UNIVERSITY OF BERGEN
DEPARTMENT OF INFORMATICS

WESTERN UNIVERSITY OF APPLIED SCIENCES
DEPARTMENT OF COMPUTER SCIENCE, ELECTRICAL ENGINEERING AND MATHEMATICAL SCIENCES

Site Sonar - A monitoring tool for ALICE's Grid Sites

Author: Even Berge Sandvik

Supervisors: Bjarte Kileng, Håvard Helstrup, Kristin Fanebust Hetland

December, 2021

Abstract

As the data collected by the LHC increases, the demand for computing resources follow. The JAliEn middleware was created to handle the high-throughput of data expected from ALICE in Run 3. However, making the sites in ALICE's Grid ready for the upgrade from the AliEn middleware to the JAliEn middleware is an laborious task. Therefore it was suggested that the JAliEn middleware could run in a containerized environment. The Site Sonar tool, designed using the Design Science Research methodology, will help the JAliEn developers discover what sites are not ready for the upgrade, and which can run a container with JAliEn. Site Sonar gives users the ability to retrieve information from sites, and has become a general tool which can be used for purposes beyond what it was first intended.

Acknowledgements

During the writing of this thesis, I have experienced valuable support and guidance. First, I wish to thank my supervisors Bjarte Kileng, Håvard Helstrup, and Kristin Fanebust Hetland for their guidance and feedback at every stage during my thesis. Your insight and expertise have made this thesis possible, and I'm very grateful to have you as my supervisors. I would also like to give a special thanks to Maksim Melnik Storetvedt, for his valuable guidance, and for always helping me with the many issues I faced during the implementation of Site Sonar and writing my thesis. I am also very grateful for the warm welcome I received from you upon arriving at CERN. I would also like to take the opportunity to thank Costin Grigoras, for guiding me during the development of Site Sonar. It would not have been possible without your expertise. Lastly, I would like to thank Latchezar Betev for showing interest in my work with Site Sonar, and Kalana Wijethunga for working with the data side of Site Sonar.

Even Berge Sandvik
16 December, 2021

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Grid Computing is a technology that focuses on distributed computing to solve large computational problems. This is essential to ALICE (A Large Ion Collider Experiment). When ALICE is running its detectors, it collects a remarkable amount of data from particle collisions. This data requires several data centers for processing, analyzing, and storage. Several countries work together to contribute by giving the ALICE collaboration the computing resources that it needs. They do so by sharing resources to the Grid. The MonALISA (MONitoring Agents using a Large Integrated Services Architecture) system provides a distributed monitoring service [10], which is used by ALICE to monitor and control its operations on the Grid. This thesis discusses how the configurations of the worker nodes in ALICE's Grid are monitored using Site Sonar, a module built upon ALICE's existing monitoring system.

1.1 CERN

CERN (The European Council for Nuclear Research) is situated at the French-Swiss border in the northwestern part of Geneva and houses the Large Hadron Collider (LHC). The Laboratory has become a prime example of international collaboration, uniting people from all over the world [11]. CERN is a research facility specializing in particle physics, which collects data from colliding particles together in an attempt to understand the physics relating to the nature of matter and radiation. A few remarkable achievements accomplished by CERN are the discovery of the Higgs Boson, creating antimatter, and birthing the World Wide Web which is known for revolutionizing communication [12].

1.1.1 The ALICE experiment

There are many projects situated at CERN, one being the ALICE experiment. The ALICE experiment is a 10,000-tonne detector [13] created to observe heavy-ion physics. It sits 56 metres underground near the village of Saint-Genis-Pouilly, France [13]. The ALICE experiment is designed to research the physics of strongly interacting matter in extreme conditions, where a phase of matter called quark-gluon plasma forms [14].

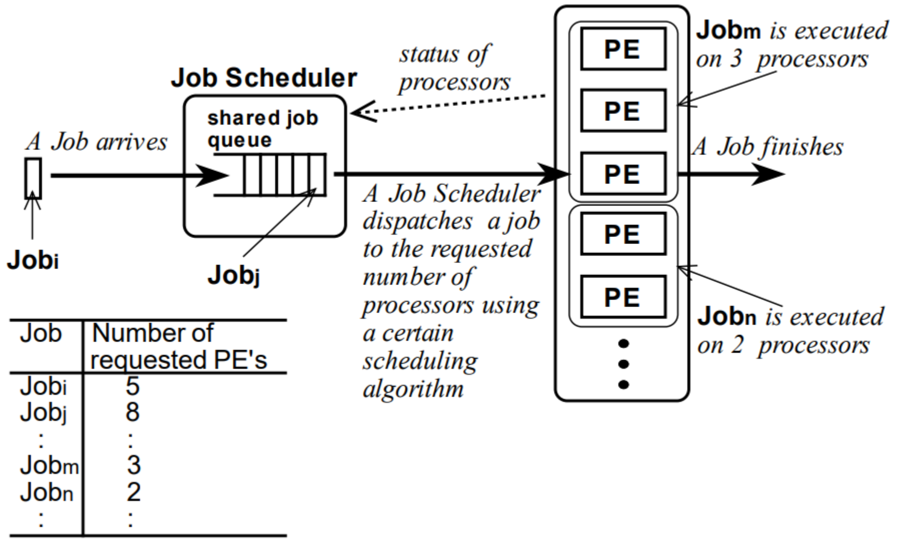
1.2 Data handling on a large scale

The ALICE experiment collects upwards of 4 gigabytes per second (Gbps) of compressed data from heavy-ion collisions. The developers at ALICE have created a *Grid* of data centers, which collaborate on processing and storing the vast amount of data. The AliEn (ALICE Environment) package is a Grid framework used to coordinate the resource sharing of the Grid, also known as a Grid *middleware*. However, the requirements and demands of the resource sharing and storing is increasing due to ongoing updates to the LHC, which in return will increase the workload on the Grid.

There are readily available solutions on the market such as cloud-based services. They offer heterogeneous computational environments with high speeds and performance. However, they are not always a replacement for Grid Computing. Clouds are commercialized and are consequently centralized through a company, giving the users less control on matters such as price, security, APIs, downtime, etc. whereas a Grid is a collaboration between individuals, organizations, and/or institutes where they contribute their available computing resources. Cloud Computing usually requires payment in return for their services. Governments and institutions are usually hesitant to use foreign data centers such as the ones Cloud Computing companies provide. They don't want to be dependent on service through a single Cloud Computing supplier. The centralized structure of the Cloud also hinders the creation of a single open Cloud for scientific use. Grids are collaborative in their nature, where they work together and decide their own regulations and policies. In Cloud Computing, the access to key administrative tasks is usually held by the Cloud Computing supplier, and not available to Cloud Computing users. In a Grid, however, the control is usually in the hand of the Grid sites, and their local administrators. A Grid site may however decide that it wants to use Cloud Computing to deliver its computing services. Thus, the Grid might implement Cloud Computing technologies, where it is up to each individual site to decide how and where they want to get their computing resources.

1.2.1 Grid middleware

Grids were first conceptualized in the 90s, where projects wanted to link supercomputers together to provide computational resources to high-performance applications. This is generally viewed as the first generation of Grids. The second-generation applies a "middleware", which allows for the integration of multiple software applications that runs in a distributed heterogeneous environment. The middleware is responsible for handling allocation, monitoring, and control of Computing Elements (CEs). Additionally, it handles file transfers, authentication, distributing access to information, and application caching.


Job Scheduler
shared job queue
A Job arrives
Job_i
Job_j
status of processors
A Job Scheduler dispatches a job to the requested number of processors using a certain scheduling algorithm
PE
PE
PE
PE
PE
Job_m is executed on 3 processors
A Job finishes
Job_n is executed on 2 processors

Job	Number of requested PE's
Job_i	5
Job_j	8
:	:
Job_m	3
Job_n	2
:	:

Figure 1.1: Illustration of a Job Scheduler [1]

1.2.2 Job scheduler

A job scheduler takes workloads and places them in a job queue (also called a batch queue). The job scheduler runs locally on a Grid node. Figure 1.1 illustrates a job scheduler. It obtains the specification of a job and attempts to put it on an idle *job slot* using a scheduling algorithm. In the figure, the job slot is a processor (PE), but depending on the system, the job slot might be an entire machine or just a processor core. One example of scheduling algorithm is *FCFS* (First Come First Serve), which is a simple form of scheduling the jobs in order of arrival. The scheduling algorithm might also use *Priority Scheduling*, where a certain priority can be given to a job, and the job with the highest priority gets scheduled first [1].

1.3 Grid computing

Dr. Ian Foster is the Director of Argonne's Data Science and Learning Division, as well as a professor in the department of computer science at the University of Chicago [15]. Foster is a prominent voice in the Grid computing community. In his paper *What is the Grid? A Three Point Checklist*, he defines a Grid as the following: "A Grid coordinates resources that are not subject to centralized control", "uses standard, open, general-purpose protocols and interfaces", and "delivers nontrivial qualities of service" [16].

1.3.1 Coordinate computing resources that are not subject to centralized control

A Grid's responsibility is to integrate and coordinate resources and users that live on different control domains. This can for example be a client's desktop versus central computing, where the client's desktop lives outside the central computing domain. Another example is a company that has different administrative units, that need to coordinate with one another [16]. To summarize, a Grid, as opposed to a local management system, works with different units that are otherwise disconnected.

1.3.2 Use standard, open, general-purpose protocols and interfaces

A Grid is built upon standard and open multi-purpose protocols and interfaces adressing matters such as authorization, authentication, resource discovery, and resource access [16]. It is important that the Grid use standardized protocols and interfaces because otherwise, it would be an application-specific system.

1.3.3 Deliver nontrivial qualities of service

The last requirement specified is that a Grid should allow constituent resources to be used in a coordinated way. It should deliver several qualities of service such as good response time, high throughput, security, availability, etc. The goal of a Grid is that the sum of its resources should be more advantageous used in a Grid than not [16].

These qualifications still leave some room for debate. One might for example differ on the definition of "centralized control" [16]. If we try Foster's checklist on ALICE's new Grid middleware, JAliEn (Java ALICE Environment), it will somewhat cover the points. There is some resource coordination of non-centralized resources on every site which are handled by the site's admins, where a component in JAliEn called VOBOX will give users an entrypoint to these resources. However, JAliEn uses centralized systems such as the Central Services, Central Job Queue and the central File Catalouge [17]. The communication protocols in JAliEn are for the most part WebSockets, JSON, and X.509, which are all standard. JAliEn also has some quality assurance of its resources. One example is that JAliEn will try to run the Grid jobs closest to the data that is required by the job execution. Additionally, it checks if any idle worker nodes on a site has the minimum requirements for running the Grid jobs. JAliEn is however meant to be a practical solution for the problems concerning ALICE and is not meant to be a general multi-purpose Grid middleware.

1.4 Grid applications

In his book *Fundamentals of Grid Computing: Theory, Algorithms and Technologies* [18] Frédéric Magoulès categorizes the different types of computing that a Grid can offer. The types are based on the main challenges that are present from the Grid architectures point of view. The 5 main categories are listed below.

Distributed supercomputing - In distributed supercomputing, applications use Grids to aggregate computational resources so that the time to completion of a job is reduced. This is often used to complete problems that can't be solved with a single system.

High-throughput computing - This type of computing allows the Grid to schedule large numbers of loosely coupled or independent tasks. The goal is to put unused processors cycles from idle workstations to work.

On-demand computing - The on-demand applications use the Grid to allow remote Grid resources into local applications. The purpose of this type of usage is to fulfill short-term computational requirements. An application using on-demand computing generally has a dynamic requirement for computing resources.

Data-intensive computing - In data intensive applications the data is maintained in geographically distributed repositories, libraries, and databases. The challenges faced by this sort of Grid application are scheduling and configuring high-volume information that flows through different admin domains and complex multiple level hierarchies [18]. Data-intensive computing is a typical Grid application in high-energy physics experiments.

Collaborative computing - The last category is collaborative computing. This type of Grid application is concerned with human-to-human interactions [18]. These applications typically provide a virtually shared space such as data archives and simulations. Examples of this can be instant messaging or video conferencing.

The most relevant Grid applications for ALICE are the data-intensive and high-throughput categories. Because ALICE has several geographically distributed sites which provide computer resources, and it schedules a large number of independent tasks (known as *Grid jobs*) to the Grid.

1.5 Research methodology

The research methodology used in this thesis focuses on designing and prototyping a software system. In the context of this thesis, the prototype is Site Sonar, a WLCG monitoring tool implemented in AliMonitor.

1.5.1 Design Science research methodology

Design Science Research (DSR) uses two major activities in its method, which are design and investigation of artefacts in its context [19]. The goal of Design Science Research is to design an artefact that solves a practical problem of interest. A practical problem is the gap between the current state and its desirable state [20]. A Design Science project consists of stakeholders, i.e people who may affect or be affected by the project. The stakeholders in this thesis are the developers of the new JAliEn Grid middleware, and the artefact is Site Sonar.

The proposed method framework for DSR has five major steps in its approach that investigates the problem, defines requirements, and demonstrates and evaluates a finished artefact. The five steps are followed in this order [20]:

1. **Explicate problem** - To formulate the initial problem, and investigate the underlying causes. The intention is to answer the question "what problem is experienced by the stakeholders, and why is it important".
2. **Define requirements** - To identify and outline an artefact that can address the explicated problem. The requirements take into account what is important to the stakeholders.
3. **Design and develop artefact** - To create an artefact fulfilling the requirements. This step includes designing both the functionality and the structure of the artefact.
4. **Demonstrate artefact** - To demonstrate the artefact in one use case, thereby proving its feasibility to solve the explicated problem.
5. **Evaluate artefact** - To evaluate how well the artefact is able to solve the problem and to what extent it fulfills the defined requirements.

Figure 1.2 illustrates the activities that are part of DSR, which are the relevance cycle, the design cycle, and the rigor cycle. The relevance cycle tries to identify the opportunities and problems presented in an application. Furthermore, it defines the acceptance criteria for the final evaluation of the research performed. When testing the application it might be necessary to have additional iterations through the relevance cycle. It might be that there are deficiencies in the DSR projects qualities such as its performance and usability, or in its functionalities [21].

The Rigor cycle wants to bring up previous knowledge relevant to the research project. It hopes to lead the DSR project to innovate. Innovations are additions to the knowledge base (KB) as a result of the DSR process. This includes all experiences, methods, and extensions to the original theories created while doing research [21]. Finally, the design cycle is the process of iterating between the construction of an artefact and the evaluation and refinements of the design. Several design alternatives can be created and evaluated based on feedback from the stakeholders until the requirements are set and the artefact is refined to a satisfactory result [21].

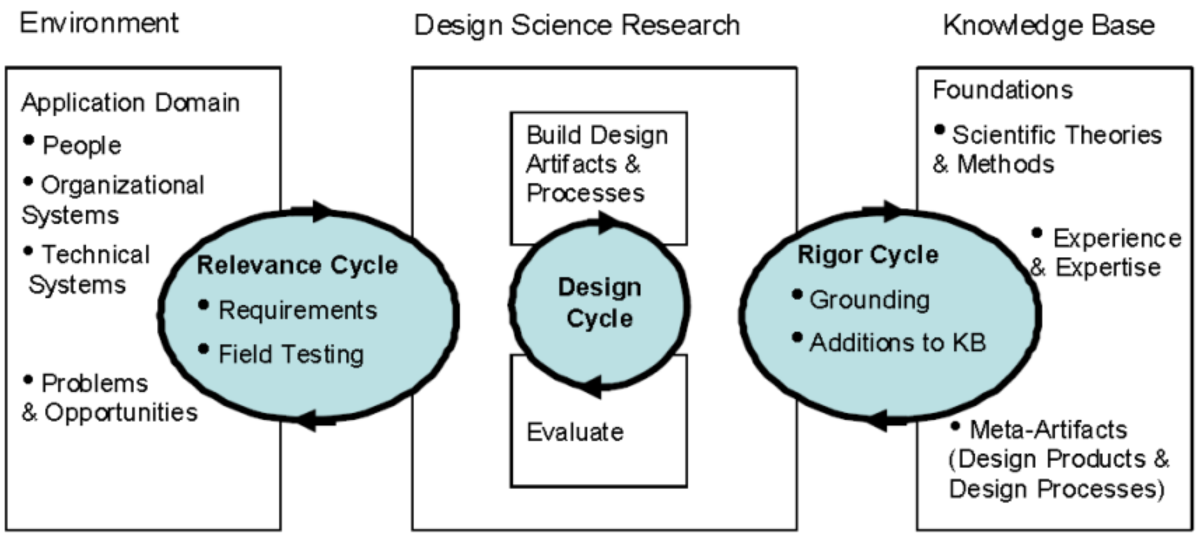


Figure 1.2: Three-Cycle View of Design Science [2]

1.6 Research question

The objective of this thesis is to re-evaluate and implement a new Site Sonar into AliMonitor. Site Sonar contributes to improving the JAliEn developers overview of ALICE's Grid by displaying information from the different sites. To achieve good results, it is essential that the requirements are discussed with the developers of the new JAliEn middleware.

The research questions in this paper are:

Research Question 1 (RQ1): What functionality and qualities are important for Site Sonar to have?

Research Question 2 (RQ2): What kind of monitoring tools exists, and which are comparative to Site Sonar?

Research Question 3 (RQ3): How can monitoring assist the JAliEn developers in distributing the JAliEn middleware?

Research Question 4 (RQ4): How can the graphical user interface design be trivial to use by users that are non knowledgeable of the database structure.

Research Question 5 (RQ5): How should the data collected from nodes on a Grid site be formatted to be practically and efficiently queried.

1.7 Outline

The remainder of the thesis is structured into 6 chapters.

Chapter 2 - Background

This chapter introduces the Grid used by the ALICE to process and analyze it's data. In addition, it covers the middleware the Grid utilizes to employ collecting and distributing so called *Grid jobs*. Furthermore, the chapter explains fundamental concepts used in Grid computing, and how the Grid is structured.

Chapter 3 - Problem Description

This chapter describes in detail the problem description for the thesis. The problem description includes issues that remain with existing solutions, and what requirements and functionality is needed by the stakeholders and users. It serves to give an overview to chapter 4 where the Site Sonar implementation is covered.

Chapter 4 - Site Sonar

This chapter explains the Site Sonar application and it's implementation. Additionally, it covers the improvements and adoptions made from the previous implementation of Site Sonar, and the changes in functionality.

Chapter 5 Evaluation

This chapter describes how software can be evaluated using software quality models. Additionally, the chapter discusses the quality criteria that are relevant to the Site Sonar project.

Chapter 6 - Conclusion

This chapter is a summarization of the thesis and the research questions. Furthermore, it covers how the projects development has used the DSR research methodology, the validity of the research and what Site Sonar has contributed to ALICE.

Chapter 7 - Future Work

The Future Work chapter addresses the ideas that have come along the process of developing Site Sonar, and the functionalities that can be implemented to improve upon it.

Chapter 2

Background

Collecting and monitoring information about the Grid is critical to meet the criteria such as server uptime, performance, and the state of the worker nodes and sites that build up the Grid. The ALICE collaboration uses AliMonitor to monitor the Grid. AliMonitor is a website that provides access to the MonALISA services, which are used to monitor and control Grid operations.

2.1 Large Hadron Collider

The Large Hadron Collider (LHC) is a 27-kilometer long particle accelerator, which makes it the largest and most powerful in the world. It consists of magnets and other structures which accelerate the high-energy particle beams close to the speed of light in an ultrahigh vacuum [22]. The particle beams are traveling in the opposite direction and are made to collide with each other at 4 locations around the accelerator. These locations are the 4 main particle detectors at the LHC, which all observe and collect data from these collisions. Figure 2.1 illustrates the four particle detectors - ALICE, ATLAS, CMS, and LHC-B. This thesis will however only be focused on the ALICE detector.

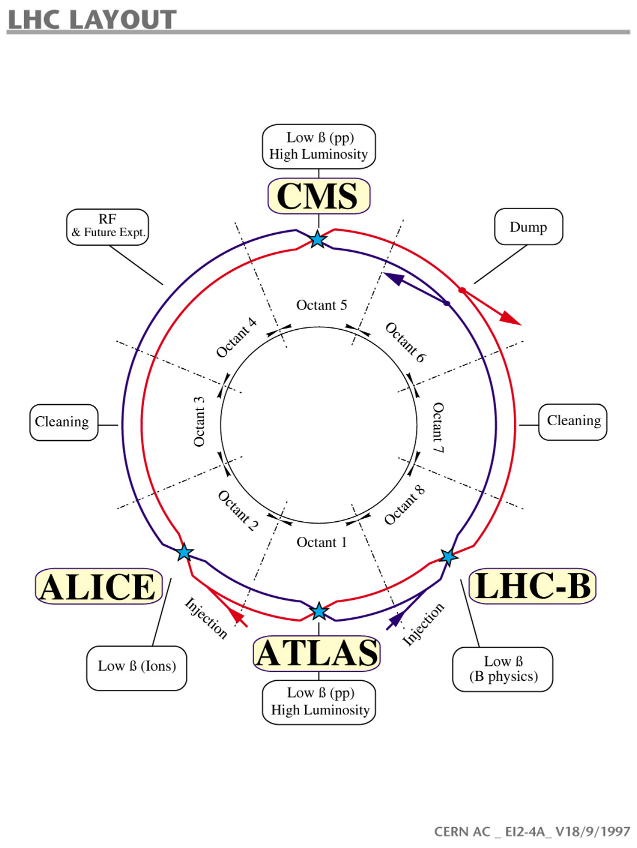


Figure 2.1: Illustration of the Large Hadron Collider layout [3]

2.1.1 ALICE

ALICE (A Large Ion Collider Experiment) uses specialized equipment to analyze colliding particles in the accelerator. There are roughly 1000 scientists working together at ALICE. Together, they are from more than 100 institutes across 30 countries. The ALICE collaboration handles the ALICE experiment, which specializes in studying the extreme conditions thought to have existed in the first millionths of a second after the Big Bang [23]. When operational, the detector receives beams of particles from the LHC [14].

To create the conditions needed for the ALICE experiment, particles called heavy ions have to be accelerated and collide head-on with each other. This creates the extreme conditions of heat and density that ALICE studies. The physics program at ALICE relies on being able to identify what type of particle they are detecting. The process of particle identification (PID) involves measuring the different ways that particles interact with matter. To do this they have specialized instrumentation that can measure these variations. The particles that live long enough to reach the sensitive detector layers are then measured by the ALICE experiment's detectors [24].

2.1.2 Virtual Organizations

Virtual Organizations (VO's) are collaborations between institutions and individuals seeking to solve the same goal. A VO regulates access to resources in the Grid environment, such as computers, equipment, and software. Grid resources are distributed among multiple sites and administrative domains that are geographically dispersed. Consequently, the Grid sites have to communicate through the use of shared networks, which introduces security-related concerns. These sites have to be cautious about illicit collection or modifications of data transfers. Thus, it is important to ensure data integrity and confidentiality. The VO has an authentication and authorization mechanism in place for performing verification that confirms a member's identity. That way a VO can grant members access to resources, files, and projects. In other words, VO's help with administrating Grid resources, and provide a form of access control. This gives limits to users on what computational assets are available to them. Members of a VO will have access to Grid resources and data linked to their respective VO, as long as that VO permits it.

2.2 Computing resource requirements

The Storage Elements (SE) provide storage and access to the data collected from the experiments at the LHC. The Storage Elements can be in the form of a disk or tape. Table 2.1 illustrates the disk and tape usage (in petabytes) for the 4 main detectors at CERN in the last year. Note that the "Disk used" and "Tape used" columns are measures of how much is stored, while the "Disk total" and "Tape total" columns are the maximum capacity available. Other variables that are important to take into consideration besides the available storage space of a SE are the read/write speed, how many failed transfers from or to the SE, and how many files are lost or corrupted[25].

Table 2.1: Storage of the Virtual Organizations (03.12.2021)

Virtual Organizations	**Disk used**	**Disk Total**	**Tape used**	**Tape Total**
ALICE	2.17 PB	2.91 PB	8,58 PB	13.71 PB
ATLAS	4.54 PB	5.98 PB	27.94 PB	33.02 PB
CMS	7.06 PB	8.73 PB	35.72 PB	39.98 PB
LHCb	2.22 PB	3.07 PB	9.28 PB	12.50 PB

2.2.1 LHC Run 3

In December 2018, LHC shut down its 4 year-long operations called Run 2. The experiments are implementing improvements during the ongoing Long Shutdown 2 (LS2)[26]. During LS2, ALICE is implementing new hardware and software improvements to the detector[27]. These installments will increase the data throughput due to the increased LHC luminosity.

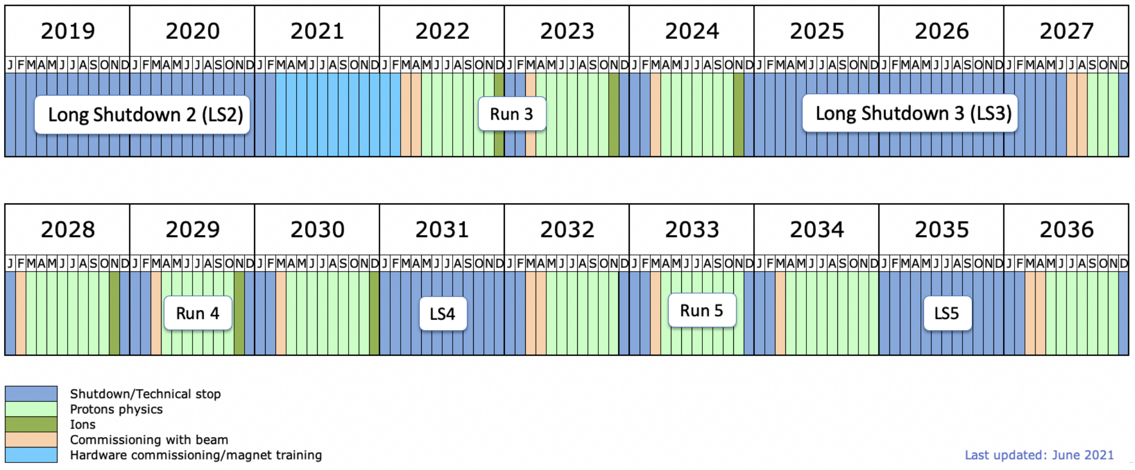


Figure 2.2: Longer term LHC schedule (from June 2021) [4]

When CERN starts up the LHC again in Run 3, the data rates and read-out will increase, and ALICE will collect many-folds more data than it did in Run 2. The ALICE experiment expects to collect 3 terabytes per second (Tbps) of raw data, which is compressed to a more manageable 90 Gbps. Comparatively, during Run 3 the CMS detector expects 50 Tbps of raw data and ATLAS with 5.2 Tbps of raw data. Only a fraction of this data is processed within the internal computing centers at CERN. To deliver enough storage and computing resources, ALICE uses the *Worldwide LHC Computing Grid*.

2.3 Worldwide LHC Computing Grid (WLCG)

WLCG is an organization of collaborating Grids that offers Grid resources. It was developed to cover the demand for computing resources needed by the LHC. WLCG is a global partnership including the European Grid Infrastructure (EGI), Open Science Grid (OSG), and the Nordic e-Infrastructure Collaboration (NeIC) [28]. Together they have roughly 170 computing centers spread across more than 40 countries. WLCG provides access to shared computing and storage resources which allows high-performance computation to its members. CERN provides about 20% of the resources in WLCG [29].

2.3.1 WLCG sites tiered architecture

The ALICE sites are structured into three tiers - Tier 0, Tier 1 and Tier 2. All the computing centers fall into one of the tiers, where each tier has several computing centers. An overview of all the computing centers in WLCG is provided in Figure 2.3. Together, they store, process, and analyze all the data produced by the LHC.

Tier 0 (T0) sites are the CERN data center and the Wigner center in Hungary. Less than 20% [30] of the data of the Grids total computing power is T0 locations. However, all the data from the LHC moves through a T0 site, before being distributed to lower tiers. T0 is responsible for keeping the raw data produced by the LHC and starts the process of reconstructing them into more meaningful and usable information. Both the raw and reconstructed data are distributed to Tier 1 sites.

Tier 1 (T1) consists of 13 computer centres [30] from 12 different countries. These centers store a large portion of the raw data passed on by T0, as well as the reconstructed data. Additionally, they store some of the data from simulations done at the Tier 2 sites. T1 also handles matters such as support, where they provide 24-7 support services for the Grid. It also is responsible for reprocessing data which is then passed to Tier 2 sites.

Tier 2 (T2) sites are usually institutes such as universities or scientific organizations. There are around 155 [30] T2 sites, which supply significant computing power and storage capacity. T2 sites handle a large share of the data and are responsible for running simulations that produce data, and reconstructing the data into meaningful information.

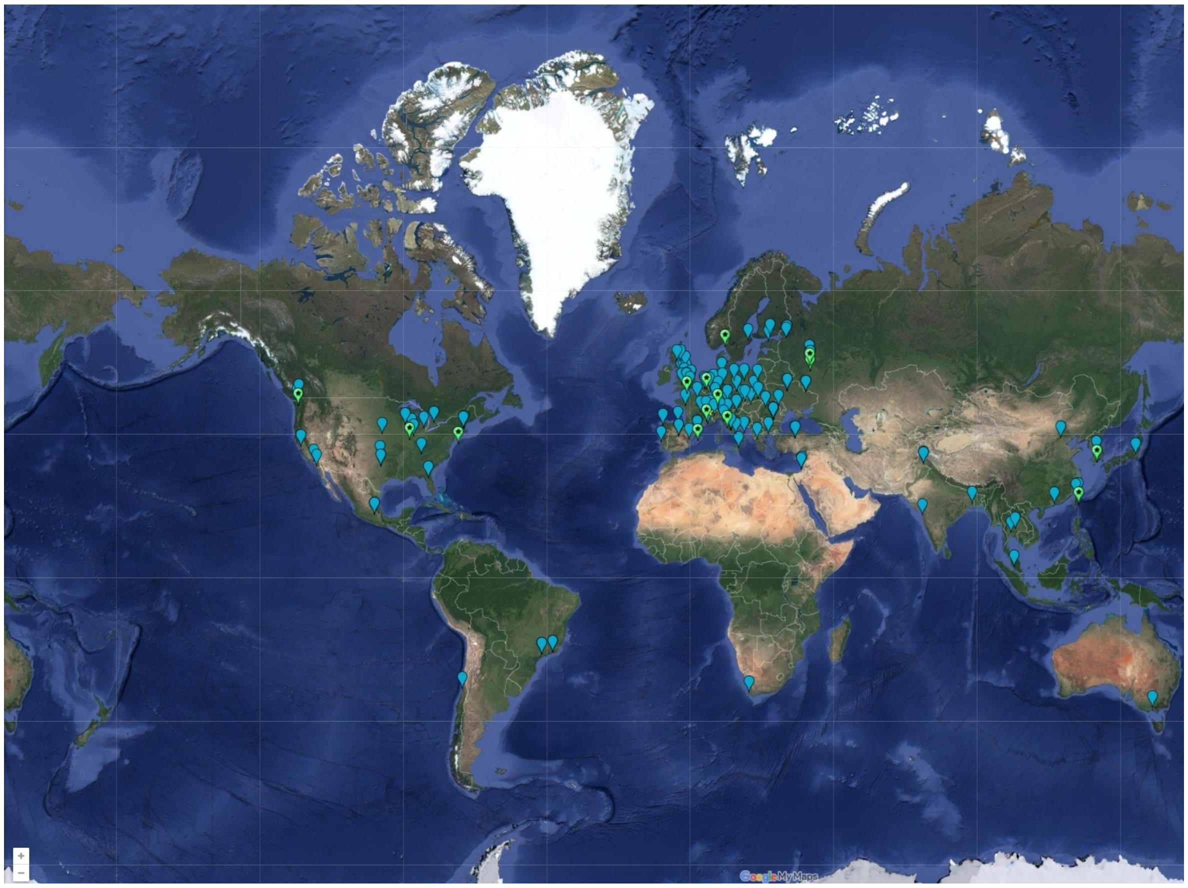


Figure 2.3: Map of all sites in the WLCG Grid [5]

2.4 JAliEn

The ALICE collaboration uses both AliEn (ALICE Environment) and JAliEn (Java ALICE Environment) as their Grid middlewares. They both leverage a heterogeneous environment of computing resources for their users [31]. JAliEn is the new Grid middleware framework evolved from the AliEn framework, and it seeks to modernize all parts of the AliEn middleware [27]. The JAliEn middleware runs on top of the WLCG within ALICE [27], and attempts to fulfil the increased computing needs created by the LHC Run 3 upgrades to the ALICE detector. A requirement set for JAliEn is to support horizontal scaling, meaning that ALICE's resource pools can add more machines. The goal is for JAliEn to be able to handle near exascale data volumes [27].

The AliEn and JAliEn middlewares accept jobs and execute them in the Grid where there are resources that match the Grid job's requirements. It also gives users access to either read or write data results on Storage Elements. Access to the information goes through a set of Central Services that authenticates users and authorizes access to the data found in the database and physical files stored on disk and tape. AliEn and JAliEn store all their experiment files, executable jobs, and scheduled data transfers in its databases [31].

It was decided that the whole AliEn framework should be re-implemented. One of the reasons was that some of the AliEn components were hard to maintain. The AliEn middleware is less complex than JAliEn in its structure, however, it is harder to continue its development. The core functionalities of AliEn are being rewritten in the Java programming language, profiting from the built-in serialization, easy-to-use fully TLS-based communication and object caching among other features [31].

JAliEn has a set of services that provide different functions. First we have the JCentral [27], which are the Central Services that takes care of the job queue and the TaskQueue [27]. The TaskQueue has several functions such as serving job submissions, splitting jobs into smaller sub-tasks, calculation, and control of job quotes for every user, handling the information of the computing centres, registering job outputs, job status transitions, job matching, and job tracing. JCentral is also responsible for data management, where it decides the file placement and file source for all Grid jobs depending on factors such as location of the user, and the topology of the network between the user and the Storage Element [27]. Clients can connect to JAliEn and use a shell prompt to execute commands such as submitting jobs. To log into JAliEn, the client has to use a Grid certificate. Grid certificates are in the format of public key infrastructure (PKI) certificates. These are digital certificates generated by CERN (or other accepted certificate providers) so that users may authorize themselves and gain access to the Grid.

2.4.1 JAliEn Grid job

The definition of a Grid job is a non-interactive application that is set to be executed in a Grid. This can for example be a program (called a *job*) written in a programming language (e.g C, Python, Java), or a package. Figure 2.4 displays how many jobs were running on 04 October 2021, with the average sum being around 17 000 jobs. In JAliEn a job is defined by the Job Description Language (JDL). The JDL file specifies attributes for execution, where an attribute has the format "Attribute = Value;". In a JDL file, some of the attributes are mandatory while others are optional. Listing 2.1 illustrates the fields which are mandatory in a JDL file. It needs to specify the executable file and the type. Additionally one has the option to specify attributes such as the input, output, and error files. The "JobType" attribute tells the job how it should execute, where "Normal" means it should execute sequentially.

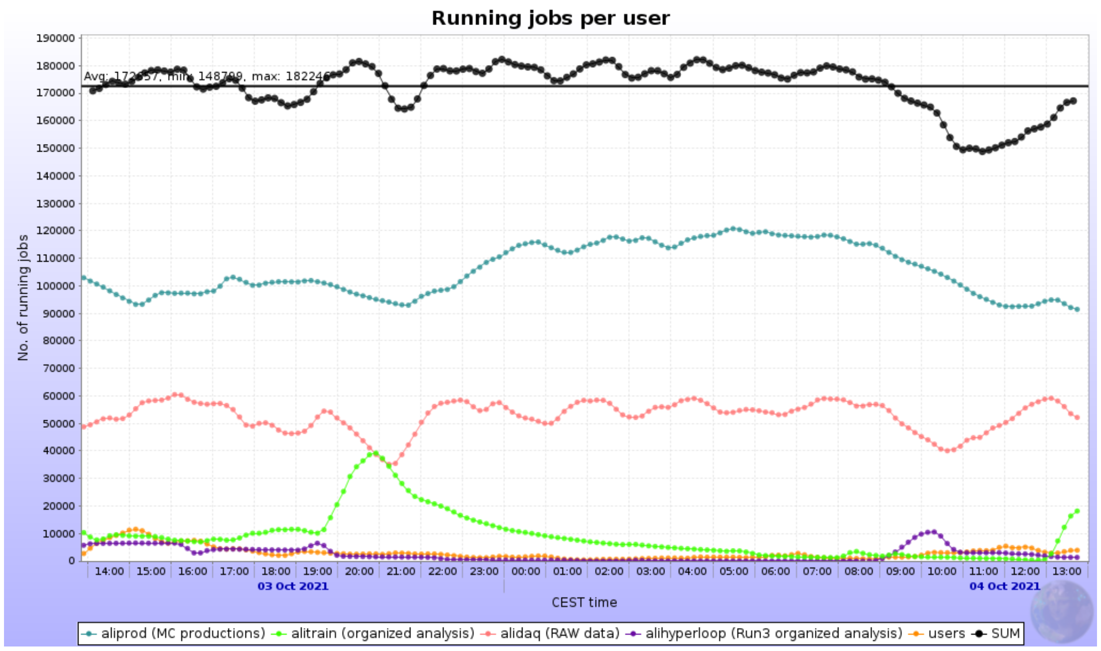


Figure 2.4: Running Grid jobs per user (04.Oct.2021) [6]

A job might require a specific running environment or hardware to be able to execute. These attributes are specified in a JDL file, where a job can give its requirements e.g memory, CPU cores, or the operating system. This ensures that a job isn't executed on a worker node that isn't capable of running it.

Listing 2.1: Mandatory and optional fields in a JDL file

```
# Mandatory attributes
Type = "Job";
Executable = "executable.sh";

# Examples of optional attributes
JobType = "Normal";
StdInput = "stdin.txt";
StdOutput = "stdout.txt";
StdError = "stderr.txt";
```

2.4.2 The JAliEn Job Agent

Job Agents run on worker nodes in the Grid, and are responsible for executing user payloads given to it by the JCentral [27]. The Job Agent is split into two components. The first component *A* has a Pilot Token Certificate. The certificate gives the Job Agent permissions to do job matchings. Component *A* also monitor payload, memory, disk, and CPU usage. The second component *B* of the Job Agent is called the Job Wrapper. The Job Wrapper handles several aspects of the payload's requirements. These requirements are the download input files, creating the job sandbox, setting the environment of the task, launching its execution, and at the end of the execution, uploading the output files and managing their registration in the JAliEn catalouge.

2.4.3 Computing Element

A Computing Element (CE) is a service that gives an entry point for a Grid job to a site. In JAliEn the CE is implemented as a class in Java, which will run with JAliEn when it is in CE mode. The CE aggregates and publishes information from worker nodes [25]. Furthermore, the CE is also responsible for authenticating users and look if there are jobs in the Central Services. When a site's local Batch Queue receives a Job Agent Script from the CE, it will give it to a worker node. A site might have many CE's, but then the site will show multiple times in the list of sites.

2.4.4 Worker node

A worker node is a job slot running under a site, and its task is to receive and execute jobs given to it by the batch queue. In the case of JAliEn, the job is the Job Agent. The environments running on the different sites are mostly heterogeneous, and therefore usually differ on aspects such as Operating System, hardware, package installations, system version, etc. However, worker nodes on the same site usually run the same environments but will vary on dynamic parameters, e.g free memory, available storage, etc. The jobs might therefore specify certain parameters that define the requirements needed from a worker node. This ensures that the job can be executed on the specified worker node. An available worker node may therefore be ignored. The Job Agent won't get the Grid Job from the Central Services if the worker node does not satisfy the requirements specified by a Grid Job.

2.4.5 VOBOX

The computing clusters in the WLCG project generally have at least one instance of VOBOX running. For ALICE this means that every Grid site has one or more VOBOX installed [25]. The VOBOX is a dedicated machine running on a site, that gives an entrypoint for a virtual organization to that particular cluster [32]. Members of virtual organizations get access to sites computing resources through the VOBOX. Additionally, the VOBOX hosts services used by the VO, such as monitoring certificate renewals, and running the Computing Element service. The VOBOX also runs the ClusterMonitor, which transfers monitoring information to the MonALISA framework and the AliEn or JAliEn Computing Element [25]. The MonALISA framework is covered in more detail in section 2.5.1.

2.4.6 JAliEn Grid job execution

JAliEn's architecture has several segments that work together to find a suitable node for a Grid Job. Figure 2.5 illustrates the process of running a Grid Job. The process starts with the VOBOX initializing a Computing Element (CE) by calling JAliEn ComputingElement. The CE first looks for a Grid Job from the Central Services and then gets a valid token with all permissions. The CE creates a Job Agent Script which it passes to the batch queue on a Grid site. The local batch queue will look for an idle resource, called a "job slot", where the Job Agent can run. Different batch queues define job slots differently, where a job slot might be a machine or just a CPU core. The Job Agent script then calls the JAliEn JobAgent function, which retrieves a Grid Job from the Central Services. There is however no guarantee that the original jobs found by the CE are still available. Finally, the Job Agent calls the JAliEn JobWrapper function, which collects all the files needed by the job and starts the job in a container.

2.4.7 Distributing JAliEn with Apptainer

JAliEn is only compatible with some of the current Grid sites. Ideally, all the worker nodes should run JAliEn before Run 3, to handle the high throughput. However, upgrading all the Grid node environments in time is not a feasible solution. Instead, JAliEn might run in a containerization tool. Containers offer a type of operating system virtualization, which can be used

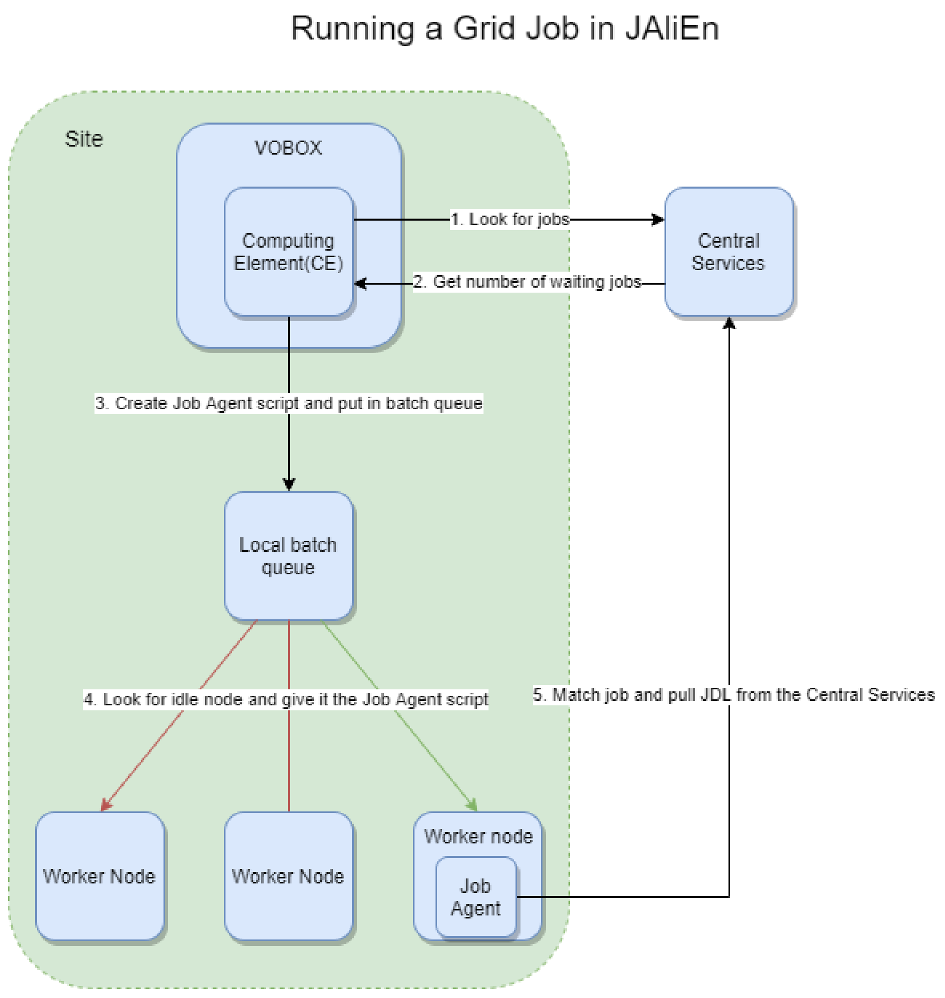

Running a Grid Job in JAliEn
Site
VOBOX
Computing Element(CE)
1. Look for jobs
2. Get number of waiting jobs
Central Services
3. Create Job Agent script and put in batch queue
Local batch queue
4. Look for idle node and give it the Job Agent script
5. Match job and pull JDL from the Central Services
Worker Node
Worker Node
Worker node
Job Agent

Figure 2.5: JAliEn Grid Job process

to run any type of service, software process, or application. All the necessary configuration files, binaries, libraries, and executables are inside the container. Containers are much more lightweight and portable compared to server or machine virtualization approaches, due to not containing an operating system image. There are several container tools, but the ones ALICE has decided to go with are Apptainer [33] (previously called *Singularity* [34]) and Docker [35]. Podman [36] is also suggested as a containerization tool, which similarly to Docker and Apptainer, can build container images. Apptainer is a containerization solution that helps to safely bring the new JAliEn middleware to the Grid nodes which are not yet ready. It is open source and is specifically made to run high-performance computing environments. It also does not require root-level permissions to run the containers, which enables it to control its own soft-

	Apptainer	**Docker**
Supports current production Linux distros	Yes	No
Internal image build/bootstrap	Yes	No
No privileged or trusted deamons	Yes	No
No additional network configurations	Yes	No
No additional hardware	Yes	Maybe
Access to host filesystem	Yes	Yes
Native support for GPU	Yes	No
Native support for InfiniBand	Yes	Yes
Native support for MPI	Yes	Yes
Works with all schedulers	Yes	No
Designed for general scientific use cases	Yes	No
Contained environment has correct perms	Yes	Yes
Containers are portable. unmodified by user	Yes	No
Admins can control and limit capabilites	Yes	No

Table 2.2: Comparing Apptainer, Docker and Podman

ware stack. A container image might either be created locally on a machine, or on Apptainer Hub. The resulting image can then execute JAliEn securely on any Grid node that has Apptainer installed[34].

Table 2.2 compares Apptainer to Docker. The table is based on the paper *Apptainer: Scientific containers for mobility of compute* [33]. Docker allows for access to the host filesystem, but with security implications. In Apptainer this is not an issue, where the host access files are available automatically. Apptainer favors integration instead of isolation, while it still maintains the security restrictions on the container[37].

2.5 Monitoring the WLCG

Monitoring surveys the state of the operations of the Grid, and the status of every Grid node (site) and worker node. It contributes to ensuring that the Grid runs in a condition that enables high performance. Multiple factors should be monitored on each of the distributed computing components, such as performance logging of CPUs, storage devices, etc. Additionally, it is necessary to check server status, determine the progress of an application, and log performance measures. As the Grid grows with more users and applications, it becomes increasingly more important to have good monitoring to handle the increased demand for resources.

2.5.1 AliMonitor

CERN hosts a monitoring system called AliMonitor, which is built on top of a MonALISA client. The interface is built on Tomcat-based dynamic pages. AliMonitor continuously collects monitoring data produced by any of the Grid sites or Central Services and stores them in a PostgreSQL database. The AliMonitor web application uses this data to create a set of pages that displays graphs, tables, lists, etc.

2.5.2 The MonALISA framework

MonALISA is a framework of several services that scale globally. Service in the context of MonALISA is a component that interacts autonomously with other services. The MonALISA framework's main objective is to monitor Grids, networks, and applications running in real-time. It monitors aspects such as system information for computer nodes and clusters, network information, the performance of applications, jobs or services [38]. Furthermore, MonALISA is capable of managing and optimizing the operational performance of Grids and its running applications [7]. Figure 2.6 visualizes the 4 layer architecture of MonALISA, where each layer provides its unique service. MonALISA has a multi-threaded execution engine that lets the services on each layer execute many monitoring tasks simultaneously. The services can also host loosely coupled agents that can (in real-time) analyze and collect information from the monitoring tasks which are then stored in databases locally [7]. Additionally, MonALISA allows for remote event subscriptions, which lets a service subscribe interest to selected sets of event types. Then when such an event occurs, the lookup discovery service will automatically notify all services that have registered interest in the event [7].

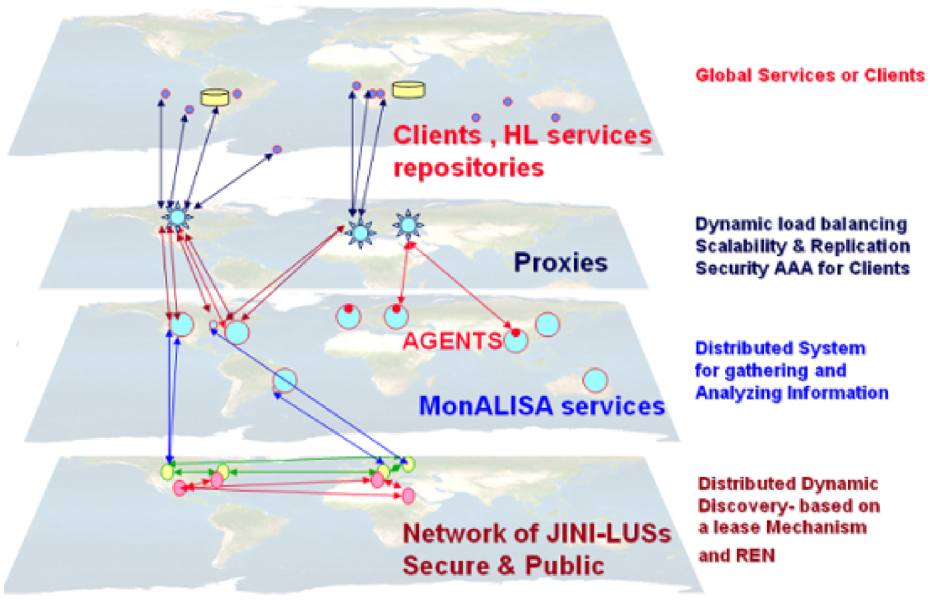

Global Services or Clients
Clients , HL services repositories
Dynamic load balancing
Scalability & Replication
Security AAA for Clients
Proxies
AGENTS
Distributed System for gathering and Analyzing Information
MonALISA services
Distributed Dynamic Discovery- based on a lease Mechanism and REN
Network of JINI-LUSs Secure & Public

Figure 2.6: The MonALISA architecture [7]

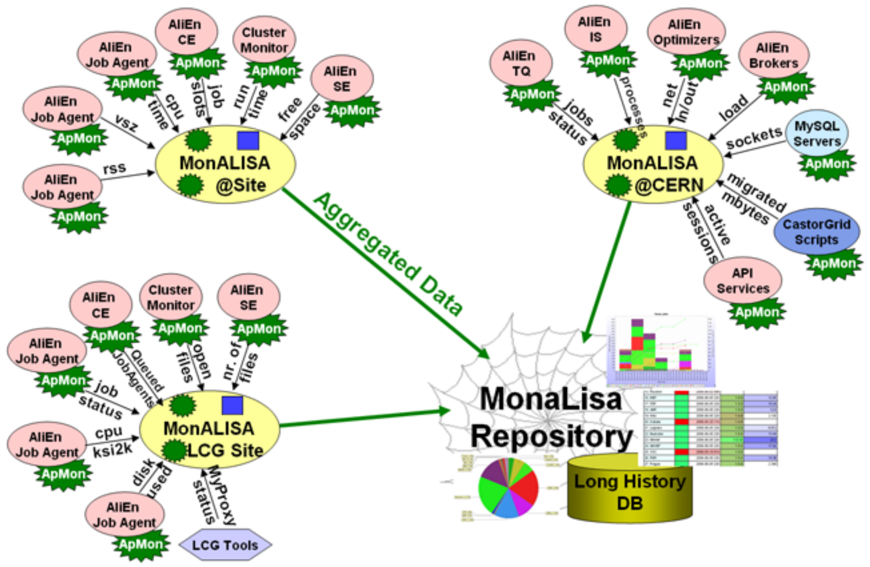


Figure 2.7: Monitoring architecture in JAliEn [8]

The Proxies layer provides intelligent multiplexing of the information requested by clients or other services [7]. It is used for reliable communication between agents. The JINI-Lookup Discovery Services (LUS), is a network that gives a registry of the distributed services. LUS thereby provides the other services and agents with dynamic registration and discovery [7]. This means that a service can interact either through dynamic proxies or agents that use self-describing protocols. The services communicate seamlessly with each other by using the lookup service and discovery and notification mechanisms through LUS [7].

The AliEn and JAliEn middlewares regularly send monitoring data to the MonALISA service running on the site where the middleware instance runs. Figure 2.7 shows the data collected from all the services, jobs, and nodes. This data is aggregated, and stored on the local site for a short period of time. Then the aggregated data is collected by the MonALISA Repository for long-term storage. This data can be viewed in AliMonitor.

Chapter 3

Problem description

Changing all the sites to a viable running environment for JAliEn in time for the start of Run 3 was proven to be a challenging task. Instead of installing a new Linux distribution and setting up a working environment on unready worker nodes, a container could be used. The environment needed to run JAliEn can be containerized beforehand, and then distributed to the Grid nodes. Then a JAliEn Job wrapper can start the job in the container. ALICE has decided to go with either Apptainer, Docker or Podman to provide this containerization. There needs to be a way to view which sites support these containerization platforms so that the ones who do not support them yet can be updated. The solution became Site Sonar, which can give an overview of what the different Grid nodes contain. However, this was only the original problem that started Site Sonar. It has since evolved into being a general tool for querying information from sites and their worker nodes. The worker nodes run programs called *test-scripts* which collect data on many topics involving computing hardware and software.

Site Sonar had a predecessor version, now called the old Site Sonar or Site Sonar v1. It was implemented as a stand-alone react web application. However, the JAliEn developers at ALICE wanted to change some of the functionality of Site Sonar, as well as have it implemented into AliMonitor. For example, the method used to collect worker node information has been changed. The old Site Sonar would send probe jobs into the Grid and wait for a response from the worker nodes. The new Site Sonar has this implemented into the Job Agent, where it will regularly run *test-scripts* on the worker nodes where the Job Agents are running.

3.1 Data

The new implementation of Site Sonar uses two PostgreSQL tables, which are the *sitesonar_tests* table, and the *sitesonar_hosts* table. The data collected from each of the worker nodes by the test-scripts are called *tests*. The name *test* comes from the old version, and is meant to be understood as "What are we **testing** the worker node for?". Some examples of a test are whether the worker node has support for Docker, whether overlay is enabled, etc.

Site Sonar table name	**Description**
sitesonar_tests	This table are the tests on the worker nodes. The columns include the *test_name*, such as os.sh, and the *test_message*. Also included in this table is a JSON formatted version of the *test_message* column. Each test has a relation to a "sitesonar_host".
sitesonar_hosts	All the worker nodes are a "host". Each host has an unique host id, and are connected to a site. Which site the host belongs to is stored in the *ce_name* column.

The test messages returned from the tests contain valuable information. However, its format is in a simple string. To be useful in Site Sonar, we have to be able to extract specific information from the message. For example, the *cpu_info.sh* test is querying CPU information from a worker node. The information returned is a string of CPU specifications. If we wish to count how many worker nodes have 4 CPU cores, it will be challenging to do so from a string where none of the specifications are categorized so that the information can be easily accessed. To solve this issue we preprocess the data to JSON and store it in a JSON column so that a specification can be accessed by using a JSON key. This allows detailed queries from Site Sonar.

3.1.1 Performance efficiency

The Site Sonar database is large, with many tests being outdated. Therefore, it is important to minimize the queries so that the application won't take minutes to load, and thereby be deemed useless by most. Site Sonar has to describe a "cut-off" for what tests should be included in the query, and which should not. This was previously done by selecting a *Run*, which are tests gathered within a specific time period. The new Site Sonar approaches this differently by continuously gathering tests and marking them by when they were gathered. Then when a query is executed, a date (represented by a UNIX timestamp) describes the maximum age of the tests that will be included.

Another aspect that can greatly affect the time performance is the sorting of the worker nodes. Site Sonar is dependent on the nodes being sorted and counted as either included or excluded based on the parameters given in the Site Sonar GUI. Additionally, they have to be sorted into their respective Grid node.

3.2 Site Sonar requirements

The original Site Sonar contains many features. The goal of the new Site Sonar is to create an integrated version of Site Sonar into AliMonitor, and change or add upon the existing functionalities while also simplifying the user interface.

3.2.1 The features in the previous implementation of Site Sonar

The old Site Sonar was implemented using React JS [39]. The querying was very customizable, but the trade-off was that it is difficult to use. The newer version simplifies the querying greatly by implementing certain features, such as having predefined filters in a dropdown menu.

The previous Site Sonar is quite customizable and has a great range of features. Some of the most important features are:

1. Users could list out worker nodes based on the parameters given in the GUI.
2. Users could specify "Run Id" which defines that only tests within a certain 48 hour timeframe should be included.
3. Users could specify the Site ID, which was what site should be queried for its worker nodes.

4. It had support multiple filters, based on alphabetical letters, where the first filter would be "A", the second "B", etc.

5. Allowed user to combine SQL queries with SQL syntax, such as "&" (known as *AND*) in the equation field. It could for example contain "A & B", where the worker node would have to support both filter A and B.

6. The users could export their results to a file.

3.2.2 Requirements for new Site Sonar

The old Site Sonar gave the groundwork and ideas that inspired the new version. However, not all the features in the previous version will be carried on to the new Site Sonar. In the Design Research Cycle, the requirements are discussed regularly with the JAliEn developers. To implement and improve upon the Site Sonar frontend, the goal was to achieve the listed requirements:

1. Display all possible test names and its JSON keys in a table so that users can easily look them up when needed.

2. Filter and group worker nodes, and then sort them into their respective sites. For example grouping them on which worker nodes has support for Docker, and then counting and displaying in a list how many worker nodes support that grouping parameter under each individual site.

3. Filter out worker nodes.

4. Display the worker nodes under a specified site in a list. A user can, for example, select the CERN site, and then see all the underlying worker nodes who have e.g support for Docker.

5. Share results with an URL.

6. Quick load time, and optimized querying of data.

3.3 Research methodology for the development of Site Sonar

The Design Cycle's main focus is to develop an artefact[21]. The process works by iterating between the construction, evaluation, and refinement phases. The artefact in this paper is the Site Sonar monitoring tool, which is implemented into AliMonitor. The DSR Relevance Cycle was based on input from the stakeholders, which helped define the requirements of the application. As mentioned, the stakeholders in the context of Site Sonar are the JAliEn developers at ALICE. The Rigor Cycle was based on the previous artefact (old Site Sonar), which gave a "proof of work". The old Site Sonar gave the theoretical proof and insight into how the new Site Sonar should be implemented. Both these cycles worked as good methods for designing the final Site Sonar artefact.

Chapter 4

Site Sonar

4.1 Overview of Site Sonar

The Site Sonar concept was first developed by Kalana Wijethunga as a standalone React[39] web-application. It was implemented using Gunicorn[40] on the server-side, and hosted on Amazon Web Services. Gunicorn is a Web Server Gateway Interface (WSGI) and serves as a way to run Python web applications. This chapter will discuss the old Site Sonar, and then discuss the reasons behind implementing the new Site Sonar. Further, the chapter will discuss the design choices, technologies used, and the technical aspects of its implementation.

AliMonitor gives many tools to its users where it displays information in a practical manner, such as lists, tables, or graphs. AliMonitor is based on the Apache Tomcat[41], which is a free and open-source framework that creates HTML pages using Java.

4.2 Old Site Sonar and GUI

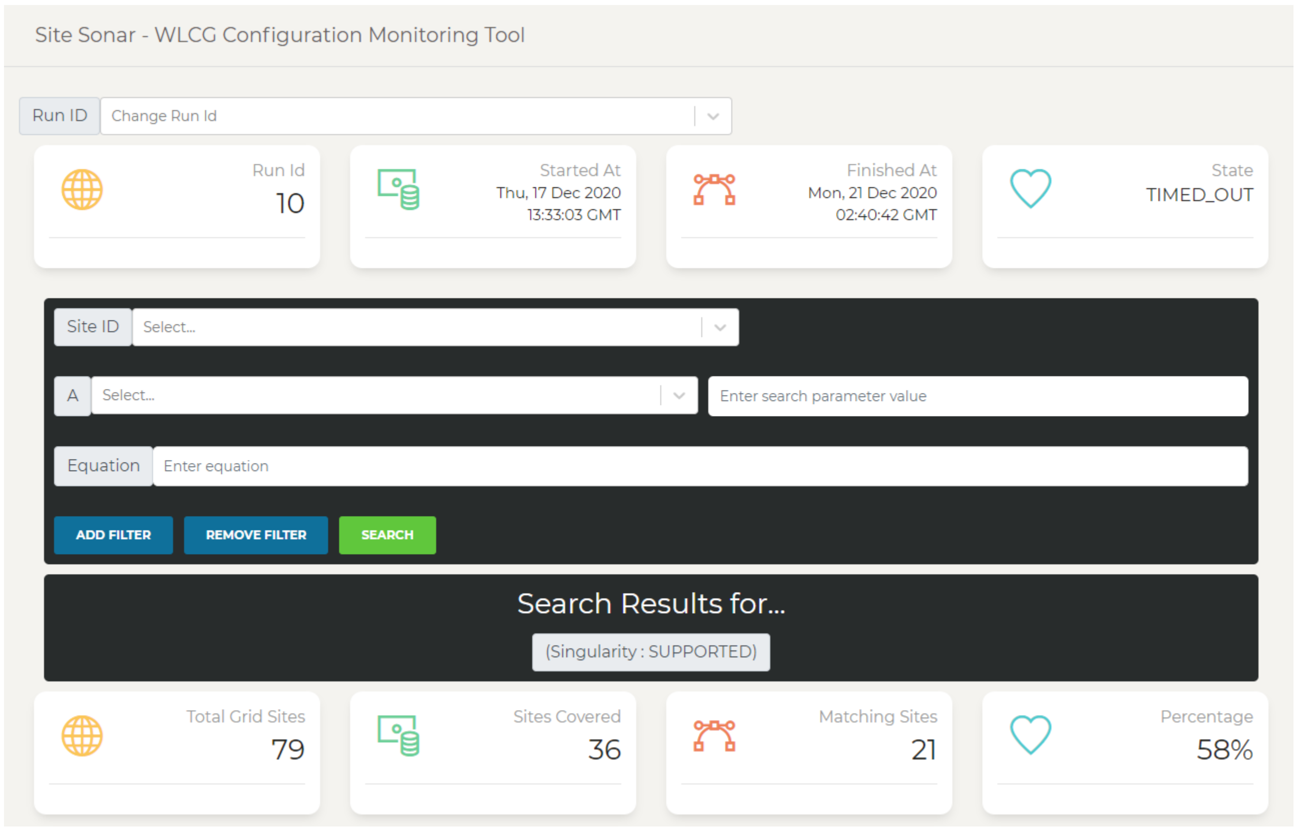
The old Site Sonar is available on github at the following four repositories:

- Site Sonar Backend Mirror[42]
- Site Sonar Mirror[43]

- Site Sonar Frontend Mirror[44]
- Site Sonar ML Client Mirror[45]

The "Site-Sonar-ML-Client-Mirror" repository sends Grid jobs (known as a *probe job*) which are distributed between ALICE's Grid nodes. These jobs read the configurations of the nodes they land on and send the information onward. It does so by using Apmon, which is a library used to transfer monitoring data to the local MonALISA instance. The next repository, "Site-Sonar-ML-client", listens to and receives information from the jobs on every node through Apmon. It also structures the data and puts it in the database. The "Site-Sonar-Backend-Mirror" and the "Site-Sonar-Frontend" repositories read the information which lies in the database and present the data in the web browser.

Figure 4.1 shows a screenshot of the old Site Sonar's interface which can list out worker nodes and sites. The "Site ID" input fields lets an end-user define which site should be included in the query. The "A" input field is a filter that first takes a test name (e.g *os.sh*, *docker.sh*, *cpu_info.sh*, etc), and then a parameter value. An example of a filter could be *singularity.sh* which has the value of "SUPPORTED". The result would then be a list with worker nodes that support Apptainer. The "Equation" input field can be used to combine queries. An example use of the "Equation" is querying all worker nodes that support Apptainer and that have CentOS 7.0. To do such a query, the user has to create one filter "A" for worker nodes that support Apptainer, and an additional filter "B" by pressing the **ADD FILTER** button, with worker nodes that have CentOS 7.0. When there are two filters, the user can combine them by writing "A & B" in the equation field, and then the list will only include worker nodes that apply both filters.


Site Sonar - WLCG Configuration Monitoring Tool

Run ID: Change Run Id

Run Id: 10

Started At: Thu, 17 Dec 2020 13:33:03 GMT

Finished At: Mon, 21 Dec 2020 02:40:42 GMT

State: TIMED_OUT

Site ID: Select...

A: Select... — Enter search parameter value

Equation: Enter equation

Search Results for...

(Singularity : SUPPORTED)

Total Grid Sites: 79

Sites Covered: 36

Matching Sites: 21

Percentage: 58%

Figure 4.1: The old Site Sonar interface

4.3 Design process

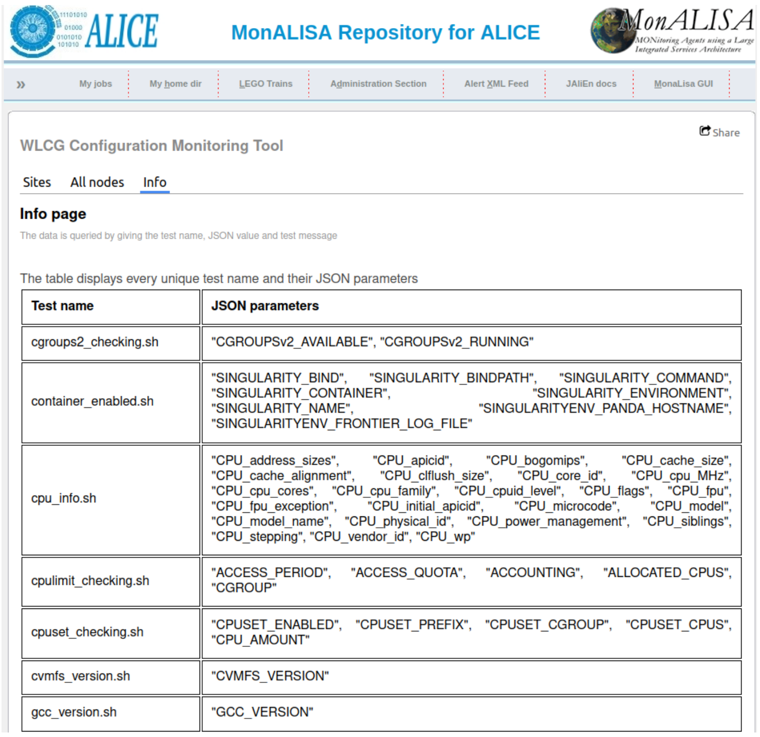
An important attribute of the new Site Sonar system was redesigning its implementation and focusing on ease of use. It should be possible for users unfamiliar with the Site Sonar implementation to do simple queries. In the previous version, a user had to know how to query information from the database. The old Site Sonar was very flexible and customizable in its way of querying data, but it was hard for an outsider to use these functionalities without some knowledge of the database.

Another drawback of the old Site Sonar was that it was dependent on sending probing jobs into the Grid to distribute the test-scripts to the worker nodes. This meant that there was no control over what or when worker nodes would run these test scripts. Therefore, there was no guarantee that it would reach all worker nodes under a site before the 48hour time limit of a

Run concluded. A site might have few resources and long queues of Grid jobs, meaning that the test scripts will never reach the worker node to collect data. Instead, this is now built into the Job Agent. It is guaranteed that the test scripts will run on all worker nodes that can run jobs since all the worker nodes who can run jobs need a Job Agent. The Job Agent runs the test scripts which are in the Site Sonar directory in the CVMFS (CernVM File System).

4.3.1 Creating a user friendly GUI

The new implementation should ease the usability for users without an overhead of knowledge of Site Sonar. The new Site Sonar supports custom user input, meaning the user can specify exactly what test and value should be queried. The new version also has examples of the most common queries which can be selected from a menu. A user can simply select an option and apply, which will reload the webpage with the updated parameters and display the resulting list.



WLCG Configuration Monitoring Tool

Info page

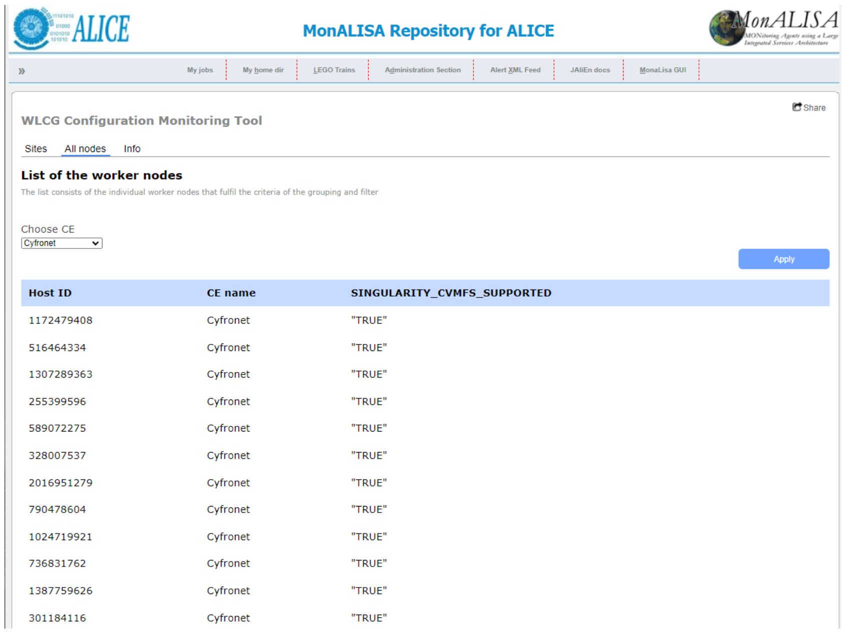
The data is queried by giving the test name, JSON value and test message

The table displays every unique test name and their JSON parameters

Test name	JSON parameters
cgroups2_checking.sh	"CGROUPSv2_AVAILABLE", "CGROUPSv2_RUNNING"
container_enabled.sh	"SINGULARITY_BIND", "SINGULARITY_BINDPATH", "SINGULARITY_COMMAND", "SINGULARITY_CONTAINER", "SINGULARITY_ENVIRONMENT", "SINGULARITY_NAME", "SINGULARITYENV_PANDA_HOSTNAME", "SINGULARITYENV_FRONTIER_LOG_FILE"
cpu_info.sh	"CPU_address_sizes", "CPU_apicid", "CPU_bogomips", "CPU_cache_size", "CPU_cache_alignment", "CPU_clflush_size", "CPU_core_id", "CPU_cpu_MHz", "CPU_cpu_cores", "CPU_cpu_family", "CPU_cpuid_level", "CPU_flags", "CPU_fpu", "CPU_fpu_exception", "CPU_initial_apicid", "CPU_microcode", "CPU_model", "CPU_model_name", "CPU_physical_id", "CPU_power_management", "CPU_siblings", "CPU_stepping", "CPU_vendor_id", "CPU_wp"
cpulimit_checking.sh	"ACCESS_PERIOD", "ACCESS_QUOTA", "ACCOUNTING", "ALLOCATED_CPUS", "CGROUP"
cpuset_checking.sh	"CPUSET_ENABLED", "CPUSET_PREFIX", "CPUSET_CGROUP", "CPUSET_CPUS", "CPU_AMOUNT"
cvmfs_version.sh	"CVMFS_VERSION"
gcc_version.sh	"GCC_VERSION"

Figure 4.2: Site Sonar page which displays the unique test names and the JSON keys

There are two additional pages in the new Site Sonar. The first page, shown in figure 4.2 displays information regarding the database structure. This information page consists of a table that has the different test names and their JSON keys. The other page, shown in figure 4.3, is where a user can see individual worker nodes. The user selects a site, and then the resulting list will be the sites worker nodes that are not filtered out.



WLCG Configuration Monitoring Tool

List of the worker nodes

The list consists of the individual worker nodes that fulfil the criteria of the grouping and filter

Choose CE: Cyfronet

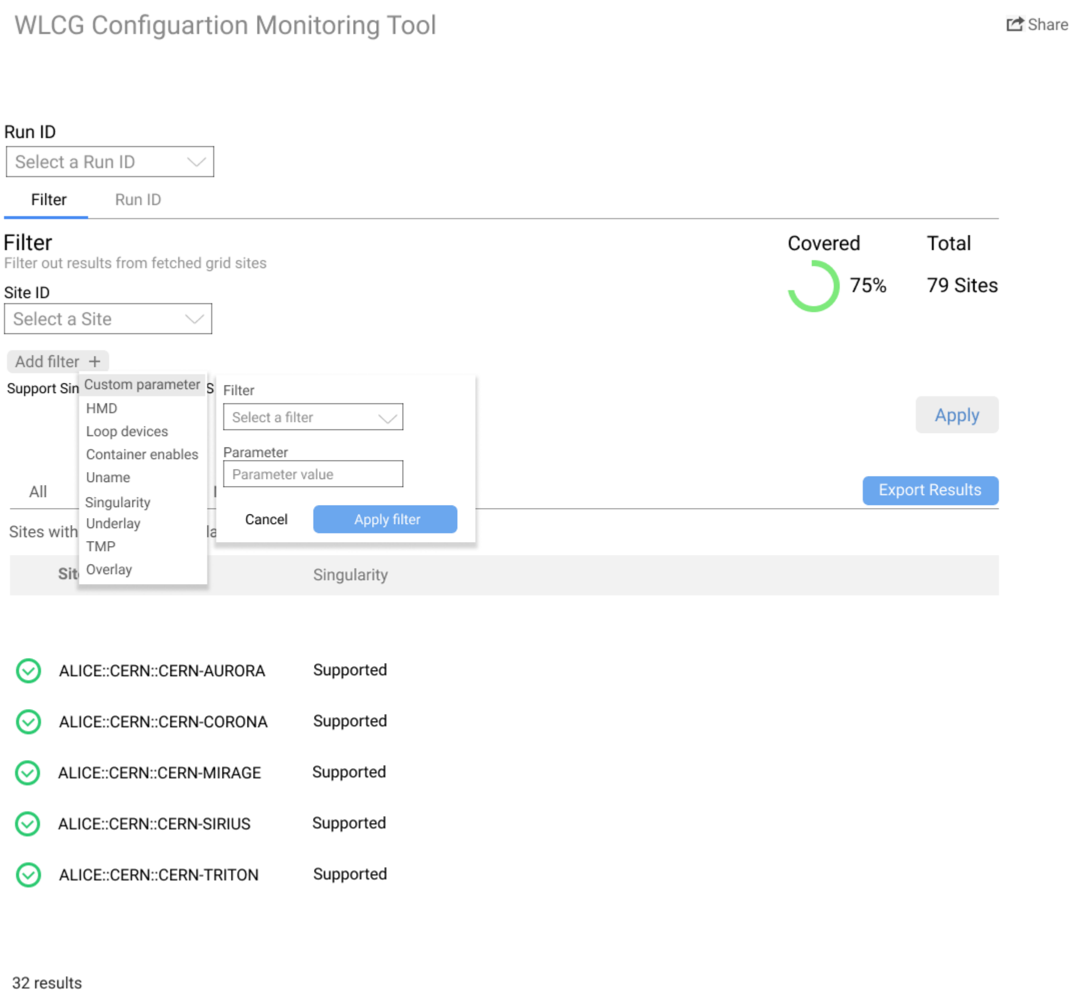
Host ID	CE name	SINGULARITY_CVMFS_SUPPORTED
1172479408	Cyfronet	"TRUE"
516464334	Cyfronet	"TRUE"
1307289363	Cyfronet	"TRUE"
255399596	Cyfronet	"TRUE"
589072275	Cyfronet	"TRUE"
328007537	Cyfronet	"TRUE"
2016951279	Cyfronet	"TRUE"
790478604	Cyfronet	"TRUE"
1024719921	Cyfronet	"TRUE"
736831762	Cyfronet	"TRUE"
1387759626	Cyfronet	"TRUE"
301184116	Cyfronet	"TRUE"

Figure 4.3: Site Sonar's All Nodes page

4.3.2 Sketching a prototype design for Site Sonar

The first prototype of the new Site Sonar was sketched in Figma[46], which is a popular tool for designing graphical user interfaces. Figure 4.4 shows the first completed draft of Site Sonar's new design. The design shows a list of all the worker nodes on a site that are not filtered out. One important change to the requirements was that instead of listing out all the worker nodes under a selected site, it should count the instances of worker nodes for a site based on a grouping (as seen in Figure 4.5). This gives a much better overview, as all the sites are displayed in the same list, and the number of worker nodes that are grouped is counted. Similarly, the worker nodes which do not support the grouping parameter are counted as "other worker nodes". This way we can see the percentage of worker nodes that support a certain grouping for each site.

One key aspect of the design in the new Site Sonar compared to the old Site Sonar is the ability for users to select a query from a menu. These predefined queries are based on the tests which are stored in Site Sonar's database, and would list out all the sites nodes, and whether they have the preferred test message or not. For example one of these filters are overlay, and the value could be "enable overlay = no". The goal of the listed filters is to cover the most frequent queries.



Site	Singularity
✓ ALICE::CERN::CERN-AURORA	Supported
✓ ALICE::CERN::CERN-CORONA	Supported
✓ ALICE::CERN::CERN-MIRAGE	Supported
✓ ALICE::CERN::CERN-SIRIUS	Supported
✓ ALICE::CERN::CERN-TRITON	Supported

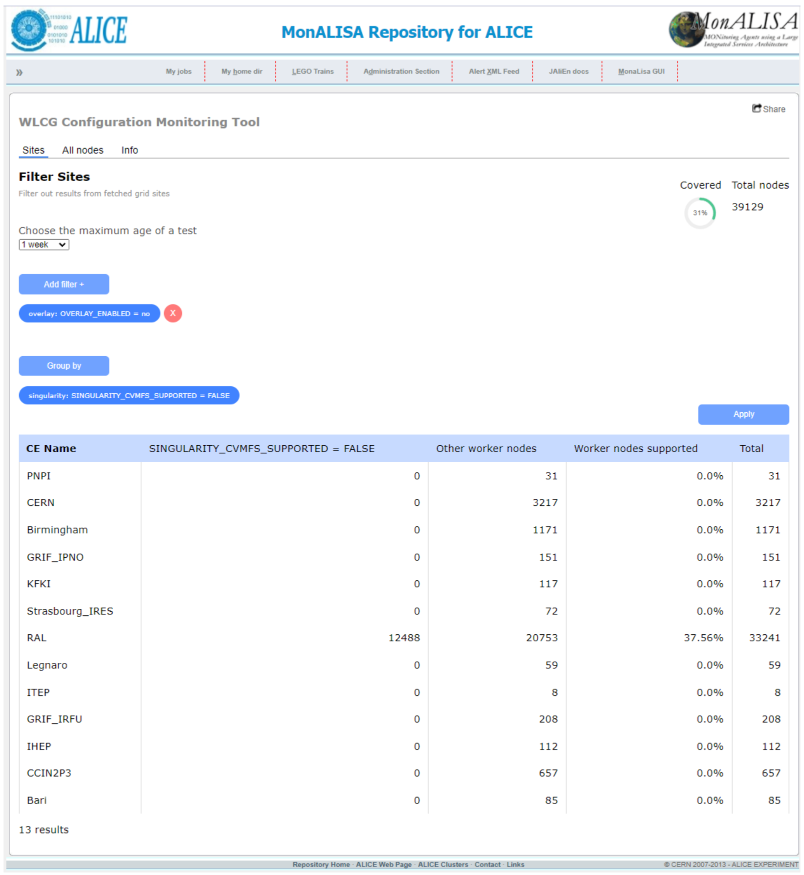
32 results

Figure 4.4: Early prototype design of Site Sonar

4.3.3 Grouping and filtering nodes

The querying method in the new Site Sonar was changed entirely from the original design. Instead of finding nodes that support the filters, a grouping parameter was added. This turned out to be more practical, as the grouping could give an overview of the condition on the sites. The user selects a parameter on which the node should be counted, called a "grouping". The results are counted and organized into their respective sites in a list. Then further filtering can be done on the results.

The example use case of Site Sonar shown in 4.5 is selecting a grouping parameter of worker nodes who don't support Apptainer, and then all the resulting worker nodes which do not enable overlay can be filtered out. The remaining list displays 13 sites. There are over 50 sites in the WLCG, but since most sites return a total of 0 nodes with the filter on, they are excluded from the list.



WLCG Configuration Monitoring Tool

Sites All nodes Info

Filter Sites

Filter out results from fetched grid sites

Covered	Total nodes
31%	39129

Choose the maximum age of a test
1 week

Add filter +

overlay: OVERLAY_ENABLED = no

Group by

singularity: SINGULARITY_CVMFS_SUPPORTED = FALSE

Apply

CE Name	SINGULARITY_CVMFS_SUPPORTED = FALSE	Other worker nodes	Worker nodes supported	Total
PNPI	0	31	0.0%	31
CERN	0	3217	0.0%	3217
Birmingham	0	1171	0.0%	1171
GRIF_IPNO	0	151	0.0%	151
KFKI	0	117	0.0%	117
Strasbourg_IRES	0	72	0.0%	72
RAL	12488	20753	37.56%	33241
Legnaro	0	59	0.0%	59
ITEP	0	8	0.0%	8
GRIF_IRFU	0	208	0.0%	208
IHEP	0	112	0.0%	112
CCIN2P3	0	657	0.0%	657
Bari	0	85	0.0%	85

13 results

Repository Home · ALICE Web Page · ALICE Clusters · Contact · Links
© CERN 2007-2013 - ALICE EXPERIMENT

Figure 4.5: New Site Sonar implemented in AliMonitor

4.4 Site Sonar implementation

Site Sonar is wrapped in an AliMonitor JSP page, meaning that it will be built using the technologies that are used in AliMonitor. The server-side of AliMonitor is written in Java Server Page (JSP) files. JSP is a language based on Java, used in the web development of dynamic web pages. It uses its tags to embed what sort of code it contains. The <% %> tags mean we want to write normal Java code, without the need for classes or objects. To write methods, the JSP embed them in <%! !%>. The server then compiles the JSP code and generates an HTML page. The project uses *.res* files, which contain HTML, JavaScript and CSS. They can be reused, where for example a list element in Site Sonar is implemented as its own *.res* file. A JSP file calls the *sonar_list.res* file several times to render the complete list. JSP files can send parameter inputs to the *.res* files by using the *append()* function.

The parameters for Site Sonar are passed through the URL. Listing 4.1 shows how the query gets the parameters in the URL with the *request.getParameter()* function, and uses it to fetch tests from the *sitesonar_tests* table. If no parameters are specified, the default values tells that the list should group on worker nodes that support Apptainer. The JSP file is responsible for sorting the worker nodes to sites, and counting whether they match the grouping parameter or not. There are two HashMaps, one for counting the supported nodes, and one for counting all other worker nodes. Listing 4.2 is taken from the source code of Site Sonar. The two hashmaps are the *countSupportForSites* and *countNotSupportForSites*. It initializes the site in the HashMap if it is the first occurrence. Each site is expressed by a Computing Element, which is why the name variable is specified as *ceName*. A possible error is that if a site has two Computing Elements, it will show the site two times. However, this has not been an issue while testing Site Sonar. Further down in listing 4.2, it loops over the tests in the **groupTestsDB** variable. The **groupTestsDB** variable contains the queried tests specified by the grouping parameter. The key in the HashMaps are the site names, and the values are initialized to 0. The data needed from the tests in this function is the host ID to identify the worker node, the test message in JSON format and the site name to cluster the worker nodes together. If the test message is the same as the grouping parameter, we count it in the *countSupportForSites* to its respective site. Otherwise it is counted in *countNotSupportForSites*.

Listing 4.1: Fetching tests based on grouping parameter

```
groupTestsDB = new DB("SELECT sitesonar_tests.host_id, test_message_json
    ↪ -> '" + request.getParameter("JSONGroup") + "', ce_name, test_name
    ↪ FROM sitesonar_tests INNER JOIN sitesonar_hosts ON
    ↪ sitesonar_hosts.host_id = sitesonar_tests.host_id WHERE
    ↪ last_updated > '" + testAge + "' AND test_name='" +
    ↪ request.getParameter("grouping") + "'");
```

Listing 4.2: Grouping worker nodes together

```

// Key: Site name, Value: counted nodes
HashMap<String, Integer> countSupportForSites = new HashMap<String,
    ↪ Integer>();
HashMap<String, Integer> countNotSupportForSites = new HashMap<String,
    ↪ Integer>();

while(groupTestsDB.moveNext()){
    String hostID = groupTestsDB.gets(1);
    String JSONMessage = groupTestsDB.gets(2);
    String ceName  = groupTestsDB.gets(3);

    //Put ce_name field if empty, init with 0
    if(!countSupportForSites.containsKey(ceName)){
        countSupportForSites.put(ceName, 0);
        countNotSupportForSites.put(ceName, 0);
    }

    //If test_message is correct, add groupMessage to already existing row
    //Note: use contains instead of equals, since some string values are
        ↪ written e.g "TRUE" instead of TRUE
    if(JSONMessage.contains(groupMessage)){
        countSupportForSites.put(ceName,
            ↪ (countSupportForSites.get(ceName)+1));

        // Add worker node to "All node" list if the Site matches.
            ↪ Default: PNPI
        if(ceName.contains(sitenameAllNodes)){
                addToNodeList(p, hostID, ceName, JSONMessage);
        }
    }
    else{
        countNotSupportForSites.put(ceName,
            ↪ (countNotSupportForSites.get(ceName)+1));
    }
}
```

Among the URL parameters are the filter and grouping values (test name, JSON key, and JSON value), the site which will list out its worker nodes, and the maximum test age. Due to the structure of the database, it complicates the process of grouping and filtering elements. This is due to the fact that the data is stored as tests instead of worker nodes. Therefore we have to gather the tests for the grouping parameter and the tests for the filter individually, and compare the host ids to see if they belong to the same worker node. As shown in listing 4.3 the filtering loop first goes through all the tests based on the filter parameter, and adds the worker node ids in a HashMap called *filterSupportMap*. This HashMap keeps track of the worker nodes that should be included. When we then loop over the tests based on the grouping parameter, then those worker nodes which are in the filter HashMap will be counted, while the worker nodes that are filtered out will be ignored.

Listing 4.3: Creating hashmap of worker node host IDs which should not be filtered out

```java
while(filterDB.moveNext()){
    String hostID = filterDB.gets(1);
    String JSONMessage = filterDB.gets(2);
    String ceName  = filterDB.gets(3);

    // Init element in hashmap for hostID
    if(!filterSupportMap.containsKey(hostID)){
        filterSupportMap.put(hostID, 0);
    }

    // Indclude the ones that pass the filter. 3 means it should be
       ↪ included
    if(JSONMessage.contains(filterTestMessage.get(0))){
        filterSupportMap.put(hostID, 3);
    }
}
```

We also have to take into account the age of the test, as they are collected continuously. The default value is that a test should be one week old at the maximum. Most sites will run the test scripts regularly, and therefore most of the worker nodes will be included in the query when the maximum age is 1 week. This saves time in the querying, grouping, and filtering functions since it reduces the number of tests. The *sitesonar_tests* table has a column for the time measured in Unix time, meaning the number of seconds passed since Jan 1, 1970, at 00:00:00 UTC. In listing 4.4 it gets the parameter in the URL for the test age, which is a number between 0 and 4. The numbers 0-4 represents a predefined option, from 1 week to 1 year. They only cover a few options, but it is assumed that queries below 1 week don't give correct results, and above one year creates a very long querying time. Based on the number given in the URL, the *testAge* variable will be updated and used in the queries for filter and grouping to cut off old tests.

Listing 4.4: Set the maximum age of a test

```java
    //Set test age
    long DAY_IN_MS = 1000 * 60 * 60 * 24;
    long testAge = (System.currentTimeMillis() - (7 * DAY_IN_MS)) / 1000l;
    if(request.getParameter("testAge") != null){
        switch(request.getParameter("testAge")) {
            case "0":
                // 1 week
                testAge = (System.currentTimeMillis() - (7 * DAY_IN_MS))
                   ↪ / 1000l;

                p.modify("ageOptionText", "1 week");
                p.modify("ageOptionValue", "0");
                break;
            case "1":
                // 2 weeks
                testAge = (System.currentTimeMillis() - (14 * DAY_IN_MS))
                   ↪ / 1000l;
                p.modify("ageOptionText", "2 weeks");
                p.modify("ageOptionValue", "1");
                break;
            case "2":
                // 1 month
                testAge = (System.currentTimeMillis() - (30 * DAY_IN_MS))
                   ↪ / 1000l;
                p.modify("ageOptionText", "1 month");
                p.modify("ageOptionValue", "2");
                break;
            case "3":
                // 6 months
                testAge = (System.currentTimeMillis() - (183 *
                   ↪ DAY_IN_MS)) / 1000l;
                p.modify("ageOptionText", "6 months");
                p.modify("ageOptionValue", "3");
                break;
            case "4":
                // 1 year
                testAge = (System.currentTimeMillis() - (365 *
                   ↪ DAY_IN_MS)) / 1000l;
                p.modify("ageOptionText", "1 year");
                p.modify("ageOptionValue", "4");
                break;
            default:
                // default to 1 week
                testAge = (System.currentTimeMillis() - (7 * DAY_IN_MS))
                   ↪ / 1000l;
                p.modify("ageOptionText", "Select option");
                p.modify("ageOptionValue", "0");
        }
    }
```

When the *countSupportForSites* and *countNotSupportForSites* HashMaps are sorted, the frontend can be generated. The frontend is based on the *index.res* file, but the logic for sorting the worker nodes and creating variables for the frontend is in the *index.jsp* file. In listing 4.5, the loop goes over all the site names. For each site (which is the key in the HashMap), the code checks that the HashMap is not empty. Sites that do not have any worker nodes will not be appended. This will only happen if none of the worker nodes on a site supports the filter. The code will then generate a list element using the *sonar_list.res* file. It will update the CE name, grouped worker nodes, other worker nodes, what percentage of the worker nodes satisfy the grouping parameter, and the total amount of worker nodes. The percentage of supported worker nodes and total worker nodes are calculated in the loop and added to the list element. It will then append the list element to the site list in the frontend.

Listing 4.5: Building the Site List for the frontend

```java
Page listElement = new Page(null, "sitesonar/sonar_list.res");
for(int i = 0; i < sites.size(); i++) {
        //Calculates percentage of sites covered by grouping parameter
        float percentSupport = 0;
        int total = 0;

        //check that values are not null
        if(countSupportForSites.get(sites.get(i)) != null &&
           ↪ countNotSupportForSites.get(sites.get(i)) != null){

            percentSupport = (countSupportForSites.get(sites.get(i)) *
               ↪ 100.0f) / (countSupportForSites.get(sites.get(i)) +
               ↪ countNotSupportForSites.get(sites.get(i)));

            total = countSupportForSites.get(sites.get(i)) +
               ↪ countNotSupportForSites.get(sites.get(i));

            //Summarize all worker nodes for all sites
            totalWorkerNodes += total;
            totalSupportedCEs += countSupportForSites.get(sites.get(i));

            //Don't display sites that have no nodes
            if(countSupportForSites.get(sites.get(i)) != 0 ||
               ↪ countNotSupportForSites.get(sites.get(i)) != 0){
                resultSites++;

                //Create list elements and put values for site-name, n
                   ↪ supported nodes, n not supported nodes etc, then
                   ↪ append the list element
                listElement.modify("site_name", sites.get(i));
                listElement.modify("supported_nodes",
                   ↪ countSupportForSites.get(sites.get(i)));
                listElement.modify("not_supported_nodes",
                   ↪ countNotSupportForSites.get(sites.get(i)));

                //Only two decimal for percent
                listElement.modify("percent_support",
                   ↪ (Math.floor(percentSupport * 100) / 100) + "%");
                listElement.modify("total", total);
                p.append("testList", listElement);
            }
        }
```

The final step is putting the values for the total worker nodes, percent of nodes supported, the grouping parameter, the message key and value, and the number of sites included in the list. These are all variables which are used in the frontend and has to be passed from the *index.jsp* to the *index.res* file. Listing 4.6 shows the function *setValuesForSiteSonarHTML()* which uses the function *modify()* to change the fields in the *index.res* file. For example the variable **totalWorkerNodes** will be passed to the *index.res* file, where the HTML has a **<<:n_workerNodes:>>** field. Then the master page appends the Site Sonar page, which wraps Site Sonar in an AliMonitor page.

Listing 4.6: Set values in the frontend

```
public void setValuesForSiteSonarHTML(Page p, int totalWorkerNodes, int
  ↪ totalSupportedCEs, String groupingName, String groupJSON, String
  ↪ groupMessage, int resultSites){
        p.modify("n_workerNodes", totalWorkerNodes);
        float percentageTotal = (totalSupportedCEs * 100.0f) /
            ↪ (totalWorkerNodes);
        p.modify("percentTotal", (int) percentageTotal);
        p.modify("groupParam", groupingName);
        p.modify("groupJSONParam", groupJSON);
        p.modify("valueParam", groupMessage);
        p.modify("result_count", resultSites);
    }
```

4.4.1 Site Sonar tests

Site Sonar has 25 unique test types. All the tests that are collected and stored in the Site Sonar database are included in table 4.1. The Job Agent on the worker nodes ensures that the test scripts are executed. Instead of sending the test scripts as a Grid Job, the Job Agent fetches them itself from the Site Sonar directory in CVMFS. Table 4.1 describes the different test-scripts executed on the worker nodes.

MonALISA also collects system information for computer nodes and clusters [38], however what is being monitored is predefined. Changing MonALISA's data collection on worker nodes requires code changes and that new versions are rolled out every time something is updated. With Site Sonar this process is simplified, where a test can be added to the Site Sonar repository in CVMFS with very few other changes. This allows Site Sonar to quickly roll out new tests when needed.

4.4.2 Parsing Test Messages to a JSON Format

There are test messages which can not be directly queried. An example of such a test is the *cpu_info.sh* test which has a long string of CPU information stored as its value. Therefore the message has to be parsed to be able to be queried. The *sitesonar_tests* table has an additional column called *test_message_json* stored as a JSON. Instead of having a message be in the string format as shown in Listing 4.7, it should instead be parsed to the example shown in listing 4.8. This allows Site Sonar to specify which part of the test message it should group and filter on. One can for example write at the end of a *sitesonar_tests* query:

Test Name	Description
cgroups2_checking.sh	Checks if cgroups v2 is available and running.
container_enabled.sh	Check whether the environment is running in a container.
cpu_info.sh	Fetches information of a worker node's CPU
cpulimit_checking.sh	Checks if it is running in a cgroup cpu limiter.
cpuset_checking.sh	Checks if it is running in a cpuset.
cvmfs_version.sh	The version of CVMFS installed on the worker node.
gcc_version.sh	The version of GNU Compiler Collection installed on the worker node.
get_jdl_cores.sh	Gets the number of cores defined in the JDL file.
home.sh	The home environment variable of the worker node.
isolcpus_checking.sh	Checks if there are CPU isolated by isolcpus.
lhcbmarks.sh	Get the LHCB Benchmark score.
loop_devices.sh	The loop devices.
lsb_release.sh	Gets the OS description and return the exit code of lsb$_{r}elease$.
max_namespaces.sh	The maximum namespace.
os.sh	Operating system information. E.g Linux distribution, system version, etc.
overlay.sh	Display whether the worker nodes enables overlay or not.
ram_info.sh	Fetches information of a worker node's RAM.
running_container.sh	What container is running. E.g No container, Docker, etc.
singularity.sh	Whether Apptainer is supported or not.
taskset_other_proccesses.sh	Checks if any of the CPUs are isolated.
taskset_own_process.sh	Checks if process is running tasksets.
tmp.sh	The TMPDIR environment variable.
uname.sh	Basic information of the operating system name and system hardware.
underlay.sh	Whether underlay is enabled or not.
wlcg_metapackage.sh	Whether the WLCG metapackage is available or not.

Table 4.1: Test names and descriptions

```
WHERE test_message_json->>'DistroMajor' = 'CentOS 7'
```

This will only retrieve tests where the *test_message_JSON* column is CentOS 7. Having the test message already parsed in the database is much more efficient than parsing the *test_message* column for each test in the backend of Site Sonar.

Listing 4.7: Example of test message from os.sh

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

Listing 4.8: Simplified example of a JSON message for os.sh

```
{
  "Distro": "CentOS",
  "DistroMajor": "CentOS 7",
  "DistroMinor": "CentOS 7.9",
  "Release": "CentOS Linux release 7.9.2009 (Core)"
}
```

4.4.3 Parsing the singularity.sh test message

The Apptainer test has two JSON keys, which are *SINGULARITY_LOCAL_SUPPORTED* and *SINGULARITY_CVMFS_SUPPORTED*. Listing 4.9 shows the BASH script used to generate the JSON message for the singularity.sh test. The parsing scripts used in Site Sonar are provided by Kalana Wijethunga[47].

Listing 4.9: Parse Singularity message to JSON

```
#!/bin/bash

# Print whether singularity is supported

SINGULARITY_EXEC="singularity exec -B /cvmfs:/cvmfs
    ↪ /cvmfs/alice.cern.ch/containers/fs/singularity/centos7 java
    ↪ -version"

CHECK_LOCAL=$($SINGULARITY_EXEC 2>&1 | grep -o "Runtime")
CHECK_CVMFS=$(/cvmfs/alice.cern.ch//containers/bin/singularity/current/bin
    ↪ /$SINGULARITY_EXEC 2>&1 | grep -o "Runtime")

if [ -z "$CHECK_LOCAL" ]
  then
      SINGULARITY_LOCAL_SUPPORTED=true
  else
      SINGULARITY_LOCAL_SUPPORTED=false
fi

if [ -z "$CHECK_CVMFS" ]
  then
```

```
        SINGULARITY_CVMFS_SUPPORTED=false
    else
        SINGULARITY_CVMFS_SUPPORTED=true
fi

echo "{ \"SINGULARITY_LOCAL_SUPPORTED\" : $SINGULARITY_LOCAL_SUPPORTED ,
    ↪ \"SINGULARITY_CVMFS_SUPPORTED\" : $SINGULARITY_CVMFS_SUPPORTED }"
```

4.4.4 Parsing the OS.sh test message

Another example of a test script that parses the test message to JSON is the one in listing 4.10. It parses the test message for *os.sh* to JSON. The *CPU_info.sh* test has 24 unique JSON keys. It quickly becomes apparent why the test messages have to be parsed to JSON. With the JSON keys in place, it can easily be specified which key and value in the *CPU_info.sh* should be queried.

Listing 4.10: Parse OS message to JSON

```
#!/bin/bash

# Print the CPU information

array=()
last_line=$(wc -l < /proc/cpuinfo)

current_line=0
index=0

JSON_OUTPUT="{ "

(while read line; do

    current_line=$(($current_line + 1))

    if [[ $current_line -ne $last_line ]]; then
        if [ -z "$line" ]; then
            index=$(($index + 1))
            continue
        fi

        if [[ $index == 0 ]];
        then
            if [[ ! "${array[@]}" =~ "${line}" || "${line}" == physical*
                ↪ || "${line}" == core* || "${line}" == apicid* ||
                ↪ "${line}" == 'cpu MHz*' ]] ; then
                key=$(echo "${line}" | cut -d ":" -f 1 | xargs | tr " " _)
                val=$(echo "${line}" | cut -d ":" -f 2- | xargs)
                # add integer values
                if [[ $key == "cpu_processor" || $key == "apicid" || $key
                    ↪ == "bogomips" || $key == "cache_alignment" || $key
                    ↪ == "clflush_size" || $key == "core_id" || $key ==
                    ↪ "cpu_MHz" || $key == "cpu_cores" || $key ==
                    ↪ "cpu_family" || $key == "cpuid_level" || $key ==
                    ↪ "model" || $key == "physical_id" || $key ==
                    ↪ "siblings" || $key == "stepping" ]]
                then
                    JSON_OUTPUT+="\"CPU_$key\" : $val ,"
                elif [[ $key == "flags" || $key == "power_management" ]]
                then
                    string_array=$(echo $val | sed 's| |\" , \"|g') #
                        ↪ convert space delimited string to json string
                        ↪ array
                    JSON_OUTPUT+="\"CPU_$key\" : [ \"$string_array\" ] ,"
                else
                    # check for boolean values
                    if [[ $val == "yes" ]]
                    then
                        JSON_OUTPUT+="\"CPU_$key\" : true ,"
                    elif [[ $val == "no" ]]
                    then
                        JSON_OUTPUT+="\"CPU_$key\" : false ,"
                    else
                        JSON_OUTPUT+="\"CPU_$key\" : \"$val\" ,"
                    fi
                fi
                array[${#array[@]}]=${line}
            fi
        fi
    else

        processor_count=$(($index + 1))
        JSON_OUTPUT+=" \"CPU_processor_count\" : $processor_count }" #
            ↪ prefixing with CPU_ to unqiuely identify these keys
        echo $JSON_OUTPUT
    fi

done < /proc/cpuinfo) 2>&1

if [ $? -ne 0 ]; then
    exit 255
fi

count=`grep -c "^processor" /proc/cpuinfo`

exit $count
```

4.5 Optimization

To optimize the time performance of Site Sonar, efforts have been made to make the least amount of calls to the database. All tests which are filtered or grouped are each queried just once. However, the tests are joined to the *sitesonar_hosts* table to collect necessary information about the worker node belonging to the test. Further development of Site Sonar should consider restructuring the database to remove the need for a SQL Join, as they can be costly in terms of performance.

The resulting data is stored in HashMaps which has O(1) running time for searching instead of an ArrayList which has O(n) running time for a search. While HashMaps will initially be slower and take more memory, it will eventually be faster for large values of n. Despite efforts being made to increase the time performance, there is an unknown reason which makes Site Sonar's loading time highly irregular. The page loading can vary from 5 seconds to a minute.

4.5.1 Indexing

Indexing sorts the values and makes it quicker to query on the indexed columns. The index *updated_and_name_idx* in listing 4.11 is a composite index, also known as a multi-column index. It is used in the Site Sonar database, where it first indexes on the *last_updated* column. This column is the age of the test, and since Site Sonar is only querying tests within a certain age limit, it should only look at the tests which are younger than the given maximum age. That way any tests which are older are "cut-off" from the query. The second column in the *updated_and_name_idx* is *test_name*. In Site Sonar the tests are queried on the *test_name* column. Indexing *test_name* makes it quick to look up for example *singularity.sh* tests.

Listing 4.11: Index on the sitesonar_tests table

```
CREATE INDEX updated_and_name_idx ON sitesonar_tests (last_updated,
    ↪ test_name);
```

Chapter 5

Evaluation

Quality models give a methodical way of evaluating a product. The model will decide which characteristics will be accounted for when evaluating the quality of a software product. Quality is the degree to which the product satisfies the criteria set by the quality model, and thus provides the stakeholders with value.

5.1 Criteria of evaluation

There are several methods for evaluating a software product. To evaluate the Site Sonar application in this thesis, the quality criteria are based on the ISO/IEC 25010 standard [48]. The ISO/IEC standard focuses on the functionality, performance, compatibility, usability, reliability, security, maintainability, and portability of a product. Figure 5.1 comprises the eight quality characteristics defined in the ISO/IEC 25010 quality model.

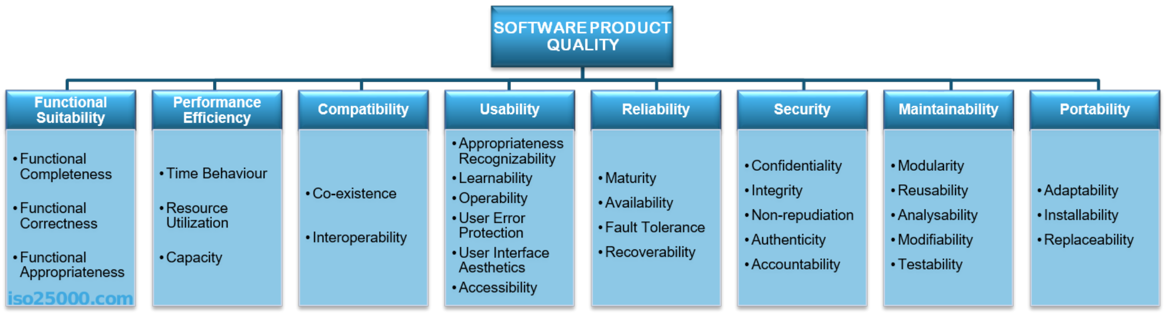


Figure 5.1: ISO/IEC 25010 quality criterias [9]

5.1.1 Software product quality

The product quality indicates to which extent the requirements of the product are satisfied. Requirements that are important to the product quality are e.g the maintainability and development cost, performance, usability, software purpose, etc. Product quality includes both internal and external quality characteristics and sub-characteristics. The first characteristic shown in figure 5.1, functional completeness, has 3 sub-characteristics which are: functional completeness, functional correctness and functional appropriateness. In total there are 47 product quality metrics in the ISO/IEC 25010 model [48]. If the product quality is insufficient, it will most likely lead to disadvantages such as high development and maintenance costs, not meeting users and stakeholders' needs, bad performance, etc.

The product quality is evaluated for both the old Site Sonar and the new version of Site Sonar. They are evaluated according to the eight quality criteria of ISO/IEC 25010. The scoring tries to be as objective as possible, where the initial score is set to the highest of 5, and then points are subtracted for lacking important aspects. Since the evaluation is based on qualitative characteristics, the final scoring might still leave room for some debate. Such as how important one aspect is (e.g time performance), and how many points should be subtracted.

5.1.2 Functional suitability

This criterion describes how the product's function meets the stated and implied needs of the stakeholder. The functional suitability characteristic is composed of the following three sub-characteristics: functional completeness, functional correctness and functional appropriateness. Functional completeness describes to what degree the functions cover all the specified tasks. Functional correctness, which describes whether the system gives correct and accurate results. Functional appropriateness, which describes how well the functions fulfill their tasks and objectives.

The functionality of Site Sonar is important to ensure that it is relevant to the stakeholders. Some functionality was added to decrease the loading time of the application, such as the ability to select how old a test should be. There is also functionality to make Site Sonar easier to use, such as the menu where users can select common search parameters.

5.1.3 Performance efficiency

ISO/IEC 25010 explains performance efficiency as how well the system performs, and how much resources it uses. In this quality model, the performance efficiency is composed of the three sub-characteristics: time behavior, resource utilization, and capacity. The time behavior characteristic determines the product's performance by measuring variables such as response time, processing time, throughput rate, etc. The second sub-characteristic, resource utilization, measures how different types of resources are used by a system during execution. Finally, the capacity sub-characteristic measures the maximum limits of the system, for example, if there is enough storage to meet the requirements.

The only sub-characteristics in performance efficiency that is important to Site Sonar, is the time behavior. Site Sonar queries a lot of data, and therefore depends on doing so efficiently. Neither the maximum limits of the system nor the resource utilization is controlled in Site Sonar. These sub-characteristics would however fit AliMonitor, which is outside the scope of this thesis.

5.1.4 Compatibility

Compatibility is described as how well the functions of the system can perform while sharing resources such as hardware, and sharing information with other systems, products, or components. The sub-characteristics of compatibility are co-existence and interoperability. Co-existence is to what degree the system can co-exist with other systems, and how it impacts other programs. Optimally it should be able to co-exist without any detrimental impact. Interoperability is how well it can exchange information with other systems, and how that information can be used. This characteristic is not crucial to Site Sonar, as it does not communicate with other system parts or share resources among other programs.

5.1.5 Usability

The usability characteristic is concerned with the user interface, and to which degree the system is effective in making the users achieve their objectives. The criteria set for a user interface is that it should be effective, efficient, free from risk, and satisfactory for any user to use. Usability is composed of these 6 sub-characteristics as described in [49].

- Appropriateness recognizability - Appropriateness recognizability is how easy an user can recognize whether the product is appropriate for their usage and needs.
- Learnability - This sub-characteristics is how easy it is to learn to use the system effectively, efficiently and have a freedom from risk.
- Operability - Operability is the degree to which the system has attributes that make it easy to control and operate.
- User error protection - This is how good the system is as protecting the users from making mistakes.
- User interface aesthetics - The aesthetics of the interface is how pretty and satisfying the design and graphics of the website are.
- Accessibility - To what degree can the system be used by the widest range of characteristics and capabilities.

Usability is a crucial aspect that has been addressed in the new implementation of Site Sonar. The user interface was an important factor in the decision to replace the old Site Sonar project. The old Site Sonar had a complex interface, which demanded a lot of prerequisite knowledge from the users. There was room for improvement when the new implementation was being designed. The new Site Sonar implementation tries to have an interface that is easier to operate and learn, and thereby increases the usage. However, measuring the usability can be challenging, as it is a highly subjective characteristic.

5.1.6 Reliability

Reliability is defined in the ISO/IEC 25010 as how well the system and its functions can perform under different conditions. It is composed of the following five sub-characteristics.

- Maturity - The degree to which a system is reliable under normal operation.
- Availability - The degree to which a system is accessible when required by users.
- Fault tolerance - The degree to which a system can operate correctly despite the presence of faults in either hardware or software.
- Recoverability - The degree to which a system can recover data and reestablish a desired state after an interruption or a failure has happened.

Reliability is not an important aspect in this project, as the software runs in the AliMonitor web application. The reliability is outside the scope of the Site Sonar application itself. However, the application has to be reliable in that it gives correct results in order to be relevant to the users.

5.1.7 Security

Security, also called cybersecurity, is how the system protects sensitive data and users harm or unauthorized access and use. Security is composed of the five following sub-characteristics.

- Confidentiality - The degree to which a system ensures that the data us accessible to only those with authorized access.
- Integrity - The degree to which a system prevents unathorized access or modification to the data or software.
- Non-repudiation - The degree to which a system can prove that an event has taken place so that the event cannot be repudiated at a later time.
- Accountability - The degree to which the actions of a user can be traced back.
- Authenticity - The degree to which the identities of users can be proven.

Security is always an important aspect of an application, especially if it is open to the internet. There are always improvements and new challenges in the security of software applications. One example being that Site Sonar does not sanitize SQL inputs, and is vulnerable to SQL injections. However in this project, there has not been an emphasis on the security aspects of the application, and therefore can't evaluate it. In the future, Site Sonar should consider any security threats existing in its implementation, and how to minimize or remove them.

5.1.8 Maintainability

Maintainability is how efficiently one can improve upon, fix or modify the system. The ISO/IEC standard composes maintainability into the 5 following sub-characteristics.

- Modularity - The degree to which a system is composed of different discrete modules, which can be individually modified with minimal impact on the other modules.

- Reusability - The degree to which parts of the system can be reused in building other assets.
- Analysability - The degree to which the system is effective in detecting or diagnosing defects or causes of errors and bugs.
- Modifiability - The degree to which the system can be changed without creating deficiencies or decreasing the product quality.
- Testability - The degree to which test criteria can be established, and be performed to determine that the system is running properly.

Maintainability is an important criterion to be able to keep Site Sonar up to the requirements. Insufficient maintainability means that it is hard to include new features. However, the new Site Sonar uses JSP which is an older technology and does not have the advantages that the old Site Sonar has by using React in its web interface. React gives a component-based architecture that has become very popular in later years because of its modularity and reusability. Further development of Site Sonar might consider implementing a component-based structure in the frontend by using shadow DOM. Additionally, it should consider using Ajax [50] to send and retrieve data from the Site Sonar database asynchronously without having to reload or interfere with the display and behavior of the loaded page.

5.1.9 Portability

Portability defines how effectively and easily a system can be transferred between environments. The sub-characteristics in portability, as defined by the ISO/IEC standard, are: adaptability, installability, and replaceability. Adaptability is the degree to which the system can be adapted to different hardware and software environments. Installability is the degree to which the system can be installed or uninstalled successfully. Replaceability is the degree to which the system can replace another application with the same purpose. There is not an emphasis on portability in Site Sonar, because it does not need to run in different environments. Site Sonar runs in the AliMonitor web application and is therefore accessed through a web browser.

5.2 Functional suitability in Site Sonar

All three sub-characteristics of functional suitability apply to Site Sonar. These sub-characteristics are: functional correctness, functional completeness, and functional appropriateness.

5.2.1 Functional completeness

Functional completeness measures how much of the functionality is implemented. The new implementation of Site Sonar has in some aspects decreased its functional completeness compared to the old Site Sonar. There was a trade-off in having a user-friendly system and including all the functionalities from the previous version. The old version uses SQL syntax in its interface to allow for advanced queries. Users can combine multiple filters, and also choose if it should combine the filter with an ’AND’ operator or an ’OR’ operator. The newer version simplifies the querying, where a user only inputs the name of the test, e.g *singularity.os*, and what category (JSON key) and value under the test it should group or filter on. However, the old Site Sonar can only query on string values, which is very limiting. In the new Site Sonar, the value of a test is parsed into JSON, so that it can choose a JSON key, e.g CPU cores, and select a value of for example 4. The new Site Sonar also has some functionality which the older version does not have, such as sharing a URL with the parameters included. This way a user can share with others what they searched for, and their results. To share the results in the old Site Sonar, a user has to export the results to a file. The new implementation also makes it easy to choose between listing all sites and their worker nodes, or just the worker nodes under a specified site.

Site Sonar gets one subtracted point for not supporting combined filters. The old Site Sonar also gets a point subtracted for lacking user-friendly features and only supporting string values instead of values parsed to JSON.

5.2.2 Functional correctness

Functional correctness is simply put whether the results are correct or not. It is hard to measure the correctness and compare it to the old Site Sonar implementation. The new Site Sonar is tested in development and production, and there have been no major bugs. For this reason, there are no subtracted points for the functional correctness in the final score.

5.2.3 Functional appropriateness

Functional appropriateness is how well the system can perform its task. The new version of Site Sonar wants to build on top of the old Site Sonar’s functionality. They both have the same goal of monitoring worker nodes but approach it in different ways. As mentioned, the old Site Sonar sent probing jobs into the Grid and hoped the job would reach as many worker nodes as possible before the time of the Run was completed. This in return gave no guarantee that all worker nodes would be reached, and was a poor way of approaching the issue. The new Site Sonar has improved this by implementing it directly into the Job Agent. When the Job Agent runs on a worker node, it will fetch the test scripts from the Site Sonar directory in CVMFS.

The old Site Sonar gets another subtracted point, giving it a final score of 3/5 for the functional suitability characteristic. The new Site Sonar is slightly better with a score of 4/5.

5.3 Performance efficiency in Site Sonar

The performance efficiency describes how well the system performs, and how much resources it uses. Since Site Sonar is a program within AliMonitor, the resource utilization and capacity sub-characteristics are not relevant. The sub-characteristic which is important, and has been a major focus in the development is the time behaviour.

5.3.1 Time behaviour

To measure and evaluate the performance, the queries have been analyzed and timed in milliseconds. The listing 5.1 is the console output of the *EXPLAIN ANALYZE* command in PostgreSQL. It provides a query plan detailing, where it specifies exactly what approach the planner took to execute the statement. The execution plan uses the *updated_and_name_idx* index to decrease its execution time. The statement provided is a common query used in the new Site Sonar where it lists out worker node tests for Apptainer. The query is also joined with the table *sitesonar_host* to get the *CE_name* for all worker nodes belonging to the tests. Joins can be time-consuming, and the database should consider changing its structure to avoid joins all together. One important consideration is that it would introduce storage redundancy by storing the site name of the test in both the *sitesonar_tests* and *sitesonar_hosts* tables.

Listing 5.1: Console Output for Explain Analyze

```

   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)
```

In listing 5.1 the execution time is 613.851 milliseconds. However, the querying time can sometimes reach 20 000 milliseconds (20 seconds). The timing is highly irregular and has been an issue of unknown causes in the development of Site Sonar. It is also difficult to compare the time performance between the old and new version of Site Sonar. The old Site Sonar was written over a 3 month period as a *Google Summer of Code* project. There was only a focus on getting it working. Due to the time constraint of the project, there were not performed any indexing or other optimization. As mentioned, the database structure was also different, making the comparison even harder.

Due to the high irregularities in the querying found in the new Site Sonar, the score for performance efficiency is 3/5. Since the old Site Sonar is not optimized, it scores slightly worse with 2/5.

5.4 Usability in Site Sonar

It is hard to measure the usability characteristics, as it is subjective. Also, due to the Covid-19 pandemic that was ongoing during the writing of this thesis, it was harder to test the application with relevant people. Instead, the usability can be evaluated by looking at the specific changes to the graphical user interface made between the old Site Sonar and the new implementation. Then the effect on usability can be estimated.

Appropriateness recognizability is how easy it is for the user to recognize if the product is appropriate for their needs. In this thesis, it is assumed that Site Sonar is only relevant to users in the ALICE organization, and its users will therefore have some prior knowledge. For this reason, the appropriateness of recognizability is not considered.

5.4.1 Usability changes to GUI in the new Site Sonar implementation

The old implementation of Site Sonar had some issues with the usability:

- The input fields were hard to understand and learn.
- Users needed knowledge of SQL syntax to combine filters.
- No easy way to share results.

The changes that affect the usability from the old to the new implementation of Site Sonar are listed in this section. The most important changes are:

- The Run ID field was removed.
- The new version only supports one filter.
- The new implementation has pre-selected filters and groupings which are based on common queries.
- The new implementation has removed the equation field found in the old Site Sonar.
- Added an information page showcasing all the test names and their "categories". That way users can look up how the data is structured.

- Users can choose between seeing the values grouped by site, or the worker nodes individually.
- Users can choose how old a test can be at the maximum. This avoids that very old tests are being queried, slowing down the page loading.

In the old version of Site Sonar, the purpose of the fields was not obvious, which made it difficult to interact with. An example of this is the equation field (Figure 4.1 on page 33) where a user can specify how the filters should be put together. This relates to how the data is queried in PostgreSQL, meaning the user would have to know some SQL syntax. Removing this feature, and making the querying simpler removes a lot of the customizability, but still upholds the requirements of the stakeholders found in section 3.2.2. This is why the new Site Sonar implements simpler input fields. A user can avoid needing any knowledge of the database in the new Site Sonar by utilizing the dropdown menu of common queries. If a user wants to make their own custom query, there is an information page where the test names and the tests JSON keys are listed.

The old version of Site Sonar also used Run IDs to specify the time-frame of the tests. A Run would send Grid jobs asking worker nodes for information, and collect that information for a maximum of 48 hours. Within that time, a certain percentage of all the Grid jobs had to finish for the Run to count and be included. When the Run finished, it would get a Run number and the date on which it finished. For example, a Run ID that started 01/05/21 might run for 48 hours, and then be marked as "Run 1 03/05/21". This was a way to somewhat represent how the ALICE Grid looked on the specified date. This had both benefits and disadvantages. It was easy to add new tests, e.g collecting information about the GPU model. The only thing needed was to add it to the job script. However, only having a short time frame for a Run meant that it was not guaranteed that all the worker nodes had time to run the Grid jobs. Often the sites had no capacity during a Run, while other sites only gave access to certain users. The consequence of this is that not all worker nodes were included in a Run. During the Design Science Research cycle, it was decided with the stakeholders that Site Sonar should move away from the concept of Run IDs. The new version of Site Sonar continuously gathers tests, and a user can specify how old the test should be at a maximum with a dropdown list. The longer the specified time frame, the more likely it is that all the worker nodes are included.

Due to the complexity of the old Site Sonar, it only gets a score of 2/5 in its usability. The new implementation has to refresh the webpage whenever a value is updated, e.g the filter parameters. Therefore the new Site Sonar gets a deducted point, with a final score of 4/5 for usability.

5.5 Maintainability in Site Sonar

The maintainability characteristic is how easily the system can be modified. The sub-characteristics described in ISO/IEC 25010 are: modularity, reusability, analyzability, modifiability, and testability.

5.5.1 Modularity and reusability

Modularity is the degree to which the system is composed of discrete modules. The modularity has decreased in the new version of Site Sonar. This is partly due to the change from having the front-end in React which is component-based, to using JSP and *.res* files to render the HTML pages. In React it would for example be simple to implement a generic list component. AliMonitor could add React to its code base, but it would be inefficient to have a JavaScript framework just for the Site Sonar tool. A better solution might be to create GUI components in the *.res* files by using plain JavaScript with shadow DOM and custom tags.

In the new Site Sonar, the list of sites and the list of individual worker nodes are written in plain HTML and JavaScript. There is some code duplication since the two lists are quite similar. The same goes for the two drop-down menus for grouping and filtering, where they are almost identical, but implemented for each instance. In the future, Site Sonar should have reusable components in the frontend to avoid code repetition and increase modularity.

5.5.2 Modifiability

The modifiable is the degree to which parts of the system can be reused in building other assets. The paper *Modifiability Tactics*[51] describes techniques which can increase a software systems modifiability. The first technique made is that responsibilities should be split up into smaller tasks, such that the cost of modifying a single responsibility is reduced. This will also reduce the probability of side effects to other responsibilities. For Site Sonar this might be that the functionality in the JSP files should be extracted to either instance of classes, or into functions that hold a small responsibility.

The next technique in the paper is that the code should be cohesive. The semantic cohesion should be maintained, meaning that the functionality that relates to a responsibility should be kept within its own respective module. Site Sonar is a small-scale system implemented into the AliMonitor web application with very little complexity, meaning that keeping Site Sonar semantically cohesive is unchallenging. Site Sonar is built so that the main JSP file *index.jsp*, fetches data, sorts it, and then builds the HTML page. The responsibilities might be split into several classes but are unnecessary unless Site Sonar's functionality grows more complex and a better class structure is deemed necessary.

The last technique is reducing the coupling. The paper gives 5 methods of achieving looser coupling. These are encapsulation, wrappers, raising the abstraction level, using an intermediary, and restricting communication paths. Encapsulation is to some degree used in all the other techniques described. The purpose of encapsulating Site Sonars functionality is to reduce the likelihood that changing a module will propagate to other modules.

5.5.3 Testability

Testability is the degree to which test criteria can be established, and be performed to determine that the system is running properly. Site Sonar has only been user-tested but could gain from having unit testing. This would automate the task of checking if the list displays the correct data, and that it sort and collects data from the database correctly.

The old Site Sonar uses components in React to build to encapsulate some of its functionality and avoids code repetition. However, it does not split all its functionality into discrete modules and has certain issues which have to be refactored. One example is the inline styling of HTML tags. Based on these characteristics, and what was discussed, the new Site Sonar gets a score of 2/5 for Maintainability. The old Site Sonar scores slightly better with a 3/5.

5.6 Quality of software

The evaluation in this chapter discussed both the old Site Sonar and the new Site Sonar implementation. Table 5.1 gives a summary of the scores for each of the quality criteria mentioned in this chapter.

Criteria	Old Site-Sonar	New Site-Sonar
Functional Suitability	3/5	4/5
Performance Efficiency	2/5	3/5
Usability	2/5	4/5
Maintainability	3/5	2/5
Total Score	**10**	**13**

Table 5.1: Comparing software quality of the old and new Site Sonar

Based on the evaluation, the overall quality has gone up in the new implementation of Site Sonar, with a score of **13**. The maintainability criteria scores are lower in the new Site Sonar. The increase in software quality happened in functional suitability, performance efficiency, and usability. The decrease in maintainability was expected, due to the timeframe of this project and that the development stack used to implement the frontend is older, and thereby not as full of features as newer frameworks are. To refactor the new Site Sonar, it should improve the modularity, modifiability and reduce code repetition.

Chapter 6

Conclusion

This thesis has identified and created an effective tool for visualizing the worker nodes belonging to the sites in ALICE's Grid. Based on both qualitative and quantitative analysis following the ISO/IEC 25010 method, it can be concluded that the new Site Sonar implementation has been a success, although still lacking in some aspects. The results from this thesis give the groundwork for the further development of Site Sonar.

6.1 Concluding the research questions

There are five activities in the DSR methodology that investigates the problems, define requirements, designs the artefact, develops the artefact, and lastly demonstrate and evaluate the artefact[20]. The research questions from section 1.6 were created through these activities.

From the DSR methodology there are five activities used in this thesis:

Activity 1 (A1) **Explicate problem**: investigate and analyze a problem and its causes.

Activity 2 (A2) **Define requirements**: find solution to the problem and how to develop the artefact to solve it.

Activity 3 (A3) **Design and develop artefact** create the artefact based on the requirements.

Activity 4 (A4) **Demonstrate artefact**: demonstrate the artefact in a real-world situation to argue that it improves or solves the problem.

Activity 5 (A5) **Evaluate artefact**: determine how well the solution fulfills the requirements.

6.1.1 RQ1: What functionality and qualities are important for Site Sonar to have?

This research question involves the relevance cycle of DSR, where it uses the A1 and A2 activities to determine the requirements for Site Sonar. The first step was to explicate the problem. Mainly it sought to solve the same issue that the old Site Sonar implementation did, but also improve upon the usability, maintainability, performance efficiency and he functional suitability. The requirements for the new implementation of Site Sonar were the following:

1. **Display all possible test names and test values** - Not all values are displayed. Dropdown menus help, and info page.
2. **Filter and group nodes by site** - E.g a user can group worker nodes on the ones who has Linux CentOS. The resulting list will then display how many worker nodes under each site support the grouping parameter.
3. **Display worker nodes under a specified site in a list** - E.g A user can specify a site, and then based on the filter and grouping parameter will get a list of the resulting worker nodes under that site.

4. **Query sites on detailed search parameters** - E.g a user can request all worker nodes who has Linux CentOS installed.
5. **Share URL with search parameters** - E.g a user can put in their search parameters, and then share those parameters through a URL.
6. **Quick load time, and optimized querying of data.** - It is important that Site Sonar does not take to long to load. The consequence of long loading times might be that the Site Sonar will not be used.

The future developers of Site Sonar should consider requirements involving encapsulating code into classes and functions to increase the modularity and reusability. Also, the time performance of the queries and response time of Site Sonar has a potential for improvement.

6.1.2 RQ2: What kind of monitoring tools exists, and which are comparative to Site Sonar?

MonALISA collects monitoring data from worker nodes on ALICE's Grid similar to Site Sonar. However, the approach is different. In MonALISA, it is predefined what it is monitoring for. Modifying this takes time, and demands that code is changed and that new versions are rolled out every time something new should be monitored. With Site Sonar this process is much simpler, where test scripts can be added to the Site Sonar directory in CVMFS and very few changes to the Job Agent are needed.

Site Sonar also has a predecessor, called the old Site Sonar. The old Site Sonar served as a "proof of concept" for the new Site Sonar which is implemented into AliMonitor. However, as discussed in this chapter, the new version improves upon the old version in several aspects.

6.1.3 RQ3: How can monitoring assist the JAliEn developers in distributing the JAliEN middleware?

By giving users the ability to sort and list worker nodes individually and by sites, it provides a helpful tool to the JAliEn developers. They have been a special focus for this project, since they are in need of a system which can get an overview of which sites can run JAliEn. Especially interesting, is knowing what worker nodes can containerize JAliEn if they can't run it in the worker nodes environment. This is part of the DSR activities A5 (evaluating the artefact) and A4 (demonstrate artefact in a real-world situation).

6.1.4 RQ4: How can the graphical user interface be designed to be trivial to use by users that are non knowledgeable of the database structure.

The dropdown menus in the Site Sonar dashboard gives users the ability to select predefined queries which are common. This way they don't need any prerequiste knowledge of how they should use it. Site Sonar also gives the ability to look up the names of the tests, and the tests JSON keys in a table under the *info page*. This is part of the DSR activities A3 (design and developing the artefact) and A2 (defining the requirements), where these requirements and ideas emerged from the design cycles.

6.1.5 RQ5: How should the data collected from nodes on a Grid site be formatted to be practically and efficiently queried.

A problem that was facing the development of Site Sonar was that the data collected stored the values from the test-scripts in a single string. The string could contain many parameters depending on the test-script. This made it hard to query on those parameters. The solution was to parse the test-script data into JSON format, so that the parameters could be fetched by a key. The data is parsed to JSON in the backend and stored in the *sitesonar_tests* table.

6.2 Validity of research

The Design Science Research methodology is a well designed approach for creating software systems. Moving through the rigor, design and relevance cycle creates a product which is appropriate to the stakeholders. However, it depends on the stakeholders being able to recognize what they want to achieve. If the stakeholders are not able to give clear requirements the outcome might be unproductive.

One should consider adding criteria that are common in any good system. For instance, during the production of the new Site Sonar, there was not a lot of focus on the security aspects of the system. In the future developers could address creating quality requirements by using the DSR methodology with for example the ISO/IEC 25010 standard [48]. This way the requirements would have to fulfil the relevant characteristics and sub-characteristics.

One part of the relevance cycle in the DSR methodology is that it should be field tested. Ideally, Site Sonar should have been tested and evaluated by some of its intended users. Due to the ongoing Covid-19 pandemic the field testing was limited. The evaluation was therefore solely based on the quality criteria from the ISO/IEC 25010 standard. Another consequence of the pandemic was that Switzerland and CERN had strict rules for visiting, which limited the possibilities of working in person with the stakeholders. To mitigate this, meetings were regularly held online.

The results of the querying time varied by about 20 seconds when tested with the same database and machine. There is an unknown factor which affects the performance. Another issue was that previous implementation of Site Sonar had no benchmarks for its performance. This made it hard to compare the two implementations.

6.3 Contribution of Site Sonar

Site Sonar was well received during the presentation on the *ALICE Software and Computing Days*. There were some viewers who mentioned how Site Sonar could benefit them. One notable discovery made using Site Sonar is that worker nodes on the Birmingham site was still running Scientific Linux 6. ALICE currently recommends sites to run CentOS 7, with a few exceptions. Another discovery brought forward by Site Sonar is that certain sites has been believed to be homogeneous, but turned out to be heterogeneous in certain aspects.

Chapter 7

Future work

This chapter contains the ideas that have emerged in the last phase and which there have not been enough time to either implement or test, that could potentially lead to further improvement of Site Sonar. In the short term, the future work is to improve upon querying optimization and the page loading time. This has to happen by investigating more effective indexes for the database, improving the database structure, and optimizing the grouping and filtering algorithms in the front end. One consideration the developers should address is changing the database structure so that the *sitesonar_tests* table doesn't have to join the *sitesonar_hosts* table to get the site of a test.

In the short term it can also be useful to perform tests on users to get valuable feedback on the user interface and the current functionality. Most likely there are improvements that can be implemented which will make it more applicable to a broader set of users. One thing that is not implemented, but could be of value to some users, is allowing multiple filters simultaneously. Site Sonar could also benefit from having unit tests, which would ease the process of troubleshooting.

In the longer term, the Site Sonar application should be more modular, increase its maintainability and code reusability. One example of reducing the code repetition could be having the grouping and filtering menus be created as generic components. Code segments with small responsibilities should also be refactored into functions, which will make Site Sonar's code easier to understand and develop. Another thing is that Site Sonar could have the option for more advanced queries, similar to what the old Site Sonar version had. This way it could both have the benefit of being very user friendly for basic needs, and give the additional tools for those who are knowledgeable of the database structure and SQL to create more advanced searches. The security should also be improved, where for example prepared statements should be used to avoid SQL injections.

Bibliography

[1] K. Aida, “Effect of job size characteristics on job scheduling performance,” in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 1–17, Springer, 2000.

[2] A. Hevner and S. Chatterjee, “Design science research in information systems,” in *Design research in information systems*, pp. 9–22, Springer, 2010.

[3] L. Stoel, M. Barnes, W. Bartmann, F. Burkart, B. Goddard, W. Herr, T. Kramer, A. Milanese, G. Rumolo, E. Shaposhnikova, and G. Switzerland, “High energy booster options for a future circular collider at cern,” 05 2016.

[4] “Longer term LHC schedule.” `https://lhc-commissioning.web.cern.ch/schedule/LHC-long-term.htm`, 2021. [Online; accessed 25-Oct-2021].

[5] “Welcome to the Worldwide LHC Computing Grid.” `https://wlcg.web.cern.ch/`, 2021. [Online; accessed 14-Oct-2021].

[6] “MonALISA Repository, running jobs per user.” `http://alimonitor.cern.ch/display?page=jpu_agg/jpu_RUNNING`, 2021. [Online; accessed 04-Oct-2021].

[7] “MonALISA System Design.” `http://monalisa.cern.ch/monalisa__System_Design.htm`, 2021. [Online; accessed 06-Oct-2021].

[8] “ALICE Grid Monitoring with MonALISA.” `https://jalien.docs.cern.ch/site/monalisa/`, 2021. [Online; accessed 06-Oct-2021].

[9] “ISO 25000 software and data quality, ISO/IEC 25010.” `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010`, 2021. [Online; accessed 02-Nov-2021].

[10] H. B. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu, "Monalisa : A distributed monitoring service architecture," *CoRR*, vol. cs.DC/0306096, 2003.

[11] C. CERN Education and O. Group, "Cern annual report 2019," 2019.

[12] "Key Achievements." `https://home.cern/about/key-achievements`, 2021. [Online; accessed 17-Sep-2021].

[13] "INFN, ALICE." `http://www.sa.infn.it/ricerca/alice.html`, 2021. [Online; accessed 23-Sep-2021].

[14] "CERN, ALICE." `https://home.cern/science/experiments/alice`, 2021. [Online; accessed 05-Sep-2021].

[15] "Ian T. Foster: Argonne National Laboratory." `https://www.anl.gov/profile/ian-t-foster`, 2021. [Online; accessed 05-Sep-2021].

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

[17] S. Schreiner, S. Bagnasco, S. S. Banerjee, L. Betev, F. Carminati, O. V. Datskova, F. Furano, A. Grigoras, C. Grigoras, P. M. Lorenzo, A. J. Peters, P. Saiz, and J. Zhu, "Securing the AliEn file catalogue - enforcing authorization with accountable file operations," *Journal of Physics: Conference Series*, vol. 331, p. 062044, dec 2011.

[18] F. Magoulès, *Fundamentals of grid computing: theory, algorithms and technologies.* CRC Press, 2009.

[19] R. J. Wieringa, *Design science methodology for information systems and software engineering.* Springer, 2014.

[20] P. Johannesson and E. Perjons, *An introduction to design science.* Springer, 2014.

[21] A. R. Hevner, "A three cycle view of design science research," *Scandinavian journal of information systems*, vol. 19, no. 2, p. 4, 2007.

[22] "The Large Hadron Collider." `https://home.cern/science/accelerators/large-hadron-collider`, 2021. [Online; accessed 05-Sep-2021].

[23] "ALICE | ALICE Collaboration." `https://alice-collaboration.web.cern.ch/`, 2021. [Online; accessed 14-Oct-2021].

[24] “Particle identification in ALICE boosts QGP studies.” `https://cerncourier.com/a/particle-identification-in-alice-boosts-qgp-studies`, 2021. [Online; accessed 05-Sep-2021].

[25] Č. Zach, “Study of the grid infrastructure and middleware of the alice project at cern and preparation for simulations of networked data processing at regional centers,” 2010.

[26] W. H. Trzaska, A. Collaboration, *et al.*, “New alice detectors for run 3 and 4 at the cern lhc,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 958, p. 162116, 2020.

[27] 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, p. 03037, EDP Sciences, 2019.

[28] “WLCG Worldwide LHC Computing Grid.” `https://wlcg.web.cern.ch/`, 2021. [Online; accessed 22-Sep-2021].

[29] “The Worldwide LHC Computing Grid (WLCG), Dealing with the LHC data deluge.” `https://home.cern/science/computing/grid`, 2021. [Online; accessed 11-Nov-2021].

[30] “MonALISA System Design.” `https://home.cern/science/computing/grid-system-tiers`, 2021. [Online; accessed 07-Oct-2021].

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

[32] M.M.Storetvedt, “PhD in writing,” 2021-.

[33] G. M. Kurtzer, V. Sochat, and M. W. Bauer, “Singularity: Scientific containers for mobility of compute,” *PloS one*, vol. 12, no. 5, p. e0177459, 2017.

[34] “Singularity: The Inner Workings of Securely Running User Containers on HPC Systems.” `https://archive.fosdem.org/2017/schedule/event/singularityhpc/`, 2021. [Online; accessed 18-Oct-2021].

[35] “Docker homepage.” `https://www.docker.com/`, 2021. [Online; accessed 12-Nov-2021].

[36] “Podman homepage.” `https://podman.io/`, 2021. [Online; accessed 12-Nov-2021].

[37] "Docker vs. Singularity for data processing: UIDs and filesystem access." `https://pythonspeed.com/articles/containers-filesystem-data-processing/`, 2021. [Online; accessed 12-Nov-2021].

[38] "MONALISA: MONitoring Agents using a Large Integrated Services Architecture." `http://monalisa.cern.ch/monalisa.html`, 2021. [Online; accessed 08-Dec-2021].

[39] "React JS homepage." `https://reactjs.org/`, 2021. [Online; accessed 26-Oct-2021].

[40] "Gunicorn homepage." `https://gunicorn.org/`, 2021. [Online; accessed 05-Nov-2021].

[41] "Tomcat homepage." `http://tomcat.apache.org/`, 2021. [Online; accessed 26-Nov-2021].

[42] "Site Sonar Backend Mirror Git repository." `https://github.com/KalanaDananjaya/Site-Sonar-Backend-Mirror`, 2021. [Online; accessed 29-Nov-2021].

[43] "Site Sonar Mirror Git repository." `https://github.com/KalanaDananjaya/Site-Sonar-Mirror`, 2021. [Online; accessed 29-Nov-2021].

[44] "Site Sonar Frontend Mirror Git repository." `https://github.com/KalanaDananjaya/Site-Sonar-Frontend-Mirror`, 2021. [Online; accessed 29-Nov-2021].

[45] "Site Sonar ML-Client Mirror Git repository." `https://github.com/KalanaDananjaya/Site-Sonar-ML-Client-Mirror`, 2021. [Online; accessed 29-Nov-2021].

[46] "Figma Homepage." `https://www.figma.com/`, 2021. [Online; accessed 30-Nov-2021].

[47] "Site Sonar Gitlab repository for parsing test message to JSON." `https://gitlab.cern.ch/jalien/site-sonar/-/tree/master/sitesonar.d`, 2021. [Online; accessed 30-Nov-2021].

[48] H. Nakai, N. Tsuda, K. Honda, H. Washizaki, and Y. Fukazawa, "A square-based software quality evaluation framework and its case study," in *2016 IEEE Region 10 Conference (TENCON)*, pp. 3704–3707, 2016.

[49] "ISO 25k documentation on Usability." `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010/61-usability`, 2021. [Online; accessed 06-Dec-2021].

[50] J. J. Garrett *et al.*, "Ajax: A new approach to web applications," 2005.

[51] F. Bachmann, L. Bass, and R. Nord, “Modifiability tactics,” tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2007.