Batch Queue Interface for the JAliEn Grid Middleware

Haakon André Reme-Ness

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,

Western Norway University of Applied Sciences

Department of Informatics,

University of Bergen

16 June 2020

Acknowledgements

I would like to give a huge thanks to my supervisors, Bjarte Kileng, Håvard Helstrup and Kristin Fanebust Hetland. The support and guidance received, while both writing the thesis and developing a solution, have been invaluable. A big thanks goes to Maxim Melnik Storetvedt as well. The insight into the inner workings of JAliEn, and the help with the local setup used in this thesis have been a blessing. To all friends that have encouraged and supported me, thanks! Lastly I would like to show my gratitude to my family. The love and encouragement received during this whole education is always appreciated.

Haakon André Reme-Ness

Drøbak, June 2020

Abstract

ALICE (A Large Ion Collider Experiment) is one of the four large experiments located on the LHC (Large Hadron Collider) at CERN. ALICE produces a lot of experimental data. To help store and analyze all the experimental data, a computing grid is used. A computing grid refers to a collection of geographically distributed data sites, used to collectively share resources to solve large and complex computing problems.

A Grid middleware is a software used to interface with a computational grid. An increase in the production of experimental data accentuated the need for a new grid middleware for ALICE. The new grid middleware under development, named JAliEn, aims to provide a more scalable solution.

Job schedulers are used to submit jobs for execution and manage the job queue. They are used on sites in a computing grid to provide a computing capacity.

This thesis aims to provide a solution for an interface between JAliEn and job schedulers. Requirements for a viable solution is described. Testing the solution revealed that not all requirements could feasibly be tested, and some were not completely fulfilled. Nonetheless, the testing indicated that interface could possibly be a viable solution for JAliEn.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Problem Introduction

The Large Hadron Collider (LHC) is the world's largest particle accelerator and built by CERN. There are four large experiments making use of the LHC. ATLAS, LHCb, CMS and ALICE. A Large Ion Collider Experiment (ALICE) is the one relevant for this thesis. ALICE is studying heavy-ion collision. Detectors in the LHC produce huge amounts of data. Storing and analyzing a large amount of data puts a demand on computing capabilities. To accommodate the demands a computing grid is used. This opens up for the possibility of spreading the load. Different sites spread around the world will be able to run a job processing or analyzing data. The sites which form the computing grid also have a local job scheduler to effectively use its own resources. A computing grid is a collection of different heterogeneous sites. This means that several different job schedulers exist in the grid. The computing grid is required to interface with the different job schedulers available on the sites.

1.2 Objective

A computing grid requires software to provide functionalities and use of available resources. ALICE is using a software called ALICE Environment (AliEn). AliEn fulfilled the old demands, but it is complex and not cheap to maintain and update[15]. A new software that is able to scale with future demands is prefered. The new software under development is called Java ALICE Environment (JAliEn). JAliEn is written in a different programming language. There are also some changes to design. An interface to communicate with local job schedulers on the sites, is still required. JAliEn therefore needs a new implementation of a job scheduler interface. Sites in the computing grid have different configurations. The new solution for the job scheduler interface must work on old site configurations. In addition the solution for a job scheduler interface must not break JAliEn. This means that errors must be handled and contained.

“How to implement a viable solution for a job scheduler interface in JAliEn that is capable of running with old site configurations”

1.3 Requirements

Requirements must be outlined to ensure a viable solution is met. JAliEn imposes requirements for the solution when it comes to functionality. The solution is also required to work on old site configurations. This increases the complexity of the solution. In addition to functionality the solution must not break or create issues for the rest of JAliEn software. To ensure this, requirements for robustness will also be formulated. The solution needs to meet both demands to be viable.

1.3.1 Functional Requirements

To function within JAliEn some methods need to be implemented.

- **Submitting jobs**
- **Getting the status of the queue, both active and queued jobs**

The implementation for the given methods should be a general solution. It should not require much site configurations. A site configuration with minimum information should still be able to function.

The solution should be backwards compatible, making sure sites with old information still works on the new solution. Site configuration working with the old software should be capable of working with the new software.

The solution should follow basic principles for quality code. An important principle is low coupling. JAliEn is a software in development and is exposed to changes. Low coupling ensures that the solution functions independently from other parts of the system.

There are a lot of different job schedulers available within the grid. In the future new job schedulers could be added to sites on the grid, while others are removed. Adding more job scheduler interfaces in JAliEn must be cheap. An interface for the required methods for the different job schedulers is a valid strategy. Each job scheduler interface must then implement the given methods.

Duplicate code should be avoided. To ensure good quality duplicated code should not be a part of the solution. This would make the code lengthy and require more time to compile and run.

1.3.2 Robustness requirements

Meeting the functional requirements is not enough to call the solution viable. The solution needs to be robust. As discussed earlier the solution should not make the rest of JAliEn crash. A software can not be expected to always run in a perfect environment. The solution can not create issues in different parts of the grid. This is imperative for a viable solution.

The solution should need minimum manual interference. If it encounters errors with the job scheduler interface this should only affect the site's ability to execute jobs.

If there is an issue with the job scheduler interface the site should not advertise to the grid that it is capable of running jobs. Ensuring the job in a computing grid is processed is a priority. A grid job should not be lost because a site is given a job while not being able to execute it.

The solution should be aware of exceptions occurring, and handle them accordingly. Exceptions must not be silent. The job scheduler interface should communicate information about exceptions.

Errors might occur because of invalid inputs, but also as a product of heavy loads. The solution needs to handle both cases.

1.4 Thesis overview

Chapter 2: Provides a background for the problem this thesis aims to solve, as well as a description of related technology.
Chapter 3: Will discuss in detail both AliEn and JAliEn and the differences. Life cycle of a job in JAliEn will then be explained.
Chapter 4: The development process will be described and a solution provided
Chapter 5: Will describe testing of both functional requirements and robustness requirements. The results of the tests will be presented.

Chapter 6: The test result will be evaluated to find out if the solution fulfilled the requirements.
Chapter 7: Will provide a conclusion to this thesis by discussing the evaluation of the tests results. At last a look at future work is provided.

Chapter 2

Background and Technologies

This chapter will start by providing a more detailed background for the problem that this thesis aims to solve. In addition, relevant technologies will be explained. Basic understanding of the technologies relevant to the problem is needed to understand the solution presented in chapter 4.

2.1 CERN

The European Laboratory for Particle Physics, or commonly abbreviated to CERN, works to uncover what the universe is made of, and the logic behind how it works[1]. The CERN laboratory is located at the Franco-Swiss border near Geneva. 23 different countries are members of CERN. CERN houses some of the worlds largest and most complex scientific instruments to study fundamental particles.These instruments vary in size and functions. Particle accelerators and detectors are examples of instruments that are used. One of the particle accelerators is the Large Hadron Collider (LHC)[2].

LHC is a particle accelerator and was first started up in 2008. The LHC is a 27 kilometer long ring, and the largest particle accelerator in the world. Inside are superconducting magnets with accelerating structures to help boost the particle energy. Two high-energy particle beams are traveling at close to the speed of light. They are then forced to collide with each other at four different locations. These locations are housing particle detectors corresponding to the four experiments: ATLAS, CMS, LHCb and ALICE[3]. ALICE is the one of interest for this thesis.

2.2 ALICE

ALICE is a heavy-ion detector located at the LHC ring. It is used to study a phase of matter called quark-gluon plasma, which is formed in conditions similar to right after the Big Bang[3]. The ALICE detector produces huge amounts of data. This detector is not continually producing data all the time, but is instead active in time periods described as Runs are active for a time period of a couple of years. Run 2 produced 4GB/s, or up to tens of petabytes in a year. Run 3 and Run 4 is expected to produce even more data[13]. A solution to store, analyze and process huge amounts of data is needed. This need is increasing since data produced by the ALICE detector is not expected to decrease over time, but increase[4][5].

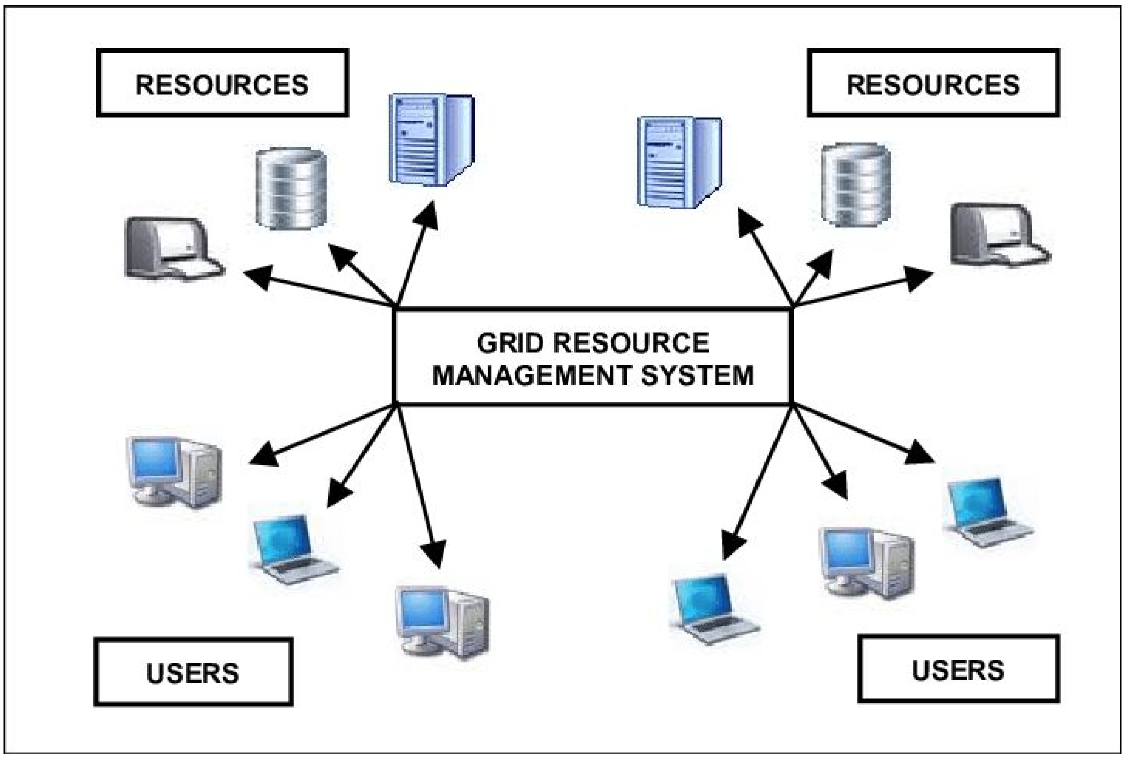


Figure 2.1: *An overview of how a computing grid can look like[25].*

2.3 Grid Computing

With the massive amounts of data produced from the experiments, one computing site would not be optimal for providing enough resources to effectively store and analyze all the data. This is where grid computing enters the picture. The concept called grid computing started in the mid 1990s, and can be compared to the electric power grid. Users are plugging into the socket and receiving electricity without having any knowledge of where the electricity is produced. In the same way a user would have the grid compute a job without having any knowledge of which resource on the grid is processing the job[6]. The figure 2.1 provides an overview of a computing grid and how a user makes use of a computing grid. Ian Foster formulated a three point checklist for the definition of a grid[7]. He describes it as a system that:

1. "...coordinates resources that are not subject to centralized control..."
2. "...using standard, open, general-purpose protocols and interfaces..."

3. “...to deliver nontrivial qualities of service”

To describe it more informally, a grid consists of resources spread over several domains using general interfaces for interaction and provides a service that the individual resources are unable to provide by itself. Clusters of connected machines spread across the world is an example of resources for a computing grid. These individual clusters, part of a computing grid, will be referred to as a site.

There exist several grids today, many serving different purposes. Examples of grids are the European Grid Infrastructure (EGI)[1] and Open Science Grid (OSG)[2]. ALICE uses grid resources provided by the Worldwide LHC Computing Grid (WLCG). WLCG is the world largest grid, and consists of 170 computing centers in over 40 different countries. WLCG is supported by a collection of other grids, including EGI and OSG. WLCG is split into different tiers. Each tier provides a specific service to the grid. Tier-0 refers to the CERN Data Centre[8].

2.4 Grid Middleware

Since a grid consists of heterogeneous sites spread geographically and in different domains, a software to interface with the grid is required. This software needs to make use of resources the grid provides, and maintain a control of the grid status. This software is called a grid middleware. It provides a framework for user access to the grid, and managing resources available and other functionalities. It is important that a grid middleware is able to utilize all available resources on the grid. Grid middlewares can provide several functionalities to the grid. Examples of such functionalities are[6]:

- Data Storage
- Data management
- Security
- Workload management

Not all grid middlewares provide all the functionalities mentioned. Several grid middlewares are available. Examples include ARC, Globus, AliEn and JAliEn. AliEn and JAliEn are the

[1] https://www.egi.eu/
[2] https://opensciencegrid.org/

two grid middlewares relevant in this thesis. AliEn was used by ALICE during Run 2. JAliEn is planned to take over for AliEn.

2.5 AliEn

Alice Environments (AliEn) is a distributed computing environment that gives the user access to resources provided by the WLCG. AliEn also provides other services, both optional and not, for a grid. Services include security, data management, data storage. AliEn is based on defined standard protocols. Web services are used for communication within the grid. Development started in 2000, and already in the end of 2001 it was deployed to sites. To prepare for Run 3 the ALICE experiment will get extensive hardware and software upgrades. Detectors will produce data fifty times more frequently compared to Run 2[13]. The amount of data produced will therefore be a lot higher in Run 3 compared to Run 2. The increase in the amount of data, and the likelihood of the amount of data increasing even further in the future, puts some demands on the system. The grid requires a grid middleware that is able to scale for the future. AliEn does not scale very well. The software is partly outdated and it is costly to make changes to AliEn. That is why the development of JAliEn started.

2.6 JAliEn

Java Alice Environment (JAliEn) is the new grid middleware planned to take over for AliEn. JAliEn is being implemented in an effort to reduce complexity and make both operation and maintenance cheaper. This will help ensure the software is able to scale for the future. JAliEn is, as the name implies, written in Java. Some of the architecture is shared by both AliEn and JAliEn, but some changes are made. In chapter 3 both AliEn and JAliEn will be discussed more in-depth to help understand how both grid middlewares function. JAliEn aims to provide the same services as AliEn.

2.6.1 Running a job on a site

One of the crucial parts of a computing grid is its ability to execute jobs. A job on the grid must be assigned to a site for execution, but only to a site capable of executing the specific job. The term grid jobs will be used to describe a job on the grid. In JAliEn a site will communicate that it is available to receive a grid job. To ensure the site receives a grid job it is able to execute, information about the site is communicated. Information about the sites are described in a server located centrally on the grid. The server that provides this service is a LDAP server.

Sites are capable of handling more than one job at a time. There is a need to make effective use of all the resources available on the site. By using a job scheduler the sites can make the most use of its resources and be capable of effectively queuing and executing several jobs simultaneously. There are several different job schedulers used by the different sites, and JAliEn needs to be able to interface with all the different job schedulers used by sites on the grid. It is important to note that the interface in JAliEn for job schedulers will be called a batch queue interface throughout this thesis.

2.7 LDAP

Since the grid used by JAliEn consists of many heterogeneous sites, JAliEn wants to maintain control of all the different sites on the grid and information describing the sites. This is solved by describing the site in a Lightweight Directory Access Protocol (LDAP) directory server. The LDAP directory server is a type of database that stores data as a tree of entries. Entries consist of an identifier called distinguished name, attributes and object class. DN is identifying and telling the position of any given entry. Attributes are holding all the data for the entry. Object classes are used to describe the type of object the entry is representing[9]. Figure 2.2 shows the architecture of an LDAP server.

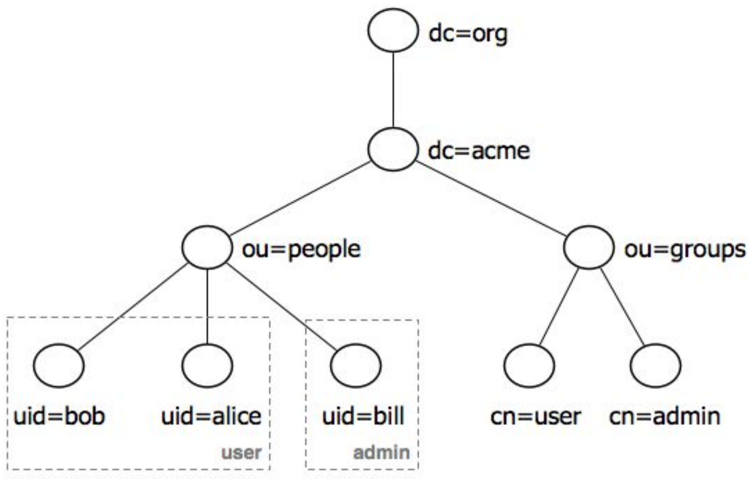


Figure 2.2 *LDAP tree.*

The information found in the LDAP server ranges from directory paths for temporary and log files, to description of the job scheduler used on the site.

2.8 Batch queue and job scheduler

Batch queue, or job queue, is a data structure in which a job is submitted to a job scheduler for a batch processing. The job remains in the batch queue until the resources required are available and it is the job's turn for batch processing. Batch processing is executing a job with no user interaction, and executing when the required resources are available. A grid job is not the same as a job submitted to the job queue. The computing grid has a queue with grid jobs to be executed by a resource on the grid. The job added to the batch queue on a site is what is termed a pilot job[39]. The pilot job works as a placeholder job ensures the resources are available before it hands the resources over to the payload, which is the grid job in this case.

The job scheduler maintains the batch queue, effectively using the resources available. Job schedulers often have to manage several machines, or nodes, within a batch queue. Jobs submitted to the queue has some mandatory attributes that describes the job [10]:

- Who submitted the job to the queue.
- The location for the input file and location where the output will be sent to.
- The executable of the job.
- When the job is supposed to run.

If not described, some of these attributes have default values. The user submitting the job to the job scheduler will always be the "who". The only parameter that needs to be explicitly specified is the executable of the job. A job can be described using a job description file. Some job schedulers also allow for the use of standard input to describe the job. A submit command might also have arguments to describe the job. In addition to the mandatory attributes, other attributes may be described. Examples of other attributes for a job are resources required for the job to be executed, if more than one instance of the job should be submitted, and specification of what resources to be used for execution. Job schedulers will also keep track of the batch queue. It will have information about all the jobs in the job queue, this includes both jobs being executed and jobs waiting. There are a lot of different job schedulers available. While all different job schedulers found on sites in the grid should be implemented in the batch queue interface for JAliEn, only three are discussed in this thesis. The reason for this is the limited time. The three job schedulers that will be discussed are SLURM, TORQUE and HTCondor.

2.9 Slurm

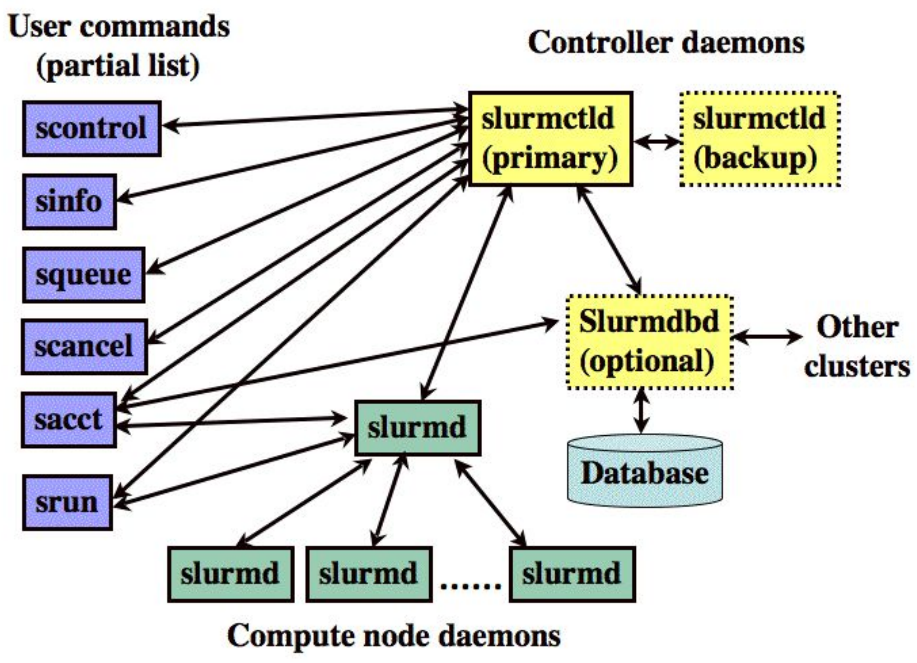


Figure 2.3: *Possible SLURM setup using several nodes[11].*

Slurm is an open source cluster manager and job scheduler for linux clusters. It maintains the job queues and allocates resources to execute jobs. Slurm also has a set of necessary APIs to manage the job queue, and monitor jobs in the job queue. Figure 2.3 describes the Slurm architecture. Jobs are executed on nodes, each running a slurmd daemon. The slurmd daemon is constantly waiting to receive more jobs. This daemon is also responsible for returning the status after a job is complete. Nodes are split into partitions, grouping nodes logically together. This is the different job queues. Jobs in Slurm consist of one or more job steps. One job step might need only part of the resources the job has been given, or all of them[11].

In addition to the worker nodes slurm has a centralized manager, the slurmctld. It monitors and controls the queue and resources available. This makes Slurm able to delegate the jobs to the appropriate node.

There are two different commands to submit a job to SLURM. They serve two different functions. First is the sbatch command. Sbatch submits the job to the job queue. The job will remain there until its turn, and the resources needed for execution is available. The other command is srun. Srun does not add the job to the job queue. It instead executes the job on the node immediately. While srun is capable of running a job by itself, it is usually used to launch a job step. This is done by declaring the Srun to run the executable in a job description . Sbatch will add the job to the job queue and srun will launch the jobstep immediately when the job is being executed. Several Srun can be used to describe more than one jobstep. Srun will have the same parameters as sbatch, unless otherwise declared.

2.10 TORQUE

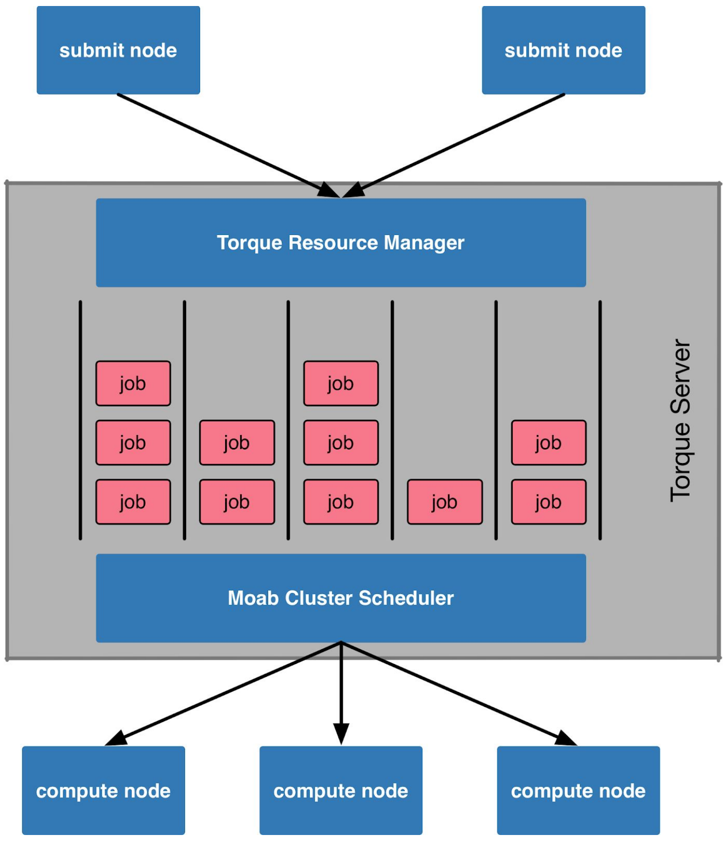


Figure 2.4: *Possible TORQUE setup[40]*

TORQUE is a resource manager that also functions as a job scheduler. Much like SLURM, TORQUE provides services to maintain a job queue. TORQUE has its own unique set of API calls. Figure 2.4 shows how the TORQUE architecture works. TORQUE consists of one master node and many computing nodes. The master node is running a *pbs_server* daemon and a *pbs_sched* daemon[12]. When a user submits a job to TORQUE, the job gets submitted to the *pbs_server* daemon running on the head node. The head node will then communicate with the *pbs_sched* daemon to find an available computing node for the job. The *pbs_sched* daemon responds with a list of available nodes. The *pbs_server* daemon will then send the job to the first computing node in the list for execution. On all of the computing nodes *a pbs_mom* is running. It is responsible for killing, starting and managing submitted jobs.

TORQUE offers the possibility of staging files when a job is executed. There are two options, stage in and stage out. Sometimes the data needed for executing a job is not available from the computing node. Staging files solve this problem by moving the data required over to the computing node for use before executing the job. When the job is finished the data is removed from the computing node. This is what in TORQUE is called stage in. Stage out is moving the output of the job to another location, then removing it from the computing node. Staging input files for execution of jobs is relevant for later implementation of a batch queue interface for JAliEn.

2.11 HTCondor

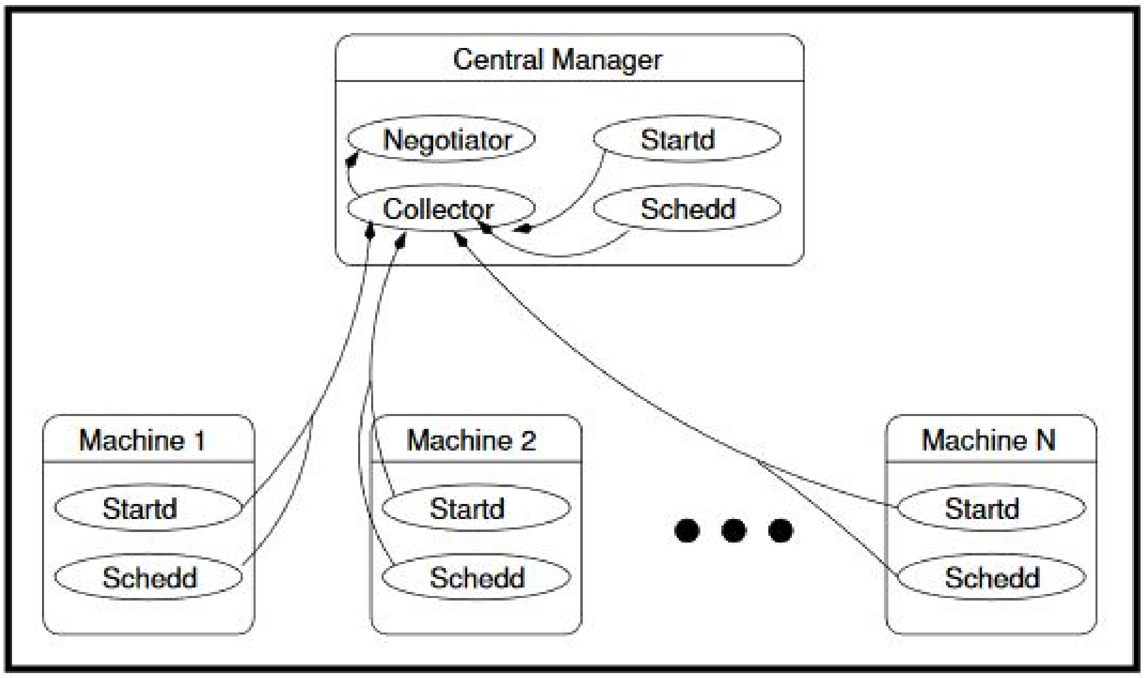


Figure 2.5: Possible HTCondor setup[36].

HTCondor is a specialized workload management system for compute-intensive jobs[30]. HTCondor was known as Condor until 2012. HTCondor is also able to function as more than a regular job scheduler, as will be shown. HTCondor provides a unique set of API calls. HTCondor has a unique way to make use of resources not dedicated to computing jobs. Resources, or machines, that have the state of idle is available for computing. When the resource is not idle anymore it can no longer be used for computing jobs. A reason for the change of status from idle can be because an input like a mouse click was detected. HTCondor might then create a checkpoint on the job[35]. The job might then be transferred to another machine on the cluster if that is optimal, or wait for the resource to become idle again. This is an effective solution for creating a cluster used for job scheduling, using non-dedicated machines. This is however not that relevant for this thesis. JAliEn sites using HTCondor will have dedicated machines on the cluster.

The figure 2.5 shows the HTCondor job scheduler architecture. HTCondor consists of a central manager and several machines, called a resource pool, used for execution of jobs. The central manager has control over the information and status of the resource pool. Both the negotiator and collector work together to maintain control over the machines available in the resource pool and distribute the jobs effectively to the resource pool.

The machines in the resource pool need to be able to share information about itself and resources available. The startd daemon will publish what is called a classad. The classad contains information about the machine concerning state, available resources and other relevant information. This will inform the central manager how many resources are available on the machine. It will then use this information to perform matchmaking between the machines and jobs. To make a user able to submit a job to the job queue on a machine, the machine needs to run the schedd daemon. The schedd daemon is also used when monitoring the job queue.

HTCondor also offers the option to use what HTCondor calls a grid universe. A universe defines an execution environment. HTCondor will then function as a frontend service receiving jobs before delegating them to a backend service that will execute the job. The backend service is called the grid resources and can be another job scheduler, like SLURM, or other grid services, like ARC. A job using the *Grid* universe must specify the grid resources to be used. This is one of two universes that HTCondor provides that are relevant to this thesis. The other universe used is *Vanilla.* This universe has few requirements and is the default universe in HTCondor. What universe to use is described in the job description.

Job routing is another service HTCondor offers. It is an additional functionality to the schedd daemon. Job routing is used to transform the job from one universe to another, to route the job to a different grid service for execution. A job router daemon sets the policy for transforming jobs. The job description decides if the job should be available for job routing

Chapter 3

AliEn and JAliEn

In this chapter a more in-depth description of both AliEn and JAliEn will be provided. AliEn will be described first, before differences between AliEn and JAliEn will be discussed. A look at the architecture of JAliEn will also be discussed. Knowledge of JAliEn will be valuable to understand how the solution for job schedulers is implemented in JAliEn. After an overview over the life cycle of a grid job in JAliEn will be described.

3.1 AliEn

As mentioned in chapter 2, AliEn is the grid middleware that was used for run 2. It is the currently distributed grid middleware on ALICE sites. AliEn consists of around 200 packages and is written in Perl, C and C++. AliEn provides functionalities like security, data management and grid job execution on the grid. Many key aspects remain the same between AliEn and JAliEn despite the difference in programming languages. Before looking at JAliEn, a rough overview of AliEn will be described. Knowledge of how AliEn works is helpful to understand how JAliEn have made changes and improvements.

3.1.1 Security

AliEn provides access to many resources spread across the world on different sites. It is important that access to these resources are secure. Security is even more important since communication between parts of AliEn partially happens over public networks. This is a result of sites being widely spread. AliEn must ensure that only valid users have access to the grid. Valid users must also have access to all the right resources. This alone is not enough to ensure a secure system. Since communications are partially done using public networks, data confidentiality and data integrity is important. It is crucial that different components of AliEn can trust each other. This provides a rough idea of what type of security is required from AliEn.

3.1.1.1 Virtual organization

A virtual organization (VO) is a collection of individuals or institutes spread across the globe, working towards a common goal. Users that are part of a VO have access to all the resources shared by this VO. AliEn uses this to control access for users. Only users that are part of the ALICE VO have access to the resources AliEn provides. The VO does not help to ensure the identity of the user and user authentication is also required.

3.1.1.2 Authentication

Utilizing Public Key Infrastructure (PKI) it is possible to satisfy some of the security requirements discussed earlier. AliEn uses PKI to manage and distribute X.509 certificates. An entity called the Certificate Authority (CA) has the responsibility of verifying certification requests and signing with its own private key[14]. All users must have X.509 certificates[23] When a user wants access to the grid through AliEn, a user certificate is used with the associated private key. The private key of a user is encrypted using a password. This functions as a login system for the users. When a user is logged on to the grid, a X.509 proxy certificate will be created and signed with the user certificate's private key. This creates a chain of trust between the user and the proxy certificate. The proxy certificate will be valid for only a short time. The proxy certificate will be used to delegate authority. When delegating authority the new proxy certificate could have full authority or limited authority. Using proxy certificates that have a chain of trust all the way up to the CA, trust between components in AliEn can be established.

The concept of PKI can also be used to achieve data confidentiality and data integrity. Messages sent over the public network will be encrypted. Public keys are shared and private keys are kept private. Messages are encrypted with the public key. Only owners of the private key can decrypt the message. This provides data confidentiality. To ensure data integrity a message digest is created using a one way hash function. The message digest is encrypted with the private key and sent with the encrypted message. The receiver uses the one way hash function on the received message and the corresponding public key on the message digest. If the two message digests are equal data integrity is maintained.

3.1.2 Data Management

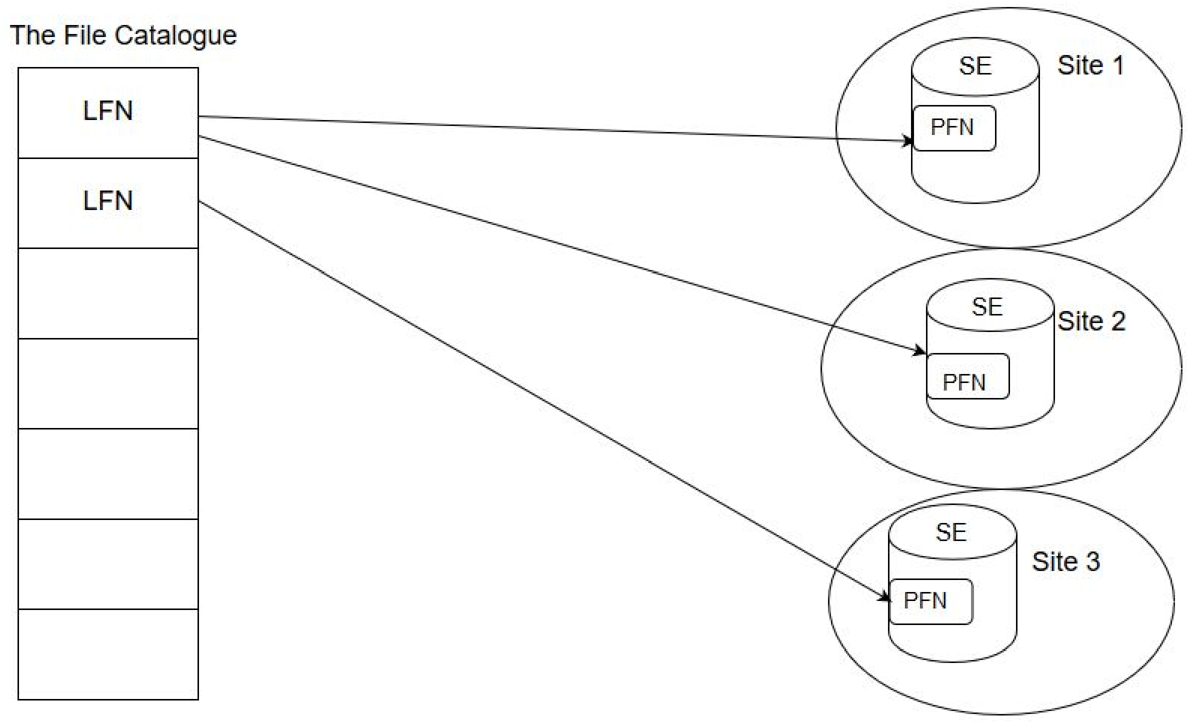


Figure 3.1: *AliEn file system overview*

Massive amounts of data needs to be stored by the ALICE experiments. Storing everything on one site is not optimal. Instead the data should be spread amongst the sites available on the grid. This requires AliEn to be able to manage all the files stored on various sites. The grid must also be able to transfer data between sites.

Storage Elements (SE) are responsible for storing grid data on the different sites. Sites available for storage are running a SE locally. The SE will be able to save and retrieve files from the site's storage. In addition to the Storage Element a File Transfer Daemons (FTD) enables transfering of files. XrootD is also commonly used to store and transfer files. SEs store data locally on the site but do not help with managing all the data stored on the grid. That is the job of the File Catalogue.

The File Catalogue works much like a Unix file system, and users have a separate privileges and a home directory. It also keeps the association between Logical File Name (LFN) and Physical File Name (PFN)[24]. LFN can be described as the grid's file system. PFN on the contrary describes the physical location of the file. This includes describing the path to Storage Element and the local path on this Storage Element. A LFN pointing to several PFN indicates that the file has replicates. The File Catalogue also provides some functionalities for grid jobs. All jobs submitted to the Task Queue will get a separate directory for standard input and output on the File Catalogue.

3.1.3 Packet Manager

A packet manager helps sites with installing, upgrading, configuring, and removing software packages from a shared area. Instead of all sites on the grid needing to do this individually, the packet manager makes this easier. A packet manager will provide the software when needed, and does not require any pre configured software

While there are different options for packet managers, like PacMan[3], the most used one is CernVM-File System (CernVM-FS). CernVM-FS is not a real packet manager, but it provides the same service to AliEn. CernVM-FS is a read only file system similar to a Unix file system[16]. CernVM-FS provides the option of executing the software directly, without having to install the software. That means it suffices to set the path to a software in CernVM-FS to run it while executing a job.

3.2 JAliEn

The goal is for JAliEn to take over for AliEn in future runs. JAliEn aims to provide the same functionalities that AliEn provides. This makes JAliEn not vastly different from AliEn. The difference is that JAliEn is being developed to combat the continuing increase in the amount of data being produced by the ALICE experiments. JAliEn is also written in a different language than ALiEn, that uses Perl[4]. JAliEn is also still under development. Changes can be made under development to improve JAliEn, making the differences larger. Many of the functionalities discussed earlier in this chapter hold true for JAliEn, but there are some

[3] https://www.archlinux.org/pacman/
[4] https://www.perl.org/

changes. An overview of JAliEn will be described, and some improvements made to JAliEn will be highlighted.

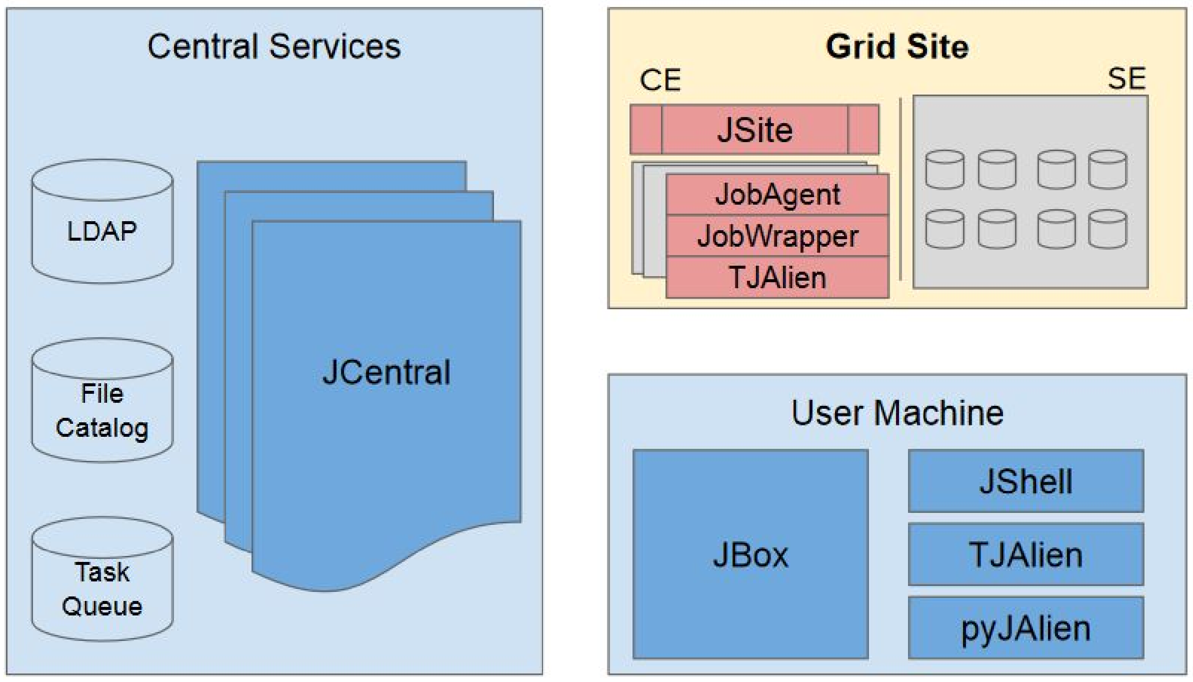


Figure 3.2: *JAliEn architecture.*

3.2.1 JAliEn Components

JAliEn is divided into three different parts that are divided logically. Figure x shows an overview of the three parts and functionalities part of them. Grid site is software running on sites part of JAliEn. User machine is software used by users when interfacing with JAliEn. Central services are services that are shared by all sites and users.

The architecture shown in figure 3.2 is mainly the same as AliEn, but there are some changes to functionalities. Most of the components are updated versions of its AliEn counterpart, with the exception of pyJAliEn, JobWrapper and JSite.

3.2.1.1 Central services

Located at the CERN Data Centre, the central services provides services shared by other components of JAliEn. It manages data concerning grid sites, grid queue and data management. One of servers that is a part of the central services, is the LDAP. The LDAP contains information about all sites on the grid. This includes information about the job scheduler the site is running. This information includes how many jobs the site is capable of running and the type of job scheduler. Information describing how to interact with the local job scheduler for the site is also stored in the LDAP server. This will be discussed more in-depth in chapter 4.

The File Catalogue's responsibility is mapping LFN to PFN for all the files stored in the grid. AliEn and JAliEn use different databases to store this data. AliEn uses a MySQL master-slave setup, while JAliEn uses a Cassandra model[5]**.** Cassandra is used to increase scalability for the increasing amount of data needed to be stored in the File Catalog in run 3.

Then we have the Task Queue which is another database that contains information related to tasks on the grid. It has control over jobs to be run, different job agents running and information concerning what resources a job needs to be successful. The Task Queue is pivotal in how Job Agents match to a job. The grid job will be executed when a Job Agent is running on the sites local job scheduler, and they match.

JCentral provides services to different sites and works as an interface for the different databases and LDAP. No other part of JAliEn has connection to the database and LDAP. There can be other JCentrals working as load balancers. The load balancers do not have direct connection to the databases. JCentral is responsible for matching jobs with different sites, and retrieving and updating information for both LDAP and the databases. Security is another aspect of JCentral's responsibilities.

3.2.1.2 User Machine

Users can interact with JAliEn much like a Unix-shell. Admin users exist, giving more permissions to manage sites. Users have access to a list of commands to interact with JAliEn. Some examples of commands are starting a grid job, monitoring running grid jobs and killing a grid job. Commands related to the file system also exist. Examples are storage, moving, copying and reading files. There are three interfaces used to interact with JAliEn for a user: TJAlien, pyJalien[6] and JShell. TJAlien works as a ROOT[7] plugin[38]. PyJalien is an interface written in python that takes advantage of websockets and JSON. JShell is written in Java and makes use of a command-line interface.

[5] https://cassandra.apache.org/
[6] https://github.com/adriansev/jalien_py
[7] https://gitlab.cern.ch/jalien/jalien-root

Jbox is a daemon running on the user’s machine. It has several responsibilities. One of the responsibilities is authentications. Jbox is used with a user certificate, user key and password to verify the user. Only when a user is verified is it possible to issue commands. The JBox will parse the commands and then establish an upstream connection with either JSite or JCentral. JBox will create a temporary file with the connection details for the instances. A connection will only be established once a command is issued.

3.2.1.3 Grid Site

Sites are able to both execute grid jobs and store data. The components used for executing a grid job are the Computing Element (CE), JobAgent and JobWrapper. It is the CE’s responsibility to get the configuration of the site from the LDAP and submitting the jobs to the local job scheduler. The CE will communicate how many slots are available on the local site’s job queue to JAliEn. If there are available grid jobs and slots on the site’s job queue, the CE will submit jobs to the job scheduler. The jobs that are submitted are generic JobAgents.

In AliEn only JobAgent exists, and it does all work of running a grid job.The functions of the AliEn JobAgent is in JAliEn split into two separate modules, JobAgent and JobWrapper. The JobAgents run by the local job scheduler tries to match with available grid jobs. When a JobAgent matches with a grid job it launches a JobWrapper. The JobWrapper will make the environment for the grid job ready, then execute it. This is different from how it works in AliEn. With JAliEn, both JobAgent and JobWrapper are containerized in two separate containers. This isolates them from the rest of the system, keeping their resources private.

If desirable a site is able to run a SE for storing data. There are no big changes from how AliEn implemented SEs to the JAliEn implementation. SE was discussed earlier in this chapter.

JAliEn must manage a lot of users and sites. Having an individual persistent connections between all of them to the central services would not be desirable. JSite is new to JAliEn and handles this problem. JSite is used to multiplex requests to the central services to a single upstream connection[29]. To achieve this JSite is running on trusted sites.

3.2.2 Security

JAliEn introduces some changes to security. JAliEn features token certificates instead of the X.509 proxy certificates that AliEn uses. There are not many changes beside the change of certificates. A CA is still used to sign and validate certificates, and users still need a certificate with its private key to login on JAliEn. The private key can be encrypted with a password.

3.2.2.1 Token Certificate

A token certificate is an extension of the X.509 model. A X.509 certificate will contain information about both the entity issuing the certificate, as well as the receiver of the certificate. Token certificates in JAliEn are as of now split into three different roles: *user, jobagent* and *jobs*. This helps a more fine tuned control of privileges. The *jobagent* token certificate gives permission to update the job status and match jobs. The *job* token certification gives permission to execute the grid job. The *user* role has all the privileges to the corresponding user.

*Subject: **OU**="queueid=1038905674/user=hremenes", **OU**=jobagent, **CN**=jobagent, **CN**=Jobs, **O**=JAliEn, **C**=ch* *Issuer: JAliEn-CS*

Table 3.1: Token certificate example in JAliEn

Table 3.1 shows how a token certificate can look like. It describes that the JobAgent can operate on the job with the grid queue id provided, and with the privileges of user "*hremenes*" .

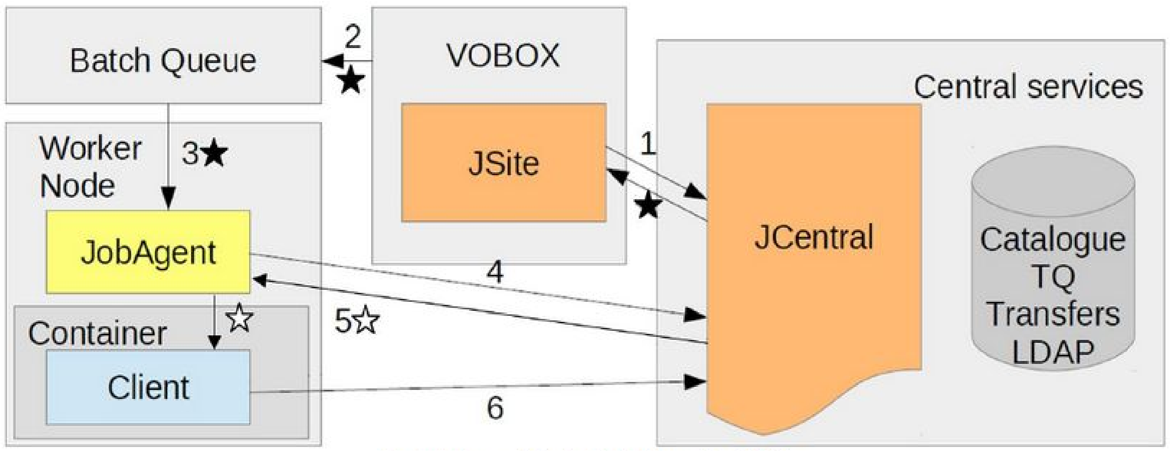


Figure 3.3: *Creation and life cycle for a job agent token certificate*

Figure 3.3 shows how a token certificate, like the token certificate described in table 3.1, is issued and used when executing grid jobs. The CE on a site will request a token certificate for a JobAgent with a generic identity when creating the JobAgent startup script. The script will be submitted to the job scheduler. When the active JobAgent is assigned a grid job it updates the token certificate with the id of the grid job and the user that submitted it. The token certificate will be passed with the payload of the grid job to JobWrapper.

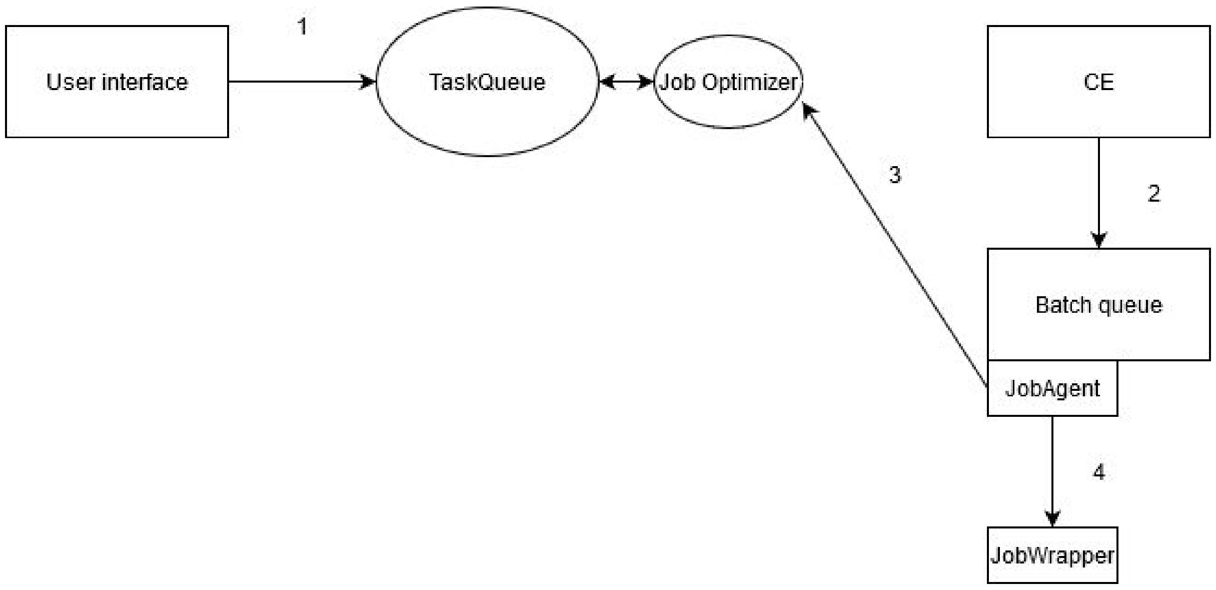


Figure 3.4: *Creation and life cycle of a grid job in the JAliEn system*

3.3 Job life

The picture above gives a rough overview over how jobs move in JAliEn before being executed. The start of a job's life in the grid happens after a user is logged on and submitting a grid job using, as an example, jShell. The job will be submitted to the grid with a Job Description Language (JDL) file describing the job. What the executable for the job is must be described. Before the job is added to the TaskQueue the JDL file will be parsed to make sure it is valid and that the executable exists. The JDL might set requirements for what is needed from the sites to being able to run the job. This could be whether or not the site has any of the files that the job needs.

Now that the job is in the TaskQueue a site needs to pick the job up and run it. To do this every site is running a CE. The CE will create a sitemap of the site using the configuration found in the LDAP server. It will then check with the JAliEn batch queue interface how many jobs are queued or active and compare it with the maximum slots available on the site. Maximum slots will be described in the site configuration. If there is room for more jobs to be submitted it will send the sitemap to the job broker. The job broker is responsible for matching sites with jobs. It matches based on resources and relevant files, as an example, available on the site. The job broker will return the number of jobs currently in the TaskQueue that matches the site map. The lowest number of matching jobs and available slots will be used for submitting JobAgents to the job scheduler. Before submitting a JobAgent, a startup script for the JobAgent must be created. This script contains a token certificate in addition to other parameters that are relevant. When the startup script for the JobAgent is complete it is passed to the JAliEn batch queue interface to be submitted to the job scheduler. The submitted JobAgents are generic ones. No grid jobs are delegated to them yet.

This generic JobAgent will again send a request to the job broker for matching grid jobs. This time they will be delegated the grid jobs for execution. If needed a job optimizer of the central services will split a job into smaller parts before execution. This will help matching more grid jobs with sites if the grid jobs require many resources.

When a JobAgent matches with a site job it is not the responsibility of the JobAgent to execute it. It will instead launch a JobWrapper**.** The JobWrapper will then set up the environment for the job. This includes downloading input files. The JobWrapper will then execute the grid job. When the grid job completes the output file is passed to the JobAgent that uploads the output files to the correct location.

Chapter 4

Development

This chapter will give a more detailed description of the functional requirements for the JAliEn batch queue interface. Discussions of the architecture of the solution will follow. Subsequently, a look at the implementation for the JAliEn interface of the job schedulers discussed in chapter 2 will be done. Finally the environment used for developing and testing will be discussed.

4.1 Functional Requirements

The other components of JAliEn enforces some requirements on the batch queue interface. The CE is the part having direct communication with the batch queue interface, and the batch queue interface needs to respond accordingly. There are four times these two components interact:

- CE initializes a batch queue interface corresponding to the sites job scheduler and passes the site configuration and logger[8]. The configuration is the information about the site collected from the LDAP server.
- CE requests the number of queued jobs on the site's batch queue.
- CE requests the number of active jobs on the site's batch queue.
- CE submits the JobAgent startup script to be submitted to the site's batch queue.

The LDAP server contains information about the local job scheduler for the site. This includes the type of job scheduler. The CE uses this to initialize the correct batch queue interface object, but it also contains information valuable for the batch queue interface. Commands or arguments for submitting or checking status of the job queue might be defined in the LDAP server. Other information important to a specific job scheduler will be discussed later. The CE passes all this information to the constructor for the batch queue interface. This information, also referred to as the site configuration, must be handled correctly by the batch queue interface.

[8] Class used for logging output and errors in JAliEn.

In addition to these requirements, set by other parts of JAliEn, the solution for the batch queue interface must satisfy the site configurations used in AliEn. To achieve this goal a comparison between the implementation of JAliEn and AliEn will be made.

4.1.1 Kill method

The TORQUE and Slurm batch queue interface in AliEn have a method called *kill.* This method is used to remove a job from the job queue on the site. The HTCondor batch queue interface does not have this method. The way this method works is that a status call is made to the job scheduler, then the first job on this list is removed. The use of this method in AliEn is not completely clear. This method can be called to remove all jobs on the job queue, but there are other ways to achieve this goal.

The CE in JAliEn has no support for this kill method. An implementation of this method in the batch queue interface would therefore not be used. Since this method has no correlation with old site parameters, and is not used by JAliEn currently, it is not implemented.

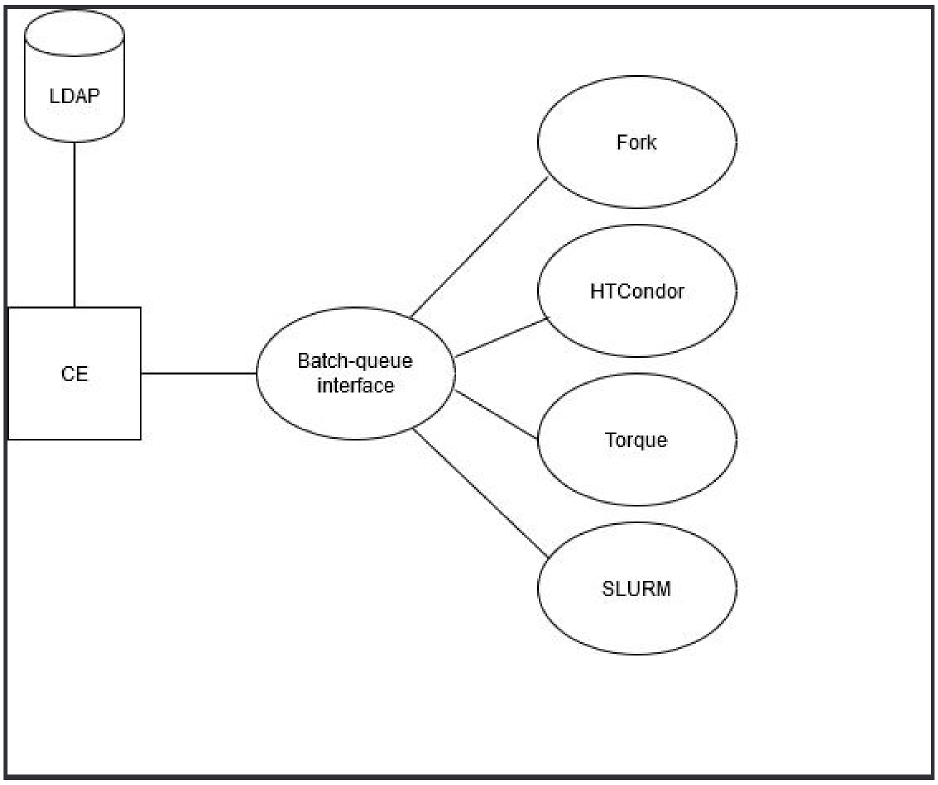


Figure 4.1: *Architecture for the JAliEn batch queue interface solution presented.*

4.2 Batch queue interface

Looking at the requirements described for the JAliEn batch queue interface some methods that are shared by all the interfaces for job schedulers appears. These are *numberActive*, *numberQueued* and *submit*. There will be a different implementation for these methods for each unique job scheduler. The figure 4.1 shows how the architecture looks. The batch queue interface does not have any implementation for the methods, it only describes them.

While implementing the different job scheduler interfaces, duplicate or similar code appeared. As discussed earlier in chapter 1, duplicated code is something that should be avoided. A common approach to solve this issue is to use a utility class for the shared code. While this is a perfectly valid solution it is not the one used. Since there already exists a batch queue interface class that serves as an interface for the different job scheduler interfaces it is possible to use this for the shared code. This way there is no need for even more classes to make it more complex. This also works because there is not too much shared code. If there was a need for many methods this approach would not be optimal since it would clutter the batch queue interface. The batch queue interface's main function is to serve as an interface for the job scheduler interfaces, not a utility class. The shared code revolves around two aspects. Communication between JAliEn job scheduler interface and local job scheduler is the first aspect. The second aspect is how the interface acquires information from the configuration.

4.2.2 ProcessBuilder

Communication with the local job scheduler at the site is crucial. As explained earlier in chapter 2, job schedulers have a set of commands to make use of their functionality. The JAliEn job scheduler interface needs a way to issue these commands to the local job scheduler. In the java.lang package we find the *ProcessBuilder*[17] and *Process*[18] classes. *ProcessBuilder* is used to create operating system processes in java. When using this class to create a process a native process will be created, and then a Process object will be created interfacing with the native process. Using *ProcessBuilder* a command to the local job scheduler can be issued. The command issued can be to submit or check the status of the queue. *ProcessBuilder* gives the option of setting the environment the process will run in. Adding to the environment of the process is not of interest for the JAliEN batch queue interface solution for either TORQUE and SLURM. HTCondor on the other hand needs this to make use of the job routing and grid resources. Using other features that JAliEn provides, it is possible to add more functionalities to the processes. *ExitStatus* is a class that already exists in JAliEn, and is a way to get the exit status of the process. This class is also used to get the standard error and standard output of the process. This was already possible using only the *Process* class, but the reason *ExitStatus* is used is to make use of *ProcessWithTimeout*. This is a class implemented in JAliEn. It adds the functionality of setting a timeout to a process. This ensures a process will be killed after a set amount of time. Without a timeout a process that has frozen might occupy resources for a long time without manual interference.

4.2.3 Information from LDAP

Getting the site configuration information that the CE passes to the batch queue interface is not as simple as in AliEn. Java is a strictly typed language and the type of a variable must be assigned. The site configuration is described as an *HashMap of Objects* in JAliEn. An attribute in the LDAP can be described using only a String, but also using a TreeSet of Strings. This means that an attribute that has the possibility of being both types must be handled in Java. AliEn did not have this issue using Perl. Perl makes it possible to send both types to a list, and then iterate through it. Methods to solve this issue in JAliEn are implemented in the batch queue interface.

Describing the commands for interacting with the job scheduler in the LDAP does not create an issue. The commands should not be possible to split up into several Strings in the LDAP. This means that collecting the commands from the site configuration should always be done using a String. It is not a requirement for the site configuration to declare a command to use on the site's local job scheduler. There is however, a need to declare a default command in the job scheduler interface for JAliEn if the command does not exist in the site configuration. The default command will always be unique to a job scheduler, and is therefore done in the constructor for the corresponding job scheduler.

In addition to storing job scheduler commands in the LDAP, optional arguments for the commands can also be stored there. A command can have several accompanying arguments. The sites in the grid store multiple command arguments in two different ways. Some store all arguments space separated in a String, and others store the different arguments as a TreeSet of Strings. The possibility of different types describing the command arguments must be handled or the batch queue interface for JAliEN will crash. It is logical to have the method for collecting command arguments from the site configuration in the batch queue interface instead of the specific job scheduler interface. Default values for arguments is an empty string, and the method does not need to be aware of the type of job scheduler the arguments is related to. When collecting the command arguments there is only one value that needs special care. This is a value used by TORQUE to describe if the site does not want to make use of file staging. If this value exists the JAliEn job scheduler interface should be made aware that the site does not want file staging, then this command argument value should be ignored. This is actually not a command argument, but is used as information to the job scheduler interface. TORQUE is the only job scheduler discussed in this thesis that makes use of this command argument value, but the check will happen for all job schedulers. The reason why this is checked on other sites is because some sites running other batch queues than TORQUE, also have been adding this value to the site information in LDAP. This happens even though it has no relevance for the site’s local job scheduler. This is in all probability occurring because some sites configurations were copied to a new site running a different job scheduler.

4.3 Job description

There are some attributes for the job shared by job schedulers. These attributes will be declared in the job description when possible, or the default value will be used if this value is right. A discussion will be made about what these attributes are and why the value was chosen for an attribute.

There is no need for several instances of the same job when submitting a job to the job scheduler. Only generic JobAgents are submitted. In addition, the CE is handling how many JobAgents should be submitted to the job queue. It is not necessary to make use of a job scheduler's ability to requeue a job. The CE is continually running, trying to add more JobAgents to the local job queue. If one fails the CE will notice there is a slot available and try to submit another.

Inputs and outputs are unecessary when submitting the JobAgent. Inputs are handled later when the JobWrapper executes the grid job. Outputs from the grid are handled by both the JobWrapper and JobAgents, ensuring this arrives at the right place. Outputs from submitting the generic JobAgent can therefore be discarded. This is the same for the error output. HTCondor is an exception. If HTCondor is making use of another grid resource or routing the job, output and error logs are more valuable. They can be used for debugging. All jobs are given a name when submitted to the job scheduler. This has no impact on JAliEn, but does make the function of the job clear if the status of the job queue is checked manually.

4.4 TORQUE

The TORQUE interface in JAliEn is pretty standard in handling commands and arguments with the exception being the submit arguments having the possibility of containing a submit argument attribute with the value "*alien_not_stage_files*" in the LDAP. If this value is set, files will not be staged when running the submit command and adding a job to the queue.

4.4.1 Submitting jobs

```
#PBS -V JobAgent22
#PBS -o /dev/null
#PBS -e /dev/null
#PBS -V
#PBS -W stagein=/tmp/jobagent.startup@jalien.hvl.no:jobagent.startup
jobagent.startup
```

Table 4.1: *TORQUE job description.*

To submit a job to the TORQUE job scheduler a job description is required. The submit method builds this job description. Table 4.1 gives an example of a job description used in the JAliEn TORQUE interface. As discussed earlier output and error for the job is not needed. They will be discarded by sending both to /dev/null. This is a special file in a Unix system that discards anything written to it. "#PBS -V" declares that all environment variables that exist in the environment that issued the submit command to be exported to the job. TORQUE has the option of staging files. Staging files is the default in the TORQUE interface for JAliEn. To force the site not to use file staging, the submit argument must contain a value *alien_not_stage_files* in the site configuration. If the site wants to stage files, the site's host jobagent startup script must be staged in. The last thing needed in the job description is the executable, the JobAgent startup script.

When submitting a job to the TORQUE job scheduler with the qsub command it is often done with the job description as an executable file. The path to this file is added as an argument in the qsub command. TORQUE gives us the option of having the qsub command read the job description from the standard input. This gives the possibility of not creating a temporary file. To achieve this the job description, created in the TORQUE interface, will be passed to the standard input of the qsub using what is called a here string[9].

4.4.2 Status of the queue

Getting the status of the queue using qstat, lacks the filtering of jobs that the TORQUE JAliEn interface wants to only get relevant results. To get the number of either queued or active jobs in the queue, a way to match with job states from the status results is needed. To achieve this the output received when querying TORQUE for the status of the job queue is processed. Each line corresponding to a job will be trimmed and split to populate an array of strings.. Each job will then be iterated through, matching jobs that correspond to the right status. When a job is successfully matched the counter increases. When all the output has been iterated through this counter is then returned.

The job states corresponding to both the number queued and number active jobs must be selected to get an accurate number. For the number of queued jobs there is the job state **Q** for queued. To get the number of active jobs, the job state **R,** for running, is the optimal choice. There are other states that could possibly be part of either of the counts. There is **T** for a job that is being moved to a different location, and **W** for a job waiting to be executed. Both states will most likely never occur and are describing a job in transition. Then there is the job state **H** for held jobs. This is yet another job state that most likely will not occur. Since the other states should not happen with a job running a JobAgent, only the states **Q** and **R** will be used.

4.5 SLURM

SLURM is not much different than other job schedulers when looking at how to handle the information received from the LDAP server. When compared to TORQUE the only difference is the command names, and there is no option of staging files. The approach to retrieving the commands for submitting and getting the status of the queue is the same as TORQUE. The general approach, described in chapter 4.1.3, to get commands and arguments is enough.

[9] https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#Here-Strings

4.5.1 Submitting jobs

```
#SBATCH -J JobAgent22
#SBATCH -o /dev/null
#SBATCH -e /dev/null
#SBATCH - N 1
#SBATCH - n 1
#SBATCH --no-requeue
srun jobagent.startup
```

Table 4.2*: SLURM job description.*

There are two different ways to add the job to the Slurm job scheduler as was previously discussed in chapter 2.9. The Slurm interface in JAliEn submits using Sbatch, and executes a job step, which is the JobAgent, using Srun.

Much like TORQUE the output and error of the job is of no relevance. Therefore the special file /dev/null is used to discard both. It is also specified that only one node is needed and only one instance of the job. This is to help the job scheduler with allocation. The no requeue parameter has already been discussed in chapter 4.2.

When submitting the job to the queue there is the option of giving the job description through standard input. This will remove the need to create a temporary file for the job description. Unfortunately feeding the job description for the submit command through standard input creates some problems. Not all the arguments in the job description are read and executed, even though the job itself runs without a problem. Therefore this is not a valid option and creating a temporary file for the job description is needed.

4.5.2 Status of the queue

Finding the number of active or queued jobs does not require the SLURM JAliEn interface to iterate through the status of the queue. Instead SLURM gives the user the option of filtering the status of the queue, using the state of the job. Using squeue with the option *"-t"* and the state we are looking for in addition to *"-h"* to remove the header, the result will be an

ArrayList with only relevant rjobs. The length of the ArrayList will be the number of either active or queued jobs in the batch queue.

SLURM have different job state codes, and it is important to choose the correct states to filter for to get a good result. For queued jobs Pending (PD) is the main one. This state says the job is waiting for resource allocation, meaning it is queued. The state code Configuring (CF) also needs to be part of the filter for queued jobs. Jobs with this state code have been allocated resources, but are waiting for them to be available for use.

To find all the active jobs the filter needs the job state code Running (R). Jobs having Running as a state code have a resource allocation and are currently active. Another state code is Completing (CG). The job is in the process of completing, but some parts may still be active and therefore take up one active slot until it changes to Completed (CD). The last state code that is in the active filter is Suspended (S). This job state code should not occur, but it is added to match the AliEn Slurm interface.

4.6 HTCondor

HTCondor shares a lot of similarities with the two job schedulers mentioned earlier, but is still a little different. Unlike SLURM and TORQUE, HTCondor gives the option of using grid resources to delegate the jobs to a different job scheduler. This makes the submit method quite different.

4.6.1 Submitting jobs

```
Cmd = jobagent.startup
Universe = vanilla
Output = /log/jobagent.out
Error = /log/jobagent.error
Log = /log/jobagent.log
+WantJobRouter = True
```

Table 4.3: *HTCondor job description for a job with job routing*

When submitting a job with the JAliEn HTCondor interface there are two options. The first option is to run the job on the HTCondor running on the site. The second option is to delegate the job to another job scheduler. This is either described in the environment parameter in the site configuration, or as an environment variable. Only the use of site configuration will be discussed going forward in this chapter. If the site configuration describes that the site will be delegating jobs, a grid resource must be described. It is important to note that in addition to delegating jobs it is also possible to use job routing. This can be valuable if the number of jobs submitted exceeds the capacity and there is a need to route the jobs to another job scheduler. This will not happen when delegating jobs to other job schedulers.

If the parameter named *GRID_RESOURCE* exists in the site configuration*,* HTCondor will delegate the job to the grid resource defined by this parameter. This will be described in the job description by setting the universe of the job to *grid* and defining the grid resources in the job description.

The other option is job routing. The local HTCondor job scheduler on the site is used, but if needed the job description can be updated and the job routed to another job scheduler for execution. If there is a parameter called *USE_JOB_ROUTER* in the site configuration job routing is available. This will be described in the job description by setting the universe to vanilla and setting the value of the attribute *Want job router* to true. It is important to note that a site can not make use of both grid resource and job routing simultaneously in the HTCondor job scheduler interface for JAliEn. This would force the job to fail. It is possible to submit a plain vanilla universe job description without job routing to HTCondor, but this is not implemented in the HTCondor job scheduler interface for AliEn. To ensure the same functionalities in the HTCondor job scheduler interface for JAliEn, this was not implemented.

Both these approaches might delegate jobs to a different job scheduler. Security and trust is therefore important. The other job scheduler must be able to recognize and trust the jobs it receives. To do this proxy certificates are used. An environmental variable called *X.509_USER_PROXY* will contain the proxy certificate needed. If this variable is not set or empty, it will be created. When the certification is fixed it will be sent with the job to the job scheduler. The other job scheduler will check if the certificate is valid. If it is valid the job will be executed. This is done by having this variable be a part of the environment the process created by ProcessBuilder is running in. Since the component that creates the process to execute the submit commands is in the batch queue interface, it must still follow a general solution for all job schedulers. The X.509 proxy variable does not exist for either SLURM or

TORQUE and trying to add a variable that is null creates a null pointer exception. The solution is to check if this variable exists before adding it to the process environment.

4.6.2 Status of the queue

To get the number of queued and the number of active jobs, the condor_status command is used. It is possible to filter for the states corresponding to active and queued.

- 0 - Unexpanded
- 1 - Idle
- 2 - Running
- 3 - Removed
- 4 - Completed
- 5 - Held
- 6 - Submission_err

For the number of queued jobs the state that is used is the **Idle** state. This means the job is waiting to be executed, meaning it is in the queue. For active jobs, **Running** is the state to filter for. There is also the option of having held jobs as a part of the active or queued filter. Jobs in the held state will start over from scratch if it is run in any other universe than the standard universe. This is the case for the JAliEn HTCondor interface. This makes it hard to actually put it in either an active or queued list. Since it can have started running then became held it could be a part of the active filter. It is starting from the beginning whether or not it was running before it got held, and it can become held from the idle state. Since this job state could potentially be used for both counts, having it in none of the filters is the approach taken. A reason for this is also that a JobAgent job should not end up in this state without manual interference.

The results condor_status produces is an ArrayList. This result also contains a header. Ignoring the header, the size of the ArrayList will be the number of jobs matching the filter.

4.7 Fork

The Fork interface in JAliEn works differently from the job scheduler interfaces previously discussed in this chapter. The Fork interface is not used by sites to run jobs, but instead used in development related tasks. The Fork interface is running on local user machines, when a user wants to test something.

In Unix systems fork is a way to create new processes. The caller process will be duplicated creating a new process. The caller process is called a parent process and the new process is called a child process. This is used when a job is submitted to the Fork interface. A new process will run the job, in this case a JobAgent. This process will be running until the TTL runs out or an error occurs, stopping the JobAgent process.

In the JaliEn Fork interface only the submit method is of any importance. Given that this is not used as a job scheduler there is no queue, or a list of active jobs at all. This means the two methods for getting the number of active and the number of queued jobs will always be returning zero. This can make a single computer take on many jobs since each time the CE will ask for how many jobs are active or queued it will always get zero and then think it is safe to submit more jobs. A grid job must still match with the sitemap for the machine. This is a valid concern. A machine that keeps on running a CE might therefore end up with a large amount of JobAgents running.

The need to submit a job is what is left, and the only important part of the Fork interface. It is important to note that there is no need to have the configuration of the site from the LDAP. There are no different commands for creating another process for the JobAgent. Instead of a command to submit the job to a scheduler, the process, created by Processbuilder, will be running the JobAgent startup script directly. Everything put together makes the actual fork interface very simple. Most of it is either handled in the batch queue interface or returns a static value. It is still an important tool for development

4.8 Environment

The environments used for development, and later testing, is not the production environment. It is made to mirror the production environment, but there are some differences. The ability to manipulate information about sites in the LDAP is required to be able to test the different job schedulers. In the production environment the LDAP used by JAliEn is located on CERN servers and shared by all sites in JAliEn. Only a select people, admins, are able to change the information on this LDAP server. To combat this issue the development environment must use a local LDAP server. This requires the development environment to use a local JCentral as well. Without a local JCentral the local LDAP server would never have been used. The local JCentral opens up for the use of local databases for the File Catalogue and Task Queue. The database used in the development environment is different from the production environment. Scalability is not an issue since the development environment does not have many sites. Instead of Cassandra the development environment uses MariaDB[10].

It is important that the JCentral used in the development environment is running on a different machine, or a different user on the same machine, from the one running the JAliEn site and jShell. Running both on the same user on the same machine creates issues with certificates. A user would automatically become an admin without a certificate. The JAliEn instance having direct access to the database can only be accessed by an admin. The JAliEn running on the site will then later try to use the certificate off the user, but this does not exist. This will create a null pointer exception, forcing JAliEn to crash.

Running a local JCentral creates some issues since the development of JCentral is not complete. Some small changes must be made manually to be able to make the CE work. A queue on a site is default set to locked. This makes the site unavailable to run grid jobs for JAliEn. An administrator for a site should be able to change the queue status to open it up to run grid jobs, but this is not implemented yet. When the CE checks for maximum available slots on the job queue for the site, it checks the database table *SITEQUEUE.* Maximum available slots for the site should be added to this database automatically from the LDAP site configuration, but this is not implemented either. Both of these issues can be fixed by manually manipulating the data stored in the database. This is possible since a local database is being used.

[10] https://mariadb.org/

Chapter 5

Testing

To ensure the solution for the JAliEn batch queue interface satisfies the requirement outlined in chapter 1, testing is necessary. To ensure a viable solution the requirements are divided into functionality and robustness. The approach to testing them is different. Creating tests to make sure the solution satisfies the functional requirements will be done using unit testing. Testing for robustness needs a different approach. Only using unit testing will not give the test coverage needed to ensure the requirements are satisfied. In this chapter both unit testing and the approach to testing robustness will be discussed and the result of the tests will be presented.

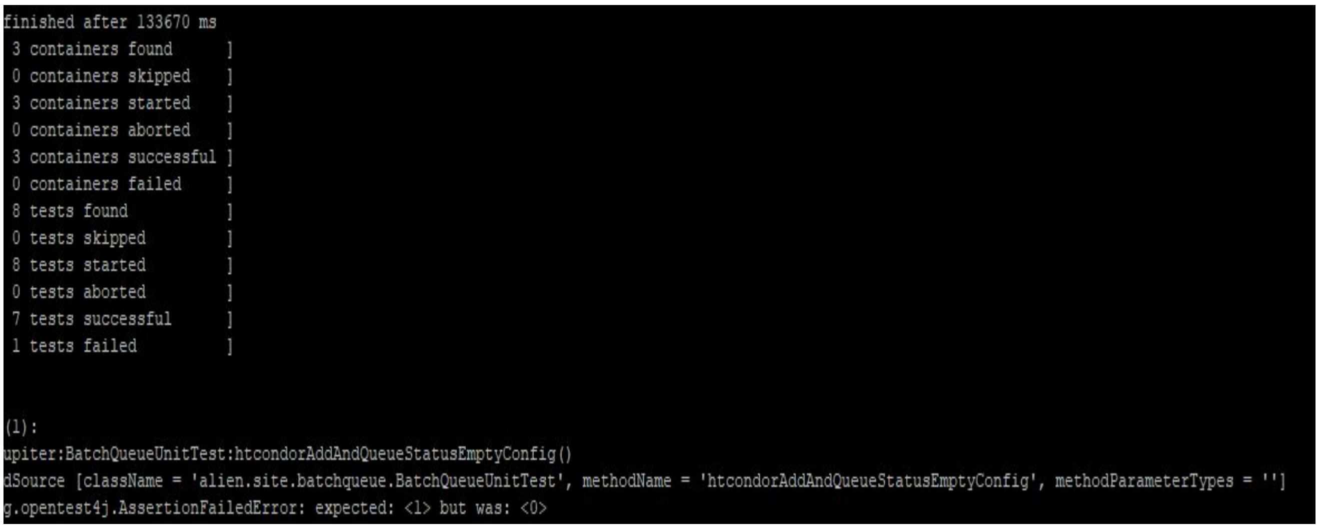
5.1 Unit testing

Unit testing is a way to separate the part you want to test from the rest of the software and ensure it behaves as expected. Errors unrelated to the component that is currently being tested might occur. Separating the component from the system and testing it makes sure this component is working as planned. When under development constant changes happen and being confident that the component is working is valuable. Unit testing also helps when making changes to the component since it needs to pass the same tests. This ensures the core functionalities still remain.

With unit testing it is possible to test if the functional requirements for the JAliEn batch queue interface is fulfilled. There are different options for doing unit testing. Since JAliEn is written in Java JUnit is the framework that is used for unit testing in this thesis.

5.1.1 Unit test cases

When creating unit tests it is important to realize what different cases should be tested to ensure good code coverage. This will help ensure the methods developed are behaving as planned. The first tests to cover are basic testing of the methods returning the number of queue and active methods with both an empty queue and an empty configuration. It also includes tests for the submit method with an empty configuration. After submitting there is a wait of five seconds to ensure the job reaches the queued state, before testing if the job was added to the queue using the method returning the number of active jobs. This wait time is exclusive to the unit tests, since jobs added to a job scheduler are not put in an active state immediately. The job submitted in all cases is a simple script putting the process to sleep for thirty seconds. This is because what kind of job being submitted is not crucial for the functionalities of the JAliEn batch queue interface. It is important to note that when using fork there will never be an increase in the number of queued and the assertions must therefore involve whether or not an error was thrown when trying to run the job.



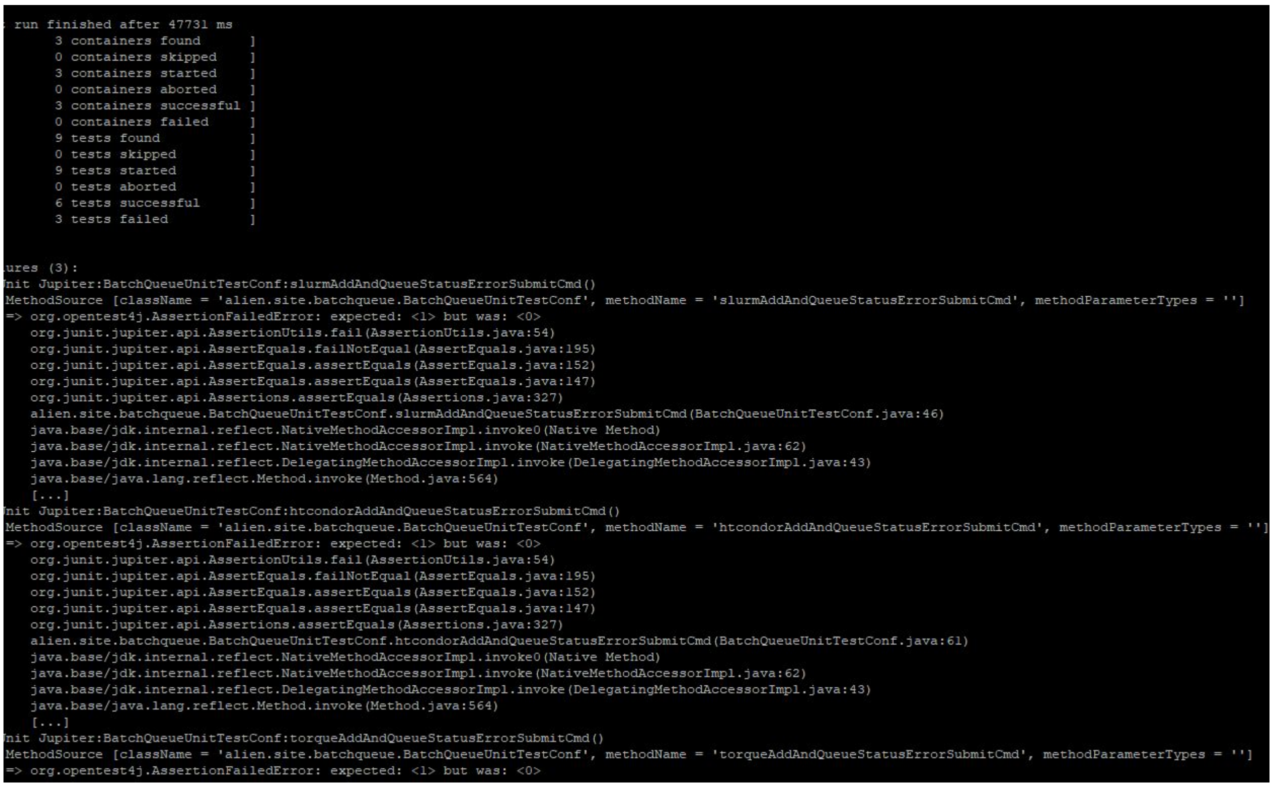
```
finished after 133670 ms
 3 containers found      ]
 0 containers skipped    ]
 3 containers started    ]
 0 containers aborted    ]
 3 containers successful ]
 0 containers failed     ]
 8 tests found           ]
 0 tests skipped         ]
 8 tests started         ]
 0 tests aborted         ]
 7 tests successful      ]
 1 tests failed          ]


(1):
upiter:BatchQueueUnitTest:htcondorAddAndQueueStatusEmptyConfig()
dSource [className = 'alien.site.batchqueue.BatchQueueUnitTest', methodName = 'htcondorAddAndQueueStatusEmptyConfig', methodParameterTypes = '']
g.opentest4j.AssertionFailedError: expected: <1> but was: <0>
```

Figure 5.1: *Unit testing with empty configuration*

As the figure 5.1 shows, all but one test succeeded. There is no surprise that submitting a job to HTCondor fails with an empty configuration. The configuration needs to include either a job routing or a grid resource value. Without either it will default to trying to use the grid universe. Trying to use a grid resource without having one declared will create an error when submitting the job description to the job queue, resulting in the job not being submitted to the job queue. The rest of the tests give safety to the functionalities under basic conditions.

Collecting the parameters from the site configuration is important, and as discussed earlier since java is strictly typed strings needs to be handled differently than lists. The tests shown in the picture below are testing different possible site configurations from the LDAP. This includes how the submit argument *"alien_no_stage_files"* is handled. The attribute used for HTCondor is a job routing. Testing done on the grid universe using grid resources with HTCondor will not be done. This is because of limitations regarding time and the development environment. Fork will not have any test associated with site configuration since that is irrelevant for the Fork interface in JAliEn.

.



```
: run finished after 47731 ms
      3 containers found      ]
      0 containers skipped    ]
      3 containers started    ]
      0 containers aborted    ]
      3 containers successful ]
      0 containers failed     ]
      9 tests found           ]
      0 tests skipped         ]
      9 tests started         ]
      0 tests aborted         ]
      6 tests successful      ]
      3 tests failed          ]


ures (3):
nit Jupiter:BatchQueueUnitTestConf:slurmAddAndQueueStatusErrorSubmitCmd()
MethodSource [className = 'alien.site.batchqueue.BatchQueueUnitTestConf', methodName = 'slurmAddAndQueueStatusErrorSubmitCmd', methodParameterTypes = '']
=> org.opentest4j.AssertionFailedError: expected: <1> but was: <0>
   org.junit.jupiter.api.AssertionUtils.fail(AssertionUtils.java:54)
   org.junit.jupiter.api.AssertEquals.failNotEqual(AssertEquals.java:195)
   org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:152)
   org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:147)
   org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:327)
   alien.site.batchqueue.BatchQueueUnitTestConf.slurmAddAndQueueStatusErrorSubmitCmd(BatchQueueUnitTestConf.java:46)
   java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
   java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
   java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
   java.base/java.lang.reflect.Method.invoke(Method.java:564)
   [...]
nit Jupiter:BatchQueueUnitTestConf:htcondorAddAndQueueStatusErrorSubmitCmd()
MethodSource [className = 'alien.site.batchqueue.BatchQueueUnitTestConf', methodName = 'htcondorAddAndQueueStatusErrorSubmitCmd', methodParameterTypes = '']
=> org.opentest4j.AssertionFailedError: expected: <1> but was: <0>
   org.junit.jupiter.api.AssertionUtils.fail(AssertionUtils.java:54)
   org.junit.jupiter.api.AssertEquals.failNotEqual(AssertEquals.java:195)
   org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:152)
   org.junit.jupiter.api.AssertEquals.assertEquals(AssertEquals.java:147)
   org.junit.jupiter.api.Assertions.assertEquals(Assertions.java:327)
   alien.site.batchqueue.BatchQueueUnitTestConf.htcondorAddAndQueueStatusErrorSubmitCmd(BatchQueueUnitTestConf.java:61)
   java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
   java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
   java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
   java.base/java.lang.reflect.Method.invoke(Method.java:564)
   [...]
nit Jupiter:BatchQueueUnitTestConf:torqueAddAndQueueStatusErrorSubmitCmd()
MethodSource [className = 'alien.site.batchqueue.BatchQueueUnitTestConf', methodName = 'torqueAddAndQueueStatusErrorSubmitCmd', methodParameterTypes = '']
=> org.opentest4j.AssertionFailedError: expected: <1> but was: <0>
```

Figure 5.2: *Unit test with site configurations*

As figure 5.2 shows, the job schedule interface in JAliEn gets the submit command from the site configuration and tries to use this command to submit jobs to the job scheduler. In these test cases an incorrect command was given to showcase that the unit test will fail, since the default command will not be used. This is what corresponds to all the failed tests in the figure 5.2. Rest of the test succeeded ensuring that information collected from configuration is done correctly, even when different types might occur. HTCondor also correctly submitted the job when the site configuration contains the value for job routing.

5.2 JAliEn under development

It is important to note that JAliEn is under development and is not currently a finished product. Since sites are planned to be distributed to sites in the summer of 2020, components running on sites are more complete than components that are part of the central services. Grid jobs submitted by users were stuck in the Task Queue, since the part of JAliEn responsible for moving the grid jobs to the database table used when matching JobAgents and grid jobs, is not yet implemented. A script running at the same time as JAliEn (created by Maksim Melnik Storetvedt) solved this issue. The script is a temporary fix that might not match the final solution. This makes testing only using the CE the logical choice.

Testing how the solution functions in a real scenario would provide valuable knowledge. JAliEn is a complex software and injecting faults to other parts of JAliEn to see how it affects the batch queue interface solution would be valuable. Both are not feasible given the current state of JAliEn. This should not be a huge issue. The batch queue interface is only communicating with the CE, which is more complete. This makes it possible to test extensively regardless. While there might be faults that carry over from other parts of JAliEn, injecting faults in the CE where it realistically will happen, will provide good coverage.

5.3 Robustness

When testing for functionalities a set of expected parameters were used. This will not always be the case. Using only “happy-path” testing, the solution might be vulnerable to parameters outside of the perfect set[19]. Examples of this is an unexpected input value. Robustness is defined as:

“the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions[20].”

Another definition is the ability of the software to react accordingly to unexpected and unforeseen circumstances[21]. If these circumstances create an issue within JAliEn the presented solution in this thesis would not be ready for deployment. It is therefore important to understand how the presented solution reacts to these circumstances. In addition, testing for robustness can help improve the quality of the code.

To ensure the robustness of the solution presented in this thesis some guidelines must be set. Some requirements for robustness were formulated in chapter 1.3.2, but this needs to be described more in detail. In addition, methods to test for robustness must be formulated. A way to measure the robustness and evaluate the results must also be provided.

5.3.1 Robustness requirements

This part will aim to describe more detailed what is an acceptable response from the current implementation. The solution should have graceful degradation even if a critical fault occurs.

This means that some services should still be provided instead of completely failing[22]. To specify even more, if there is an issue with the batch queue interface in JAliEn, this should not affect the rest of JAliEn outside the local grid site. The CE is directly connected to the batch queue interface in JAliEn and it is therefore acceptable that it is affected. The main function for the CE is submitting jobs to the job scheduler, if this fails it will be unable to perform this task. This is the most important requirement for robustness, the faults should be contained to the CE and JAliEn batch queue interface.

If the batch queue interface in JAliEn fails, this should be properly communicated. Error messages help developers pinpoint where and why it occurred. This is valuable for debugging and fixing faults that should not happen. Some circumstances should be handled instead of producing a fault. This is the case for sites that have the attribute indicating that files should not be staged with a local job scheduler that does not support staging files. This is a real situation as some sites using SLURM does have this attribute in the submit arguments file. Instead of the job scheduler interface in JAliEn producing a fault when submitting this can be handled, and the attribute ignored.

One of the balances that needs to be maintained is that all exceptions should be handled, but not every single exception handled individually. Handling too many exceptions takes up more resources. Often catching several exceptions and handling them the same way is the correct way to go.

There are three main vulnerabilities for the implementation presented.

- **Invalid inputs:** The JAliEn batch queue interface gets input for a wide array of functions. Commands from the LDAP server for the local job scheduler is one example..
- **Peak load:** A solution can not expect a steady amount of activity. Cases of extreme load on the system is possible, and should be considered.
- **Faults with the local job scheduler:** Faults can occur with the local job scheduler. How the JAliEn batch queue interface behaves in this circumstance is relevant.

With some requirements outlined a method to test for robustness can be formulated.

5.3.2 Test method

The most common way to test for robustness is by using fault injection. Fault injection is to inject the faults into the software and observe how the software handles it. The standard approach to fault injection is generating a set of invalid inputs and injecting them to the system.

While there is software for automatic tests of robustness available, this will not be used for robustness testing in this thesis. The robustness tests will be done manually. Manual testing does not scale well, and if the whole of JAliEn was to be tested, doing it manually would not be feasible. Since only a part of JAliEn, namely the batch queue interface, will be tested this is not a problem to do manually. Manual testing makes it easier to tailor the robustness tests for software that will be tested. It is also easier to understand the results of a test, and where the fault occured.

In practice not all of the fault that is being tested for will occur, and if it does occur, it might not be as severe. It is still important to have an extensive robustness test. The software can not be expected to run without encountering faults of some kind. The tests will give an indicator of how the software responds to faults that might occur. This includes faults that was not tested for.

5.3.3 Penetration testing

Penetration testing is a part of robustness testing. It is possible for inputs to be malicious. Penetration testing can be done by testing known malicious inputs and see how the system reacts. It is important to keep the system safe, since hackers will be looking for weak points to exploit. It is therefore important to be aware of where such malicious inputs can occur. There are two points relevant to job scheduling where user inputs might cause harm. The first being user submitted jobs. A user will submit a JDL file describing the job to the grid. This JDL will be parsed to make sure it is correct. Any wrong or malicious code will be found out while parsing this JDL file, but the JDL file contains an executable. This executable does not go through a parser like the JDL file, making this a potential security issue. Faults in the executable are handled, but not malicious input specifically. If someone were to inject malicious code into the executable it would affect the site running the job, but not JAliEn as a whole. Checking for malicious input in the grid job is impossible to do in the JAliEn batch queue interface. The executable is run after the JobAgent matches with the job in the

TaskQueue. Submitting a job to the batch queue happens earlier, when a generic JobAgent is submitted to the queue.

The other way to inject malicious input is by having the input in configuration of a site in LDAP. Some attributes like submit arguments will only be added to the submit command making the command fail without actually doing any harm. The command for submit, status or kill on the other hand will be able to run directly in the bash shell. This again is a security issue. Difference from the other security issue is that this is possible to take into account when implementing the JAliEN batch queue interface. This does not mean it is logical to test every input. The only one that is capable of adding attributes and configurations to the LDAP are administrators. This is such a small group that it is very unlikely that the LDAP would contain malicious code. Taking this into account, choosing to not to test site configuration received from LDAP is a valid approach. This is the approach that is used in the implementation of the JAliEn batch queue interface described in chapter 4. Being aware that this is a potential weakness is valuable.

5.3.4 Robustness benchmark

When testing there needs to be a way to evaluate the results of the tests. This is not always as easy to do when testing for robustness as it might be when doing unit tests. To do this the most common robustness benchmark will be used. This benchmark is called CRASH. CRASH describes how the fault affects the system. There are five different groups used to describe the faults. Each group corresponds to a letter in CRASH.

- C Catastrophic (OS crashes/multiple tasks affected)
- R Restart (task/process hangs, requiring restart)
- A Abort (task/process aborts, e.g. segmentation violation)
- S Silent (no error code returned when one should be)
- H Hindering (incorrect error code returned)

CRASH is used to give an indication of the severity of the errors that might occur. It is a rough benchmark and is only giving an indication. Errors must be evaluated individually as CRASH does not ensure that an error is severe or not. This changes by system and error produced. It is also important to note that CRASH does not tell anything about why an error occurs, only what type of error. Every test must be evaluated based on why it happens and what the impact on the system is. CRASH will be used to give an overview over all the errors, indicating if there are severe errors that repeatedly occur.

5.3.5 Planned tests

Requirements for robustness are already described, and potential vulnerabilities have been identified. To create good tests of robustness, the testing must be tailored to the software. Planned tests must be defined to ensure the software satisfies the requirements defined. Table 5.1 describes the collection of tests that will be done to help ensure robustness.

Wrong input from LDAP
Wrong input type from LDAP
Wrong path for JobAgent script
Fault within the JobAgent script
Status when job queue has unrelated jobs running
Faults with status results
Faults with job scheduler
Peak loads

Table 5.1: *Planned tests for robustness*

5.3.6 Expected results

Expectations are that most of the tests will create problems submitting jobs to the job scheduler. The CE and JAliEn batch queue interface on the site might crash, but will not have further impact on JAliEn. It is expected to communicate when errors occur, and not crash without giving valuable information about the crash to the developer. Peak load is expected to not create any big issues. If it fails to submit a job, the CE will try again. If the load forces a crash with the JAliEn batch queue interface, this is expected to be contained to the site.

5.3.7 Test results

Wrong input from LDAP: Having input that is not valid in the LDAP server can occur. Mistakes can be made when adding information, or the information might be wrong. Changes can have been made to the site, making the information outdated. As expected this stops the JAliEn batch queue interface from interacting with the job scheduler. It does not create any issue beyond that. The Processbuilder class is aware that it failed running the command and will throw an exception. It is unlikely, but if it freezes the ProcessBuilder when trying to run the command there is a stopper. ProcessBuilder has been assigned a timeout that makes it exit if reached. The CE will keep on running while not functioning. If it fails getting the state of the job queue it will not try to submit any JobAgents. If that does not fail and the job scheduler has available slots it will try to submit a JobAgent. This will fail, and the CE starts over from the beginning after a delay. While the CE keeps on running, the site is not usable for running grid jobs on.

Wrong input type from LDAP: This is an issue because Java is a strictly typed language. The two main types in the LDAP are either a TreeSet of Strings or just a String. They both need to be handled differently. A different type than expected showing up in the configuration most likely will happen in arguments for a job scheduler command. There can be several arguments for one command. The most efficient way is still having all the arguments as one string, but not all the sites have done this. There is also the possibility for the value indicating if a file should be staged, discussed in chapter 4. This value is set for some sites that do not run a job scheduler capable of staging files. Testing was also done on commands found in the LDAP. Commands should never be a TreeSet given that a command is only “one of”, but still tested as it can happen. No exceptions occur when testing with only arguments. This case is already handled, and it still works. Fetching job scheduler commands from the site configuration described in the LDAP server, with TreeSet as a type will throw an exception. This exception happens in the constructor of the JAliEn batch queue interface. This will create a null pointer exception in the CE. This causes the CE to crash, requiring a manual reboot.

Wrong path for JobAgent startup script: It is unlikely that the CE will provide a wrong path for the JobAgent startup script without encountering an error in the CE. Still tested to ensure that changes to the sites filesystem while running does not create a huge issue. This produces the same errors as when submitting a job with wrong input. It does not create an issue beyond the CE, but the site will not be usable for grid jobs.

Fault within the JobAgent startup script: JobAgent startup script consists of either values fetched from the LDAP or default values if the attributes are not described in the LDAP site configuration. These values can be wrong, but are still added to the script. Submitting the job is done without issues. The job itself will fail when running, but the standard error is discarded in the JAliEN job scheduler interface. The JobAgent will not start, but no exceptions are thrown or communicated. Since the job will fail, the slot is available again on the job scheduler. The CE will continue to submit JobAgents. This is an issue. No grid jobs will be given to the site since JobAgent will not be running, but the site is not functional while not communicating this. The way to solve this would be to parse the JobAgent script ahead of submitting it, but this will rarely happen and parsing the whole script is time and resource constricting.

Status when job queue has unrelated jobs running: Other jobs that are not relevant for JAliEn might be submitted to the same job queue. This would create an issue where there are less slots available for JAliEn to use since it will show up when using status commands. There is the possibility of filtering the result of the status to show only jobs submitted by a specific user. If the same user submitted unrelated jobs to the job queue there is nothing that would signal this. It could be possible to match patterns specific to JAliEn jobs. This would take more time and resources on something that is unlikely to occur, and does not create huge issues if it does.

Faults with status results: A fault with the status result could be the result of an error when using ProcessBuilder to run the status command. Instead of returning zero *numberActive* and *numberQueued* return -1. When the CE checks for slots available on the job queue it checks if the value is zero or above. A value below zero would trigger a log printing that an error with the status has occurred. No JobAgent will be submitted and the CE will continue to operate.

Faults with job scheduler: Problems might happen to the job scheduler used by a site that is a part of JAliEn. Connection to the job scheduler might be broken or the job scheduler might have internal issues. This is not a huge problem for the JAliEn batch queue interface. Failing to communicate with the job scheduler will throw an error when the CE tries to receive the status of the job queue. If the job scheduler goes down after checking for status, the CE might try to submit the job. This will result in an error, but not crashing the CE. Again the site is not suitable for grid jobs.

Peak loads: Given the test environment available there are limits to testing peak loads. Some sites on the grid are capable of running a thousand active and queued jobs at the same time. The setup used in this thesis is not capable of that. It is possible to queue over a thousand jobs, but the active jobs are very limited. Testing with a lot of active jobs would give more confidence that the solution can handle it, but is not strictly necessary. When submitting jobs it is handled by the job schedulers before executing it. This means that if every job is queued correctly any fault while the job is under transition from queued to active is happening at the job scheduler and not the JAliEn batch queue interface. Peak load was tested by submitting five thousand (**5000**) jobs. Every job was submitted correctly.

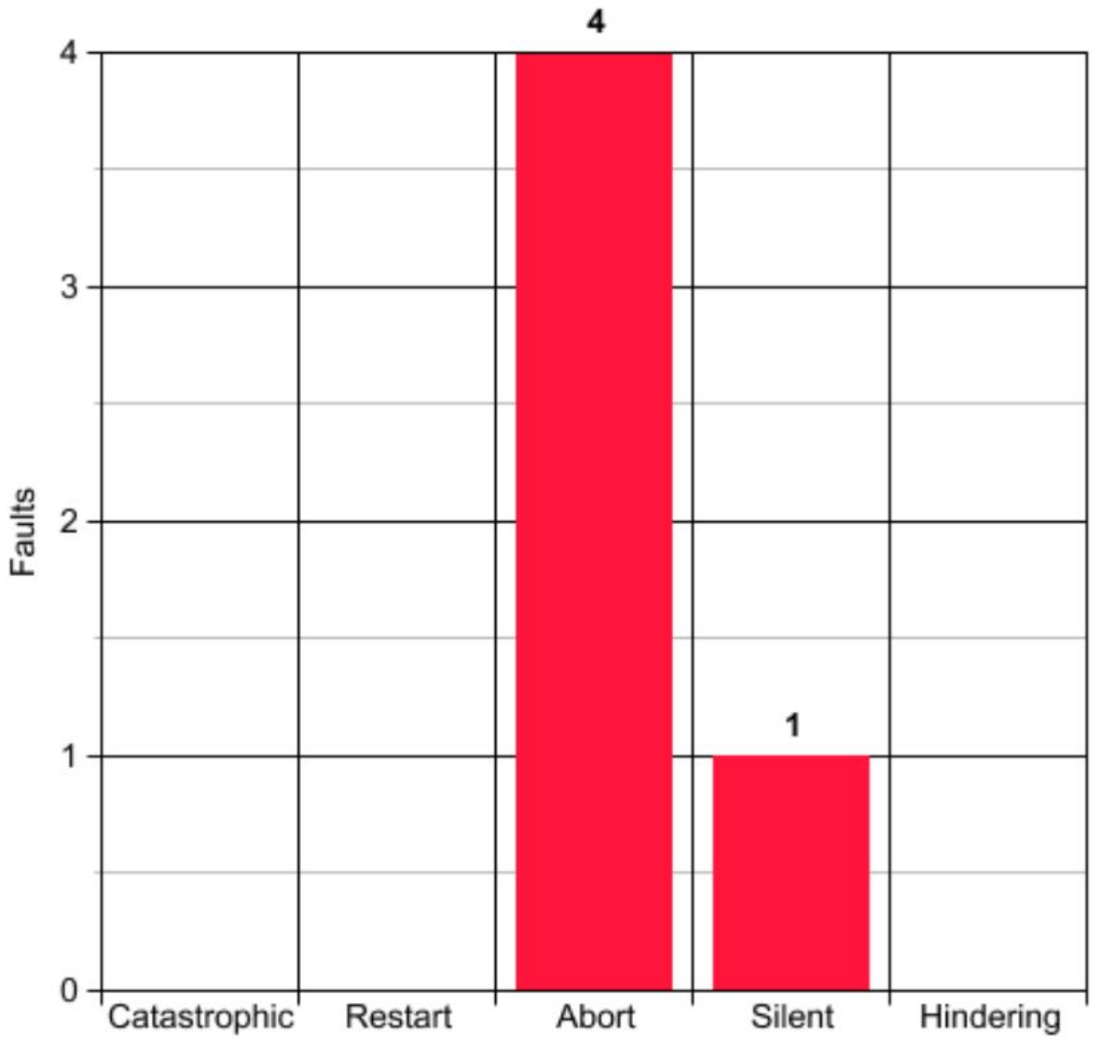


Figure 5.4: *Results of robustness testing using the CRASH benchmark.*

5.3.8 Result

Figure 5.4 shows the result of the robust testing using the *CRASH* benchmark. It does not give us a complete picture of the severity of the faults, but it is a good indicator. It shows that most of the fault occurring under the testing is *Abort*. These faults make the sites not usable for JAliEn in regards to running grid jobs, but do not affect JAliEn outside of that. There is one Silent fault. This is an issue as the CE and JAliEn batch queue interface continues to seemingly work correctly while this is not true. This did not impact any other part of the JAliEn than the site experiencing this fault. This is because the JobAgent is not executed, and will therefore not match with a grid job. It does make the site unavailable for running grid jobs without indicating that it is unavailable. Outside of the one silent fault, the expected results held true.

Chapter 6

Evaluation

This chapter aims to compare the requirements and criteria set in chapter 1 with the results obtained in chapter 5. An evaluation of the JAliEn batch queue interface solution will be made based on this, followed by a discussion on the viability of the solution.

6.1 Evaluation of functional requirements

The unit testing done in chapter 5 shows that the core functionalities regarding submitting jobs to the job scheduler, and getting the status of the job queue is achieved. Testing for backwards compatibility with old sites was not done. There are so many parameters that are unique for a site. Recreating environments used by all sites in JAliEn to test the solution presented is not feasible. Instead comparisons between the batch queue interface in JAliEn and AliEn are made. This has been done with the goal of achieving the same functionalities in mind. While this does not completely ensure that the solution fulfills the requirement of being backwards compatible it is nonetheless an indicator that this might hold true.

The requirement for a general solution holds true for all the job schedulers except HTCondor. None of the other job scheduler interfaces in JAliEn presented needs any site configuration from LDAP to interface with the job schedulers. This can be observed when performing unit tests with completely empty configuration. HTCondor on the other hand needs to know if it should be using job routing or grid resources. It is possible to achieve this with the HTCondor job scheduler interface in JAliEn, but a choice to keep the same functionality as the old AliEn code was made.

Few tests have been done on quality of the code like duplication and low coupling. Choices have been made under development that is aimed at achieving this goal. It is easy to see the low coupling when creating unit tests for the batch queue interface in JAliEn. There are only three parts the JAliEn batch queue interface needs to function. A configuration, logger and *JobAgent* startup script. The configuration can be empty for the other job scheduler than HTCondor. The logger class is used to correctly log errors or other information of relevance for users and developers of JAliEn. Lastly is the *JobAgent* startup path that is submitted to the job schedulers. Duplicated code on the other hand has not been tested but an effort to avoid it was made under development. Shared code was collected in the batch queue interface.

Being able to add a new job scheduler interface to the JAliEn batch queue interface cheaply is partially achieved by having low coupling and using an interface to only describe the methods that require a unique implementation. In addition, collecting duplicate and shared code in the batch queue interface ensures a new job scheduler interface in JAliEn needs less code to be implemented. All of this together makes it cheap to add more job scheduler interfaces in JAliEn. It is possible that some of the new job schedulers require something more from the environment the ProcessBuilder is running in, or when collecting information from the site configuration described in LDAP. Ways to achieve this is overriding the methods in the JAliEn batch queue interface and implementing a unique solution for the given job scheduler.

To be completely sure the solution fulfills all the functional requirements some unit tests are missing, and therefore the solution cannot be said to fulfill all the functional requirements without a method to validate it.

6.2 Evaluation of robustness requirements

The test done in chapter 5 shows that none of the faults injected into the JAliEn batch queue creates an issue beyond the site's ability to execute jobs. This includes both faults created by wrong input and peak loads. No JobAgent is started when an error occurs regarding submitting jobs to the local job scheduler. The site will be unavailable for grid jobs, but JAliEn does not crash and can make use of other available sites. The requirement of containing the faults is therefore met.

One silent error did occur, and one of the requirements for robustness is that errors should be handled or communicated. The silent error is a result of a job being submitted to the job scheduler with an executable that has faults. The JobAgent will therefore never be started. The issue is that the site is unavailable for executing grid jobs, but acting like it is available. This means the issue might go unnoticed for a time period. No error message also makes it more difficult to locate why and where it occurs. All other faults are properly communicated. The solution is not perfect, but it is adequate for its purpose. While robustness testing aims to cover everything, this cannot be expected. The tests done however indicate that it should be capable of handling unexpected faults occurring within the requirements outlined.

6.3 Determining viability

All testing is done in a development environment. Testing using the production environment was not an option, since JAliEn is not rolled out to sites yet. The solution should not behave differently in the production environment. The CE used when testing is up to date, and databases and LDAP servers mimics the setup found on the central servers. This means there is no way to guarantee the solution presented will work in the production environment when it is finished. Nonetheless, expectations are positive, and indicators show that it most likely will.

Given that not all the functional requirements have been tested and every test is done on an isolated development environment, it is not possible to guarantee the solution is viable. Manual testing and observations on how the solution interacts are indicating that the solution might be viable, but there are not enough tests to back this up.

Chapter 7

Conclusion

Grid computing is used to provide resources from all around the world to compute large and complex problems. Grid middlewares are interfacing with grids to provide services on the grids behalf. ALICE makes use of computing grids, and to be able to scale for the future, the new grid middleware JAliEn is under development.

This thesis aimed to provide a viable solution for an interface between JAliEn and job schedulers on sites. Requirements for the solution were outlined. This included requirements for both functionalities and robustness. To understand how the batch queue interface works within JAliEN, both AliEn and JAlien were discussed.

A solution was developed for Slurm, HTCondor and TORQUE, using the individual job schedulers characteristic. Duplicate codes were handled and old AliEn batch queue interfaces were studied and compared to not lose functionalities.

The solution presented was tested for both functionality and robustness. Functionalities were tested using unit testing. A strategy for testing and evaluating robustness was described, and the result of the tests were discussed. The test results were evaluated, comparing the results with the requirements described in chapter 1. Some functional requirements were found to not be completely fulfilled. The reason for this was not enough test to verify the solution fulfilled the requirements.

While indicators show that the solution might be a viable solution, this needs to be verified by evaluating test results. The combination of using a test environment completely isolated from the production environment, and some missing tests for functionality draws the conclusion that the solution cannot be called viable currently. Nevertheless, manual testing while developing and observations does show positive indicators.

7.1 Further work

Further work should be focused on more extensive tests of the solution to provide a viable solution. The tests should focus on the functional requirements. In addition, only three job schedulers have been discussed and developed in this thesis. The ALICE computing grid has several other job schedulers running on different sites. Adding more job scheduler interfaces to the batch queue interface in JAliEn is needed to make use of all the sites on the grid.

Appendices

Appendix A

Installation

When developing the JAliEn batch queue interface a local JAliEn was used in addition to a local setup of JCentral and LDAP. To achieve this a few different technologies are needed. Since JAliEn is under development and therefore under constant change the newest working version is prefered. Usually database, LDAP and JCentral is handled by the central services, and to run a site only a part of the JAliEn code is needed in addition to adding the site, with its corresponding parameters, to the central LDAP.

JAliEn

To get the newest version of JAliEn all that is needed is to have git installed on the computer and clone the directory. When a new version is pushed to the git repository, a git pull will update the code.

Since the JCentral will be running locally as well, JAliEn has to be installed twice, using different users or machines. If JCentral and the site is a single installation under a user it creates a lot of problems like setting a default user without any certificates, which creates errors in the site part of JAliEn, causing null pointer errors, and crashing the program.

Configuration

JAliEn needs to know how to connect to the LDAP and databases, this is done in the different properties files found in “/jalien/config/” folder. Copy this folder over to the “~/.j” folder. The properties files default to the database and LDAP to CERN Central?, so it needs to be changed to the local ones that are used in this setup. This needs to be done on both the JAliEn that will be running the JCentral and the one running the site. On the user running JCentral create a password.properties file and write password=”database password”, making the JCentral having direct database connection and not being a load balancer.
There still exists some code which is hardcoded to use the configuration found in the “jalien/config” folder so to remove potential issues the files will match the ones found in the other “.j” folder.

Certificates

Certificates are important to assure that only authorized users are granted access, and the setup needs the bare minimum to make sure JAliEn is capable of running. Copy the certificates from "/jalien/installation/trusts/trusts" folder into the ".j/trusts" folder. In both the JCentral and site installation we need an Alien.p12 certificate. What certificate authority used is of no importance, and a self-signed one will be used. The password used in the Alien.p12 certificate needs to be added to the config.properties file as ca.password="password". It is also important that the Alien.p12 certificate has the alias "Alien". After this is done the certificate will be added to the ".globus" folder. In this folder we also need host and user certificates. On the JCentral installation the user has to be equal to the host certificate.

Run and compile

To be able to run and compile, JAliEn JDK 12 or newer needs to be available. There are different ways to do this. It can be installed locally or we can let the JAVA_HOME and PATH environment variable point to the latest JDK in the CVMFS. The latest JDK in the CVMFS will be supported and is a safe way to be sure it is compatible, and is the approach used in the environment used in this paper.

JAliEn has its own shell script both for compiling and running. Compiling is done by running the compile.sh and works without any changes to it, but a change was made due to it forcing the use of an old JDK version. Making it use the newest JDK instead speeds up the process of compiling considerably. First time compiling, use the tag "all", to get all the needed jars. To run JAliEn a running instance of JCentral is needed and then using the jalien shell script, which in turn will run the run.sh script. This will not work before you remove "*-XX:+UseCompressedOops"* from run.sh. To get the JCentral up and running, run the JCentral.sh script from the corresponding JAliEn.

Database and LDAP

Both database and LDAP have to mirror the setup found on central services. The database needs to be tweaked manually as there are deficiencies in the JCentral. It should be populated dynamically by JAliEn, but not everything that is needed for running the CE is added to the database. To get the ComputingElement running, it needs to find the maximum number of jobs the site is capable of running and queueing, which is set to null, and needs to manually be changed to the wanted number in the "HOSTS" table in the "process" database.

The queue also needs to be open, but it is default set to blocked. The way to open it is to use the JSh and the “queue” command, but as that is not implemented yet, again the it needs to be changed manually in the “SITEQUEUE” table in the “process” database.

Bibliography

1. CERN, *About CERN*. https://home.cern/about. Accessed: 14 June 2020
2. CERN, *The Large Hadron Collider.* https://home.cern/science/accelerators/large-hadron-collider. Accessed: 14 June 2020
3. ALICE, *ALICE*. https://home.cern/science/experiments/alice. Accessed: 14 June 2020
4. CERN, *Processing: What to record?* https://home.cern/science/computing/processing-what-record. Accessed: 14 June 2020
5. P. Buncic, J. F. Grosse-Oetringhaus, A. J. Peters, P. Saiz, *The architecture of the AliEn system,* CERN, 2005
6. B Jacob, M Brown, K Fukui, N Trivedi, *Introduction to grid computing*, p. 3-29, IBM redbooks, 2005.
7. Ian Foster, *What is the Grid? A Three Point Checklist,* GRID today. 1, 2002
8. CERN, *Worldwide LHC Computing Grid*. https://wlcg-public.web.cern.ch/. Accessed: 14 June 2020
9. LDAP, *Learn About LDAP.* https://ldap.com/learn-about-ldap/ Accessed: 14 June 2020
10. Talend, *Beginner's Guide to Batch Processing*. https://www.talend.com/resources/batch-processing/. Accessed: 14 June 2020
11. SchedMD, *SLURM Workload Manager - Overview.* https://slurm.schedmd.com/overview.html. Accessed: 14 June 2020
12. Adaptive Computing, *TORQUE Resource Manager - Getting Started.* http://docs.adaptivecomputing.com/torque/6-1-2/adminGuide/torque.htm#topics/torque/0-intro/gettingStarted.htm#ResourceManager. Accessed: 14 June 2020
13. P. Vande Vyvre, *ALICE computing strategies for Run3,* CERN-EP, 2018
14. Carlisle Adams, Steve Lloyd, *Understanding PKI : concepts, standards, and deployment considerations*, p. 249 - 263, Addison-Wesley Professional, 2003
15. G Grigoras et al, *JAliEn – A new interface between the AliEn jobs and the central services*, Journal of Physics: Conference Series. 523. 012010, 2014

16. CERN, *CernVM File System (CernVM-FS)* https://cernvm.cern.ch/portal/filesystem. Accessed: 14 June 2020
17. Oracle, *ProcessBuilder Documentation,* https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html. Accessed: 14 June 2020
18. Oracle, *Process Documentation* https://docs.oracle.com/javase/7/docs/api/java/lang/Process.html. Accessed: 14 June 2020
19. Julie Cohen, Dan Plakosh, Kristi Keeler, *Robustness Testing of Software-Intensive Systems: Explanation and Guide,* Carnegie Mellon University, 2005
20. IEEE Std 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology,* 1990
21. Jean-Claude Fernandez, Laurent Mounier, Cyril Pachon, *A model-based approach for robustness testing,* Verimag, 2005
22. Institute for Telecommunication Sciences (ITS), *Graceful Degradation*, http://www.its.bldrdoc.gov/fs-1037/dir-017/_2479.htm. Accessed: 14 June 2020
23. Jianlin Zhu et al, *Enhancing the AliEn Web Service Authentication,* J. Phys.: Conf. Ser.331 062048, 2011
24. Andreas Peters,Pablo Saiz, Predrag Buncic, *AliEnFS - a Linux File System for the AliEn Grid Services*. CoRR. cs.DC/0306071, 2003
25. Ku Ruhana Ku-Mahamud, Husna Jamal Abdul Nasir. *Ant Colony Algorithm for Job Scheduling in Grid Computing*. Asia International Conference on Modelling & Simulation. p. 40-45, 2010
26. Pablo Saiz,Predrag Buncic, Andreas Peters. *The AliEn system, status and perspectives*. CoRR. cs.DC/0306067, 2003.
27. Pablo Saiz, Predrag Buncic, Andreas Peters. *AliEn Resource Brokers.* CERN, 2003
28. Predrag Buncic et al, *Middleware for the next generation Grid infrastructure,* EGEE-PUB-2004-002, 2004
29. Alina Grigoras, Grigoras Adriana, M. Pedreira, P. Saiz, S. Schreiner, *JAliEn – A new interface between the AliEn jobs and the central services*. Journal of Physics: Conference Series. 523. 012010, 2014
30. HTCondor, *What is HTCondor?,* https://research.cs.wisc.edu/htcondor/description.html. Accessed: 14 June 2020
31. S Bagnasco et al, *AliEn: ALICE environment on the GRID,* J. Phys.: Conf. Ser.119 062012, 2003

32. Ali Shahrokni, Robert Feldt, *A systematic review of software robustness*, Information and Software Technology volume 55, p. 1–17, 2013
33. Zoltan Micskei et al, *Robustness Testing Techniques and Tools,* In Resilience Assessment and Evaluation of Computing Systems, Springer, p. 323--339,2012
34. Maksim Melnik Storetvedt, *Applying Cloud Technologies for Grid Analysis,* Bergen University College / University of Bergen, 2016
35. Douglas Thain, Todd Tannenbaum, Miron Livny, *Condor and the Grid,* University of Wisconsin-Madison, 2003
36. Todd Tannenbaum, Derek Wright, Karen Miller, Miron Livny, *Condor - A Distributed Job Scheduler,* University of Wisconsin-Madison, 2001
37. Miguel Pedreira, Costin Grigoras, Volodymyr Yurchenko, Maksim Storetvedt,*The Security model of the ALICE next generation Grid framework,* EPJ Web of Conferences 214, 03042, 2019
38. Nikola Hardi, *jAliEn user client and new ROOT classes,* CERN, 2018
39. Matteo Turilli, Mark Santcroos, Shantenu Jha, *A Comprehensive Perspective on Pilot-Job Systems*. ACM Comput. Surv. 51, 2, Article 43, 2018
40. Donders Centre for Cognitive Neuroimaging, *Running computations on the Torque cluster* https://dccn-hpc-wiki.readthedocs.io/en/latest/docs/cluster_howto/compute_torque.html. Accessed 15 June 2020