a job in a Grid node contains multiple steps.

1. Acquiring an execution slot in the worker node
2. Initiating the relevant services in the node
3. Downloading packages and dependencies
4. Executing the job
5. Uploading the results
6. Cleaning up the worker node
7. Replicating results between multiple nodes if necessary

Each of these steps has a considerable overhead and spending valuable compute time of a Computing Grid just for data collection is a considerable waste. Site Sonar v2.0 considerably minimizes this overhead as it runs as a part of the JobAgent instead of an actual job. This eliminates the requirement to submit an entirely new job for monitoring and does monitoring as a part of actual jobs that contain useful computations.

Batch name	Site Sonar v1.0(core hours)	Site Sonar v2.0(core-hours)
Batch 1	272.22	5.83
Batch 2	283.89	7.78
Batch 3	1024.72	7.78
Batch 4	1176.39	5.83
Batch 5	1236.67	5.83
Batch 6	295.56	7.78
Batch 7	283.89	7.78
Batch 8	1242.50	7.78
Batch 9	1495.28	5.83
Batch 10	1256.11	7.78

TABLE 5.4: No. of core hours spent for Data Collection in Site Sonar

The number of core hours wasted for monitoring in Site Sonar v1.0 and Site Sonar v2.0 is presented in 5.4 and plotted in chart 5.6. The chart uses the same setup described in section 5.3.1.1. The execution time of the 2 services on the JAliEn replica is used to do the calculation and the total wastage in each job batch is calculated by

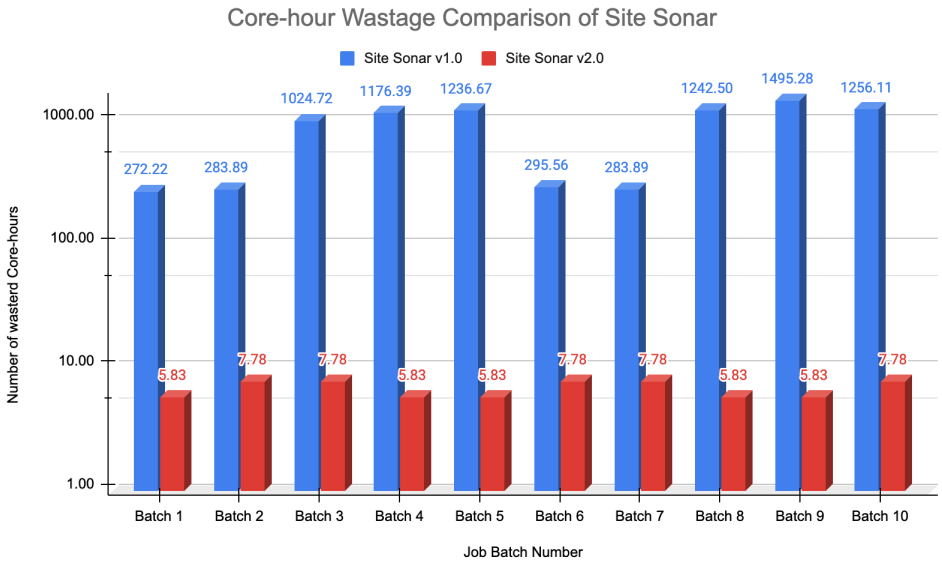

Fig. 5.6: Core-hours wastage in Site Sonar v1.0 vs v2.0

assuming each job takes 1 core. The average core-hour wastage per batch is then multiplied by 7,000 to project an estimate for the actual core-hours that will be lost if the setup was run in the actual ALICE Computing Grid.

It can be clearly noticed that Site Sonar v1.0 wastes a considerable amount of core-hours, sometimes wasting more than 1000 core hours for a single data collection run. The average lost core-hours can be seen as 856.72 in Site Sonar v1.0 whereas it is mere 6.81 hours in Site Sonar v2.0. This is a 99.67% reduction of core hour wastage compared to the existing system.

Note that the numbers in an actual computing Grid could be different from the estimates due to different factors like network speed, different CPU models etc. However, in total Site Sonar v1.0 would still have a larger amount of core hours because of the overhead associated with its design. If we consider the actual total number of lost core-hours in a single data collection round, Site Sonar v1.0 will report a much larger wastage because it submits jobs in the scale of twice the number of the worker nodes in the Grid which would essentially increase the wastage by a factor of 2. This leads to the conclusion that Site Sonar v1.0 is much less efficient than Site Sonar v2.0.

5.3.1.3 Query Performance evaluation

A major drawback of the Site Sonar v1.0 was the time taken to query the data which had a considerable impact on the usability of the system. A single data retrieval query could take up to 20 seconds. Hence using the system to explore the data with small adjustments to the data like changing the date ranges, adding/removing filters etc. and building conclusions out of it was extremely difficult. To address this problem, the

proposed architecture included moving the database engine from PostgreSQL to Elasticsearch as discussed in detail in section 3.2.3 which improved the querying time of the system to a couple of seconds for any query. This experiment compares the time taken to retrieve the data in each system from the database backend.

The experiment was done on actual ALICE Grid servers that hold the same data in both PostgreSQL database and Elasticsearch backends. A predefined set of queries that intends to output the same results by querying data in a specific time period(16/04/2023 - 23/4/2023) were executed on each server and the time taken to execute the query and return the results were collected. The set of queries that were run are shown in Table 5.5. The queries are simplified(not the actual query) for readability and the uniqueness of each query is also described.

Number	Uniqueness	Example
Query 1	All data	Select *
Query 2	All values of a metric	Select CGROUPv2_AVAILABLE
Query 3	Specific value of a metric	Select CGROUPv2_AVAILABLE:true
Query 4	Specific values of multiple metrics	Select (CGROUPv2_AVAILABLE:true) AND (CPU_CORES:16)
Query 5	Multiple values of a specific metric	Select CPU_FLAGS: is one of [avx,avx512]
Query 6	Combination of Query 4 & 5	Select (CGROUPv2_AVAILABLE:true) AND (CPU_CORES:16) AND (CPU_FLAGS: is one of [avx,avx512])

TABLE 5.5: Simplified queries used for the evaluation

Observed values from the experiment are plotted in Fig. 5.7. It can be observed that all the queries provide faster results with Elasticsearch-backed Site Sonar v2.0 and all the results are returned under 4 seconds. It is to be noted that PostgreSQL also return results with an acceptable latency for basic queries like query 2 and 3. The performance gap is highlighted when many complex queries are executed in the two systems leading Site Sonar v1.0 to report very high latency values up to 14 seconds for the considered dataset. Therefore, it can be concluded that Site Sonar v2.0 performs better in terms of Data retrieval speed as compared to Site Sonar v1.0.

5.3.1.4 Time taken for Introducing a new matching parameter

As discussed in section 3.5.1, it requires large changes in core parts of the Grid middleware to introduce a new job matching parameter. Due to the number of changes and

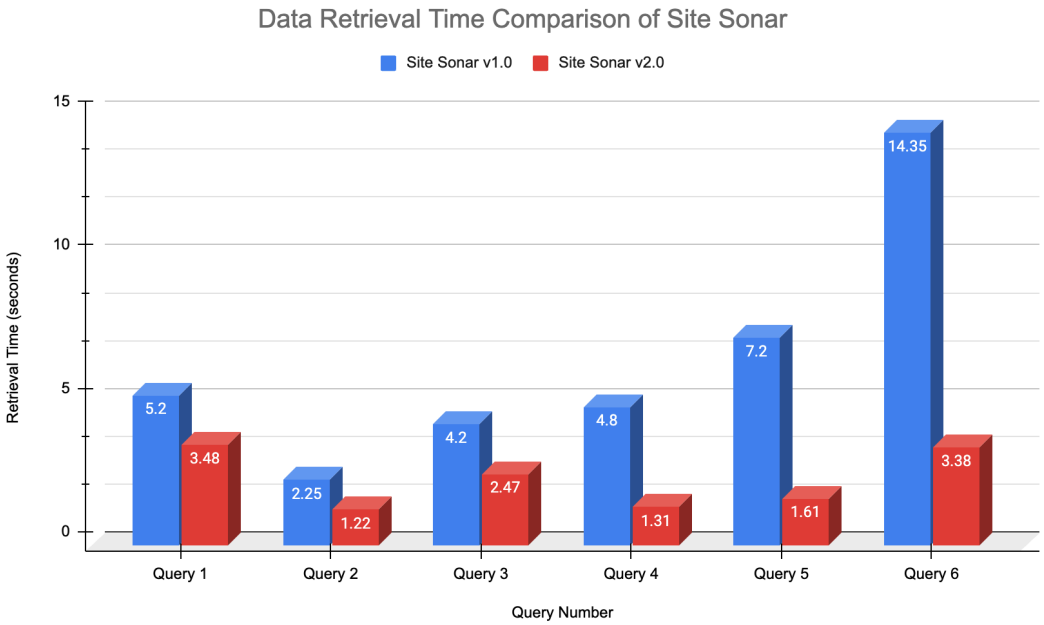

Fig. 5.7: Data Retrieval Time in Site Sonar v1.0 vs v2.0

effort needed, introducing new matching parameters is done very carefully and infrequently. The new architecture addresses this problem by allowing the introduction of parameters without any code changes.

The time taken to introduce a new parameter in the existing and new design of JAliEn is evaluated in this section. The deployment and testing times are recorded by observing the time taken by the Grid administrators to follow the software deployment lifecycle and create releases. Table 5.6 presents the time taken to add a new parameter to the JAliEn Grid middleware following the existing practices and table 5.7 presents the time taken to add a new parameter in the improved version of JAliEn Grid middleware that is integrated with Site Sonar v2.0.

Action	Time taken
Update the central server code to accept values for the new parameter	1 day
Test the functionality of the updated code in a central server	1 day
Deploy and test the update code across the central server farm after testing	3 days
Deploy the new pilot job code in few sites and test	1-3 days
Deploy the new pilot job code across all Grid sites	1 week

TABLE 5.6: Deployment time for adding a new job matching parameter in the existing design

As shown in Table 5.6, adding a new job matching parameter in the existing JAliEn implementation can usually take 1-3 weeks. This is because adding the new parameter will require updating both central servers and the JobAgent, and each of these com-

Action	Time taken
Deploy the new probe in CVMFS and allow distribution	1 day
Define the new constraint in the database and check data integrity	6 hours
Update the constraint servlet to include the new parameter	6 hours

TABLE 5.7: Deployment time for adding a new job matching parameter in the proposed design

ponents has its own deployment cycle. Changes to central servers are tested on one server and rolled out across the server farm whereas changes to JobAgent are tested on one site and the update is gradually propagated across other sites over a span of approximately 1 week ensuring updated sites are not showing abnormal behaviors.

Since Site Sonar v2.0 completely avoids the requirement to change any of the code related to central servers and JobAgent, adding a new parameter is much easier. It requires updating one of the database tables with the new constraint and updating the HTTP endpoint(that hot deploys the changes to its source code) that outputs the results from this database. Since both these components are completely independent from the critical components: Central servers and the JobAgent in the Grid WMS, there is no such rigorous testing or deployment processes which reduces the new parameter introduction time to about 2 days. Therefore, it can be seen that Site Sonar v2.0 drastically reduces the time taken to introduce a new job matching parameter from 1-3 weeks to 2 days while avoiding the need to update critical components of Grid middleware.

5.3.2 Qualitative Evaluation

The proposed architecture is mainly focused on providing new and important features that cannot be provided by using the existing architectures. Therefore, the system provides more features that can be compared qualitatively instead of quantitatively. This section intends to qualitatively evaluate the proposed architecture.

5.3.2.1 Extensibility

One of the 2 major goals in designing the new monitoring system is the possibility to easily extend the system. This is a critical feature because all the existing systems are limited by the fact that they cannot be extended as required. This is addressed in Site Sonar v2.0 data collection framework with the use of the distributed file system used by the relevant Grid. The data collection scripts will reside on the distributed file system and be called on demand at the data collection step. This allows to add or remove data collection probes as necessary or enable/disable them on demand.

The discussed monitoring tools in this study are divided into two models: the Data Pull Model and the Data Push Model. The monitoring systems that follow data pull model like GridIce, Paryavekshanam etc. also offer some extensibility, but their model

of collecting data introduces many limitations in terms of functionality as discussed in section 3.1.3 and performance issues as discussed in section 5.3.1.1. So the pull model cannot be used to achieve the goals of this research. While the systems with data push model like MonALISA, MONIT etc. aligns with some of the goals, they are unable to provide this level of flexibility because they run agents on nodes and updating those agents with new probes often require new rollouts and updates to all the nodes which considerably hinder the extensibility of the system. Therefore, it is clear that the new design provides higher extensibility than the existing systems.

5.3.2.2 Flexibility

The other main goal of designing the new monitoring system is to offer high flexibility in terms of data collection. This refers to allowing changing of the data structure and the data type of data collection and allowing the collection of arbitrary data.

Site Sonar v2.0 provides these features by the use of an ELK stack. Due to the extensibility of the system, monitoring probes can be updated and deployed on demand. Any of these changed parameters can be collected without any changes in code in Site Sonar v2.0. Further, it allows doing high-level analysis of any data without additional changes. This leads to the provision of Post data filtering feature which is described in section 3.2.2 that is very helpful in things like Grid debugging which is explained in section 5.2.3.

All of the discussed systems except MONIT are based on SQL databases. This imposes a critical limitation on their flexibility because SQL requires all the data types to be predefined. Different data types cannot be collected unless all are defined as strings which limits their indexing capabilities. While changing parameter keys can be handled with SQL by creating a table that accepts any key, this would largely restrict analysis and visualizations using that data as it makes it hard to group different monitoring parameter values by their name. Therefore, providing the level of flexibility given by Site Sonar v2.0 will require the support of a noSQL database, and out of the discussed systems, only MONIT is capable of achieving the desired level of flexibility.

5.3.2.3 Supported parameters

Very low time taken for parameter introduction and zero impact to the core framework allows the improved JAliEn version to support any number of parameters without a problem. However, due to the limitations discussed in 3.5.2, usual Grid middleware supports only a limited set of parameters.

The table 3.1 shows the number of parameters that can be supported by the existing Grid middleware compared to the improved JAliEn. It can be seen that all the existing systems can support only a limited set of infrastructure parameters with a maximum of

around 6-10 parameters. These are hardcoded in the Pilot job code and updated very infrequently.

Improved JAliEn integrated with Site Sonar presents the possibility of introducing an unlimited no. of infrastructure properties in the job matching process. Due to this reason, it can be concluded that the introduced design is much more versatile than the existing designs.

5.3.3 Associated projects

Since its introduction, Site Sonar v2.0 has been a very useful tool for Grid administrators to understand the nature of the Grid in addition to the improved job matching it provides. In ALICE Computing Grid, multiple studies are being undertaken to improve different aspects of the Grid usage and some of them have already started using the data collected from Site Sonar to base their observations on.

5.3.3.1 CPU Oversubscription

One of the studies in ALICE Computing Grid presently is to devise a dynamic scheduling strategy that identifies the idle resources in the nodes that are executing jobs and oversubscribe the node to run extra jobs This requires up-to-date information about the CPU cores, memory and other hardware information of the worker node to understand if the node has extra resources and how much of them can be reused. Such hardware information is collected from Site Sonar and they are used to plan the dynamic scheduling strategy which has yielded promising results.

5.3.3.2 Open File Limit Comparison

A study in ALICE Grid has been undertaken on why some jobs with large I/O requirements are being killed in some Grid sites without a reason. These sites have been identified as very large size with enough I/O capabilities to process the job, yet fail to do so. Upon observing the infrastructure of the worker nodes, it has been noted that the worker nodes use a OS level default value for the maximum number of open files and hence they kill the job when the job exceeds the set limit even if it has the capability to fully execute it. These levels set in different worker nodes are collected via Site Sonar and the relationship between the number of open files and the number of cores in the node is being studied to propose new maximum open file limits and other relevant parameters to the sites.

5.3.3.3 Profiling and Duration Estimation of the MonteCarlo Jobs

An ongoing project in the ALICE Grid aims to improve the scheduling algorithms for the CPU-intensive jobs (MonteCarlo simulations) based on their run history. The

project attempts to determine the components that impact the job performance. Thus, it combines data extracted from job traces (time spent by the job, site and hostname) with information extracted from SiteSonar such as CPU model and its flags, host configurations (containerization, job scheduler applications, as well as other metrics). Moreover, thanks to SiteSonar's extensibility, new scripts can be easily integrated into the platform to extract additional information that may be necessary during the analysis (virtualization, vulnerability mitigations, DMI configuration).

CHAPTER 6

CONCLUSION

6.1 Contribution

This research focused on studying distributed Computing Grids to propose a highly flexible and extensible Grid infrastructure monitoring architecture design and a new infrastructure aware job matching architecture for them. The research studied the existing Grid monitoring tools and Grid workflow managements systems and the lack of interoperability between these systems which restricts providing infrastructure aware job matching in computing Grids.

The research identified that the existing Grid infrastructure systems are not equipped to achieve the present monitoring requirements and discussed the limitations of these systems. A new Grid monitoring architecture that is based on noSQL backend with the ability to extend and change monitoring parameters on-demand was proposed in this research to address this. The proposed design is used to develop a tool called "Site Sonar v2.0" that is currently being used in production to monitor 7,000+ worker nodes in 60+ Grid sites across the ALICE Computing Grid. It is a valuable addition to the Grid and proves to achieve the required flexibility and extensibility, leading it to be used successfully even as a Grid debugging tool. The introduced tool has also led to new findings about the ALICE Computing Grid which is not possible with traditional monitoring systems and has surpassed its predecessor Site Sonar v1.0 in both performance and usability perspectives.

This research recognized the importance of the new Grid monitoring capabilities introduced in the proposed design to match jobs to worker nodes based on their infrastructure properties and proposed a novel architecture to integrate Grid monitoring tools with Grid workflow management systems. The new design bridges the gap between the Grid monitoring systems and workflow management systems allowing the monitoring information to be used in the job matching criteria. The proposed design is implemented in JAliEn Grid workflow management system that is used to handle workflows in the ALICE Computing Grid. Implementation has proven to be very useful allowing the Grid administrators to match jobs based on any infrastructure metric of the worker nodes. Monitoring systems and WMSs have been largely independent so far and we intend that the new design will inspire advanced job matching designs in the future in light of the new parameters introduced in the job matching process in this design. Both the tools have been successfully implemented and are running in production across the nodes and servers in ALICE Computing Grid leading to a new way of monitoring worker nodes and matching jobs to worker nodes in distributed computing Grids.

6.2 Limitations and Future Work

This research focused on providing a modernized way of monitoring Distributed Computing Grids with a new perspective of job matching in Computing Grids. The proposed designs can be seen as novel concepts and hence there are aspects that can be considerably improved but are out of the scope of this research. This section describes the future work that can be done based on this research that can contribute to the advancement of the Grid Computing domain.

Identifying anomalies in the monitoring data can be seen as a significant contribution to the Grid monitoring domain. When the data is used for analyzing the issues in the ALICE Computing Grid, it was noticed that the anomalies in the data could help to understand the abnormal behavior of jobs in specific sites. For example, the absence of specific monitoring parameters in some worker nodes could hint that the worker nodes or the whole site are incorrectly configured to run jobs. In such cases, job failures can be directly mapped to the incorrect nodes by correlating the two factors. Although this can be done manually at the moment, it would be a good addition to automate such anomaly detection and notify relevant parties about the behavior automatically.

The proposed job matching design introduces imposing constraints on the job description about the infrastructure capabilities a worker node should suffice to be able to run that job. This allows the job to define what kind of worker nodes can run the job. From the studies, we have identified that it is also important for the sites to be able to impose constraints declaring which kind of jobs they would prefer to accept. This is becoming important with the addition of new special sites like GPU equipped Grid sites that would prefer to accept jobs that can execute parallelly leading to better utilization of the site resources. The current design can be extended to achieve this by improving the job matching process further allowing both the job and the Grid site to define their constraints and preferences of each other.

This research lays the initial steps for bridging the gap between Grid infrastructure monitoring systems and Grid WMSs. However, we have not considered how the job monitoring systems can also be integrated with the infrastructure monitoring systems. While the current integration provides the basic functionality, integration with a job monitoring system would allow the Grid administrators to get a full picture of the Grid infrastructure and Grid jobs and correlate between them easily to narrow down and identify the root cause of Grid issues. This would also be an important research area because none of the existing systems are seen to have both infrastructure and job monitoring data correlated with each other while being used for job matching as well.

REFERENCES

[1] L. Evans, "The large hadron collider," *New Journal of Physics*, vol. 9, no. 9, p. 335, 2007.

[2] CERN, "CERN Annual report 2019," CERN, Geneva, Tech. Rep., 2019. [Online]. Available: http://cds.cern.ch/record/2723123

[3] T. J. Berners-Lee, "Information management: A proposal," Tech. Rep., 1989.

[4] "CERN Annual report 2016," CERN, Geneva, Tech. Rep., 2017. [Online]. Available: https://cds.cern.ch/record/2270805

[5] P. Buncic, M. Krzewicki, and P. Vande Vyvre, "Technical Design Report for the Upgrade of the Online-Offline Computing System," Tech. Rep., 2015. [Online]. Available: https://cds.cern.ch/record/2011297

[6] A. Grigoras, C. Grigoras, M. Pedreira, P. Saiz, and S. Schreiner, "Jalien–a new interface between the alien jobs and the central services," in *Journal of Physics: Conference Series*, vol. 523, no. 1. IOP Publishing, 2014, p. 012010.

[7] B. Jacob, M. Brown, K. Fukui, N. Trivedi *et al.*, "Introduction to grid computing," *IBM redbooks*, pp. 3–6, 2005.

[8] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *The International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[9] I. Foster, "What is the grid? a three point checklist," *GRID today*, vol. 1, pp. 32–36, 01 2002.

[10] A. Weerasinghe, K. Wijethunga, R. Jayasekara, I. Perera, and A. Wickramarachchi, "Resource aware task clustering for scientific workflow execution in high performance computing environments," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2020, pp. 255–262.

[11] J. Nabrzyski, J. M. Schopf, and J. Weglarz, "Grid resource management: state of the art and future trends," 2012.

[12] E. Elmroth and J. Tordsson, "Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions," *Future Generation Computer Systems*, vol. 24, no. 6, pp. 585–593, 2008.

[13] S. Jang, X. Wu, V. Taylor, G. Mehta, K. Vahi, and E. Deelman, "Using performance prediction to allocate grid resources," *Texas A&M University, College Station, TX, GriPhyN Technical Report*, vol. 25, 2004.

[14] R. Brun, P. Buncic, F. Carminati, A. Morsch, F. Rademakers, and K. Safarik, "Computing in alice," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 502, no. 2-3, pp. 339–346, 2003.

[15] S. Fartoukh, I. Efthymiopoulos, R. Tomas Garcia, R. Bruce, H. Timko, G. Arduini, N. Mounet, Y. Papaphilippou, B. Salvant, S. Redaelli *et al.*, "Lhc configuration and operational scenario for run 3," Tech. Rep., 2021.

[16] S. Zanikolas and R. Sakellariou, "A taxonomy of grid monitoring systems," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 163–188, 2005.

[17] R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, B. Tierney, and R. Wolski, "A grid monitoring architecture," *Recommendation GWD-I (Rev. 16*, 2002.

[18] S. Andreozzi, C. Aiftimiei, G. Cuscela, S. Dal Pra, G. Donvito, V. Dudhalkar, S. Fantinel, E. Fattibene, G. Maggi, G. Misurelli *et al.*, "Next steps in the evolution of gridice: a monitoring tool for grid systems," in *Journal of Physics: Conference Series*, vol. 119, no. 6. IOP Publishing, 2008, p. 062010.

[19] S. Andreozzi, M. Sgaravatto, and C. Vistoli, "Sharing a conceptual model of grid resources and services," *arXiv preprint cs/0306111*, 2003.

[20] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (xml)," *World Wide Web Journal*, vol. 2, no. 4, pp. 27–66, 1997.

[21] D. H. Karuna, N. Mangala, B. Prahlada Rao, and N. Mohan Ram, "Paryavekshanam: a status monitoring tool for indian grid garuda," in *24th NORDUnet2008 Conference–The Biosphere of Grids and Networks, Espoo, Finland*, 2008, pp. 9–11.

[22] B. Prahlada Rao, S. Ramakrishnan, M. Raja Gopalan, C. Subrata, N. Mangala, and R. Sridharan, "e-infrastructures in it: A case study on indian national grid computing initiative–garuda," *Computer Science-Research and Development*, vol. 23, no. 3, pp. 283–290, 2009.

[23] I. Legrand, C. Cirstoiu, C. Grigoras, R. Voicu, M. Toarta, C. Dobre, and H. Newman, "Monalisa: An agent based, dynamic service system to monitor, control and optimize grid based applications," 2005.

[24] J. Balcas, D. Kcira, A. Mughal, H. Newman, M. Spiropulu, and J.-R. Vlimant, "Monalisa, an agent-based monitoring and control system for the lhc experiments," *Journal of Physics: Conference Series*, vol. 898, p. 092055, 10 2017.

[25] D. Petković, "Json integration in relational database systems," *Int J Comput Appl*, vol. 168, no. 5, pp. 14–19, 2017.

[26] A. Aimar, A. A. Corman, P. Andrade, J. D. Fernandez, B. G. Bear, E. Karavakis, D. M. Kulikowski, and L. Magnoni, "Monit: monitoring the cern data centres and the wlcg infrastructure," in *EPJ Web of Conferences*, vol. 214. EDP Sciences, 2019, p. 08031.

[27] M. Babik, I. Fedorko, N. Hook, H. T. Lansdale, D. Lenkes, M. Siket, and D. Waldron, "Lemon-lhc era monitoring for large-scale infrastructures," in *Journal of Physics: Conference Series*, vol. 331, no. 5. IOP Publishing, 2011, p. 052025.

[28] J. Andreeva, M. Boehm, B. Gaidioz, E. Karavakis, L. Kokoszkiewicz, E. Lanciotti, G. Maier, W. Ollivier, R. Rocha, P. Saiz *et al.*, "Experiment dashboard for monitoring computing activities of the lhc virtual organizations," *Journal of Grid Computing*, vol. 8, no. 2, pp. 323–339, 2010.

[29] B. Elasticsearch, "Elasticsearch," vol. 6, no. 1, 2018.

[30] Y. Gupta, *Kibana essentials*. Packt Publishing Ltd, 2015.

[31] N. Garg, *Apache kafka*. Packt Publishing Birmingham, UK, 2013.

[32] L. Gardi, "Hardware monitoring with collectd," Tech. Rep., 2018.

[33] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[34] D. Borthakur *et al.*, "Hdfs architecture guide," *Hadoop apache project*, vol. 53, no. 1-13, p. 2, 2008.

[35] M. Chakraborty and A. P. Kundan, "Grafana," in *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*. Springer, 2021, pp. 187–240.

[36] D. Piparo, E. Tejedor, P. Mato, L. Mascetti, J. Moscicki, and M. Lamanna, "Swan: A service for interactive analysis in the cloud," *Future Generation Computer Systems*, vol. 78, pp. 1071–1078, 2018.

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

[38] E. B. Sandvik, “Site sonar-a monitoring tool for alice’s grid sites,” Master’s thesis, The University of Bergen, 2021.

[39] W. Barth, *Nagios: System and network monitoring*. No Starch Press, 2008.

[40] M. L. Massie, B. N. Chun, and D. E. Culler, “The ganglia distributed monitoring system: design, implementation, and experience,” *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[41] P. Saiz, L. Aphecetche, P. Bunčić, R. Piskač, J.-E. Revsbech, V. Šego, A. Collaboration *et al.*, “Alien—alice environment on the grid,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 502, no. 2-3, pp. 437–440, 2003.

[42] R. Cruceru, “Enabling distributed analysis for alice run 3,” *arXiv preprint arXiv:2211.12276*, 2022.

[43] A. Collaboration, G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, A. A. Abdelalim, O. Abdinov, R. Aben, B. Abi *et al.*, “A particle consistent with the higgs boson observed with the atlas detector at the large hadron collider,” *Science*, vol. 338, no. 6114, pp. 1576–1582, 2012.

[44] T. Maeno, K. De, A. Klimentov, P. Nilsson, D. Oleynik, S. Panitkin, A. Petrosyan, J. Schovancova, A. Vaniachine, T. Wenaus *et al.*, “Evolution of the atlas panda workload management system for exascale computational science,” in *Journal of Physics: Conference Series*, vol. 513, no. 3. IOP Publishing, 2014, p. 032062.

[45] F. H. B. Megino, A. Alekseev, F. Berghaus, D. Cameron, K. De, A. Filipcic, I. Glushkov, F. Lin, T. Maeno, and N. Magini, “Managing the atlas grid through harvester,” in *EPJ Web of Conferences*, vol. 245. EDP Sciences, 2020, p. 03010.

[46] “Atlas production task brokerage,” https://panda-wms.readthedocs.io/en/latest/advanced/brokerage.html#id6, accessed: 2023-04-29.

[47] I. Sfiligoi, “glideinwms—a generic pilot-based workload management system,” in *Journal of Physics: Conference Series*, vol. 119, no. 6. IOP Publishing, 2008, p. 062044.

[48] D. Bradley, O. Gutsche, K. Hahn, B. Holzman, S. Padhi, H. Pi, D. Spiga, I. Sfiligoi, E. Vaandering, F. Würthwein *et al.*, “Use of glide-ins in cms for production and analysis,” in *Journal of Physics: Conference Series*, vol. 219, no. 7. IOP Publishing, 2010, p. 072013.

[49] M. M. Pedreira, C. Grigoras, and V. Yurchenko, "Jalien: the new alice high-performance and high-scalability grid framework," in *EPJ Web of Conferences*, vol. 214. EDP Sciences, 2019, p. 03037.

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

[51] D. Álvarez, K. Sala, and V. Beltran, "nos-v: Co-executing hpc applications using system-wide task scheduling," *arXiv preprint arXiv:2204.10768*, 2022.

[52] E. B. Sandvik, "Site sonar - a monitoring tool for alice's grid sites," 2021.

[53] A. Reelsen, "Using elasticsearch, logstash and kibana to create real-time dashboards," *Dostupné z. Available online: https://speakerdeck.com/elasticsearch/using-elasticsearch-logstash-and-kibana-to-create-realtimedashboards*, 2014.

[54] R. Rosen, "Resource management: Linux kernel namespaces and cgroups," *Haifux, May*, vol. 186, p. 70, 2013.

APPENDIX A

FULL DATA SET COLLECTED FROM A WORKER NODE

This appendix shows the full set of data collected from a single worker node in the ALICE Computing Grid using the Site Sonar Data Collection Framework introduced in this research.


```
{
  "_index": "new-mapping-sitesonar-2023.04",
  "_type": "_doc",
  "_id": "314CwIcBwP3PkOBT6JBF",
  "_version": 1,
  "_score": null,
  "_source": {
    "port": 59266,
    "host_id": 83851870,
    "@timestamp": "2023-04-26T09:57:28.000Z",
    "ce_name": "NIPNE",
    "addr": "81.180.86.124",
    "@version": "1",
    "test_results_json": {
      "ram_info": {
        "RAM_kB_Inactive(anon)": 6936452,
        "RAM_HugePages_Surp": 0,
        "RAM_kB_Cached": 17984532,
        "RAM_kB_DirectMap2M": 19728384,
        "RAM_kB_Hugepagesize": 2048,
        "RAM_kB_Active(anon)": 13954640,
        "RAM_kB_WritebackTmp": 0,
        "RAM_HugePages_Total": 0,
        "RAM_kB_SwapFree": 31962860,
        "RAM_kB_HardwareCorrupted": 0,
        "RAM_kB_Slab": 905852,
        "RAM_kB_CmaFree": 0,
        "RAM_kB_Bounce": 0,
        "RAM_kB_Mlocked": 0,
        "RAM_kB_Buffers": 224,
        "RAM_kB_AnonHugePages": 0,
        "RAM_kB_Active": 22863220,
```

```
    "RAM_kB_KernelStack": 16336,
    "RAM_kB_SUnreclaim": 272568,
    "RAM_kB_VmallocTotal": 34359738367,
    "RAM_kB_Inactive": 15686420,
    "RAM_HugePages_Rsvd": 0,
    "RAM_kB_PageTables": 126560,
    "RAM_kB_MemTotal": 65780016,
    "RAM_kB_SReclaimable": 633284,
    "RAM_kB_DirectMap4k": 176512,
    "RAM_kB_VmallocUsed": 404104,
    "RAM_kB_DirectMap1G": 47185920,
    "RAM_kB_CmaTotal": 0,
    "RAM_kB_SwapCached": 29676,
    "RAM_kB_Active(file)": 8908580,
    "EXITCODE": 0,
    "RAM_kB_Writeback": 12,
    "RAM_kB_NFS_Unstable": 4,
    "RAM_kB_SwapTotal": 32767996,
    "RAM_kB_Shmem": 326164,
    "EXECUTION_TIME": 899,
    "RAM_kB_Inactive(file)": 8749968,
    "RAM_kB_MemAvailable": 42660680,
    "RAM_kB_Unevictable": 0,
    "RAM_kB_Percpu": 17408,
    "RAM_kB_AnonPages": 20537956,
    "RAM_kB_VmallocChunk": 34300112892,
    "RAM_HugePages_Free": 0,
    "RAM_kB_CommitLimit": 65658004,
    "RAM_kB_Dirty": 2336,
    "RAM_kB_Mapped": 229224,
    "RAM_kB_MemFree": 25304120,
    "RAM_kB_Committed_AS": 43775252
  },
  "os": {
    "OS_ANSI_COLOR": "0;31",
    "OS_CPE_NAME": "cpe:/o:centos:centos:7",
    "OS_PRETTY_NAME": "CentOS Linux 7 (Core)",
    "OS_CENTOS_MANTISBT_PROJECT": "CentOS-7",
    "OS_NAME": "CentOS Linux",
    "EXITCODE": 0,
```

```
  "OS_VERSION": "7 (Core)",
  "OS_REDHAT_SUPPORT_PRODUCT": "centos",
  "EXECUTION_TIME": 503,
  "OS_BUG_REPORT_URL": "https://bugs.centos.org/",
  "OS_HOME_URL": "https://www.centos.org/",
  "OS_ID": "centos",
  "OS_ID_LIKE": "rhel fedora",
  "OS_VERSION_ID": "7",
  "OS_REDHAT_SUPPORT_PRODUCT_VERSION": "7",
  "OS_CENTOS_MANTISBT_PROJECT_VERSION": "7"
},
"underlay": {
  "UNDERLAY_ENABLED": "yes",
  "EXECUTION_TIME": 146,
  "EXITCODE": 0
},
"gcc_version": {
  "GCC_VERSION": "gcc (GCC) 7.3.0",
  "EXECUTION_TIME": 174,
  "EXITCODE": 0
},
"cpuset_checking": {
  "CPUSET_CPUS": "0-31",
  "EXECUTION_TIME": 2366,
  "EXITCODE": 3,
  "CPU_AMOUNT": 32,
  "CPUSET_PREFIX": "/",
  "CPUSET_ENABLED": true,
  "CPUSET_CGROUP": "/sys/fs/cgroup/cpuset"
},
"profiling_checking": {
  "PTRACE_SCOPE": "0",
  "VTUNE_PROFILING_ENABLED": false,
  "EXECUTION_TIME": 129,
  "PERF_EVENT_PARANOID": "2",
  "EXITCODE": 0
},
"cvmfs_version": {
  "CVMFS_VERSION": 15271,
  "EXECUTION_TIME": 290,
```

```
  "EXITCODE": 0
},
"singularity_debug": {
  "SINGULARITY_CVMFS_SUPPORTED_BOOL": true,
  "SINGULARITY_LOCAL_SUPPORTED_BOOL": true,
  "EXECUTION_TIME": 1366,
  "EXITCODE": 0
},
"uname": {
  "EXECUTION_TIME": 248,
  "UNAME": "Linux",
  "EXITCODE": 0
},
"vulnerabilities": {
  "EXECUTION_TIME": 564,
  "kernel_vulnerabilities": [
    {
      "output": "permission_error",
      "name": "itlb_multihit"
    },
    {
      "output": "permission_error",
      "name": "l1tf"
    },
    {
      "output": "permission_error",
      "name": "mds"
    },
    {
      "output": "permission_error",
      "name": "meltdown"
    },
    {
      "output": "permission_error",
      "name": "mmio_stale_data"
    },
    {
      "output": "permission_error",
      "name": "retbleed"
    },
```

```
{
  "output": "permission_error",
  "name": "spec_store_bypass"
},
{
  "output": "permission_error",
  "name": "spectre_v1"
},
{
  "output": "permission_error",
  "name": "spectre_v2"
},
{
  "output": "permission_error",
  "name": "srbds"
},
{
  "output": "permission_error",
  "name": "tsx_async_abort"
},
{
  "output": "Present",
  "name": "FEATURE IBPB_SUPPORT"
},
{
  "output": "Not Present",
  "name": "FEATURE SPEC_CTRL"
},
{
  "output": "not found in dmesg",
  "name": "MDS"
},
{
  "output": "not found in dmesg",
  "name": "MMIO Stale Data"
},
{
  "output": "Mitigation: untrained return thunk",
  "name": "RETBleed"
},
```

```
      {
        "output": "Mitigation: Full retpoline",
        "name": "Spectre V2"
      },
      {
        "output": "not found in dmesg",
        "name": "SRBDS"
      },
      {
        "output": "not found in dmesg",
        "name": "TAA"
      }
    ],
    "EXITCODE": 0
  },
  "cpu_info": {
    "EXITCODE": 0,
    "CPU_clflush_size": 64,
    "CPU_power_management": [
      "ts",
      "ttp",
      "tm",
      "100mhzsteps",
      "hwpstate",
      "cpb"
    ],
    "CPU_flags": [
      "fpu",
      "vme",
      "de",
      "pse",
      "tsc",
      "msr",
      "pae",
      "mce",
      "cx8",
      "apic",
      "sep",
      "mtrr",
      "pge",
```

```
"mca",
"cmov",
"pat",
"pse36",
"clflush",
"mmx",
"fxsr",
"sse",
"sse2",
"ht",
"syscall",
"nx",
"mmxext",
"fxsr_opt",
"pdpe1gb",
"rdtscp",
"lm",
"constant_tsc",
"art",
"rep_good",
"nopl",
"nonstop_tsc",
"extd_apicid",
"amd_dcm",
"aperfmperf",
"pni",
"pclmulqdq",
"monitor",
"ssse3",
"cx16",
"sse4_1",
"sse4_2",
"popcnt",
"aes",
"xsave",
"avx",
"lahf_lm",
"cmp_legacy",
"svm",
"extapic",
```

```
      "cr8_legacy",
      "abm",
      "sse4a",
      "misalignsse",
      "3dnowprefetch",
      "osvw",
      "ibs",
      "xop",
      "skinit",
      "wdt",
      "lwp",
      "fma4",
      "nodeid_msr",
      "topoext",
      "perfctr_core",
      "perfctr_nb",
      "cpb",
      "hw_pstate",
      "ssbd",
      "rsb_ctxsw",
      "ibpb",
      "vmmcall",
      "retpoline_amd",
      "arat",
      "npt",
      "lbrv",
      "svm_lock",
      "nrip_save",
      "tsc_scale",
      "vmcb_clean",
      "flushbyasid",
      "decodeassists",
      "pausefilter",
      "pfthreshold"
    ],
    "CPU_cpuid_level": 13,
    "CPU_initial_apicid": "103",
    "CPU_cpu_MHz": 2400,
    "EXECUTION_TIME": 5160,
    "CPU_address_sizes": "48 bits physical, 48 bits virtual",
```

```
    "CPU_vendor_id": "AuthenticAMD",
    "CPU_physical_id": 3,
    "CPU_model": 1,
    "CPU_cache_size": "2048 KB",
    "CPU_fpu_exception": true,
    "CPU_cpu_cores": 4,
    "CPU_stepping": 2,
    "CPU_fpu": true,
    "CPU_core_id": 3,
    "CPU_bogomips": 5199.3,
    "CPU_model_name": "AMD Opteron(TM) Processor 6212",
    "CPU_microcode": "0x600063e",
    "CPU_wp": true,
    "CPU_TLB_size": "1536 4K pages",
    "CPU_cpu_family": 21,
    "CPU_siblings": 8,
    "CPU_processor": "31",
    "CPU_cache_alignment": 64,
    "CPU_processor_count": 32,
    "CPU_apicid": 135
  },
  "cpulimit_checking": {
    "CGROUP": "/sys/fs/cgroup/cpu",
    "ACCESS_QUOTA": "",
    "ACCESS_PERIOD": "",
    "EXECUTION_TIME": 2223,
    "EXITCODE": 2,
    "ALLOCATED_CPUS": ""
  },
  "taskset_other_processes": {
    "PERCENTAGE_PINNED_CPUS": 0,
    "ENTRIES": "ffffffff",
    "ENTRY_COUNT": 1,
    "EXECUTION_TIME": 808,
    "EXITCODE": 0
  },
  "running_container": {
    "RUNNING_IN": "no container",
    "EXECUTION_TIME": 204,
    "EXITCODE": 0
```

```
},
"container_enabled": {
  "EXECUTION_TIME": 326,
  "EXITCODE": 1
},
"tmp": {
  "TEMP_DIR": "/mnt/scratch/arc/gMJNDm2zm/ALICE/tmp",
  "EXECUTION_TIME": 146,
  "EXITCODE": 0
},
"cpu_architecture": {
  "CPUs_node6": "24-27",
  "EXITCODE": 0,
  "NUMA_NODES": "8",
  "CPUs_node7": "28-31",
  "CPUs_node0": "0-3",
  "CPUs_node4": "16-19",
  "EXECUTION_TIME": 245,
  "CPUs_node2": "8-11",
  "CPUs_node3": "12-15",
  "CPUs_node1": "4-7",
  "CPUs_node5": "20-23"
},
"taskset_own_process": {
  "EXITCODE": 0,
  "PERCENTAGE_PINNED": 0,
  "EXECUTION_TIME": 370,
  "USING_PINNING": 0,
  "TASKSET": "ffffffff"
},
"lhcbmarks": {
  "LHCbMarks": 8.041,
  "EXECUTION_TIME": 154297,
  "EXITCODE": 0
},
"get_jdl_cores": {
  "EXECUTION_TIME": 83,
  "ALIEN_JDL_CPUCORES": "",
  "EXITCODE": 0
},
```

```
"cvmfs_cache_size": {
  "CVMFS_CACHE_SIZE": 49,
  "EXECUTION_TIME": 247,
  "EXITCODE": 0
},
"overlay": {
  "OVERLAY_ENABLED": "try",
  "EXECUTION_TIME": 991,
  "EXITCODE": 0
},
"wlcg_metapackage": {
  "EXECUTION_TIME": 617,
  "WCLG_METAPACKAGE": "HEP_OSlibs-7.3.1-2.el7.cern.x86_64",
  "EXITCODE": 0
},
"isolcpus_checking": {
  "ISOLATED_CPUS": "",
  "EXECUTION_TIME": 77,
  "EXITCODE": 1
},
"processes_visibility": {
  "EXECUTION_TIME": 708,
  "USERS_COUNT": 12,
  "EXITCODE": 0
},
"connectivity": {
  "CURL_IPv6_Status": 7,
  "IPv4_ICMP_Status": 0,
  "EXECUTION_TIME": 1681,
  "EXITCODE": 2,
  "CURL_IPv4_Status": 0,
  "IPv6_ICMP_Status": 2,
  "IPv4_ICMP_avg_rtt": 34.811
},
"lsb_release": {
  "LSB_RELEASE": "CentOS Linux release 7.9.2009 (Core)",
  "EXECUTION_TIME": 298,
  "EXITCODE": 0
},
"ulimit": {
```

```
    "ulimit_current_max_memory_size": 999999999,
    "ulimit_hard_virtual_memory": 999999999,
    "ulimit_hard_max_user_processes": 256612,
    "ulimit_hard_max_memory_size": 999999999,
    "ulimit_nropen": 1048576,
    "EXITCODE": 0,
    "ulimit_current_core_file_size": 999999999,
    "ulimit_current_scheduling_priority": 0,
    "ulimit_hard_open_files": 32768,
    "ulimit_hard_stack_size": 999999999,
    "ulimit_hard_pending_signals": 256612,
    "EXECUTION_TIME": 662,
    "ulimit_current_pipe_size": 8,
    "ulimit_current_data_seg_size": 999999999,
    "ulimit_hard_core_file_size": 999999999,
    "ulimit_current_virtual_memory": 999999999,
    "ulimit_hard_POSIX_message_queues": 819200,
    "ulimit_current_max_user_processes": 256612,
    "ulimit_current_cpu_time": 999999999,
    "ulimit_current_stack_size": 999999999,
    "ulimit_hard_pipe_size": 8,
    "ulimit_current_file_locks": 999999999,
    "ulimit_hard_max_locked_memory": 64,
    "ulimit_current_max_locked_memory": 64,
    "ulimit_hard_data_seg_size": 999999999,
    "ulimit_current_file_size": 999999999,
    "ulimit_hard_scheduling_priority": 0,
    "ulimit_current_pending_signals": 256612,
    "ulimit_filemax": 6472526,
    "ulimit_hard_cpu_time": 999999999,
    "ulimit_hard_real-time_priority": 0,
    "ulimit_current_POSIX_message_queues": 819200,
    "ulimit_hard_file_size": 999999999,
    "ulimit_current_open_files": 32768,
    "ulimit_current_real-time_priority": 0,
    "ulimit_hard_file_locks": 999999999
  },
  "home": {
    "HOME": "/mnt/scratch/arc/gMJNDm2zm",
    "EXECUTION_TIME": 118,
```

```
      "EXITCODE": 0
    },
    "loop_devices": {
      "shared_loop_devices": "no",
      "EXECUTION_TIME": 807,
      "EXITCODE": 0,
      "max_loop_devices": 256
    },
    "derived_fileds": {
      "ulimit_hard_max_user_processes_per_core": 64153,
      "ulimit_filemax_per_core": 1618131,
      "ulimit_hard_openfiles_per_core": 8192,
      "ulimit_nropen_per_core": 262144,
      "ulimit_current_max_user_processes_per_core": 64153,
      "ulimit_current_openfiles_per_core": 8192
    },
    "max_namespaces": {
      "MAX_NAMESPACES": 10000,
      "EXECUTION_TIME": 302,
      "EXITCODE": 0
    },
    "cgroups2_checking": {
      "CGROUPSv2_AVAILABLE": false,
      "EXECUTION_TIME": 412,
      "CGROUPSv2_RUNNING": false,
      "EXITCODE": 0
    }
  },
  "host": "gateway",
  "last_updated": 1682503048,
  "hostname": "wn59.nipne.ro"
},
"fields": {
  "last_updated": [
    "2023-04-26T09:57:28.000Z"
  ],
  "@timestamp": [
    "2023-04-26T09:57:28.000Z"
  ]
},
```

```
  "sort": [
    1682503048000
  ]
}
```

APPENDIX B

SITE SONAR V2.0 COMPONENT TEMPLATE

This appendix presents the complete component template that is used to index Site Sonar v2.0 data in Elasticsearch.

```
{
  "component_templates" : [
    {
      "name" : "new_mapping_sitesonar_template",
      "component_template" : {
        "template" : {
          "mappings" : {
            "properties" : {
              "hostname" : {
                "type" : "text",
                "fields" : {
                  "keyword" : {
                    "ignore_above" : 256,
                    "type" : "keyword"
                  }
                }
              },
              "last_updated" : {
                "format" : "epoch_second",
                "type" : "date"
              },
              "port" : {
                "type" : "integer"
              },
              "test_results_json" : {
                "properties" : {
                  "wlcg_metapackage" : {
                    "properties" : {
                      "EXECUTION_TIME" : {
                        "type" : "integer"
                      },
                      "WCLG_METAPACKAGE" : {
                        "type" : "text",
```

```
      "fields" : {
        "keyword" : {
          "ignore_above" : 256,
          "type" : "keyword"
        }
      }
    },
    "EXITCODE" : {
      "type" : "integer"
    }
  }
},
"lhcbmarks" : {
  "properties" : {
    "LHCbMarks" : {
      "type" : "float"
    },
    "EXECUTION_TIME" : {
      "type" : "integer"
    },
    "EXITCODE" : {
      "type" : "integer"
    }
  }
},
"ram_info" : {
  "properties" : {
    "RAM_kB_Inactive(file)" : {
      "type" : "long"
    },
    "RAM_kB_AnonHugePages" : {
      "type" : "long"
    },
    "RAM_kB_Bounce" : {
      "type" : "long"
    },
    "RAM_kB_Hugepagesize" : {
      "type" : "long"
    },
    "RAM_kB_Active(anon)" : {
```

```
    "type" : "long"
  },
  "RAM_kB_SReclaimable" : {
    "type" : "long"
  },
  "RAM_kB_WritebackTmp" : {
    "type" : "long"
  },
  "RAM_kB_FileHugePages" : {
    "type" : "long"
  },
  "RAM_kB_SwapCached" : {
    "type" : "long"
  },
  "RAM_kB_KReclaimable" : {
    "type" : "long"
  },
  "RAM_kB_Shmem" : {
    "type" : "long"
  },
  "RAM_kB_Inactive(anon)" : {
    "type" : "long"
  },
  "EXECUTION_TIME" : {
    "type" : "integer"
  },
  "RAM_kB_CmaTotal" : {
    "type" : "long"
  },
  "RAM_kB_Active(file)" : {
    "type" : "long"
  },
  "RAM_kB_MemAvailable" : {
    "type" : "long"
  },
  "RAM_kB_PageTables" : {
    "type" : "long"
  },
  "RAM_kB_DirectMap2M" : {
    "type" : "long"
```

```
},
"RAM_kB_Unevictable" : {
  "type" : "long"
},
"RAM_kB_MemFree" : {
  "type" : "long"
},
"RAM_kB_KernelStack" : {
  "type" : "long"
},
"RAM_kB_Mlocked" : {
  "type" : "long"
},
"RAM_kB_HardwareCorrupted" : {
  "type" : "long"
},
"RAM_kB_NFS_Unstable" : {
  "type" : "long"
},
"RAM_kB_ShmemPmdMapped" : {
  "type" : "long"
},
"RAM_kB_AnonPages" : {
  "type" : "long"
},
"RAM_kB_VmallocChunk" : {
  "type" : "long"
},
"RAM_kB_SwapTotal" : {
  "type" : "long"
},
"EXITCODE" : {
  "type" : "integer"
},
"RAM_kB_Dirty" : {
  "type" : "long"
},
"RAM_kB_SwapFree" : {
  "type" : "long"
},
```

```
"RAM_HugePages_Free" : {
  "type" : "long"
},
"RAM_kB_Buffers" : {
  "type" : "long"
},
"RAM_kB_CmaFree" : {
  "type" : "long"
},
"RAM_kB_Cached" : {
  "type" : "long"
},
"RAM_kB_DirectMap1G" : {
  "type" : "long"
},
"RAM_kB_DirectMap4k" : {
  "type" : "long"
},
"RAM_kB_VmallocUsed" : {
  "type" : "long"
},
"RAM_kB_FilePmdMapped" : {
  "type" : "long"
},
"RAM_kB_SUnreclaim" : {
  "type" : "long"
},
"RAM_kB_Hugetlb" : {
  "type" : "long"
},
"RAM_kB_VmallocTotal" : {
  "type" : "long"
},
"RAM_kB_Writeback" : {
  "type" : "long"
},
"RAM_HugePages_Total" : {
  "type" : "long"
},
"RAM_kB_Active" : {
```

```
        "type" : "long"
      },
      "RAM_HugePages_Surp" : {
        "type" : "long"
      },
      "RAM_kB_Slab" : {
        "type" : "long"
      },
      "RAM_kB_MemTotal" : {
        "type" : "long"
      },
      "RAM_kB_Percpu" : {
        "type" : "long"
      },
      "RAM_kB_Committed_AS" : {
        "type" : "long"
      },
      "RAM_kB_Inactive" : {
        "type" : "long"
      },
      "RAM_HugePages_Rsvd" : {
        "type" : "long"
      },
      "RAM_kB_CommitLimit" : {
        "type" : "long"
      },
      "RAM_kB_Mapped" : {
        "type" : "long"
      },
      "RAM_kB_ShmemHugePages" : {
        "type" : "long"
      }
    }
  },
  "cpu_info" : {
    "properties" : {
      "CPU_bugs" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
```

```
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "CPU_apicid" : {
    "type" : "integer"
  },
  "CPU_clflush_size" : {
    "type" : "integer"
  },
  "CPU_cpu_cores" : {
    "type" : "integer"
  },
  "CPU_cpu_family" : {
    "type" : "integer"
  },
  "CPU_lflush_size" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "CPU_wp" : {
    "type" : "boolean"
  },
  "CPU_power_management" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "EXECUTION_TIME" : {
    "type" : "integer"
```

```
},
"CPU_cpuid_level" : {
  "type" : "integer"
},
"CPU_microcode" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"CPU_cache_alignment" : {
  "type" : "integer"
},
"CPU_processor" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"CPU_stepping" : {
  "type" : "integer"
},
"CPU_cache_size" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"CPU_TLB_size" : {
  "type" : "text",
  "fields" : {
```

```
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "CPU_initial_apicid" : {
    "type" : "integer"
  },
  "CPU_cpu_MHz" : {
    "type" : "float"
  },
  "CPU_fpu" : {
    "type" : "boolean"
  },
  "CPU_address_sizes" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "CPU_p" : {
    "type" : "boolean"
  },
  "EXITCODE" : {
    "type" : "integer"
  },
  "CPU_fpu_exception" : {
    "type" : "boolean"
  },
  "CPU_flags" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
```

```
    }
  },
  "CPU_model_name" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "CPU_processor_count" : {
    "type" : "long"
  },
  "CPU_physical_id" : {
    "type" : "integer"
  },
  "CPU_siblings" : {
    "type" : "integer"
  },
  "CPU_core_id" : {
    "type" : "integer"
  },
  "CPU_model" : {
    "type" : "integer"
  },
  "CPU_bogomips" : {
    "type" : "float"
  },
  "CPU_pu_exception" : {
    "type" : "boolean"
  },
  "CPU_hysical_id" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
```

```
      },
      "CPU_vendor_id" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      }
    },
    "singularity_debug" : {
      "properties" : {
        "SINGULARITY_CHECK_SECCOMP" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "EXECUTION_TIME" : {
          "type" : "integer"
        },
        "SINGULARITY_CVMFS_DEBUG" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "SINGULARITY_LOCAL_SUPPORTED_BOOL" : {
          "type" : "boolean"
        },
        "SINGULARITY_LOCAL_DEBUG" : {
          "type" : "text",
```

```
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "EXITCODE" : {
        "type" : "integer"
      },
      "SINGULARITY_CVMFS_SUPPORTED_BOOL" : {
        "type" : "boolean"
      }
    }
  },
  "gcc_version" : {
    "properties" : {
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      },
      "GCC_VERSION" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      }
    }
  },
  "max_namespaces" : {
    "properties" : {
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "MAX_NAMESPACES" : {
```

```
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
  },
  "underlay" : {
    "properties" : {
      "UNDERLAY_ENABLED" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
  },
  "cgroups2_checking" : {
    "properties" : {
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "CGROUPSv2_AVAILABLE" : {
        "type" : "boolean"
      },
      "CGROUPSv2_RUNNING" : {
        "type" : "boolean"
      },
      "EXITCODE" : {
        "type" : "integer"
      }
```

```
      }
    },
    "taskset_other_processes" : {
      "properties" : {
        "PERCENTAGE_PINNED_CPUS" : {
          "type" : "float"
        },
        "ENTRY_COUNT" : {
          "type" : "integer"
        },
        "EXECUTION_TIME" : {
          "type" : "integer"
        },
        "EXITCODE" : {
          "type" : "integer"
        },
        "ENTRIES" : {
          "type" : "text"
        }
      }
    },
    "container_enabled" : {
      "properties" : {
        "SINGULARITY_BINDPATH" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "SINGULARITYENV_PANDA_HOSTNAME" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
```

```
},
"SINGULARITY_COMMAND" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"SINGULARITY_BIND" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"EXECUTION_TIME" : {
  "type" : "integer"
},
"SINGULARITY_NAME" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"SINGULARITY_CONTAINER" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
```

```
        "SINGULARITY_ENVIRONMENT" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "SINGULARITYENV_FRONTIER_LOG_FILE" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "EXITCODE" : {
          "type" : "integer"
        }
      }
    },
    "cpulimit_checking" : {
      "properties" : {
        "ALLOCATED_CPUS" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "EXECUTION_TIME" : {
          "type" : "integer"
        },
        "ACCOUNTING" : {
          "type" : "text",
          "fields" : {
```

```
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "ACCESS_QUOTA" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "ACCESS_PERIOD" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "EXITCODE" : {
          "type" : "integer"
        },
        "CGROUP" : {
          "type" : "text"
        }
      }
    },
    "tmp" : {
      "properties" : {
        "EXECUTION_TIME" : {
          "type" : "integer"
        },
        "TEMP_DIR" : {
          "type" : "text",
          "fields" : {
```

```
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
  },
  "singularity" : {
    "properties" : {
      "SINGULARITY_CVMFS_SUPPORTED" : {
        "type" : "boolean"
      },
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      },
      "SINGULARITY_LOCAL_SUPPORTED" : {
        "type" : "boolean"
      }
    }
  },
  "taskset_own_process" : {
    "properties" : {
      "USING_PINNING" : {
        "type" : "integer"
      },
      "TASKSET" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
```

```
    },
    "EXECUTION_TIME" : {
      "type" : "integer"
    },
    "PERCENTAGE_PINNED" : {
      "type" : "float"
    },
    "EXITCODE" : {
      "type" : "integer"
    }
  }
},
"cvmfs_cache_size" : {
  "properties" : {
    "EXECUTION_TIME" : {
      "type" : "integer"
    },
    "CVMFS_CACHE_SIZE" : {
      "type" : "integer"
    },
    "EXITCODE" : {
      "type" : "integer"
    }
  }
},
"get_jdl_cores" : {
  "properties" : {
    "ALIEN_JDL_CPUCORES" : {
      "type" : "integer"
    },
    "EXECUTION_TIME" : {
      "type" : "integer"
    },
    "EXITCODE" : {
      "type" : "integer"
    }
  }
},
"os" : {
  "properties" : {
```

```
"OS_REDHAT_SUPPORT_PRODUCT" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"OS_SUPPORT_URL" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"OS_REDHAT_BUGZILLA_PRODUCT_VERSION" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"OS_VERSION_ID" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"OS_CPE_NAME" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
```

```
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "OS_NAME" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "OS_HOME_URL" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "OS_PRETTY_NAME" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "EXITCODE" : {
    "type" : "integer"
  },
  "OS_REDHAT_BUGZILLA_PRODUCT" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
```

```
        "type" : "keyword"
      }
    }
  },
  "OS_VERSION_CODENAME" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "OS_REDHAT_SUPPORT_PRODUCT_VERSION" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "OS_ID" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
  "OS_CENTOS_MANTISBT_PROJECT_VERSION" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
        "ignore_above" : 256,
        "type" : "keyword"
      }
    }
  },
```

```
"OS_VERSION" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"EXECUTION_TIME" : {
  "type" : "integer"
},
"OS_ID_LIKE" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"OS_BUG_REPORT_URL" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"OS_ANSI_COLOR" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "ignore_above" : 256,
      "type" : "keyword"
    }
  }
},
"OS_PRIVACY_POLICY_URL" : {
```

```
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "OS_CENTOS_MANTISBT_PROJECT" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "OS_UBUNTU_CODENAME" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      }
    }
  },
  "overlay" : {
    "properties" : {
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "OVERLAY_ENABLED" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
```

```
        }
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
  },
  "uname" : {
    "properties" : {
      "UNAME" : {
        "type" : "text"
      }
    }
  },
  "derived_fileds" : {
    "properties" : {
      "ulimit_current_openfiles_per_core" : {
        "type" : "integer"
      },
      "ulimit_nropen_per_core" : {
        "type" : "integer"
      },
      "ulimit_hard_openfiles_per_core" : {
        "type" : "integer"
      },
      "ulimit_current_max_user_processes_per_core"
        "type" : "integer"
      },
      "ulimit_hard_max_user_processes_per_core" : {
        "type" : "integer"
      },
      "ulimit_filemax_per_core" : {
        "type" : "integer"
      }
    }
  },
  "isolcpus_checking" : {
    "properties" : {
      "EXECUTION_TIME" : {
        "type" : "integer"
```

```
      },
      "ISOLATED_CPUS" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
  },
  "cpuset_checking" : {
    "properties" : {
      "CPU_AMOUNT" : {
        "type" : "integer"
      },
      "CPUSET_CGROUP" : {
        "type" : "text"
      },
      "CPUSET_ENABLED" : {
        "type" : "boolean"
      },
      "CPUSET_CPUS" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
```

```
      },
      "CPUSET_PREFIX" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      }
    },
    "ulimit" : {
      "properties" : {
        "ulimit_current_stack_size" : {
          "type" : "integer"
        },
        "ulimit_current_max_user_processes" : {
          "type" : "integer"
        },
        "ulimit_current_max_locked_memory" : {
          "type" : "integer"
        },
        "ulimit_hard_max_memory_size" : {
          "type" : "integer"
        },
        "ulimit_hard_core_file_size" : {
          "type" : "integer"
        },
        "ulimit_hard_file_size" : {
          "type" : "integer"
        },
        "ulimit_hard_cpu_time" : {
          "type" : "integer"
        },
        "ulimit_filemax" : {
          "type" : "integer"
        },
        "ulimit_current_data_seg_size" : {
          "type" : "integer"
```

```
},
"ulimit_current_max_memory_size" : {
  "type" : "integer"
},
"ulimit_hard_max_user_processes" : {
  "type" : "integer"
},
"EXECUTION_TIME" : {
  "type" : "integer"
},
"ulimit_nropen" : {
  "type" : "integer"
},
"ulimit_current_file_size" : {
  "type" : "integer"
},
"ulimit_current_virtual_memory" : {
  "type" : "integer"
},
"ulimit_current_open_files" : {
  "type" : "integer"
},
"ulimit_hard_max_locked_memory" : {
  "type" : "integer"
},
"ulimit_hard_virtual_memory" : {
  "type" : "integer"
},
"EXITCODE" : {
  "type" : "integer"
},
"ulimit_current_pipe_size" : {
  "type" : "integer"
},
"ulimit_hard_data_seg_size" : {
  "type" : "integer"
},
"ulimit_hard_pipe_size" : {
  "type" : "integer"
},
```

```
      "ulimit_current_cpu_time" : {
        "type" : "integer"
      },
      "ulimit_current_core_file_size" : {
        "type" : "integer"
      },
      "ulimit_hard_open_files" : {
        "type" : "integer"
      },
      "ulimit_hard_stack_size" : {
        "type" : "integer"
      }
    }
  },
  "process_visibility" : {
    "properties" : {
      "USERS_COUNT" : {
        "type" : "integer"
      },
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
  },
  "loop_devices" : {
    "properties" : {
      "shared_loop_devices" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      },
      "EXECUTION_TIME" : {
        "type" : "integer"
```

```
      },
      "max_loop_devices" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
  },
  "home" : {
    "properties" : {
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      },
      "HOME" : {
        "type" : "text",
        "fields" : {
          "keyword" : {
            "ignore_above" : 256,
            "type" : "keyword"
          }
        }
      }
    }
  },
  "cvmfs_version" : {
    "properties" : {
      "CVMFS_VERSION" : {
        "type" : "integer"
      },
      "EXECUTION_TIME" : {
        "type" : "integer"
      },
      "EXITCODE" : {
        "type" : "integer"
      }
    }
```

```
},
"vulnerabilities" : {
  "properties" : {
    "EXECUTION_TIME" : {
      "type" : "integer"
    },
    "kernel_vulnerabilities" : {
      "properties" : {
        "output" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        },
        "name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "ignore_above" : 256,
              "type" : "keyword"
            }
          }
        }
      }
    },
    "EXITCODE" : {
      "type" : "integer"
    },
    "kernel_flags" : {
      "type" : "text"
    }
  }
},
"running_container" : {
  "properties" : {
    "EXECUTION_TIME" : {
      "type" : "integer"
```

```
          },
          "RUNNING_IN" : {
            "type" : "text",
            "fields" : {
              "keyword" : {
                "ignore_above" : 256,
                "type" : "keyword"
              }
            }
          },
          "EXITCODE" : {
            "type" : "integer"
          }
        }
      },
      "lsb_release" : {
        "properties" : {
          "LSB_RELEASE" : {
            "type" : "text",
            "fields" : {
              "keyword" : {
                "ignore_above" : 256,
                "type" : "keyword"
              }
            }
          },
          "EXECUTION_TIME" : {
            "type" : "integer"
          },
          "EXITCODE" : {
            "type" : "integer"
          }
        }
      }
    }
  },
  "ce_name" : {
    "type" : "text",
    "fields" : {
      "keyword" : {
```

```
                    "ignore_above" : 256,
                    "type" : "keyword"
                  }
                }
              },
              "host" : {
                "type" : "text",
                "fields" : {
                  "keyword" : {
                    "ignore_above" : 256,
                    "type" : "keyword"
                  }
                }
              },
              "test_code" : {
                "type" : "integer"
              },
              "addr" : {
                "type" : "ip"
              },
              "host_id" : {
                "type" : "long"
              }
            }
          }
        }
      }
    }
  ]
}
```