Università degli Studi di Ferrara

DOTTORATO DI RICERCA IN
Matematica e Informatica

CICLO XXVII

COORDINATORE Prof. Massimiliano Mella

A general purpose framework for Grid resource exploitation

Settore Scientifico Disciplinare INF/01

Dottorando
Dott. Fella Armando

(firma)

Tutore
Prof. Tomassetti Luca

(firma)

Anni 2012/2015

Contents

Introduction

Many research activities from several fields, such as high energy, nuclear and atomic physics, biology, medicine, geophysics and environmental science, rely on data simulation, which are high CPU-consuming task. Sequential computation may require months or years of CPU time, so a loose-parallel distributed execution is expected to give benefits to these applications. In fact, the large number of storage and computation resources offered by a distributed computing environment allow to consistently reduce the amount of task completion time by splitting it in several parts and executing each part on a single node.

The potential intensive use of distributed computing resources by a large variety of research communities is reduced by the difficulties that researchers can encounter in using the complex software infrastructure at the base of distributed computing itself. High energy physics experiments made a pioneer work [1, 2, 3] on this but their results are still hardly available to small and mid-size organizations that may have similar computational requirements, mostly due to the large amount of needed technical expertise.

This document presents a prototype software-suite intended to be an interface to distributed computing resources such as LHC [4] Computing Grid LCG [5] [6] or Cloud [7] stacks. The lightweight and general-purpose framework built on standard functionality has been designed for organizations requiring an easy-to-use interface to such resources, but cannot afford the costs of a specialized software environment in terms of expertise, development and maintenance.

This work is the outcome of the collaboration of various institutions taking part of the High Energy Physics experiment SuperB [8]; it was born to be able to perform the huge amount of simulation tasks requested for SuperB detector design, subsequently it grown up as an autonomous, general-purpose project. I participated to the architecture definition, project design and developing of key functionality with the collaboration of Ferrara INFN and University groups.

The dissertation is composed by the following chapter structure:

- Chapter one is entirely dedicated to the description of LCG model and its components; it can be useful to non expert reader for the comprehension of the subject. Besides this vademecum of the Grid, the reader can refer to the glossary appendix A at the end of the document

- Chapter two contains the description of the project background. The original proto-framework was born to accomplish the computing requirements of SuperB experiment to allow the simulation of detector design in a distributed computing environment

- Third chapter hosts the description of the core elements of the project. The goals, the conceptual design and Grid integration have been introduced as starting point, subsequently the workload and metadata management systems have been analyzed in deep touching all the architectural and functional aspects. The description of the Job wrapper and the Operations Web Portal conclude the chapter

- The description of the project integration with Dirac[1] platform is reported in fourth chapter.

- The document concludes with the report of the framework results in a real utilization case, the SuperB simulation campaigns

[1]DIRAC [9] (Distributed Infrastructure with Remote Agent Control) suite is a software framework for distributed computing providing a complete solution to one (or more) user community requiring access to distributed resources

Chapter 1

Fundamentals of LHC Computing Grid

The "LHC Computing Grid, Technical design report" [10] contains a very clear and complete description of the LCG project; the goals, the distributed infrastructure, the services and the main components have been discussed at a level of detail permitting to embrace the entire complexity of the model avoiding detailed technical description. Despite the age of this document it remains a complete and general view of the LHC distributed computing model and fully satisfy the needs of this thesis. I'm citing here portions of the "LHC Computing Grid, Technical design report" [1] to contextualize the Ph.D. work which is tightly dependent on LCG middleware.

Virtual Organization (VO) is defined as "a temporary or permanent coalition of geographically dispersed individual, group, organizational units or entire organizations that pool resources, capabilities, and information to achieve common objectives" [11].

"The LHC Computing Grid Project (LCG) was approved by the CERN Council on 20 September 2001 to develop, build and maintain a distributed computing infrastructure for the storage and analysis of data from the four LHC experiments[2].

[1]In specific parts of chapter 3 section 1 and chapter 4 section 1 have been reported

[2]The data from the LHC experiments will be distributed around the globe, according

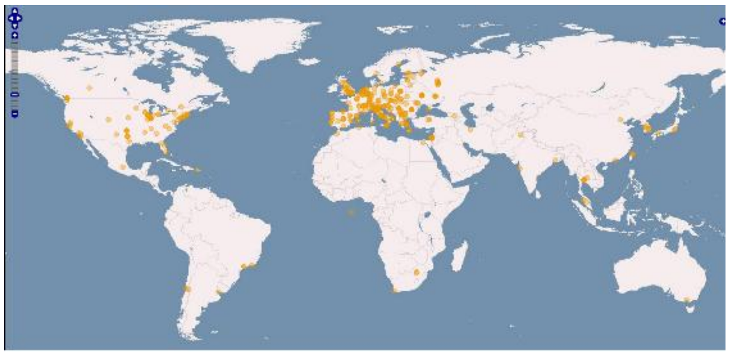


Figure 1.1: The Worldwide LHC Computing Grid (WLCG) project [12] is a global collaboration of more than 170 computing centres in 40 countries, linking up national and international grid infrastructures. The mission of the WLCG project is to provide global computing resources to store, distribute and analyse the 30 Petabytes (30 million Gigabytes) of data annually generated by the Large Hadron Collider (LHC) at CERN on the Franco-Swiss border.

Developing common applications software for all experiments is an important part of the LCG Project. This includes core software libraries, tools and frameworks for data management and event simulation as well as infrastructure and services for software development, and the support of analysis and database management.

to hierarchical model. The raw data emerging from the data-acquisition systems will be recorded on tape and initially processed at the Tier-0 centre of LCG located at CERN. The data will be distributed to a series of Tier-1 centres, large computer centres with sufficient storage capacity for a large fraction of the data, and with round-the-clock operation. Analysis tasks requiring access to large subsets of the raw, processed, and simulated data will take place at the Tier-1 centres. The Tier-1 centres will make data available to Tier-2 centres, each consisting of one or several collaborating computing facilities, which can store sufficient data and provide adequate computing power for end-user analysis tasks and Monte Carlo simulation. Individual scientists will also access these facilities through Tier-3 computing resources, which can consist of local clusters in a University Department or even individual PCs.

1.1 Basic LCG architecture

The LCG architecture will consist of an agreed set of services and applications running on the Grid infrastructures provided by the LCG partners. These infrastructures at the present consist of those provided by the Enabling Grids for E-sciencE (EGEE) [13] project in Europe, the Open Science Grid (OSG) [14] project in the U.S.A. and the Nordic Data Grid Facility in the Nordic countries [15]. The EGEE infrastructure brings together many of the national and regional Grid programmes into a single unified infrastructure. In addition, many of the LCG sites in the Asia-Pacific region run the EGEE middleware [16] stack and appear as an integral part of the EGEE infrastructure.

1.1.1 Grid Functionality and Services

The set of services that should be made available to the experiments have been discussed and agreed in the Baseline Services Working Group set up by the LCG Project Execution Board in February 2005. The report of the group identified the services described here. The full details of the services, the agreed set of functionality, and the interfaces needed by the experiments is described fully in the report of the working group.

1.1.2 Storage Element Services

A Storage Element (SE) is a logical entity that provides the following services and interfaces: Mass storage system, either disk cache or disk cache front-end backed by a tape system. Mass storage management systems currently in use include CASTOR, Enstore-dCache, HPSS and Tivoli for tape/disk systems, and dCache [18], LCG-dpm [19], and DRM for disk-only systems.

- SRM [17] interface to provide a common way to access the MSS no matter what the implementation of the MSS. The Storage Resource

Manager (SRM) defines a set of functions and services that a storage system provides in an MSS-implementation independent way. The Baseline Services working group has defined a set of SRM functionality that is required by all LCG sites. This set is based on SRM v1.1 with additional functionality (such as space reservation) from SRM v2.1. Existing SRM implementations currently deployed include CASTOR-SRM, dCache-SRM, DRM/HRM from LBNL, and the LCG dpm.

- GridFTP service to provide data transfer in and out of the SE to and from the Grid. This is the essential basic mechanism by which data is imported to and exported from the SE. The implementation of this service must scale to the bandwidth required. Normally the GridFTP transfer will be invoked indirectly via the File Transfer Service or through srmcopy.

- Local POSIX-like input/output facilities to the local site providing application access to the data on the SE. Currently this is available through rfio, dCap, aiod, rootd, according to the implementation. Various mechanisms for hiding this complexity also exist, including the Grid File Access Library in LCG-2, and the gLiteIO service in gLite. Both of these mechanisms also include connections to the Grid file catalogues to enable an application to open a file based on Logical File Name (LFN) or Global Unique IDentifier (GUID).

- Authentication, authorization and audit/accounting facilities. The SE should provide and respect ACLs for files and datasets that it owns, with access control based on the use of extended X509 proxy certificates with a user DN and attributes based on VOMS [20] roles and groups. It is essential that a SE provide sufficient information to allow tracing of all activities for an agreed historical period, permitting audit on the activities. It should also provide information and statistics on the use of the storage resources, according to schema and policies to be defined.

A site may provide multiple SEs providing different qualities of storage. For example, it may be considered convenient to provide an SE for data intended to remain for extended periods and a separate SE for data that is transient - needed only for the lifetime of a job or set of jobs. Large sites with MSS-based SEs may also deploy disk-only SEs for such a purpose or for general use.

1.1.3 File Transfer Services

Basic-level data transfer is provided by GridFTP. This may be invoked directly via the globus-url-copy command or through the srmcopy command which provides 3rd-party copy between SRM systems. However, for reliable data transfer it is expected that an additional service above srmcopy or GridFTP will be used. This is generically referred to as a reliable file transfer service (rfts). A specific implementation of this - this gLite FTS has been suggested by the Baseline Services Working group as a prototype implementation of such a service. The service itself is installed at the Tier-0 (for Tier-0-Tier-1 transfers) and at the Tier-1s (for Tier-1-Tier-2 transfers). It can also be used for 3rd-party transfers between sites that provide an SE. No service needs be installed at the remote site apart from the basic SE services described above. However, tools are available to allow the remote site to manage the transfer service. For sites or Grid infrastructures that wish to provide alternative implementations of such a service, it was agreed that the interfaces and functionality of the FTS will be taken as the current interface. The gLite File Placement Service (FPS) takes data movement requests and executes them according to defined policies. It maintain a persistent transfer queue thus providing reliable data transfer even in the case of network outage and interacts fully with the Fireman catalogue. The File Placement service can be used without the interaction with the catalogue and is then referred to as the File Transfer Service [21] (FTS)[3].

[3]FTS release 3 is now in use by all the LHC experiments and by all the research communities making use of Grid infrastructure

1.1.4 Compute Resource Services

The Computing Element (CE) is the set of services that provide access to a local batch system running on a compute farm. Typically the CE provides access to a set of job queues within the batch system. How these queues are set up and configured is the responsibility of the site and is not discussed here. A CE is expected to provide the following functions and interfaces:

- A mechanism by which work may be submitted to the local batch system. This is implemented typically at present by the Globus gatekeeper in LCG-2 and Grid/Open Science Grid. NorduGrid (the ARC middleware) uses a different mechanism.
- Publication of information through the Grid information system and associated information providers, according to the GLUE schema, that describes the resources available at a site and the current state of those resources. With the introduction of new CE implementations we would expect that the GLUE schema, and evolutions of it, should be maintained as the common description of such information.
- Publication of accounting information, in an agreed schema, and at agreed intervals. Presently the schema used in both LCG-2 and Grid3/ OSG follows the GGF accounting schema. It is expected that this be maintained and evolved as a common schema for this purpose.
- A mechanism by which users or Grid operators can query the status of jobs submitted to that site.
- The Computing Element and associated local batch systems must provide authentication and authorization mechanisms based on the VOMS model. How that is implemented in terms of mapping Grid user DNs to local users and groups, how roles and subgroups are implemented, may be through different mechanisms in different Grid infrastructures. However, the basic requirement is clear - the user presents an extended X509 proxy certificate, which may include a set of roles, groups, and

subgroups for which he is authorized, and the CE/batch system should respect those through appropriate mappings locally.

It is anticipated that a new CE from gLite, based on Condor-C, will also be deployed and evaluated as a possible replacement for the existing Globus GRAM-based CEs within LCG-2 and Open Science Grid.

1.1.5 Workload Management

Various mechanisms are currently available to provide workflow and workload management. These may be at the application level or may be provided by the Grid infrastructure as services to the applications. The general feature of these services is that they provide a mechanism through which the application can express its resource requirements, and the service will determine a site that fulfills those requirements and submit the work to that site. It is anticipated that on the timescale of 2006-2007 there will be different implementations of such services available, for example, the LCG-2 Resource Broker, and the Condor-G mechanism used by some applications in Grid3/OSG, and new implementations such as that coming from gLite implementing both push and pull models of job submission. The area of job workflow and workload management is one where there are expected to be continuing evolution over the next few years, and these implementations will surely evolve and mature.

1.1.6 VO Management services

The VOMS software will be deployed to manage the membership of the VOs. It will provide a service to generate extended proxy certificates for registered users which contain information about their authorized use of resources for that VO.

1.1.7 Database Services

Reliable database services are required at the Tier-0 and Tier-1 sites, and may be required at some or all of the Tier-2 sites depending on experiment configuration and need. These services provide the database back-end for the Grid file catalogues as either central services located at CERN or local catalogues at the Tier-1 and Tier-2 sites. Reliable database services are also required for experiment-specific applications such as the experiment metadata and data location catalogues, the conditions databases and other application-specific uses. It is expected that these services will be based on scalable and reliable hardware using Oracle at the Tier-0, Tier-1 and large Tier-2 sites, and perhaps using MySQL on smaller sites. Where central database services are provided, replicas of those databases may be needed at other sites. The mechanism for this replication is that described by the 3D project in the applications section of this report.

1.1.8 Grid Catalogue Services

The experiment models for locating datasets and files vary somewhat between the different experiments, but all rely on Grid file catalogues with a common set of features. These features include:

- Mapping of Logical file names to GUID and Storage locations (SURL)
- Hierarchical namespace (directory structure)
- Access control

The deployment models also vary between the experiments, and are described in detail elsewhere in this document. The important points to note here are that each experiment expects a central catalogue which provides look-up ability to determine the location of replicas of datasets or files. This central catalogue may be supported by read-only copies of it regularly and frequently replicated locally or to a certain set of sites. There is, however, in all cases a single master copy that receives all updates and from which the replicas are generated. Obviously this must be based on a very reliable database service. The central catalogues must also provide an interface to the various workload management systems. These interfaces provide the location of Storage Elements that contain a file (or LHC COMPUTING GRID Technical Design Report dataset) (specified by GUID or by logical file name) that the workload management system can use to determine which set of sites contain the data that the job needs. This interface should be based on the StorageIndex of gLite or the Data Location Interface of LCG/CMS. Both of these are very similar in function. Any catalogue providing these interfaces could be immediately usable by, for example, the Resource Broker or other similar workload managers. The catalogues are required to provide authenticated and authorized access based on a set of roles, groups and sub-groups. The user will present an extended proxy-certificate, generated by the VOMS system. The catalogue implementations should provide access control at the directory level, and respect ACLs specified by either the user creating the entry or by the experiment catalogue administrator. It is expected that a common set of command-line catalogue management utilities be provided by all implementations of the catalogues. These will be based on the catalogue-manipulation tools in the lcg-utils set with various implementations for the different catalogues, but using the same set of commands and functionalities.

1.1.9 Job Monitoring Tools

The ability to monitor and trace jobs submitted to the Grid is an essential functionality. There are some partial solutions available in the current systems (e.g., the LCG-2 Workload Management system provides a comprehensive logging and book-keeping database), however, they are far from being full solutions. Effort must be put into continuing to develop these basic tools, and to provide the users with the appropriate mechanisms through which jobs can be traced and monitored.

1.1.10 Interoperability

This section has outlined the basic essential services that must be provided to the LHC experiments by all Grid implementations. The majority of these deal with the basic interfaces from the Grid services to the local computing and storage fabrics, and the mechanisms by which to interact with those fabrics. It is clear that these must be provided in such a way that the application should not have to be concerned with which Grid infrastructure it is running on. At the basic level of the CE and SE, both EGEE and Grid3/OSG use the same middleware and implementations, both being based on the Virtual Data Toolkit. In addition, both use the same schema for describing these services, and have agreed to collaborate in ensuring that these continue to be compatible, preferably by agreeing to use a common implementation of the information system and information providers. Common work is also in hand on other basic services such as VOMS and its management interfaces. In addition, both EGEE and OSG projects are defining activities to ensure that interoperability [22] remain a visible and essential component of the systems. The EGEE Resource Broker, as it is based on Condor-G, can submit jobs to many middleware flavours including ARC (NorduGrid). When the Glue2 information system schema, being defined jointly by several Grid projects, is available this will enable the EGEE Resource Broker to schedule resources at sites running ARC. Further steps towards interoperability in the areas of workload management and data management are planned by the NorduGrid Collaboration. Other activities are being undertaken by the developers of ARC to foster and support standards and community agreements. These include participation in the Rome Compute Resource Management Interfaces initiative and in the Global Grid Forum. These activities will improve interoperability between different middleware implementations, and in the longer term we can expect standards to emerge and be supported by future versions of the software. For the medium term, however, the approach taken by the LCG Project was to set up the Baseline Services Working Group to define a set of basic services that can be deployed in all of the existing Grid infrastructures, taking account of their different technical constraints. In some cases the services are defined in terms of standard interfaces, while in other cases a specific implementation is identified. In this way sites providing resources to LCG will be able to provide these essential services in a transparent way to the applications.

1.2 EGEE Middleware

The EGEE middleware deployed on the EGEE infrastructure consists of a packaged suite of functional components providing a basic set of Grid services including job management, information and monitoring and data management services. The LCG2.x middleware, currently deployed in over 100 sites worldwide originated from Condor, EDG, Globus, VDT and other projects. It is anticipated that the LCG-2 middleware will evolve in summer 2005 to include some functionalities of the gLite middleware provided by the EGEE project. This middleware has just been made available as this report is being written, and has not yet passed certification . The rest of this chapter will describe, respectively, the LCG-2 middleware services and the gLite ones. The middleware can in general be further categorized into site services and Virtual Organization (VO) services as described below.

1.2.1 Site Services

Security

All EGEE middleware services rely on the Grid Security Infrastructure (GSI). Users get and renew their (long-term) certificate from an accredited Certificate Authority (CA). Short-term proxies are then created and used throughout the system for authentication and authorization. These short-term proxies may be annotated with VO membership and group information obtained from the Virtual Organization Membership Services (VOMS). Access to (site) services is controlled by the Java authorization framework

(Java services) and LCAS (C services). When necessary, in particular for job submission, mappings between the user Distinguished Names (DN) and local account are created (and periodically checked) using the LCAS and LCMAPS services. When longer-term proxies are needed, MyProxy services can be used to renew the proxy. The sites maintain Certificate Revocation Lists (CRLs) to invalidate unauthorized usage for a revoked Grid user.

Computing Element

The Computing Elements (CEs), often dubbed head nodes, provide the Grid Interfaces to Local Resource Managers (a.k.a. site batch systems). They normally require external network connectivity.

LCG-2 Computing Element The LCG-2 Computing Element (CE) handles job submission (including staging of required files), cancellation, suspension and resume (subject to support by the Local Resource Management System - LRMS), job status inquiry and notification. It only works in push mode where a job is sent to the CE by a Resource Broker (RB). Internally the LCG-2 CE makes use of the Globus gatekeeper, LCAS/LCMAPS and the Globus Resource Allocation Manager (GRAM) for submitting jobs to the LRMS. It also interfaces to the logging and book-keeping Services to keep track of the jobs during their lifetime. An updated version is due in summer 2005. The LCG-2 CE interfaces with the following LRMS: BQS, Condor, LSF, PBS and its variants (Torque/Maui), and many others.

gLite Computing Element The gLite Computing Element (CE) handles job submission (including staging of required files), cancellation, suspension and resume (subject to support by the LRMS), job status inquiry and notification. The CE is able to work in a push model (where a job is pushed to a CE for its execution) or in a pull model (where a CE asks a known Workload Manager - or a set of Workload Managers - for jobs). Internally the gLite CE makes use of the new CondorC technology, GSI and LCAS/LCMAPS, as well as the Globus gatekeeper. The CE is expected to evolve into a VO-based scheduler that will allow a VO to dynamically deploy their scheduling agents. The gLite CE also make use of the 3ogging and book-keeping services to keep track of the jobs during their lifetime.

The gLite CE interfaces with the following LRMS: PBS and its variants (Torque/Maui), LSF and Condor. Work to interface to BQS (IN2P3) and SUN Grid Engine (Imperial College) is under way.

Storage Element

The Storage Element (SE) provides the Grid interfaces to site storage (can be Mass Storage or not). SEs normally require external network connectivity.

LCG-2 Storage Elements The LCG-2 SE can either by a ”classic” SE or an SRM SE. The classic SE provides a GridFTP (efficient FTP functionality with GSI security) interface to disk storage. The RFIO protocol can be used for accessing directly the data on a classic SE. An SRM SE provides the GridFTP interface to a Storage Resource Manager (SRM), a common interface to Mass Storage Systems such as the CERN Advanced Storage Manager (CASTOR) or dCache/Enstore from DESY and FNAL. Recently, a more lightweight and simpler SRM has been made available, the LCG Disk Pool Manager (DPM), which is targeted at smaller disk pools. The DPM is a natural replacement for the classic SE.

GFAL The Grid File Access Library (GFAL) is a POSIX-like I/O layer for access to Grid files via their Logical Name. This provides open/read/write/-close style of calls to access files while interfacing to a file catalogue. GFAL currently interfaces to the LFC [23] and the LCG-RLS catalogs. A set of command line tools for file replication called lcg-utils have been built on top of GFAL and catalogue tools supporting SRMs and classic SEs.

gLite Storage Element A gLite Storage Element consists of a SRM (such as CASTOR, dCache or the LCG Disk Pool Manager) presenting a SRM 1.1

interface, a GridFTP server as the data movement vehicle and gLite I/O for providing a POSIX-like access to the data. gLite itself does not provide a SRM nor a GridFTP server which must be obtained from the standard sources.

Monitoring and Accounting Services

The monitoring and accounting services retrieve information on Grid services provided at a site as well as respective usage data, and publish them. User information (in particular related to job execution progress) may be published as well.

gLite Monitoring and Accounting Services gLite relies on the same services as described in Section 0. In addition, an R-GMA-based service discovery system is provided. The gLite accounting system (DGAS) is subject to evaluation. DGAS collects information about usage of Grid resources by users, groups of users (including VO). This information can be used to generate reports/billing but also to implement resources quotas. Access to the accounting information is protected by ACLs. More information on DGAS is available at Ref [24].

1.2.2 VO or Global Services

Virtual Organization Membership Service

The Virtual Organization Membership Service (VOMS) annotates short-term proxies with information on VO and group membership, roles and capabilities. It originated from the EDG project. It is in particular used by the Workload management System and the FireMan catalogue for ACL support to provide the functionality identified by LCG. The main evolution from EDG/LCG is support for SLC3, bug fixes and better conformance to IETF RFCs. A single VOMS server can serve multiple VOs. A VOMS Administrator Web interface is available for managing VO membership through the use of a Web browser. There is no significant functional difference between the VOMS in LCG-2 and in gLite. VOMS 1.5 and higher supports both MySQL and Oracle. For a detailed description of VOMS and its interfaces, see Refs. [25] and [26].

Workload Management Systems

gLite Workload Management System The Workload Management system in gLite is an evolution of the one in LCG-2. As such, it relies on BDII as an information system. It is interoperable with LCG-2 CEs. The Workload Management System (WMS) operates via the following components and functional blocks: The Workload Manager (WM) or Resource Broker, is responsible of accepting and satisfying job management requests coming from its clients. The WM will pass job submission requests to appropriate CEs for execution, taking into account requirements and preferences expressed in the job Description. The decision as to which resource should be used is the outcome of a matchmaking process between submission requests and available resources. This not only depends on the state of resources, but also on policies that sites or VO administrators have put in place (on the CEs). Interfaces to Data Management allowing the WMS to locate sites where the requested data is available are available for LCG RLS, the Data Location Interface (DLI - used by CMS) and the StorageIndex interface (allowing for querying catalogs exposing this interface - a set of two methods listing SEs for a given LFN or GUID, implemented by the FiReMan and AliEn catalogs). The WMproxy component, providing a Web service interface to the WMS as well as bulk submission and parametrized job capabilities is foreseen to be available before the end of the EGEE project. The user interfaces to the WMS using a Job Description Language based on Condor Classads is specified at Ref. [27]. The user interacts with the WMS using a Command Line Interface or APIs. Support of C++ and Java is provided (for a detailed description of the WMS and the interfaces, see Refs. [28] and [29])

File Catalogs

Files on Grids can be replicated in many places. The users or applications do not need to know where the files actually are, and use Logical File Names (LFNs) to refer to them. It is the responsibility of file catalogs to locate and access the data. In order to ensure that a file is uniquely identified in the universe, Global Unique Identifiers (GUIDs) are usually used.

EDG and RMS The services provided by the RMS, originating from EDG, are the Replica Location Service (RLS) and the Replica Metadata Catalogue (RMC). The RLS maintains information about the physical location of the replicas. The RMC stores mappings between GUIDs and LFNs. A last component is the Replica Manager offering a single interface to users, applications or Resource Brokers. The command line interfaces and APIs for Java and C++ are respectively available from Refs. [30] and [31]. It is anticipated that the EDG RMS will gradually be phased out.

LCG File Catalogue The LCG File catalogue (LFC) offers a hierarchical view of logical file name space. The two functions of the catalogue are to provide Logical File Name to Storage URL translation (via a GUID) and to locate the site at which a given file resides. The LFC provides Unix style permissions and POSIX Access Control Lists (ACL). It exposes a transactional API. The catalogue exposes a so-called Data Location Interface (DLI) that can be used by applications and Resource Brokers. Simple metadata can be associated with file entries. The LFC supports Oracle and MySQL databases. The LFC provides a command line interface and can be interfaced through Python.

Information Services

Information services publish and maintain data about resources in Grids. This information in LCG is modelled after the Grid Laboratory Uniform Environment schema (GLUE).

BDII The Berkeley Database Information Index (BDII) [32] is an implementation of the Globus Grid Index Information Service (GIIS), but allowing for more scalability. Information provided by the BDII adheres to the GLUE information model. Interfacing with BDII is made of ldap operations for which commands and API exist. Both LCG-2 and gLite currently rely on BDII for proper operation.

Logging and Bookkeeping The Logging & Bookkeeping services (LB), which tracks jobs during their lifetime in term of events (important points of job life, such as submission, starting execution, etc.) gathered from the WM's and the CE's (they are instrumented with LB calls). The events are first passed to a local logger then to bookkeeping servers. More information on the Logging and Bookkeeping services are available at Ref. [33].

At present time the project organization in charge for the LCG development is called EGI-Inspire [34] versus EGEE the one reported in the document.

Chapter 2

Project background, the SuperB experiment case

I'm giving here a short presentation of SuperB experiment in terms of computational needs and user community aspects. It will allows the reader to better understand the origin of the thesis project. In particular the needs of SuperB VO in terms of distributed computing tools will be described trying to characterize the target VO for our general-purpose framework.

2.1 The SuperB experiment

The SuperB Project was formally born in 2005 when INFN inserted in its three-years planning document the intention of building a high luminosity [35] flavour particle factory in Italy. In the course of the years SuperB has evolved from an intention into a full-fledged project, with a Conceptual Design Report published in 2007, progress reports in 2009, and a formal collaboration structure setup in 2010 with hundreds of members from several countries. All aspects of the project, physics potential, accelerator ring design, detector design, successfully passed several international reviews. In 2010 SuperB was inserted in the Italian Research Ministry National Research Plan as Flagship Project, and a good fraction of the required funds were al-

Table 2.1: Computing resource estimate for the first 5 years of SuperB data taking.

	Year	**1**	**2**	**3**	**4**	**5**	
Luminosity	Peak	**0.25**	**0.7**	**1.0**	**1.0**	**1.0**	$10^{36} cm^{-2} s^{-1}$
	per Year	3.75	10.51	15.01	15.01	15.01	ab^{-1}
	integrated	3.75	14.26	29.26	44.27	59.28	ab^{-1}
Data Sets	Raw Data	39.4	149.7	307.3	464.9	622.4	PB
	Mini	1.2	4.5	9.2	14.0	18.7	PB
	Micro	2.8	10.6	21.8	33.0	44.1	PB
	User	0.4	1.3	2.8	4.2	5.6	PB
Storage	Disk	9.4	31.4	51.3	68.5	77.1	PB
	Tape	43.7	168.0	348.2	530.7	713.1	PB
CPU	Reconstr.	0.09	0.34	0.69	1.04	1.40	MHS06
	Simulation	0.28	1.06	2.18	3.30	4.42	MHS06
	Skimming	0.06	0.26	0.59	0.95	1.32	MHS06
	Analysis	0.11	0.43	0.88	1.33	1.78	MHS06
	Total	**0.54**	**2.09**	**4.34**	**6.63**	**8.91**	MHS06

located, although not the full amount. The decision to build SuperB on the land of the University of Rome Tor Vergata led, in 2011, to the formation of the Cabibbo Laboratory consortium [36] between INFN and TorVergata university, with the explicit mission of constructing and managing a new research infrastructure for flavour physics. A ministerial cost and schedule review of the accelerator project was held in fall 2012. A combination of a more realistic cost estimates and the unavailability of funds due of the global economic climate led to a formal cancellation of the project on Nov 27, 2012.

For a number of years, an Italian led, INFN hosted, collaboration of hundreds of scientists from Canada, Italy, Israel, France, Norway, Spain, Poland, UK, and USA has worked to the detector and accelerator design.

2.2 Computing requirements

During 5 years of data taking, the SuperB detector was expected to produce more than 500PB of raw data. To this, event reconstruction and Monte Carlo simulation [37] will add another 300PB.

Table 2.1 shows the computing resource estimate for the first 5 years of SuperB data taking. Integrated luminosity parameter is a measure of efficiency of a collider experiment as the higher the integrated luminosity, the more data is available to analyze. The data set and storage sizes included replication factors, contingency, on-disk fractions and support storage. Raw data was expected to be processed at the same rate at which they was produced, with no more than 48 hours of latency; the corresponding simulated data sets were produced within a month. As a consequence, the CPU required for these activities was proportional to the peak luminosity. The CPU required to reprocess the data collected in the previous years scaled with the integrated luminosity.

To cope with these large data volumes, SuperB relied on predictable progress in computing technology to provide cost reduction and performance increase. The effective exploitation of computing resources on the Grid has become well established in the LHC era, and would have been enable SuperB to access a huge pool of world wide distributed computing resources.

Crucial issues for SuperB were the ability to efficiently use modern CPU architectures, and to efficiently and reliably access very large amounts of data spread over multiple geographically distributed sites.

So far, the SuperB computing effort has been devoted to the development and support of simulation software tools and computing infrastructure needed to design and validate the detector, to the initial definition of a computing model, and to computing R&D.

The SuperB collaboration has developed a set of tools to perform fairly detailed and sophisticated detector and physics studies. This toolset included a detailed Monte Carlo simulation (Fullsim [38]) based on Geant4 [39], a fast parametric simulation (Fastsim [40]) which directly leverages the original analysis code base, and a production system that could exploit the computing resources available on the European and US Grids to perform very large-scale simulation productions.

2.2.1 Full Simulation

The core simulation software of FullSim has been re-written from scratch, aiming at having more freedom to better profit from both the Babar[1] [41] legacy and the experience gained in the development of the full simulation for the LHC experiments. Geant4 was the underlying technology and C++ the programming language.

In the following, are summarized the most important characteristics. The geometry description was based on GDML [42]. The event generators can be either be embedded into the simulation software or interfaced through an intermediate exchange format. The simulation output was saved in ROOT [43] files; hits from the different sub-detectors, which represent the simulated event as seen from the detector, was saved for further processing. A staged simulation has been implemented, where snapshots of particles taken at a specific detector boundaries, can be read back and used to start a new simulation process without the need of re-tracking particles through sub-detectors that sit at inner positions.

FullSim was also used to generate detailed background frames that could be used by FastSim executable by propagating those particle though the simplified detector geometry and overlaying the resulting hits to the ones coming from signal events.

The typical computation time was of the order of $100s$ per event.

2.2.2 Fast Simulation

The FastSim relied on simplified models of the detector geometry, materials, response, and reconstruction to achieve an event generation rate a few orders of magnitude faster than was possible with a Geant4-based detailed simulation, but with sufficient detail to allow realistic physics analyses. In

[1]Babar experiment can be considered the SuperB precursor. It collected physics data in the period 1999-2008. FullSim and FastSim codes have been written for the first time during the BaBar life-time

order to produce more reliable results, FastSim incorporated in the some measure the effects of expected machine and detector backgrounds. It was easily configurable, allowing different detector options to be selected at runtime, and was compatible with the BaBar analysis framework, allowing sophisticated analyses to be performed with minimal software development. In brief, the simulation proceeded through four main steps: particles generation, detector configuration and response, particles reconstruction and analysis of the event.

The typical computation time was of the order of $1s$ per event[2] when all background mixing was activated.

2.3 Distributed computing tools

Even at early stage in its lifetime, the SuperB experiment needed very large samples of Monte Carlo simulated events to evaluate the machine background, to optimize the detector design, and to estimate the physics analysis performances. Since producing these samples was beyond the capacity of a single computing farm, the SuperB collaboration developed a suite of tools to fully exploit the existing world wide Grid computing infrastructure. The SuperB distributed computing tools were built upon the LHC Computing Grid which is widely adopted in the High Energy Physics (HEP) community and has a long term support by the Grid initiatives. The SuperB Virtual Organization (VO) was enabled at several sites located in countries participating in the project; both Grid flavors, EGI and OSG were in use within the SuperB VO, so all services needed to be able to support multi-flavor Grid middlewares.

Figure 2.1 shows the main elements of the SuperB distributed computing system: actual processing and data movement were performed in the Simula-

[2]In particle physics, an event refers to the results just after a fundamental interaction took place between subatomic particles, occurring in a very short time span, at a well-localized region of space

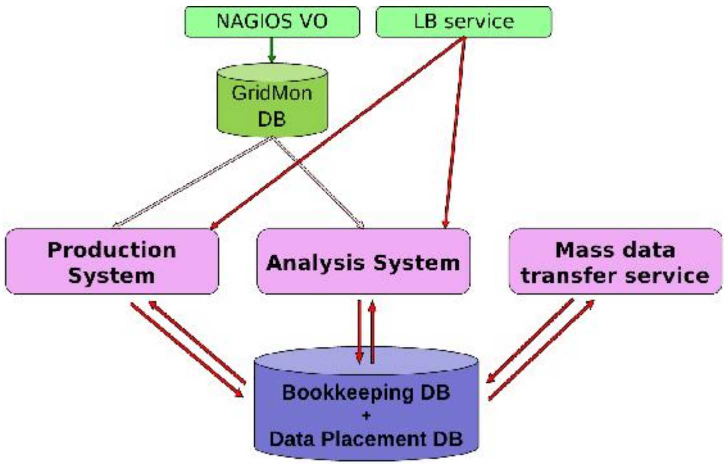


Figure 2.1: Distributed systems bird's-eye view.

tion Production System, the distributed Analysis System and the Mass data transfer service. The Bookkeeping and Data Placement databases holded metadata related to Fastsim and FullSim applications, and information about dataset structures and data placement on Grid resources. The monitoring system used the Nagios [44] tools suite to implement a Service Availability Monitoring (SAM) [45] to maintain a per-VO status of the Grid elements in the GridMon database. Finally, the Logging and Bookkeeping (LB) service provided detailed job status information from the Grid-internal processes.

The Simulation Production System, the monitoring tool-suite and the information systems just introduced are the proto-components of the general-purpose framework has been developed.

The Simulation Production System has been developed to manage the production of very large Monte Carlo samples in a distributed environment. It supported both Fastsim and FullSim, is tightly integrated with the bookkeeping database, permitting a fine tuning of operational tasks via web portal.

The web interface provided basic monitoring features by means of querying the bookkeeping database and interacting with the Logging and Bookkeeping gLite service. The user could retrieve the list of jobs as a function of their unique identifier (or range of them), their specific parameters, the execution site, status, and so forth. The monitor section provided a per job list of output files and a direct access to the corresponding log files. The output file size, execution time, computing resource load status, and the list of the last finished jobs (successfully or with failures) were also provided.

A simple authentication and authorization layer, based on an LDAP [46] directory service allowed to apply a user-role based policy to access the system.

2.3.1 Distributed computing resources

The LHC Computing Grid (LCG) architecture was adopted to provide the minimum set of services and applications upon which the SuperB distributed production system has been built. Authentication and authorization is provided by VOMS service, LFC is the file catalog, WMS was used for brokering purpose and for Grid flavor interoperability features, transfers were performed via Lcg-Utility, gLite CLI was used for job submission tasks.

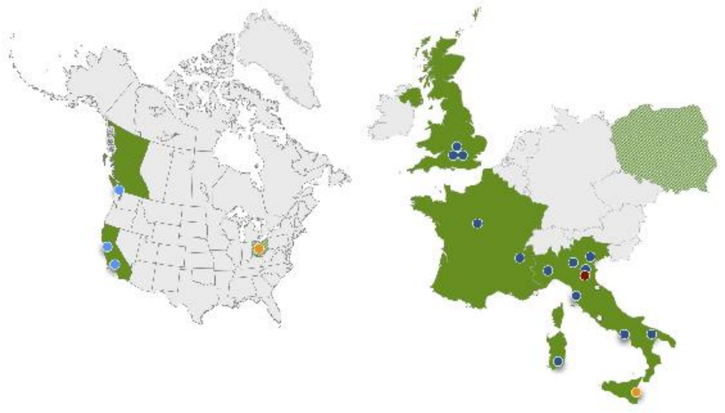
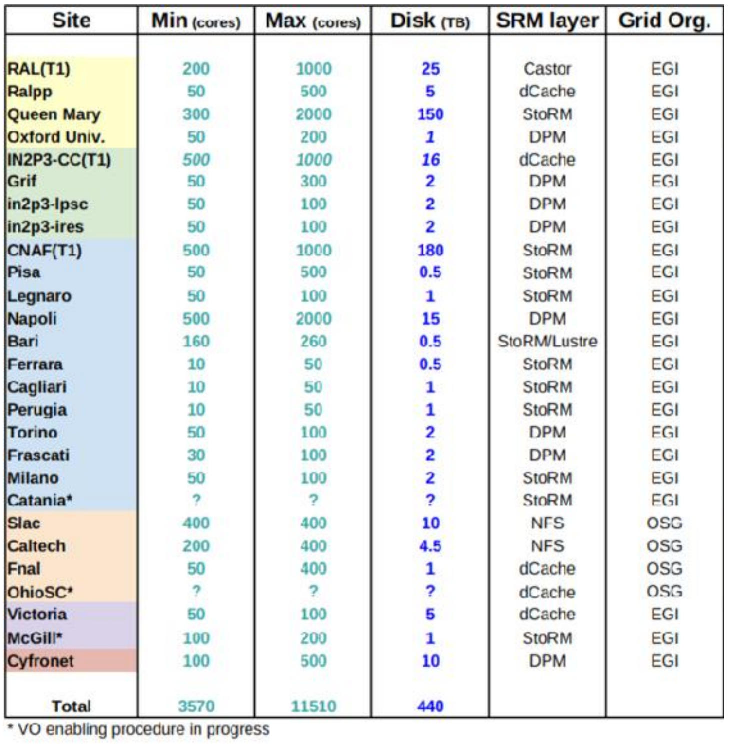


Figure 2.2: SuperB VO computing resources: the enabled LCG data centers.

The SuperB distributed computing infrastructure included several sites in Europe and North America, see Fig. 2.2. About thirty sites have been enabled for the SuperB Virtual Organization, among which four were Tier1 and 15 Tier2.EGI and OSG Grid flavour resources were available at the time of the simulation production system development.



Site	Min (cores)	Max (cores)	Disk (TB)	SRM layer	Grid Org.
RAL(T1)	200	1000	25	Castor	EGI
Ralpp	50	500	5	dCache	EGI
Queen Mary	300	2000	150	StoRM	EGI
Oxford Univ.	50	200	1	DPM	EGI
IN2P3-CC(T1)	500	1000	16	dCache	EGI
Grif	50	300	2	DPM	EGI
in2p3-lpsc	50	100	2	DPM	EGI
in2p3-ires	50	100	2	DPM	EGI
CNAF(T1)	500	1000	180	StoRM	EGI
Pisa	50	500	0.5	StoRM	EGI
Legnaro	50	100	1	StoRM	EGI
Napoli	500	2000	15	DPM	EGI
Bari	160	260	0.5	StoRM/Lustre	EGI
Ferrara	10	50	0.5	StoRM	EGI
Cagliari	10	50	1	StoRM	EGI
Perugia	10	50	1	StoRM	EGI
Torino	50	100	2	DPM	EGI
Frascati	30	100	2	DPM	EGI
Milano	50	100	2	StoRM	EGI
Catania*	?	?	?	StoRM	EGI
Slac	400	400	10	NFS	OSG
Caltech	200	400	4.5	NFS	OSG
Fnal	50	400	1	dCache	OSG
OhioSC*	?	?	?	dCache	OSG
Victoria	50	100	5	dCache	EGI
McGill*	100	200	1	StoRM	EGI
Cyfronet	100	500	10	DPM	EGI
Total	3570	11510	440		

* VO enabling procedure in progress

Figure 2.3: SuperB VO computing resources: the enabled LCG data centers.

2.3.2 The simulation production system

The design of the simulation production suite foresaw CNAF Tier1 as central service site including the unique submission point, the production tools, the bookkeeping database and the Grid service permitting authentication, file cataloging and data management. The jobs were submitted using Ganga [47] suite to all the remote sites via WMS brokering. It was duty of job during running stage to update bookkeeping DB and manage for input accessing at closest SE. At job completion time the output files were transferred to the central and unique target site, CNAF; all the data handling operations were performed via LCG Utilities making use of LFC file catalogue service. A detailed description of the workflow is illustrated in Fig. 2.4.

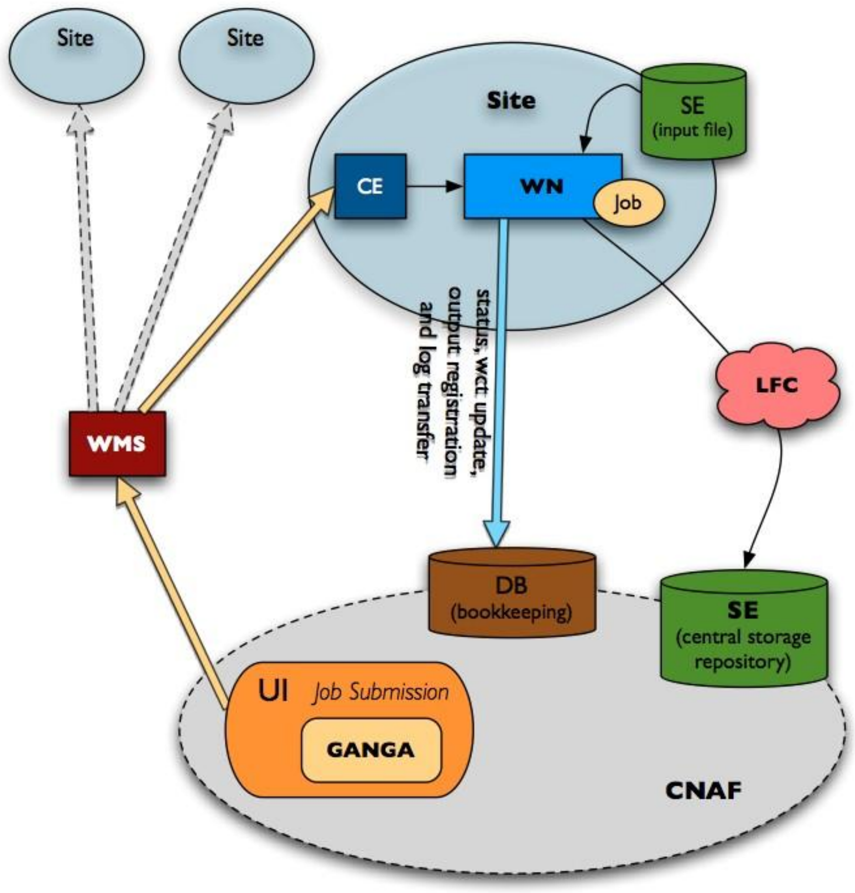

Site
Site
Site
SE
(input file)
CE
WN
Job
status, wct update, output registration and log transfer
LFC
WMS
DB
(bookkeeping)
SE
(central storage repository)
UI
Job Submission
GANGA
CNAF

Figure 2.4: The site centralized architecture of SuperB production system

The job workflow included also procedures for correctness check, monitoring, data handling and bookkeeping metadata communication. A replication mechanism allowed to store the job output to the local site SE (CNAF). The job input data management included an off production step: the application software release and background files were transferred to all involved sites and accessed by the jobs at running time. The job submission procedure included a per site customization to adapt the job actions to site peculiarities: e.g. file transfer from/to different data handling systems StoRM [48], dCache, DPM and Hadoop [49].

The figure 2.5 shows from left to right the interactions between the web portal and the job wrapper with the bookkeeping DB. The yellow boxes represent the actions taken by the web portal at job definition and completion time.

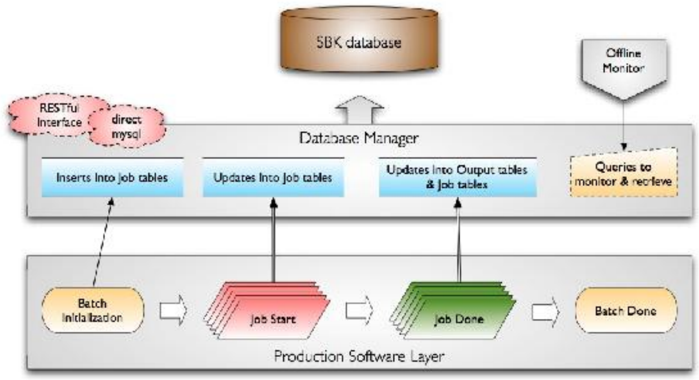


Figure 2.5: Bookkeeping DB interactions during the job life cycle

Chapter 3

A general-purpose framework for Grid resource exploitation

The general-purpose suite presented here is the evolution of the one developed during the SuperB experience. The components of the project that more then other have been refactored are the bookkeeping data base, the web user interface (Operations web portal) and the job wrapper application. In fact the new platform has to be seamlessly customizable by VO specific requirements in terms of application environment and parameters, metadata structure and distributed resources to be involved in computation campaign.

We can now take as reference for the VO that can benefit from the framework adoption the SuperB VO. The storage resource needs, computational power needs, the community size, members geographical distribution and application task to be run on computing infrastructure in SuperB VO can be taken as upper limit for the definition of the target VO for the tool suite we are describing.

The effort in developing a distributed simulation production system for SuperB experiment has brought to the study and the development of a general purpose design based on minimal and standard set of Grid services, capable to fit the requirements of many different Virtual Organizations.

The generalization process applied to the SuperB framework includes the following main components:

- design and implementation of a new bookkeeping DB structured to allow the management of generic computation process
- development of a new web portal for operation management which is capable to fit dynamically the DB information
- development of a job wrapper able to manage the new requirements in terms of flexibility and extensibility

The definition of the ER schema for the new DB and the development of the new web user interface (from now on the operations web portal) proceeded in parallel with the rewriting of the job wrapper software.

In the following sections we will deepen the design of the new database, we will see an overview of the functioning of the job wrapper and analyze in depth the new web portal for the operations management, from now on the operations web portal.

3.1 Project goal, conceptual design and requirements

The project described in this thesis refers, as background computational environment, to the science Grid and more in specific to the LHC Computing Grid. The project intends to provide an all-in-one tool suite permitting the exploitation of distributed resources to small-/mid-size VO such as experiments, organizations, or communities in general. Such a tool suite should allow an easy, quick and highly customizable access to the Grid resources while keeping the technical details hidden to the client organization. The underlying idea is to minimize the platform requirements in terms of hardware and human resources, development efforts and Grid service customization and configuration. The project was designed to manage client applications built-up to perform Parallel, Embarrassingly parallel (EP), Coarse-grained (often EP) calculations. The design has been kept light and it is based on a minimum set of standard Grid services.

The operations web portal provides a session section dedicated to VO specific task definition. The session interface compilation should be the first act of the VO manager permitting to customize its specific work environment; this step will brought the framework to be adapted to the VO specification. More in detail it includes the setup/configuration of job run time environment, VO application parameters, input and output dataset and distributed resources. The following list summarizes the technical framework requirements:

- The Grid initiatives that can host the framework are EGI and all the ones compatible with EGI middleware: Open Science Grid(OSG), West Grid, Nordu Grid, Latin American Grid initiative (GISELA), other
- The user VO need to be defined and enabled at the involved Grid sites
- Jobs should run where the data to be accessed reside (data driven paradigm)
- Job input files should be of the order of 10GB, Job output files should be lesser then of 3GB size each, maximum Job RAM consumption has been tested is 1GB
- A gLite User Interface should host the framework suite including the backend sub-systems: Ganga, PostgreSQL9, Apache, Lcg-Util library and CLI (see next sections for details)
- At least 1 Storage Element should be available to store jobs result
- At least 1 Storage Element should be available for job input retrieval per site

3.2 Distributed infrastructure: LCG services integration

The system design includes a main EGI site hosting the job submission manager, the bookkeeping database and the operations web portal, all residing in a User Interface Grid element. Jobs submitted to remote sites transfer their output to the Storage Element of a predefined target site and update the bookkeeping database. Each site may implement different Grid flavors. One of the main problem interfering with the Grid concept itself regards the cross Grids interoperability: many steps forward have been done toward a solution and nowadays the choice of using the EGI Workload Management System (WMS) permits to transparently manage the jobs life through the different Grid middlewares. The Grid services involved in system work-flow resulted to be widely adopted and long-term supported by Grid initiatives, moreover a particular attention has been given in identifying the multi-flavor Grid services compliance. A briefly description follows:

- Job brokering service: the Workload Manager System (WMS) in addition to the job brokering specific tasks, manages jobs across different Grid infrastructures (OSG, EGI, NorduGrid, WestGrid, etc), performs job routing, bulk submission, retry policy, job dependency structures (Direct acyclic graph DAG job), etc.

- Authentication and accounting system: Virtual Organization Membership System (VOMS) is a service developed to solve the problems of granting users authorization to access the resources at the Virtual Organization level, providing support for group membership and roles.

- File meta-data catalog: the LCG File Catalog (LFC) is a catalog containing logical to physical file mappings. In the LFC, a given file is represented by a Grid Unique Identifier (GUID). Data handling: LCG-Utils permits to perform data handling tasks in a fully LFC/SRMV2 compliant solution.

- Job management system: GANGA is an easy-to-use front-end for job definition and submission management implemented in Python. It provides interfaces for different backends (Local Resource Management System (LRMS) like LSF or PBS, gLite, Condor [50], UNICORE [51], Dirac, etc) and includes a light job monitor system with a user-friendly interface. The framework has been configured to use LCG back-end, cross-compatibility among different Grid-middleware is guaranteed by the WMS service.

- Storage resource manager: SRM provides data management capabilities in a Grid environment to share, access and transfer data among heterogeneous and geographically distributed data centres. StoRM, dCache, DPM, Lustre [52] and Hadoop are some implementations in use by the remote sites involved in the production distributed system deployment at present time.

3.3 Workload management system

3.3.1 Job workflow

The structure of services and job workflow follow a semi-centralized design, as shown in figure 3.1: job management service, bookkeeping database and default storage repository are hosted in a central site. Jobs executed into remote sites update the bookkeeping database with status, logging and timing information and transfer their output back to central repository or to a predefined site, discriminating on execution metadata. The submission model is based on direct job brokering on computational resource without the use of pilot model (see Chapter 4). The resource availability monitor permits the system to individuate free resources and efficient sites since the adoption of multiple algorithms: Nagios service for service availability monitor, effective site banning procedure based on last job bunch efficiency per site. The level of simplicity the system and use cases can maintain assured very good whole performance in real scenario, see Chapter 5. Job stage-in and stage-out[1] have been managed via pure LCG-Utility tool suite; SRMV2 protocol is at the base of all the file access tasks, a fail-over command chain and retry mechanism guarantee a high efficiency in transfers operations, see 3.5.

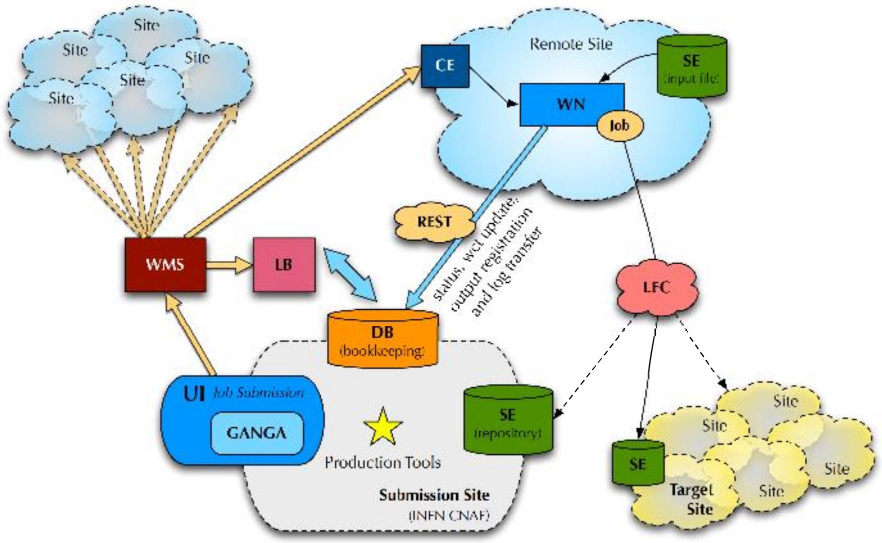


Figure 3.1: Job life-cycle schema.

The job work-flow shown in figure 3.1 is synthesized in the following three steps:

- Pre-submission steps: the VO managers should install the VO specific software all over the enabled Grid sites into the official software area[2]. The job input files that need to be accessed during stage-in phase by the job should be transferred to the site's SEs, these transfers should be performed via ClientSRM or LCG-Utils commands taking care the files will be registered into the LFC catalog using replication model.

[1]The stage-in and stage-out phases are respectively the action of application input files retrieving on the WN and the application output files transfer to the target site SE

[2]CVMFS [53] system provide a unique storage area exported all over the sites with the scope of collecting the VO software areas

- Job preparation and submission: operations web portal provides the interfaces for job preparation in terms of definition of specific session setup, executable parameters and distributed resources selection. The portal interacts with LCG resources with an official VOMS proxy identity, it allows job submission and monitor tasks to be performed in complete integration with proper Grid environmentSubmission type is limited to bulk approach and it is managed by a Ganga engine specifically configured for such a task.

- Job running time: Ganga engine submits the job to the WMS service capable to route it to the requested CE and takes care of Grid middlware interoperability. The job will be scheduled on a WN by the Computing Element[3] and starts running. It performs a general environment check, subsequently performs the transfer of input files from local SE to the WN disk area reserved for it's own computation, launches the VO specific executable and finally sends back to target site SE the generated output and log files. During the entire life cycle the job communicates its own status updates to the database via REST [56] based web service. Input and output file transfers are performed by LCG-Utilities commands allowing file registration to the catalog service, LFC.

3.3.2 Grid job submission via Ganga system

Ganga [54] is a user interface which supports large-scale data analysis in both distributed environments and local clusters. Ganga software architecture has been designed to be modular, plug-able and in general adaptable to fulfill the VO requirements. The software implementation is object based and the used programming language is Python.

As it is shown in figure 3.2 Ganga provides various interfaces (plug-in)

[3]The CE provide the submission request to the LRMS which performs the scheduling on site computational resources

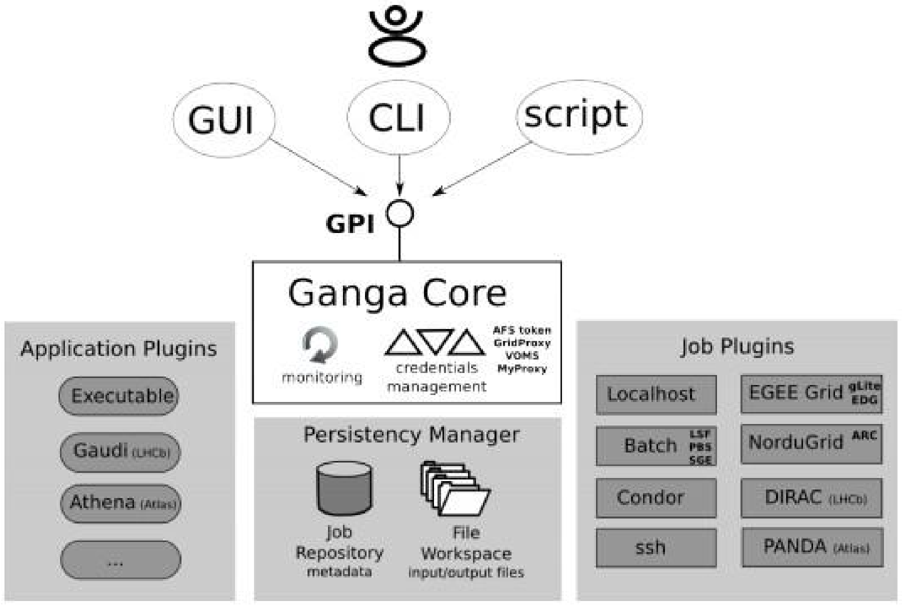


Figure 3.2: Ganga architecture schema: user can interact with the Ganga Public Interface (GPI) via Ganga User Interface (GUI) or command line interface using iPython [55] language or Python scripting procedure.

allowing job submission to a very comprehensive set of environments: jobs can be executed on user local host, via common LRMSs (LSF, PBS, SGE, Condor, other), to the main Grid middlwares (gLite, ARC and OSG) and act as bridge to application environment specifically developed in HEP scenario like Dirac and Panda. Moreover Ganga is able to maintain a state-full picture of the jobs submitted by the user, in fact the job repository and File Workspace permit the Ganga system to keep a persistent and historical job metadata record in terms of job characteristics and i/o operations. Advanced features like automatic job resubmission can be performed thanks to this repository implementation. Finally Ganga system provides a fruitful job definition/debugging environment taking advantage of the python native capabilities. Grid computational resources have been accessed via gLite suite with the use of WMS when the target site is not the local, otherwise Ganga is able to identify the specific batch system (LRMS) and submit directly to it.

The brokering system permits to distinguish CREAM CEs [57] from LCG2 CEs. The WMS functionality include migration of stuck jobs to new, JDL defined, sites and in general to access Logging and Bookkeeping job historical information. This will increase brokering quality. The framework we are discussing here will take advantage of these new features just maintaining Ganga submission engine up to date as Ganga project is tightly integrated with LCG middleware developing and releasing process.

The job submission management is delegated entirely to Ganga system. Various studies and configuration tests have been performed with the aim of customize Ganga system to be able to work as a simple and efficient submission manager. The lines of intervention can be summarized in the following main groups:

- sub services clean-up procedure: the deactivation of all the services around the core submission routine (job monitoring daemons, user interactive interface, job specific feedbacks, integrity checks, automatic Ganga specific resubmission policy)
- bulk as unique active submission method: specialization in bulk submission procedure included decreasing of Ganga submission response time
- Grid job specific information collection: Grid job Id is an example of information the bookkeeping DB need to store, it can be retrieved from the submission process itself

Ganga developer team expressed interests in this use case and an active collaboration on the specific subject started. A Ganga specific script has been developed to allow run time customization and site specific JDL file generation.

The results of the use of Ganga system in this particular role have been optimal in terms of submission reliability and robustness: a negligible failure rate in submission operations has been registered during stress tests and real case simulation job submission.

A typical bulk job submission has a size range since 50 to 500 single jobs per site; each job is identified by a run number (run-number) so the bulk job is identified by the first run-number and last run-number. For each set of VO parameters characterizing a set of submissions (called request) and for each target grids site, the job wrapper receives as input a specific configuration file and the run-numbers included in the bulk submission range(see sction 3.4 for a detailed description). Ganga provides the application programming interface (API) allowing the fine-grained control of job submission in terms of input/output sandbox management, data handling and submission methods[4]. A python script has been developed to perform the job wrapper submissions, it receives as input the following parameters:

1. path of job wrapper configuration file generated by the operations web portal
2. the configuration file name
3. minimum run-number
4. maximum run-number
5. grid site name
6. the name of the application production: a reference of the application and its parameters (see section 3.6)
7. the name of the session: a reference to the VO specific requirements (see section 3.6)
8. the type of job: test or normal

The script implements a loop on run-number using the script parameters (3) and (4), the script code is reported in the following. The loop at line 25 configures the submission: the i-th job takes as parameters the configuration file, the run-number and the job type (normal or test). During the loop cycle

[4]Ganga provides the same level of expression as the Job Definition Language (JDL) provides

Ganga allows to select few other additional parameter as the computational backend like localhost, LCG, Dirac, LFC, etc. At submission task completion the script waits for each Grid job id in return from each job submission; as last action the script runs an UPDATE query to the bookkeeping DB providing the Grid job id.

This is the python script has been developed performing the job submission via Ganga:

```
#!/usr/bin/envpython
import sys
import os
import time
import commands
import string

from GangaTest.Framework.utils import sleep_until_state
from Ganga.Utility.Config import makeConfig, getConfig

conf_path=sys.argv[1]#pathofconfigurationfileforseverus
conf_file=sys.argv[2]#configurationfileforseverus
minrun=sys.argv[3]#minimumrunnumber
maxrun=sys.argv[4]#maximumrunnumber
site=sys.argv[5]#sitename
prod_series=sys.argv[6]#productionseries
session_name=sys.argv[7]#sessionname
mode=sys.argv[8]#jobtype

if (len(sys.argv)<8):
  print "usage: base_ganga.gpi <conf_path> <file.conf> <
      min_run> <max_run> <site> <prod_series> <session_name>
       <mode>"
  sys.exit()

split=ArgSplitter()
for i in range(int(minrun), int(maxrun)+1):
  split.args.append([conf_file,str(i),mode])
  #first argument: file.conf, second argument: runnumecome
  #third argument: the mode(normal or test)

#Merge the output of the collection
m=TextMerger()
m.files=['stdout.gz','stderr.gz']
m.ignorefailed=True
#Createthejob
j=Job ()
j.name="%s  %s  %s " % (prod_series, str(minrun), str(
   maxrun))
j.application=Executable(exe=File( '/data1/script_webui/
   severus/exe.sh'))
j.inputsandbox = [File( '/data1/script_webui/severus/
   severus.tgz' ), File(conf_path + ' / ' + conf_file)]
j.splitter = split
j.backend = "LCG"

if site==" GRIF":
  j.backend.requirements.other=['other.GlueCEUniqueId == "
     grid36.lal.in2p3.fr:8443/creampbssuperbvo.org"']

elif site==" UKILT2QMUL":
  j.backend.requirements.other=['other.GlueCEUniqueId == "
     ce03.esc.qmul.ac.uk:2119/jobmanagerlcgsgelcg_long"']

elif site=="INFNT1":
  j.backend.requirements.other=['RegExp("cr.cnaf.infn.it",
     other.GlueCEUniqueId ) && !(RegExp("ce04lcg",other.
     GlueCEUniqueId))']

elif ...

else:
  print " \n\n SITE NAME NOT RECOGNIZED, CHECK THE SITE
     LIST IN base_ganga.gpi\n\n"
  jobs.clean(confirm=True,force=True)
  sys.exit()
```

```

j.merger=m
j.submit()

print "\n\n WAITING FOR GRID JOB IDs\n\n"
for a in j.subjobs:
  while a.backend.id== ' ':
    runMonitoring(jobs=jobs.select(j.id,j.id))
    time.sleep(1)
    print "JOB"
    print a.application.args[0]
    print a.backend.id
    whoami=commands.getoutput("whoami")
    cmd_db=" /usr/bin/mysql -usbkprodw -psbkFromFe -host=
       bbrserv09.cr.cnaf.infn.it --database=sbk4 e \" start
        transaction ; UPDATE %s _J ob SET grid_ job_id=\'%s
        \' WHERE prod_series=\'%s \ ' AND runnum=%s ;
       UPDATE %s_Job SET status =\'submitted \ ' ,
       ts_submitted=NOW( ) WHERE prod_series=\'%s \ ' AND
       runnum=%s and status =\' prepared \ ' ; commit ; \"
       " % (session_name,a.backend.id, prod_series, a.
       application.args[1],session_name, prod_series, a.
       application.args[1])

    print cmd_mysql
    out_mysql=commands.getoutput(cmd_mysql)
    print out_mysql

print "End Ganga"
```

Listing 3.1: Ganga script for job wrapper submission using LCG backend

3.4 Metadata management

3.4.1 The bookkeeping database

Both the job submission system and the individual user require a way to identify data files of interest and to locate the storage system element holding them. Moreover the prompt availability of information on the execution status of jobs and their specific meaning and parameters is crucial to the users in order to plan their activities and summarize the results. To make this possible, the developed framework needs a data bookkeeping system to store the semantic information associated to data files and keep track of the relation-ships between executed jobs and their parameters and outputs. This same bookkeeping database is extensively used by the job management system itself in order to schedule subsequent submissions and bring completion level of requests and site availability information up to date. The bookkeeping database was modeled according to the general requirements of a typical simulation production application; its design is sufficiently general to accommodate several use cases from many fields of science although being self-consistent and structured at the same time. Moreover, the schema can be easily extended in order to take into consideration new applications specificity, nevertheless by keeping core functionality unaffected. At the moment, its design adheres to the relational model and the current implementation makes use of PostgreSql rDBMS in a centralized way.

As discussed in the next section 3.5, the bookkeeping database needs to interact either with the operations web portal or the job in execution on the Worker Nodes. Depending on the sender/receiver these communications are therefore managed by a direct interface to PostgreSQL or a REST interface. The latter case is required from remote sites because typically only outbound communication over http/https is allowed. Strong authentication, by means of X509 proxy certificates over https, is used to grant jobs access to the database. It is important to stress that such an intensive use of the bookkeeping database by our framework is crucial and permits to distinguish it from others portal-like solutions available to the Grid communities.

The bookkeeping DB developed within the SuperB life time has been completely re factored introducing a level of abstraction called "session" and cleaning-up all the SuperB customizations. With respect to the SuperB use cases, the new DB has to include two, dynamically defined, sessions, the FastSim and the FullSim. The Entity-Relationship schema of the old SuperB DB is reported in figure 3.3; the schema can be divided in three subset of entities: the ones related to FastSim, the ones related to FullSim and the ones modeling the functions shared by the two sessions.

Analyzing the FastSim and FullSim entities we can find a symmetric schema, but for one entity of difference in FullSim sub-schema. The shared entities between the two use cases are: Job, Log, Output, Production, Soft_Ref. The entities structure is the same, but for few fields in the Job one.

Said that, we try to lead back the Fast and Full use cases in a unique set of entities which could model a general-purpose job production. The common characteristics identifiable between the two use cases are:

- the existence of an application to be executed in a local or distributed computing resource environment.
- the VO activity organization is structured in macro set called "productions", see next section for a detailed description of the components of a job submission campaign modeled in the bookkeeping DB
- the goal of both the use cases is to keep track of job life cycle information, application execution and data management

Observing in particular the FastSim entities we can add other important elements for the generalization process:

- the "request" entity represents single queries within the same production in terms of size and kind of application output and application parameters (the description of the entire bookkeeping DB functional schema will be presented in the next section)

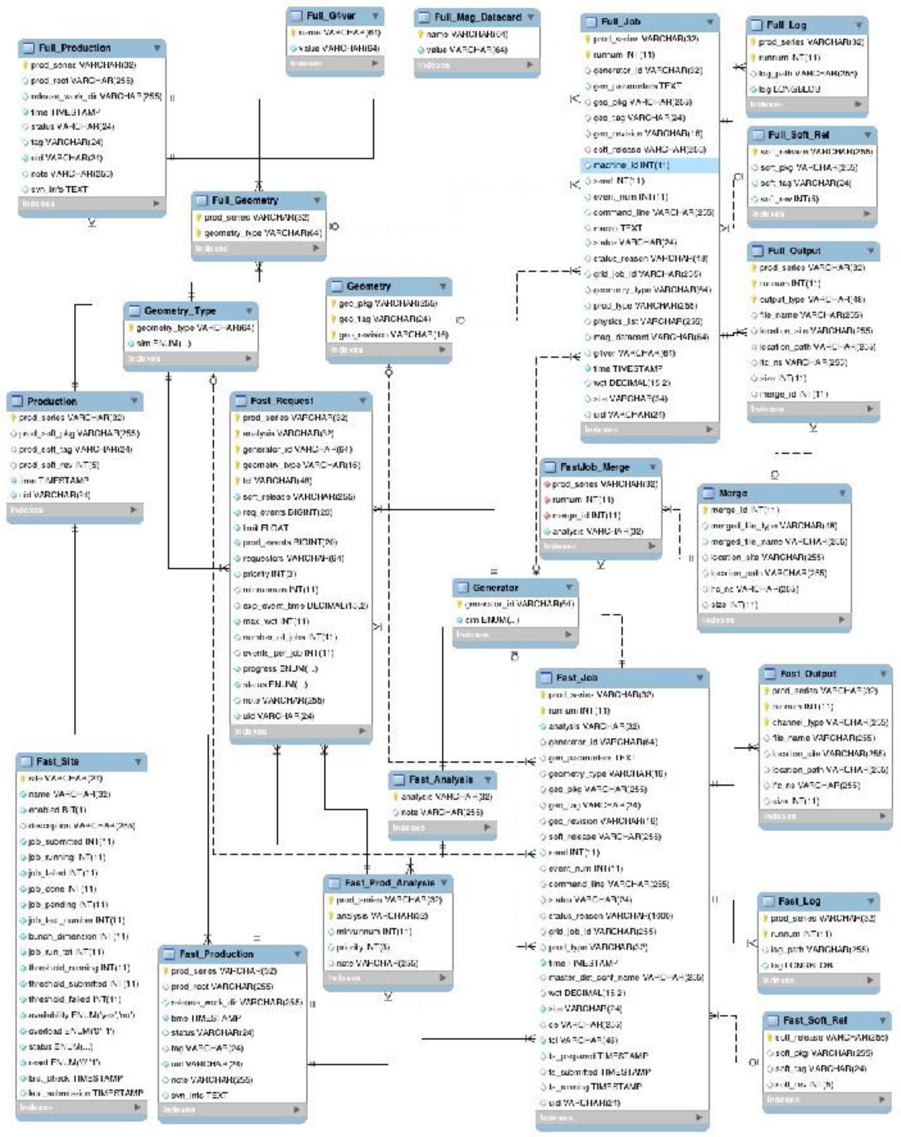


Figure 3.3: The model Entity-Reletionship for the SuperB bookkeeping DB.

- the presence of a parametric application
- a set of distributed computing resources along with load and availability calculation algorithm

3.4.2 The generalization step: the session concept

The VO manager switching on the operations web portal for the first time is asked to create a session and to define a set of Grid resources. These are the minimal set of information that the framework needs to be able to start a job submission campaign (hereinafter called: production). A session is an entity including all the information regarding a production in relationship with a specific application release and application environment setup.

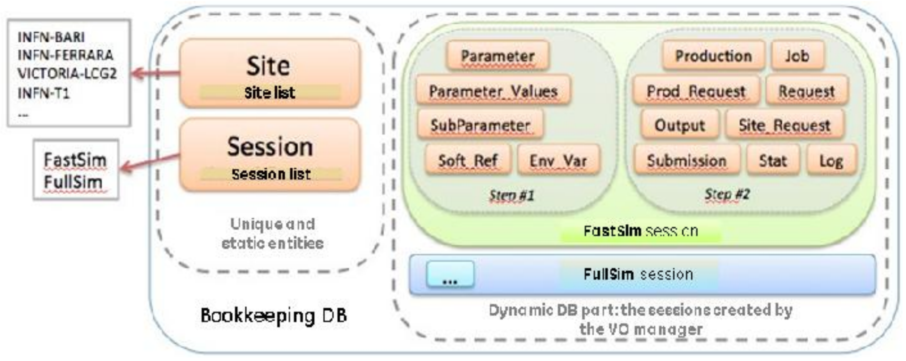


Figure 3.4: Representation of static and dynamic parts of bookkeeping DB: session entities on the right side are created dynamically in two steps by the VO manager using the operations web portal.

The figure 3.4 shows a conceptual representation of the bookkeeping DB divided in static and dynamic entities. The static entities contains all the information related to distributed computing resources (Site box) and information regarding the Session management, list of sessions and sessions metadata (Session box). The FastSim and FullSim SuperB use cases are two meaningful examples of sessions. The dynamic entities are the sessions themselves which contain application and production information; a session is divided in two sub boxes identifying the chronological order in which the information will be provided by the VO manager (in two steps).

Let's proceed with describing an overview of the entities modeling a job campaign in the Session container. In the figure 3.5 the boxes in the rows above contain the boxes underlying it. We can see a set of Production entities contained in a session, each Production identify a time period in which a massive job submission will be performed using an established application release.

A set or Request is contained in a Production. The Request concept has been introduced for the FastSim use case to define the required number of physics events to be simulated, the number of events per job and the number of jobs. These three information are tightly related one to the other and dependent by the job execution time.

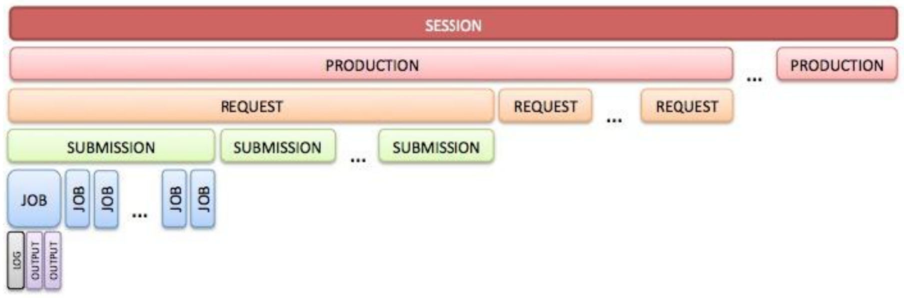


Figure 3.5: A representation of Session entities: the entities in the row above contain the entities in the row below.

In a generic VO scenario the event concept can be setup to be meaningful or not, in any case the number of jobs and the duration time per job will be defined besides all the information regarding job input and output files location and application parameters. VO manager can choose among different ways of access the input file by the job for the specific Request he/she is defining: the access can be of type NONE, DIR, FILE or LIST:

- NONE: the job do not need any input file, the input_path environment variable will be empty
- DIR: input_path contains the logical file name of the directory containing the input file; the job is in charge to select the right input file
- FILE input_path contains the logical file name of the input file

The following Logical File Name represent the template composition for an output file reference; subsequently the output LFN for the FastSim session. For the latter the application parameters PRODSCRIPT, DG and GENERATOR will take part of the LFN path:

- output_dir_lfn = lfn:/grid/VO_NAME/production/SESSION_ NAME/PROD_SERIES/output/RUNNUM/
- output_dir_lfn = lfn:/grid/superbvo.org/production/fastsim /first_cycle/output/PRODSCRIPT/DG/GENERATOR/RUNNUM/

The last two setup regarding the Request entity are the priority and the minimum run-number. The operations web portal, discussed in detail in the next session, can be configured to perform automatically the job submission taking care of the size of the job bulk, the submission target site, and the priority request to be referred for job definition. The latter choice is taken using the priority request parameter. Finally the minimum run-number is the starting integer that identify univocally the job in a request, can be used by the VO manager with a job classification purpose or as a meaningful parameter for the application itself: this is the case of the FastSim session, the run-number in the run-number range defined in the Requests is used as the seed for the Monte Carlo simulation algorithm.

The Production and the Request entities have a status and its management is delegated to the system policy or to the VO manager. The Jobs entities have been gathered in Submission entities. Each Submission is identified by the couple: submitter user id and submission timestamp. The ER schema for the entities Submission, Job, Output and Log will be reported and commented in the following section.

The entities Session and Site have a n:n relation-ship, it is represented in figure 3.6. The session entity has the following fields:

- session name

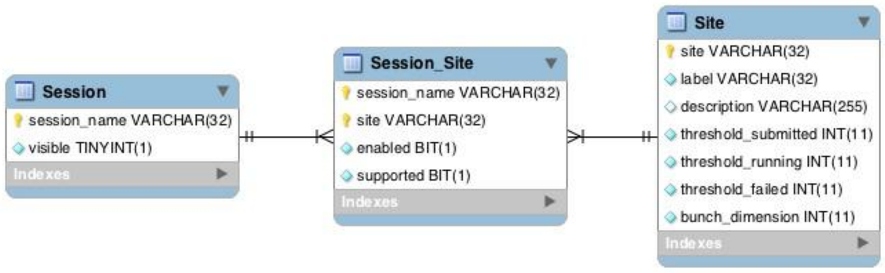


Figure 3.6: Session e Site: entities and relationship.

- visible: flag used to report the session access by the operations web portal. It is useful to hide obsolete sessions which is necessary to be kept inaccessible, but still have a value from the historical point of view.

The entity Session-Site have been collected information regarding the enabling/disabling of a site for a session:

- session name: name of the session
- site: name of the site
- enabled: flag used to declare site enablement for a certain session
- supported: flag declaring the application support availability at a certain site. It is used during the distributed computing resource setup phase and during the resources validation phase, vice-versa during a sustained production cycle the sites will stay in supported AND enables status or in disable status.

A session in the bookkeeping DB is composed by a fixed set of entities; let's give in table 3.1 the complete list with a description:

3.4.3 Job and submission modeling

The relationships and entities modeling the job and the submission concepts have been represented in figure 3.8. As we shortly reported in the

Table 3.1: Entities belonging by a Session in bookkeeping DB

Tables	**Description**
Soft_Ref	Application information: version, tag, release, logical file name, etc
Env_Var	Environment variables related to the execution task
Parameter	The execution parameters for the application in this session
Parameter_Values	The default values or the list of possible values for each parameter
SubParameter	Define the dependence link among parameters
Production	The list of production defined in the session
Prod_Site	The list of enabled site for a production
Request	Define a specific set of application parameters. A Production can contain more Requests
Prod_Request	List of Requests defined in a specific Production and other requirements
Site_Request	List of Sites in connection with a specific Request
Submission	List of Submission
Job	Contains information about every single submitted job in the session
Output	Every job can generate one or many output files
Log	For each job a log file is generated
Stat	Statistical information about job submitted

previous section the Job entities have been gathered in Submission entities. Each Submission is identified by the couple: submitter user id and submission timestamp. The Submission layer between the Request and the Job layers has been introduced to allow a high level of control on a unique, critical action that can involve several sites and an elevated number of identical jobs: during testing phase for resource validation purpose or for application validation purpose is very important to be able to manage such a level of job collection.

At submission time the record insertion in submission table is managed automatically by a trigger that intercept the BEFORE INSERT event for each job.

Let's now describe how the job status system has been conceived during the job life-cycle, since the job creation to the job completion time. The job status-flow schema is reported in figure 3.7.

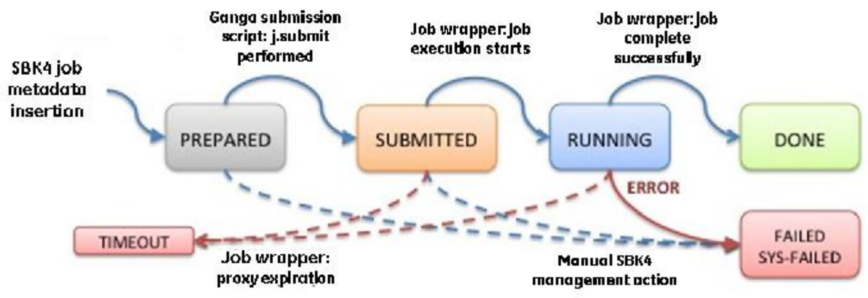

SBK4 job metadata insertion
Ganga submission script: j.submit performed
Job wrapper:Job execution starts
Job wrapper:Job complete successfully
PREPARED
SUBMITTED
RUNNING
DONE
ERROR
TIMEOUT
FAILED SYS-FAILED
Job wrapper: proxy expiration
Manual SBK4 management action

Figure 3.7: The graphical schema of job status during the job life cycle.

Every job in the bookkeeping DB could be in a status included in: PREPARED, SUBMITTED, RUNNING, DONE, FAILED, SYS-FAILED and TIMEOUT; allover the job status system a set of timestamp have been defined in Job entity:

- ts_prepared: the timestamp of creation/insertion of job record in bookkeeping DB, the job status is PREPARED, it is used with the user-id to identify univocally the related session
- ts_submitted: the timestamp at status update to SUBMITTED
- ts_running: the timestamp at status update to RUNNING
- time: the timestamp updated to the last job record modification, it reports the transition to DONE or FAILED status. It is used to calculate the job wall clock time (wct)

Other fields of relevance in job entity are:

- status_reason: it is used as a text buffer transferred by the job wrapper to the bookkeeping DB reporting information about the software execution. It is of particular interest for the production manager because it can receive important information about the progress of job during the running phase

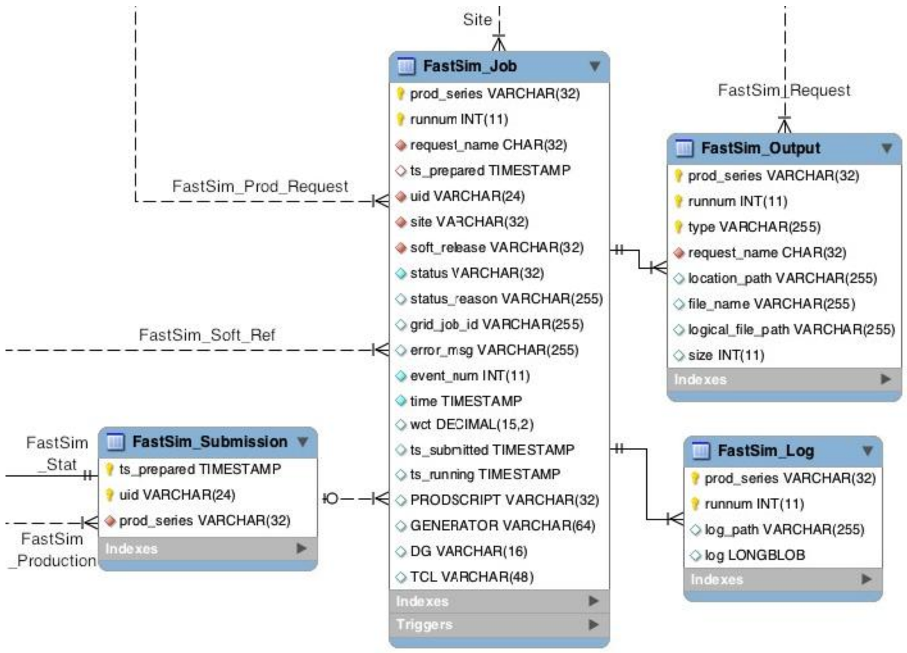


Figure 3.8: ER schema for job, submission, output and log entities.

- error_msg: provide the error message for the intercepted exceptions during the job wrapper execution
- grid_job_id: the unique string identifying the job in LCG context. It is provided as result of Ganga job submission

For what is regarding the Log and Output entities, the logical file names of the files have been stored in logical_file_path field for Output entity and in log_path for Log entity. For all the test jobs and the jobs selected by expert section in the web portal the log file content has been recorded entirely in the filed log. It allows a quicker and sure method to provide job execution information to the VO experts.

The session ER schema

The entire ER schema for the FastSim session has been reported in figure 3.9 as a representative example of a dynamically generated session.

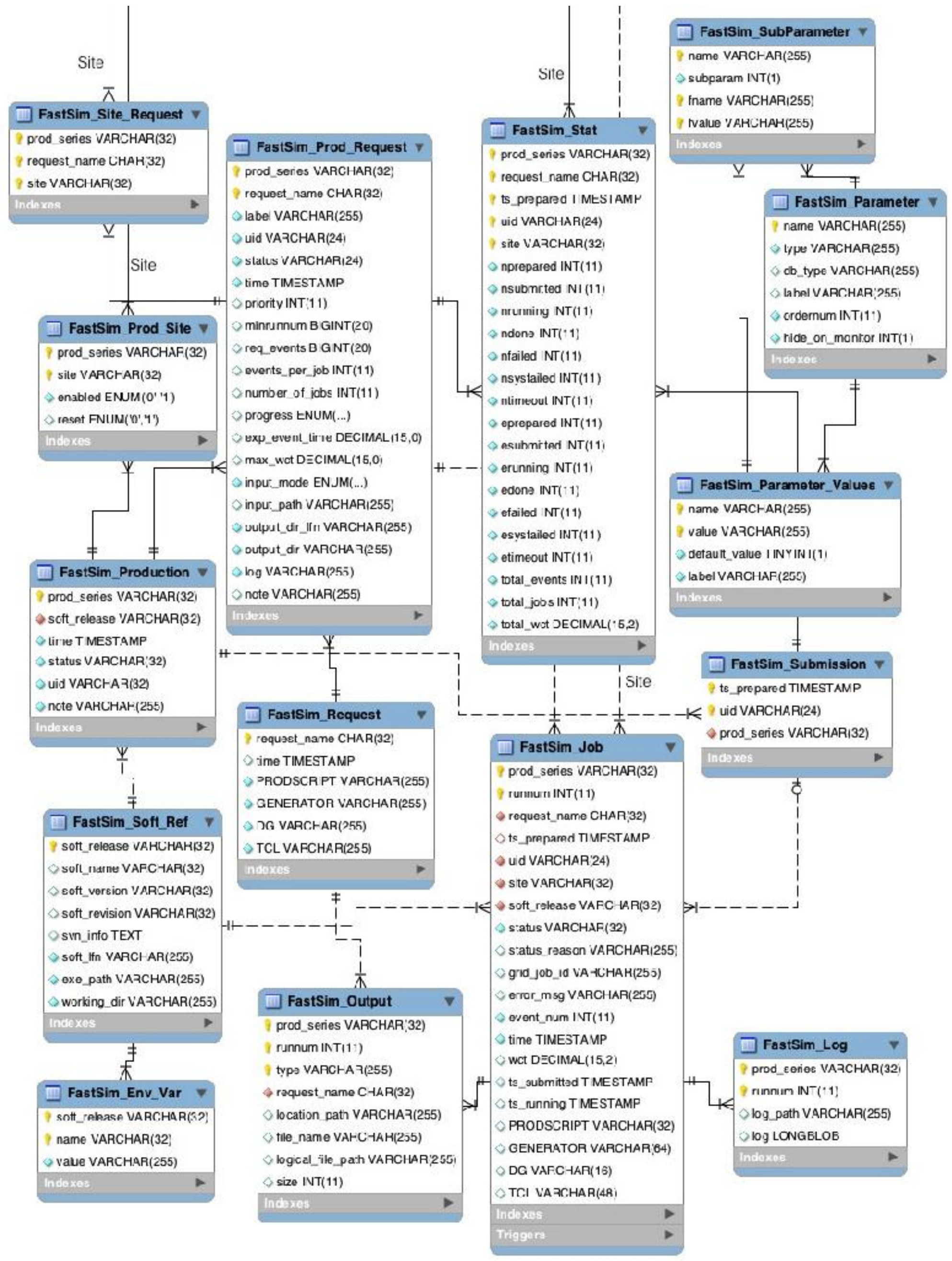


Figure 3.9: The FastSim session ER schema in new bookkeeping DB.

3.5 The job wrapper script

The development of a job wrapper that could run on Grid or local resources is justified by several reasons. A general-purpose project needs to be able to manage the widest set of scenarios in terms of:

- application setup: parameters and sub-parameters with complex dependence schema and environment setup (specific environment variables, scope environment variable like PATH and LD_LIBRARY_PATH, etc)
- input and output management: an heterogeneous set of transfer methods need to be implemented in response to the heterogeneous Grid site setup: direct access (network file system protocols), direct transfer via standard protocols or via Grid middleware typical methods (Lcg-utility)
- process monitoring capabilities: standard output and error general management and signal handling
- bookkeeping DB communication: REST methodology implementation

The first job wrapper version has been developed in Bash scripting language. The porting from Bash to Python language occurred as a solution for lack of software maintainability and flexibility. Python is an object oriented interpreted language fulfilling the project requirements in terms of facility in conducting bug-fixing operations, facility in code extension/evolution and robustness as the ability of managing failures (i.e.:exception management).

The job wrapper name is Severus, the following listed dot is the step-by-step workflow of the job wrapper:

- environment setup: to satisfy application needs and specific job requirements
- application software installation: transfer the package to the worker node hosting the job execution

- SE selection for input file retrieval: the nearest SE (in terms of Round Trip Time (RTT)) is identified
- transfer of input file from the SE to the WN where the job has been scheduled
- application launch
- job status update to RUNNING in bookkeeping DB
- at application completion time check the exit status
- copy of the log file to the target SE or to the bookkeeping DB depending on job type (test or normal)
- update the bookkeeping DB with the logical file name or the physical file name of the log file
- copy of the output files to the target SE
- update the bookkeeping DB with the logical file name or the physical file name of the output files
- job status update to DONE or FAILED in bookkeeping DB and application wall clock time estimation

3.5.1 Communications with bookkeeping DB

We decided to develop a communication layer between the job wrapper and the bookkeeping DB to provide the framework of on-premise monitor system. An evaluation phase has been performed to design the monitor architecture of the general purpose framework allowing to choose the simpler and quickly architecture to capture job life-cycle status and logs at run-time. The logging and bookkeeping service provided by LCG infrastructure have been subject of various measurements test and framework integration tests. We finally decided to collect status and logs information directly from the job wrapper. The framework uses these information both to populate the monitor section of the operations web portal and as information base for submission decision system (see section 3.6).

The communication layer implements the REpresentational State Transfer (REST) model. REST is a set of practices describing an architectural style for web service implementation; it has been chosen due to its characteristics: the simplicity, robustness and standardization (long-aged http protocol based). It could be considered a light weight alternative to RPC, SOAP, CORBA solutions, maintaining the same power of expressiveness. The set of architectural principles that REST defines among others are:

- the model application is client-server
- the functionality and the states of the service are represented by resources, each resource is identified by an URI. In web service design the resources in REST implementation are the methods in SOAP/CORBA/RPC implementation
- the client and server communications must be stateless: each communication has to include all the necessary information (context, parameters, data) to be understood with no need of counting on other information support increasing the system scalability and availability
- the application status must evolve as a flux of hyper link
- client and server interfaces need to be uniform in terms of representation of resources allowing an independent development process
- REST model encourage the use of cache mechanism to mitigate the loss of performance due to the stateless constraint
- the system can be stratified adding inter-layer services/devices between client and server (proxy, cache service, gateway, other). So to add new characteristics to the system like security, Quality of Service, scalability, High Availability, etc

The framework implementation of REST model regards the communications between job wrapper and bookkeeping DB; the used REST flavor is an implementation in PHP language for SQL resource management called PHP REST SQL [58]. The HTTP methods (GET, PUT, POST, DELETE) have been used on a resource identified by an URI, the following is an example of such an URI: http://server-host/table1/table2/field1/field2. The client side REST layer sends the request via an http-client implementation like curl or wget, the server side elaborates the request (method, URI) and obtain an SQL query to be execute at information system level.

All the communications between jobs and DB in the framework need to be authenticated and authorized. The LCG standards are based on x509 certificate in a Public Key Infrastructure (PKI) using the Secure Socket Layer (SSL) as communication channel for HTTP protocol. Every job in LCG is created with an x509 certificate with a limited expiration time to 12 hours called proxy certificate associated with the submitter user. For interoperability reason, such a certificate has been extended with VOMS user information obtaining, as side effect, the non-compliance with rfc standards. The "mediator" of all the authentication process in a web service scenario is the web server, in our case Apache web server has been used as web portal publisher and REST bookkeeping DB frontend. Unfortunately the proxy certificate as it is described above is not processable just with Apache mod_ssl module due to the non-compliance with rfc. The mod_gridsite [59] Apache module has been developed within LCG community to permit the fully comprehension of the proxy certificate by the Apache web server. It enables a native Grid Security Infrastructure(GSI) layer and permits to configure Grid Access Control List (GACL) [60] based on certificate VOMS extension.

The figure 3.10 shows the relation-ships among the Apache environment and Gridsite specific functionality. In particular the cooperation between mod_gridsite and mod_ssl allows the system to overcome the problem of certificate non-rfc compliance. The second layer from the bottom, the green one shown the mod_gridsite implementation of Grid Access Control list.

Said that the authentication and authorization functionality for the job-DB communication has been based on Grid Security Infrastructure that rely

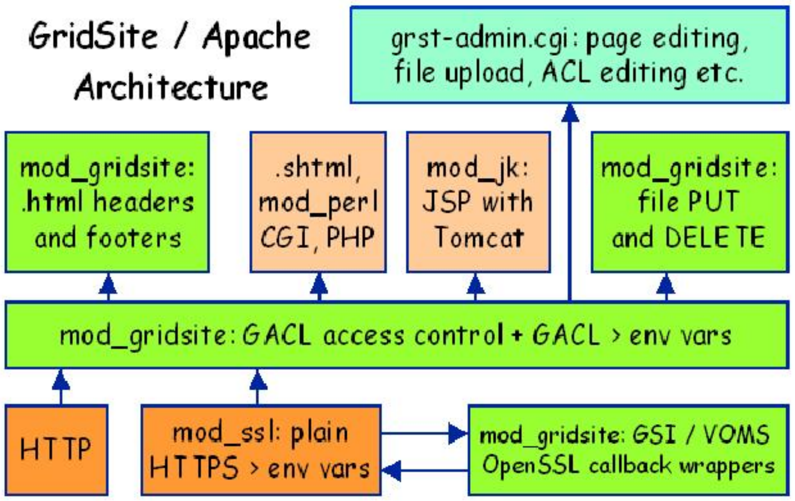

GridSite / Apache Architecture
grst-admin.cgi: page editing, file upload, ACL editing etc.
mod_gridsite: .html headers and footers
.shtml, mod_perl CGI, PHP
mod_jk: JSP with Tomcat
mod_gridsite: file PUT and DELETE
mod_gridsite: GACL access control + GACL > env vars
HTTP
mod_ssl: plain HTTPS > env vars
mod_gridsite: GSI / VOMS OpenSSL callback wrappers

Figure 3.10: Gridsite/Apache architecture.

on Apache ssl and gridsite modules for VOMS proxy certificate management and rely on Gridsite capability in implementing access control list natively on certificate VOMS extension. The following code is a real example of Grid ACL file used in the project:

```
<?xml version="1.0"?>
<gacl version="0.1.0">

<entry>
    <voms>
      <fqan>/superbvo.org/Role=ProductionManager/Capability=NULL
         </fqan>
    </voms>
    <allow>
      <read/><list/><write/>
    </allow>
</entry>

<entry>
    <voms>
```

```
      <fqan>/superbvo.org/Role=NULL/Capability=NULL</fqan>
    </voms>
    <allow>
      <read/><list/>
    </allow>
</entry>

</gacl>
```

Listing 3.2: Grid ACL file for REST communications between job and DB

The ProductionManager role has been granted to write,read and list access to DB resource, a NULL role authentication has been granted for read and list actions only.

3.5.2 Job wrapper algorithm description

Severus job wrapper is launched by the Ganga submission engine (see section 3.3.2) with the following input parameters:

- the configuration file path
- the job run-number
- the job type (normal or test)

Reporting here shortly the description of the configuration file content is of help for understanding the script execution steps. The configuration file is divided into macro sections, I'm reporting here a subset of such sections:

- OPTIONS contains the general parameters
  - Modulename: session name
  - Prodseries: production name
  - Requestname: the request including the submission for this job
  - Savelogdb: log file storing policy (into DB yes/no)

  - Test: job type (test or normal)

- SOFTWARE: contains all the parameters allowing the job wrapper to identify and transfer the correct version/revision and the executable path
- REST: contains the information related to layer REST like Hostname, Port and Proxy certificate path
- TARGETSITE: contains the information related to the Site and its Storage Element defined as the target for the application output files and logs
- INPUT: contains the input mode defined in the Request (see section 3.4.2) and the input path
- OUTPUT: contains the Logical File Name of the location for output storing task, the output path (the physical one (PFN)) to be used in the case the direct access by the WN is available
- $MODULENAME: takes the name by the content of the parameter MODULENAME in the macro section OPTION. It contains the application parameters. The number of parameters is variable

The figure 3.11 shown the Severus execution steps (from left to right) reporting the involved objects and files. The first performed action is the Logger object instantiation, it will take care to collect log information during job execution time. Once the configuration file has been parsed the following objects will be instantiated:

- the REST manager object
- the Execute manager object for the specific session
- the Site object, the correct object implementation will be selected depending by the configuration file information loading the class <sitename>.py

- the Session object, same as for the Site object loading the class <session>.py
- one ore more Targetsite object representing the site where the output files will be transferred

Now the object FileManager is initialized, such an object is on duty for file transfer operations, it implements the lcg-util functionality or eventually the direct file access wherever it is available. Hence Severus performs the transfer of application package (compressed archive) from the site SE to the job working area, it starts the transfer of input files. At this point the script is ready to launch the VO application, the method <sessionname>.execute() is launched accessing the application parameters from the SESSION section of the configuration file. At application completion time the FileManager will take care of the transfers of output and log files to the TARGETSITE. Throughout the script execution the job status is being updated in book-keeping DB via REST communication using the homonym object.

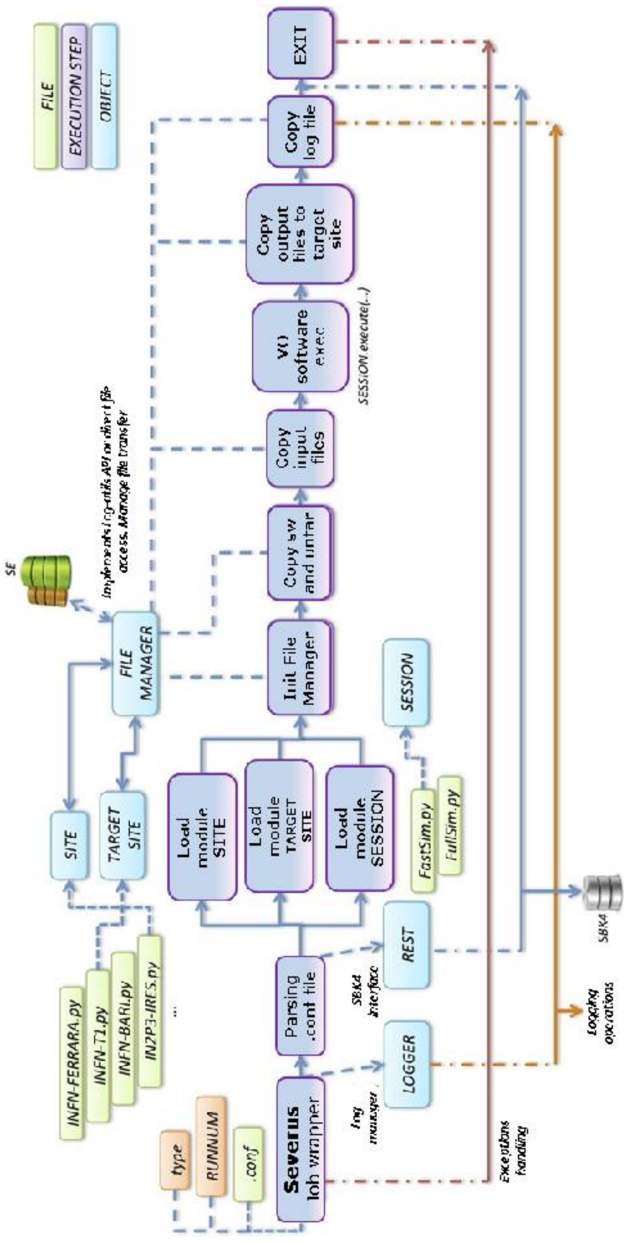


Figure 3.11: Block diagram for Severus job wrapper execution flow.

Filemanager class

As we described in the previous section the Filemanager object is on duty for file transfer tasks. It checks the transfer method (the one defined in the Request) and acts in respect of this. In the majority of the cases files have been defined as entry in LCG File Catalog (LFC) so we can refer to that via multiple identifiers:

- LFN - Logical File Name: define the file independently by its location
- GUID - Global Unique IDentifier: a unique string created at file generation time
- SURL - Storage URL: also known as physical file name (PNF) contains information about the SE or the SRM related to the file
- TURL - Transport URL: contains information about the transport layer like protocol and port besides SE hostname and absolute pats as in the SURL

The Filemanager class takes advantage of the LCG Utility suite[5] to handle the file movement operations to/from SE. Is of worth mentioning here, among the other implemented mechanisms, the data transfer fail-over strategy. It allows to minimize the transfer failures in the case of files registered in LFC.

In the diagram reported in figure 3.12 is shown the step-by-step decision process regulating the launch of lcg-utilities methods in various failures scenarios. The first run of lcg-util suite is the lcg-cr, it copies the file to/from

[5]"LCG Utils is a suite of client tools for data movement written for the LHC Computing Grid. The tools are based on the Grid File Access Library [61], which is also included. The tools allow users to copy files between CE (Compute Element), WN (Worker Node) and a SE (Storage Element) and to register entries in the file catalog and replicate files between SEs. Some commands use logical file names and require a connection to a BDII-based catalog; exceptions are file copies and deletions, which take endpoints based on the SRM URL. This document does not contain examples of commands that require access to file catalog. The usage of such command is explained in the LCG User Guide", citation from OSG twiki page, https://twiki.grid.iu.edu/bin/view/Documentation/Release3/LcgUtilities.

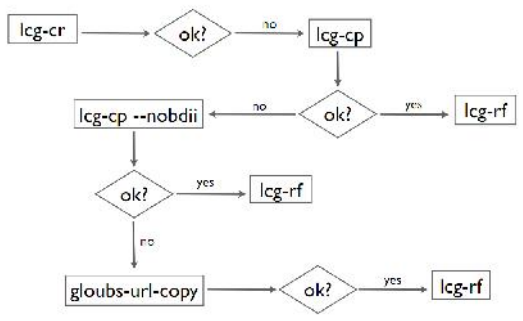


Figure 3.12: Fail-over flux diagram for file transfers to/from SEs.

the SE and register it into the LFC. This method can fail due to different reasons: the BDII[6] or the SE or the LFC are not available. We decided to perform separately a file copy (lcg-cp) and the file registration in LFC (lcg-rf). If lcg-cp fails we perform a copy specifying taht the bdii must not be included in the process (lcg-cp –nobdii), if it works we conclude with the file registration (lcg-rf). If the lcg-cp –nobdii fails we finally proceed with the low level copy via gridftp tool (globus-url-copy) and subsequently with the file registration step.

3.6 The operations web portal

An automated submission procedure is of the utmost importance in order to speed up and simplify the user operations completion. As a matter of fact, the major hurdle in accessing the Grid infrastructure for non-expert users is given by its intrinsic complexity and their lack of expertise.

To accomplish this task, a web-based user interface has been developed, which takes care of the bookkeeping database interactions, the job wrapper script preparation according to the VO manager's input, and the job management; it also provides advanced monitor functionality and procedures to query the stored metadata.

[6]Berkeley Database Information Index Read, used to maintain information for the Grid computing infrastructure, http://en.wikipedia.org/wiki/BDII

3.6.1 Web portal technology and design

The operations web portal has the typical architecture of a web application project: the user's browser accesses the site on the web server, the server side code builds the web pages relying on information residing on a back-end DB. As for project design the web server resides in the same Operating System (the gLite User Interface) as the information system so there is no interest in creating a web service layer allowing web server to communicate with the back-end DB.

The following list contains the technologies adopted for the development of the operations web portal, the server and the client sides:

- Server side
  - PHP - versione
  - PHP database wrapper
  - Smarty Template Engine
- Client side
  - jQuery
  - jQuery UI
  - jQuery DataTables
  - Google Charts tools

Smarty template engine provides the possibility to separate the logic programming (pure PHP) by the interface programming (HTML, Javascript, jQuery [62] [63]); in particular introduces the concept of templates: files containing interface programming code associated with PHP files. The figure 3.13 shows the relational schema among Smarty templates, PHP files and Javascript files during the access to the portal monitoring section.

The jQuery suite was born with the aim of facilitating the Javascript writing code implementing complex techniques like Ajax. jQuery UI has been developed in particular for graphical effects programming and complex interactions like drag and drop, elements ordering and widget. The DataTables functionality include advanced capabilities in managing tables like length dynamic paging, data loading via Ajax and multi-colomn ordering.

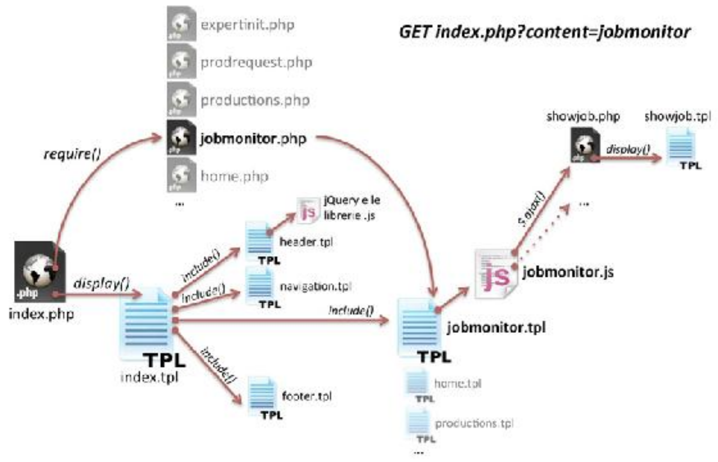


Figure 3.13: Diagram example of relation-ships among PHP scripts, the Smarty templates and the Javascripts for the jobmonitor request in the web operations portal.

3.6.2 The portal architecture

The web portal is structured in the following main sections:

- home: the user's home page containing the status of last activity
- session: the section for the management and creation of a Session (designed, still not implemented)
- productions: the section for the management and creation of a Production
- prodrequest: the section for the management, creation and update of the Request

- expertinit: the section for the job submission dedicated to expert users
- shiftinit: the section for the job submission dedicated to users on shift program
- jobmonitor: the section dedicated to the monitoring functionality
- sitemgr: the section for distributed resource management

As we discussed in section 3.4 the framework has to be adaptable respecting the VO requirements in terms of session definition, the web portal has to adapt too. The figure 3.14 shows the main blue bar containing two VO defined sessions FastSim and FullSim, each one has a private menu allowing the access to all the portal sections. Each link in the menu has the parameter *sn* identifying the session to be considered.

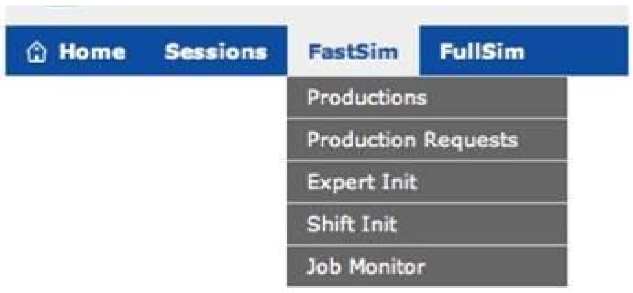


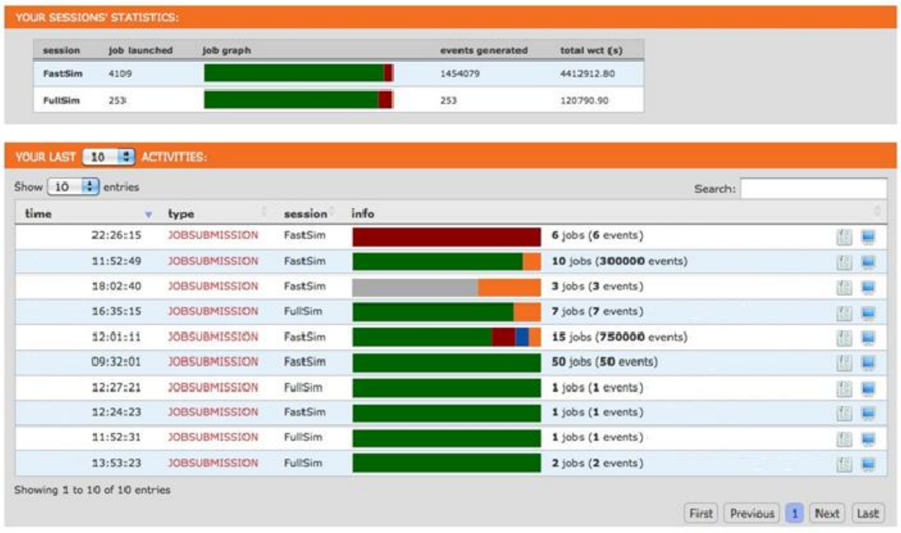
Figure 3.14: The operations portal modifies its own main menu structure with respect to the defined sessions.

User home page

The home page provides information about user groups and privileges, statistics of job submission per session and last activity list.

As it is shown in the figure 3.15, in the left side box the user information have been collected:

- user identifier



session	job launched	job graph	events generated	total wct (s)
FastSim	4109		1454079	4412912.80
FullSim	253		253	120790.90

time	type	session	info	
22:26:15	JOBSUBMISSION	FastSim		**6** jobs (**6** events)
11:52:49	JOBSUBMISSION	FastSim		**10** jobs (**300000** events)
18:02:40	JOBSUBMISSION	FastSim		**3** jobs (**3** events)
16:35:15	JOBSUBMISSION	FullSim		**7** jobs (**7** events)
12:01:11	JOBSUBMISSION	FastSim		**15** jobs (**750000** events)
09:32:01	JOBSUBMISSION	FastSim		**50** jobs (**50** events)
12:27:21	JOBSUBMISSION	FullSim		**1** jobs (**1** events)
12:24:23	JOBSUBMISSION	FastSim		**1** jobs (**1** events)
11:52:31	JOBSUBMISSION	FullSim		**1** jobs (**1** events)
13:53:23	JOBSUBMISSION	FullSim		**2** jobs (**2** events)

Showing 1 to 10 of 10 entries

Figure 3.15: The home page provides information about user groups and privileges, a per session job submission statistics and the list of activity.

- the user proxy certificate validity is shown by a green/red led
- the LDAP groups the user belongs to
- a per session user's privileges

The box SESSSIONS' STATISTICS includes information like the total submitted jobs, the total amount of generated output (here in terms of events) and the wall clock time used. The stripe graph shows the summary of the job status. The LAST ACTIVITY box includes the actions of Submission creation, Request creation and Production creation. For each Submission activity the direct link to the related monitoring web page is provided.

Session management

The section of the Operations web portal dedicated to the VO Session definition and configuration is not still developed, but has been designed.

The Session class and its functions are completely integrated in the server side software. The Session web section is shown in figure 3.16.

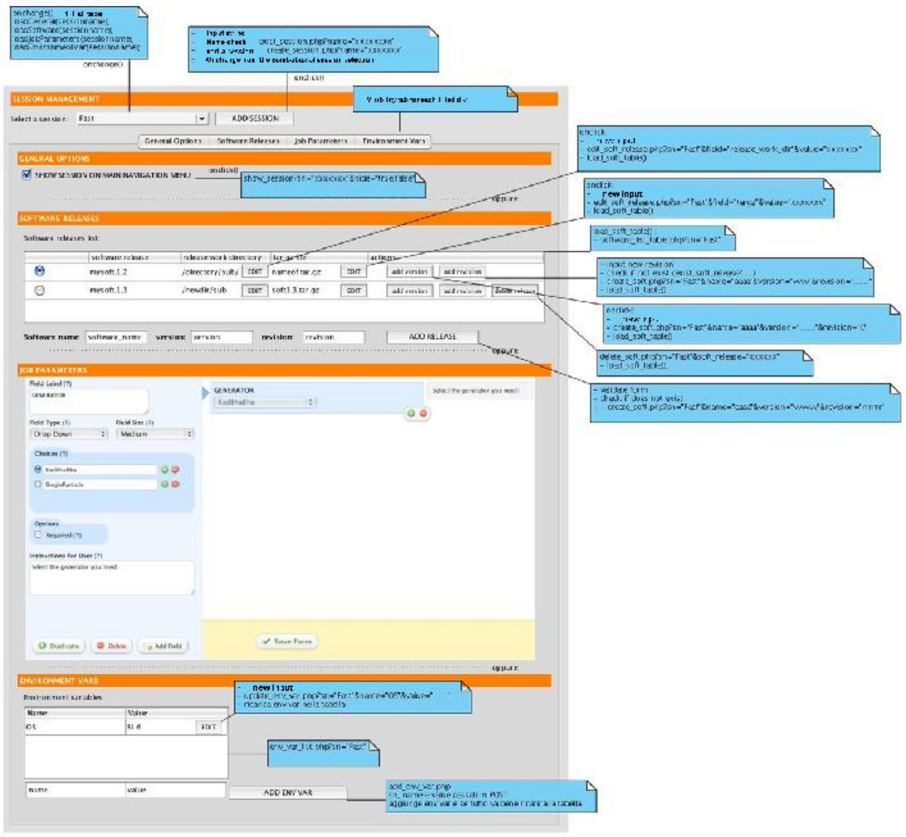
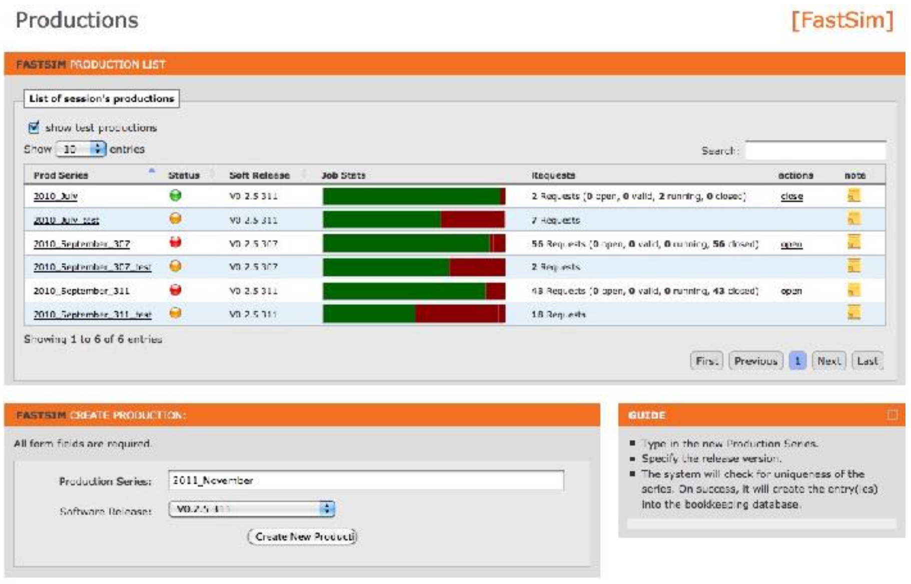


Figure 3.16: Draft schema of the web page dedicated to the Session definition and configuration. The blue boxes show implementation details.

The SOFTWARE RELEASE box at the top of the web page is dedicated to the insertion and visualization of application releases; the revision, working directory and package name can be edited and selected. The ENVIRONMENT box at the bottom of the web page allows the insertion and visualization of Application environment variable like JAVA_HOME or LD_LIBRARY_PATH per Operating System version. The JOB PARAMETERS box is the harder to be designed in fact its goal is to provide a form capable to model the insertion of any kind of application parameters with parameters dependence. The shown example includes the possibility of parameter creation, type and size, one level parameters dependence, required flag and user instructions. Finally the box GENERAL OPTIONS dedicated to usability and interface customization actions like enable/disable of session entry in navigation menu.

Production and request management

The figure 3.17 shows Production web interface for the Session FastSim.


Productions [FastSim]

FASTSIM PRODUCTION LIST

List of session's productions

☑ show test productions

Show 10 entries Search:

Prod Series	Status	Soft Release	Job Stats	Requests	actions	note
2010_July	●	V0 2.5 311	🟩🟥	2 Requests (0 open, 0 valid, 2 running, 0 closed)	close	
2010_July_test	●	V0 2.5 311	🟩🟥	7 Requests		
2010_September_307	●	V0 2.5 307	🟩🟥	56 Requests (0 open, 0 valid, 0 running, 56 closed)	open	
2010_September_307_test	●	V0 2.5 307	🟩🟥	2 Requests		
2010_September_311	●	V0 2.5 311	🟩🟥	43 Requests (0 open, 0 valid, 0 running, 43 closed)	open	
2010_September_311_test	●	V0 2.5 311	🟩🟥	18 Requests		

Showing 1 to 6 of 6 entries

FASTSIM CREATE PRODUCTION:

All form fields are required.

Production Series: 2011_November

Software Release: V0.2.5 [illegible]

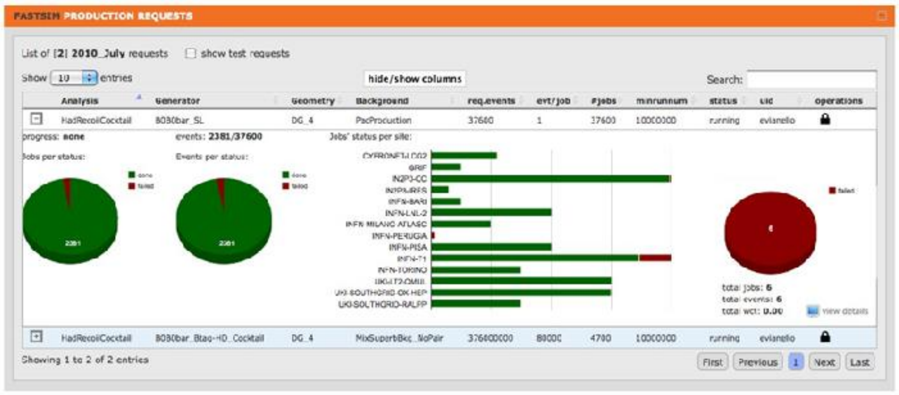
GUIDE

- Type in the new Production Series.
- Specify the release version.
- The system will check for uniqueness of the series. On success, it will create the entry(ies) into the bookkeeping database.

Figure 3.17: The web section for Production creation and management.

The web page includes the section of the operations portal dedicated to the Production management (PRODUCTION LIST box) and a section for the creation of new Production(CREATE PRODUCTION box). The Production list has been obtained using the plugin datatables (see section 3.6.1) by the productionlist.php script in JSON format. At Production creation time a couple of Productions will be created, the official one and its twin the test Production. Enabling the flag "show test productions" the test Production for each official one will be shown with the suffix name "_test".

Statistics about job status of the submitted jobs per Production are shown in the stripe graph. Production creation needs the insertion of Production name and application software release/revision. The Productions with no associated Requests only can be canceled. A Production can be set to status open or closed and vice versa with no logical constraints by the VO manager; a Production in "closed" status can not be modified in terms of Requests or metadata. The visualization/status control and creation are the two main sections of the Request management interface. The section REQUESTS LIST is dedicated to Request management per Production, the section CREATE REQUEST is dedicated to the configuration and Request creation process.



List of [2] 2010_July requests

Analysis	Generator	Geometry	Background	req.events	evt/job	#jobs	minrunnum	status	uid	operations
HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	37600	1	37600	10000000	running	evianello	🔒
HadRecoilCocktail	B0B0bar_Btag-HD_Cocktail	DG_4	MixSuperbBkg_NoPair	376000000	80000	4700	10000000	running	evianello	🔒

progress: none

events: 2381/37600

total jobs: 6
total events: 6
total wct: 0.00

Showing 1 to 2 of 2 entries

Figure 3.18: The section REQUEST LIST for the Request management interface in operations web portal.

The REQUESTS LIST section is composed by a table containing all the Requests for the selected Production, its own parameters and various statistical graphs. The figure 3.18 reports an example of such a section. The graphs have been created using Google Chart Tools API, the datatable jQuery plugin allows to collect statistical information.

Operating on "Lock" icon on the right side of each Request raw the VO manager can modify the configuration of a Request in Open state, validate, delete or close the Requests observing the constraints reported by the following figure 3.19, the Request status diagram.

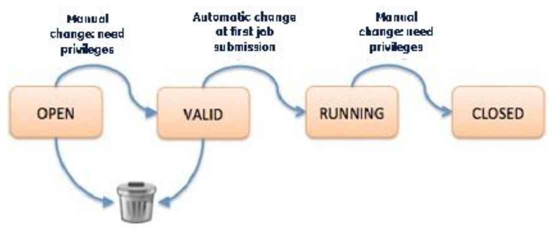

Manual change: need privileges
Automatic change at first job submission
Manual change: need privileges
OPEN
VALID
RUNNING
CLOSED

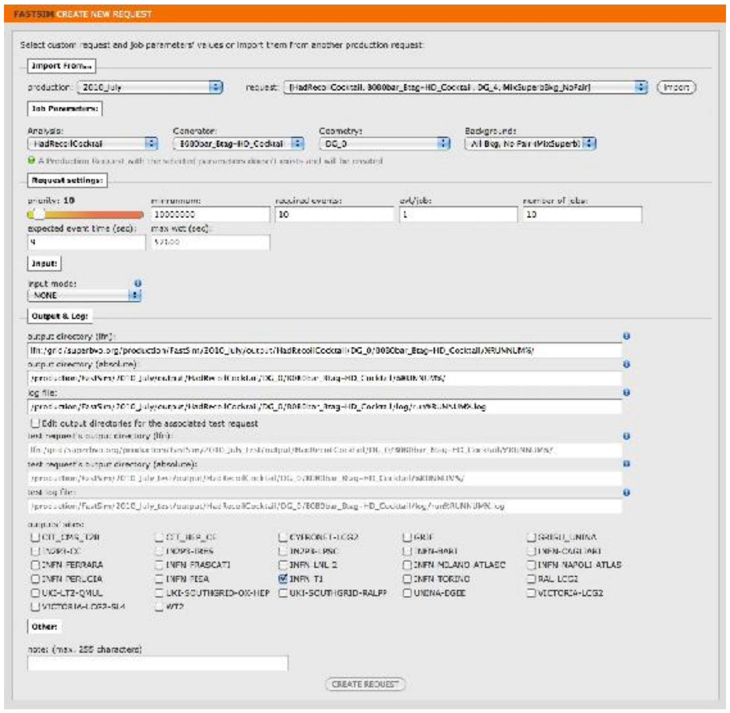
Figure 3.19: The Request status diagram.

Let's now analyze the steps of the Request configuration and creation process. The section CREATE REQUEST is shown in figure 3.20 The section is divided in various parts:

- IMPORT: the request configuration can be imported choosing among the Requeste already created in this Session;
- JOB PARAMETERS: application parameter section;
- REQUEST SETTINGS: includes the following set of fields
  - priority
  - min-runnum for all the jobs in the Request
  - amount of output elements (events in FastSim Session) to satisfy the Request
  - amount of output elements per job
  - the estimated number of jobs to be submitted
  - the estimated job duration
- INPUT: the input mode need to be selected among NONE, DIR, FILE or LIST. In case of a value different by NONE the LFN need to be inserted
- OUTPUT and LOG: includes the following set of fields
  - the output directory (LFN)

  - the log file path (LFN)
  - the output and log path for the associated test Request (as for the Production, the Request is created with the test twin ¡request-name¿_test)
  - one or more sites as targets for output transfers

- OTHER: free note


FASTSIM CREATE NEW REQUEST

Select custom request and job parameters' values or import them from another production request:

Import From...

production: 2010_July
request: [HadRecoilCocktail, 8080bar_Btag-HD_Cocktail, DG_4, MixSuperbBkg_NoPair]

Job Parameters:

Analysis: HadRecoilCocktail
Generator: 8080bar_Btag-HD_Cocktail
Geometry: DG_0
Background: All Bkg, No Pair (MixSuperb)

A Production Request with the selected parameters doesn't exists and will be created

Request settings:

priority: 10
run number: 10000000
required events: 10
evt/job: 1
number of jobs: 10
expected event time (sec): 4
max wct (sec): 57600

Input:

input mode: NONE

Output & Log:

output directory (lfn):
lfn:/grid/superbvo.org/production/FastSim/2010_July/output/HadRecoilCocktail/DG_0/8080bar_Btag-HD_Cocktail/%RUNNUM%/

output directory (absolute):
/production/FastSim/2010_July/output/HadRecoilCocktail/DG_0/8080bar_Btag-HD_Cocktail/%RUNNUM%/

log file:
/production/FastSim/2010_July/output/HadRecoilCocktail/DG_0/8080bar_Btag-HD_Cocktail/log/run%RUNNUM%.log

☐ Edit output directories for the associated test request

test request's output directory (lfn):
lfn:/grid/superbvo.org/production/FastSim/2010_July_test/output/HadRecoilCocktail/DG_0/8080bar_Btag-HD_Cocktail/%RUNNUM%/

test request's output directory (absolute):
/production/FastSim/2010_July_test/output/HadRecoilCocktail/DG_0/8080bar_Btag-HD_Cocktail/%RUNNUM%/

test log file:
/production/FastSim/2010_July_test/output/HadRecoilCocktail/DG_0/8080bar_Btag-HD_Cocktail/log/run%RUNNUM%.log

output sites:

☐ CIT_CMS_T2B	☐ CIT_HEP_CE	☐ CYFRONET-LCG2	☐ GRIF	☐ GRISU_UNINA
☐ IN2P3-CC	☐ IN2P3-IRES	☐ IN2P3-LPSC	☐ INFN-BARI	☐ INFN-CAGLIARI
☐ INFN FERRARA	☐ INFN FRASCATI	☐ INFN LNL 2	☐ INFN MILANO ATLASC	☐ INFN NAPOLI ATLAS
☐ INFN PERUGIA	☐ INFN PISA	✓ INFN T1	☐ INFN TORINO	☐ RAL LCG2
☐ UKI-LT2-QMUL	☐ UKI-SOUTHGRID-OX-HEP	☐ UKI-SOUTHGRID-RALPP	☐ UNINA-EGEE	☐ VICTORIA-LCG2
☐ VICTORIA-LCG2-SL4	☐ WT2			

Other:

note: (max. 255 characters)

Figure 3.20: The CREATE REQUEST section allowing the insertion of a new Request.

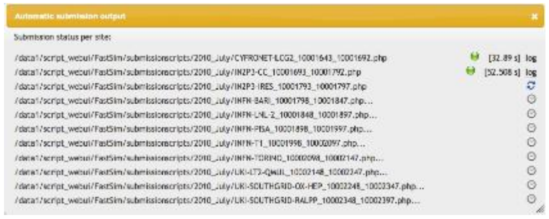
Please refer to Metadata management section 3.4 for detailed information about Request related entities.

Job submission

Before starting describing the involved interfaces let's recall few concepts and strategies the job submission is based on in the framework: we implemented a unique submission method among the ones gLite middlware makes available, it is the bulk. One bulk submission is in direct correspondence with the entity Submission, subsequently one Request is composed by one or more bulk submissions to one ore more target sites. Application software and application parameters are the same for each bulk submission. Said that whatever web page designated for job submission has to perform an ajax call to the submissioninit.php script providing:

- the Session name, sn
- the action to be performed:
  - summary: returns the HTML preview of the list of the bulk submissions set up
  - postpone_submit: inserts the jobs metadata (with status "prepared") in the bookkeeping DB and creates the submission scripts (this action is for development purpose only, can be inhibited)
  - submit: performs the postpone_submit actions and create the proxy certificate that will be used by Ganga to perform the gLite job submission
- the javascript object "subparam" containing all the job metadata

Subsequently the system will generate n submission scripts (n is the number of target sites) which are on duty for Ganga script launch, n Severus configuration files and a general launch script that will execute all the submission scripts. The list of all generated files is passed to the submissioninit.php script that will execute the general launch script. The figure 3.21 reports the progress submission window shown to the VO manager during the submission operations.



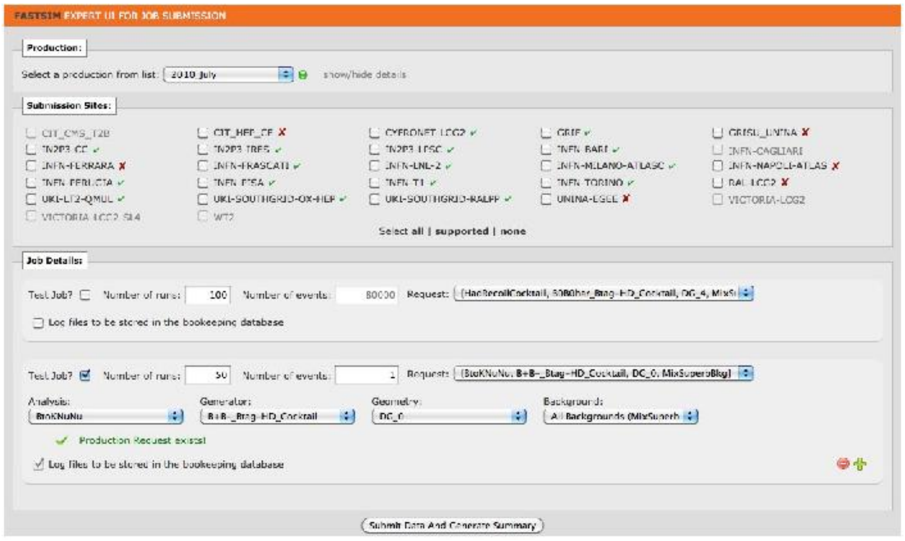
```
Submission status per site:

/data1/script_webui/FastSim/submissionscripts/2010_July/CYFRONET-LCG2_10001643_10001692.php      [32.89 s] log
/data1/script_webui/FastSim/submissionscripts/2010_July/IN2P3-CC_10001693_10001792.php          [52.508 s] log
/data1/script_webui/FastSim/submissionscripts/2010_July/IN2P3-IRES_10001793_10001797.php
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-BARI_10001798_10001847.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-LNL-2_10001848_10001897.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-PISA_10001898_10001997.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-T1_10001998_10002097.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/INFN-TORINO_10002098_10002147.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/UKI-LT2-QMUL_10002148_10002247.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/UKI-SOUTHGRID-OX-HEP_10002248_10002347.php...
/data1/script_webui/FastSim/submissionscripts/2010_July/UKI-SOUTHGRID-RALPP_10002348_10002397.php...
```

Figure 3.21: Progress window shown during automatic bulk submissions.

Submission web section for expert users The Expert Init section is the job submission interface for expert users (figure 3.22). The user has to select one or more target site, every Session enabled site can be selected. The user can define one or more bulk submissions, for each one need to provide:

- the number of jobs to be submitted
- the Request of reference
- if the job need to be inserted directly in the bookkeeping DB
- if it is a test job or not

If the user chose to submit a test bulk the interface will give the opportunity to detach the submission from the Request. The entire set of job parameters can be redefined.



FASTSIM EXPERT UI FOR JOB SUBMISSION

Production:

Select a production from list: 2010 July

Submission Sites:

CIT_CMS_T2B	CIT_HEP_CE ✗	CYFRONET-LCG2 ✓	GRIF ✓	GRISU_UNINA ✗
IN2P3-CC ✓	IN2P3-IRES ✓	IN2P3-LPSC ✓	INFN-BARI ✓	INFN-CAGLIARI
INFN-FERRARA ✗	INFN-FRASCATI ✓	INFN-LNL-2 ✓	INFN-MILANO-ATLASC ✓	INFN-NAPOLI-ATLAS ✗
INFN-PERUGIA ✓	INFN-PISA ✓	INFN-T1 ✓	INFN-TORINO ✓	RAL-LCG2 ✗
UKI-LT2-QMUL ✓	UKI-SOUTHGRID-OX-HEP ✓	UKI-SOUTHGRID-RALPP ✓	UNINA-EGEE ✗	VICTORIA-LCG2
VICTORIA-LCG2-SL4	WT2			

Select all | supported | none

Job Details:

Test Job? ☐ Number of runs: 100 Number of events: 80000 Request: (HadRecoilCocktail, 5080bar_Btag-HD_Cocktail, DG_4, MixS

☐ Log files to be stored in the bookkeeping database

Test Job? ☑ Number of runs: 50 Number of events: 1 Request: (BtoKNuNu: B+B-_Btag-HD_Cocktail, DC_0, MixSuperbBkg)

Analysis: BtoKNuNu Generator: B+B-_Btag-HD_Cocktail Geometry: DG_0 Background: All Backgrounds (MixSuperb

✓ Production Request exists!

☑ Log files to be stored in the bookkeeping database

Figure 3.22: The Expert Init interface.

Submission web section for shift users The Shift Init section is the job submission interface for users on shift. When the Production have to be performed during a long period of time (a certain number of weeks) due to large amount of requested output, the VO can organize a cycle shifts for job submission management.

Before starting the interface description we need to present the method adopted to calculate the site status. The status of a site is its ability to schedule a job and run it all over the available worker nodes. In other words, if the site farm is empty the jobs will be scheduled as soon as they will be routed by WMS, CE and local batch system, on the contrary a certain number of jobs will remain in pending status waiting for resource availability. Severus job wrapper communicates to the bookkeeping DB when it is start running so the framework can classify the site in three status GREEN, YELLOW and RED. If number of pending jobs (in prepared or submitted status) is lesser then the half of the number of submitted jobs (half bulk size) the site status is GREEN, if it is in between the half and the submitted jobs, the site status is YELLOW, otherwise it is RED. The site entity in the bookkeeping DB collects per site the "bunch_dimension", the "failed" and the "max_running" values: the fist one is the fixed bulk size for the specific site[7], the second is the number of accepted job failures before the site could be flagged as "unavailable" and the third one is the maximum number of jobs in running state at the specific site.

The framework calculates the following three values for each site:

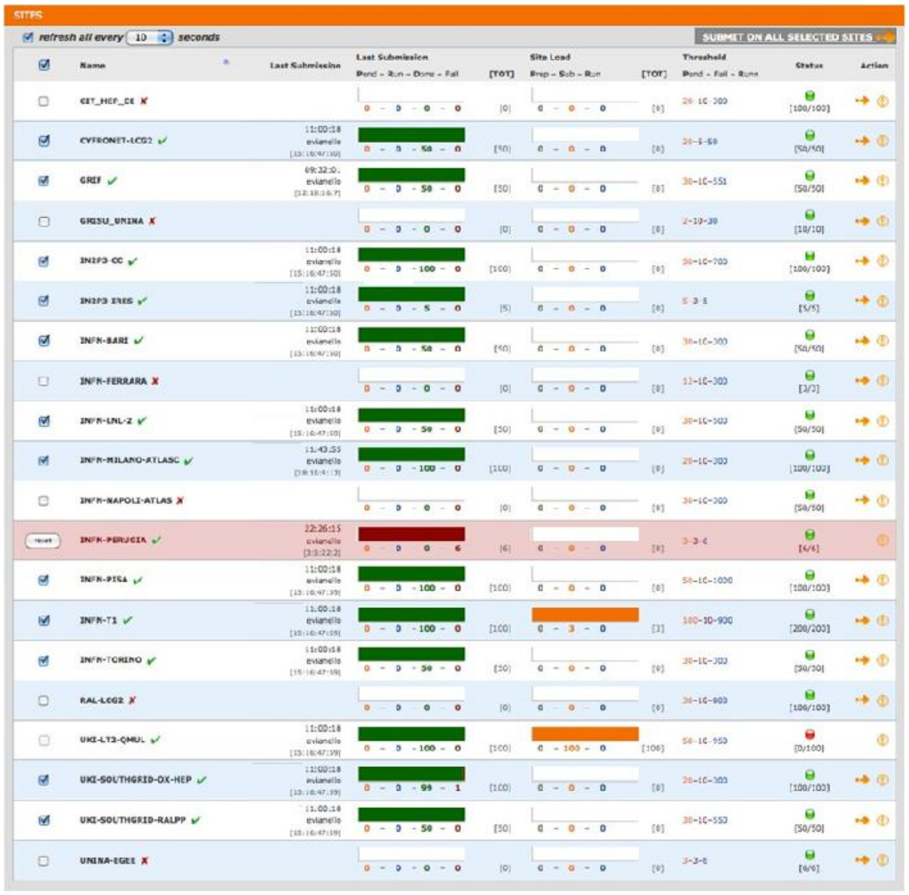
- site status: GREEN, YELLOW and RED will be the led's color per site in shift init web interface
- site availability: a site becomes unavailable if and only if the number of failed jobs exceeds the "failed" value for that site in bookkeeping DB
- overload: a site is considered overloaded if the number of job running exceeds the threshold value "max_running" for that site in bookkeeping DB

The next submission on a certain site will have a full bulk size given by "bunch_dimension" if the site status is GREEN, half of it if the site status is YELLOW and zero if the site status is RED. Submission is enabled all over the non overloaded sites in function of their statuses. As last submission control parameter defined in bookkeeping DB we needed to introduce a maximum running job for all the jobs running all over the sites enabled in a Session. This parameters allows to establish a framework and Grid infrastructure limit in managing job communications with bookkeeping DB.

The main differences between the Expert Init and the Shift Init interfaces are:

- the selection of target sites Shift Init interface is bound by the site status residing in bookkeeping DB
- the number of jobs for each bulk submission varies in function of the site "bunch_dimension" and the site status

[7]The bulk size varies with respect to the reserved resourced for the VO per site


SITES

☑ refresh all every 10 seconds

SUBMIT ON ALL SELECTED SITES

☑	Name	Last Submission	Last Submission Pend - Run - Done - Fail	[TOT]	Site Load Prep - Sub - Run	[TOT]	Threshold Pend - Fail - Runs	Status	Action
☐	CIT_HEP_CE ✗		0 - 0 - 0 - 0	[0]	0 - 0 - 0	[0]	20-10-300	● [100/100]	➜ ⓘ
☑	CYFRONET-LCG2 ✓	11:00:18 evianello [15:16:47:50]	0 - 0 - 50 - 0	[50]	0 - 0 - 0	[0]	20-5-50	● [50/50]	➜ ⓘ
☑	GRIF ✓	09:32:01 evianello [12:18:16:7]	0 - 0 - 50 - 0	[50]	0 - 0 - 0	[0]	30-10-551	● [50/50]	➜ ⓘ
☐	GRISU_UNINA ✗		0 - 0 - 0 - 0	[0]	0 - 0 - 0	[0]	2-10-30	● [10/10]	➜ ⓘ
☑	IN2P3-CC ✓	11:00:18 evianello [15:16:47:50]	0 - 0 - 100 - 0	[100]	0 - 0 - 0	[0]	50-10-700	● [100/100]	➜ ⓘ
☑	IN2P3-IRES ✓	11:00:18 evianello [15:16:47:50]	0 - 0 - 5 - 0	[5]	0 - 0 - 0	[0]	5-3-5	● [5/5]	➜ ⓘ
☑	INFN-BARI ✓	11:00:18 evianello [15:16:47:50]	0 - 0 - 50 - 0	[50]	0 - 0 - 0	[0]	30-10-300	● [50/50]	➜ ⓘ
☐	INFN-FERRARA ✗		0 - 0 - 0 - 0	[0]	0 - 0 - 0	[0]	12-10-300	● [3/3]	➜ ⓘ
☑	INFN-LNL-2 ✓	11:00:18 evianello [15:16:47:50]	0 - 0 - 50 - 0	[50]	0 - 0 - 0	[0]	30-10-500	● [50/50]	➜ ⓘ
☑	INFN-MILANO-ATLASC ✓	11:43:55 evianello [3:8:16:4:13]	0 - 0 - 100 - 0	[100]	0 - 0 - 0	[0]	20-10-300	● [100/100]	➜ ⓘ
☐	INFN-NAPOLI-ATLAS ✗		0 - 0 - 0 - 0	[0]	0 - 0 - 0	[0]	30-10-300	● [50/50]	➜ ⓘ
reset	INFN-PERUGIA ✓	22:26:15 evianello [3:5:22:2]	0 - 0 - 0 - 6	[6]	0 - 0 - 0	[0]	3-3-6	● [6/6]	ⓘ
☑	INFN-PISA ✓	11:00:18 evianello [15:16:47:39]	0 - 0 - 100 - 0	[100]	0 - 0 - 0	[0]	50-10-1000	● [100/100]	➜ ⓘ
☑	INFN-T1 ✓	11:00:18 evianello [15:16:47:59]	0 - 0 - 100 - 0	[100]	0 - 3 - 0	[3]	100-30-900	● [200/200]	➜ ⓘ
☑	INFN-TORINO ✓	11:00:18 evianello [15:16:47:59]	0 - 0 - 50 - 0	[50]	0 - 0 - 0	[0]	30-10-300	● [50/50]	➜ ⓘ
☐	RAL-LCG2 ✗		0 - 0 - 0 - 0	[0]	0 - 0 - 0	[0]	30-10-900	● [100/100]	➜ ⓘ
☐	UKI-LT2-QMUL ✓	11:00:18 evianello [15:16:47:39]	0 - 0 - 100 - 0	[100]	0 - 100 - 0	[100]	50-10-950	● [0/100]	ⓘ
☑	UKI-SOUTHGRID-OX-HEP ✓	11:00:18 evianello [15:16:47:39]	0 - 0 - 99 - 1	[100]	0 - 0 - 0	[0]	20-10-300	● [100/100]	➜ ⓘ
☑	UKI-SOUTHGRID-RALPP ✓	11:00:18 evianello [15:16:47:59]	0 - 0 - 50 - 0	[50]	0 - 0 - 0	[0]	30-10-550	● [50/50]	➜ ⓘ
☐	UNINA-EGEE ✗		0 - 0 - 0 - 0	[0]	0 - 0 - 0	[0]	3-3-8	● [0/0]	➜ ⓘ

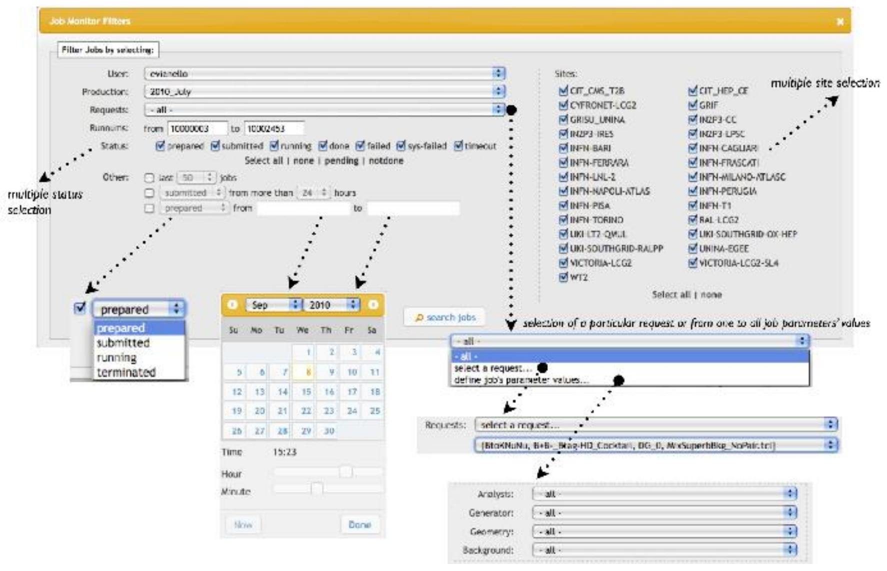
Figure 3.23: The ShiftInit interface with the list of selectable sites and submission management functions.

- the number of output elements per job is fixed to the one defined in the Request
- the job parameters can not be redefined, they are locked to the related Request

The Shif Init interface is organized in three main tabs showing the list of Requests (submission will be performed as the selected Request), the list of sites with submission management and the submission preview window. The figure 3.23 shows the tab containing the list of sites including site status, availability, last submission, job status graph, site load and the submission action buttons (per site or ”On all selected sites” in the up-right corner).

Job monitor

One of the main purpose of the operations web portal is the monitoring of the framework in terms of jobs all over the distributed resources. This function is implemented in the Job Monitor section. At first access this interface presents the window allowing job filtering and search. The figure 3.24 shows the filter box, it includes filters for user-id, production name, request name, run-number range, job status, sites, job in status submitted or running since number of hours, job in selectable status in a time range.


Job Monitor Filters
Filter Jobs by selecting:
multiple status selection
multiple site selection
selection of a particular request or from one to all job parameters' values
search jobs
Select all | none | pending | notdone
Select all | none
prepared
submitted
running
terminated
Su Mo Tu We Th Fr Sa
Now
Done
- all -
select a request...
define job's parameter values...

User: evianello
Production: 2010_July
Requests: - all -
Runnums: from 10000003 to 10002453
Status: ✓ prepared ✓ submitted ✓ running ✓ done ✓ failed ✓ sys-failed ✓ timeout
Other: ☐ last 50 jobs
☐ submitted from more than 24 hours
☐ prepared from to

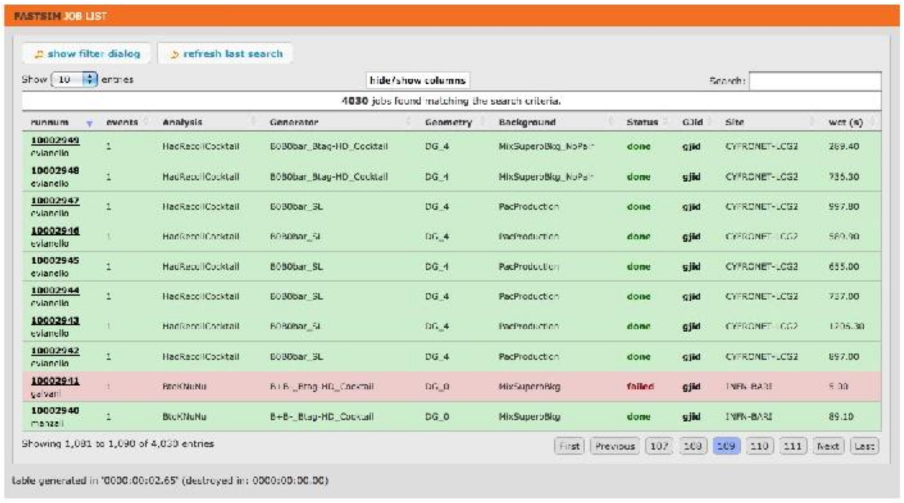
Sites:
✓ CIT_CMS_T2B ✓ CYFRONET-LCG2 ✓ GRISU_UNINA ✓ IN2P3-IRES ✓ INFN-BARI ✓ INFN-FERRARA ✓ INFN-LNL-2 ✓ INFN-NAPOLI-ATLAS ✓ INFN-PISA ✓ INFN-TORINO ✓ UKI-LT2-QMUL ✓ UKI-SOUTHGRID-RALPP ✓ VICTORIA-LCG2 ✓ WT2
✓ CIT_HEP_CE ✓ GRIF ✓ IN2P3-CC ✓ IN2P3-LPSC ✓ INFN-CAGLIARI ✓ INFN-FRASCATI ✓ INFN-MILANO-ATLASC ✓ INFN-PERUGIA ✓ INFN-T1 ✓ RAL-LCG2 ✓ UKI-SOUTHGRID-OX-HEP ✓ UNINA-EGEE ✓ VICTORIA-LCG2-SL4

✓ prepared
Sep 2010
Time 15:23
Hour
Minute

Requests: select a request...
{BtoKNuNu, B+B-_Btag-HD_Cocktail, DG_0, MixSuperbBkg_NoPair.tcl}

Analysis: - all -
Generator: - all -
Geometry: - all -
Background: - all -

Figure 3.24: The window for the filtered search of jobs in the Job Monitor section.



FASTSIM JOB LIST

show filter dialog | refresh last search

Show 10 entries | hide/show columns | Search:

4030 jobs found matching the search criteria.

runnum	events	Analysis	Generator	Geometry	Background	Status	GJId	Site	wct (s)
10002949 evianello	1	HadRecoilCocktail	B0B0bar_Btag-HD_Cocktail	DG_4	MixSuperbBkg_NoPair	done	gjid	CYFRONET-LCG2	289.40
10002948 evianello	1	HadRecoilCocktail	B0B0bar_Btag-HD_Cocktail	DG_4	MixSuperbBkg_NoPair	done	gjid	CYFRONET-LCG2	735.30
10002947 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	597.80
10002946 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	580.90
10002945 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	655.00
10002944 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	737.00
10002943 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	1206.30
10002942 evianello	1	HadRecoilCocktail	B0B0bar_SL	DG_4	PacProduction	done	gjid	CYFRONET-LCG2	897.00
10002941 galvani	1	BtoKNuNu	B+B-_Btag-HD_Cocktail	DG_0	MixSuperbBkg	failed	gjid	INFN-BARI	5.00
10002940 manzali	1	BtoKNuNu	B+B-_Btag-HD_Cocktail	DG_0	MixSuperbBkg	done	gjid	INFN-BARI	89.10

Showing 1,081 to 1,090 of 4,030 entries

table generated in '0000:00:02.65' (destroyed in: 0000:00:00.00)

Figure 3.25: Example of search result performed in Job Monitor section.

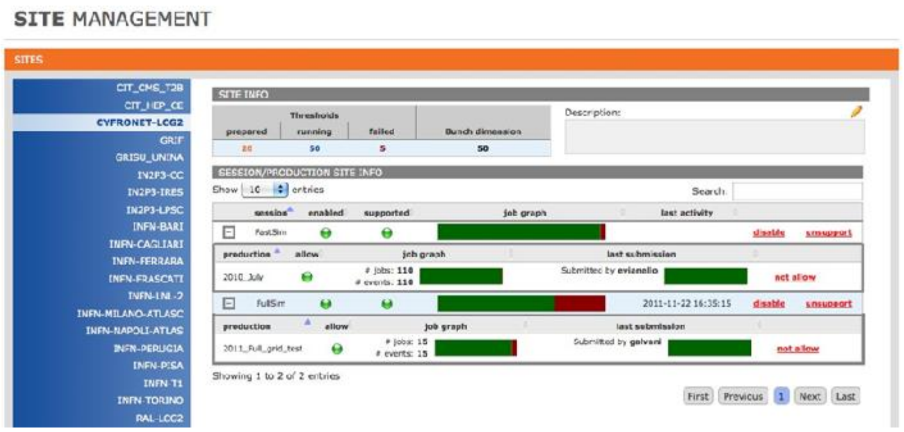
The result of a search is shown in figure 3.25. The job status reason communicated by Severus job wrapper can be directly accessed as a bubble tip moving the mouse on the status value for each job. Clicking the job run-number the job detail view is shown; it includes all job metadata information, the job running log (if available) and job output files in terms of LFN,GUID and SRM.

Distributed resource management

The Site Management portal section allows to insert a new Site characterized by the following metadata:

- site thresholds: limits for prepared, running and failed jobs
- bunch_dimension
- the enabled sessions
- the supported sessions
- the enabled Production for each sessions

The interface shown in figure 3.26 allows the selection from the left side column of the site allowing its own metadata update operations.



Thresholds			
prepared	running	failed	Bunch dimension
20	50	5	50

session	enabled	supported	job graph	last activity		
FastSim	●	●			disable	unsupport

production	allow	job graph	last submission	
2010_July	●	# jobs: 110 # events: 110	Submitted by evianello	not allow

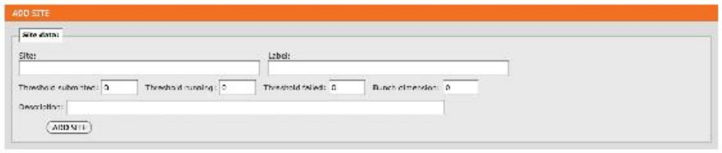
session	enabled	supported	job graph	last activity		
FullSim	●	●		2011-11-22 16:35:15	disable	unsupport

production	allow	job graph	last submission	
2011_Full_grid_test	●	# jobs: 15 # events: 15	Submitted by galvani	not allow

Showing 1 to 2 of 2 entries

Figure 3.26: Site Management web section. VO manager can enable a site for a Session or a Production.

The section ADD SITE allows the VO manager to insert new sites in bookkeeping DB, it is shown in figure 3.27



Threshold submitted: 0 Threshold running: 0 Threshold failed: 0 Bunch dimension: 0

Figure 3.27: The Site Management section dedicated to new site definition.

Chapter 4

Integration with Dirac system

Dirac (Distributed Infrastructure with Remote Agent Control) starts within the LHCb [64] experiment as a framework for Monte Carlo production in a Grid Computing environment. Since its first release, Dirac used Pilot Job paradigm[1] the consequence of this approach was a considerable increase in the overall efficiency of the production system. Over the years, Dirac was expanded by adding advanced job management, data management, user management and many other features, until Dirac became the official LHCb tool for handling the entire stack of distributed computing operations, including the user analysis jobs.

Since 2012 a large part of Dirac development has been dedicated to port the core system to be able to satisfy general purpose requirements, agnostic with respect to the specificity of LHCb. At present time Dirac can be customized to specific requirements adding new plugins. In fact all specific functions are integrated in Dirac via an Extension.

Dirac framework resulted to share the goal with the project discussed in

[1]In pilot-based infrastructures, users submit their jobs to a centralized queue or job repository. These jobs are handled asynchronously by submitted Grid jobs (called "pilot"). Pilot jobs start them execution asking to a pilot aggregator to obtain a user task from the centralized queue, when the pilot job completes the task requests a new user task to the pilot aggregator and so on until the queue is empty. A pilot job it is not committed to any particular task, and may not be even bound to a particular user.

this dissertation, it is a general-purpose framework for distributed computing resource exploitation oriented to big sized VO. Dirac architecture is complex and several sub components technology have been self-own developed like the Dirac File Catalog and the monitoring system. On the other hand Dirac is capable to exploit cloud stacks, is integrated with File Transfer System (FTS), adopt an efficient job workload management (pilot job, filling mode described in the following sections). We decided to perform a Dirac evaluation phase with the goal of integrating the bookkeeping DB, monitor system and site management in Dirac architecture.

4.1 Dirac system introduction

Resource management

Dirac is capable to interact with EGI, OSG and ARC Grid flavors. The gLite middleware is fully supported (which ensures full interoperability with EGI and OSG), support to the ARC middleware has recently been added. An additional module named VMDirac [65] ensures support for Cloud Computing resources. Originally developed as part of the experiment Belle II [66], VMDirac provides an interface to mostly used Cloud Computing platforms: the Amazon Elastic Compute Cloud [67] (EC2) and Open Cloud Computing Interface [68] (OCCI), a standard interface adopted by several Cloud Computing platform like OpenStack [69], OpenNebula [70] and CloudStack [71]. Configuring an SSH access, Dirac can use even resources like desktop PCs or clusters. Dirac can interact directly with most used batch systems such as LSF, BQS, SGE, PBS/Torque and Condor. Dirac, by means of two open source software, BOINC [72] (platform for distributed computing on a desktop computer) and VirtualBox [73] (virtualization software), can execute job payloads even on desktop and laptop computers. In this way a middle-big sized organization can use computational resources just available but often under-used (e.g. computers used for secretarial tasks), increasing the computing power in a manner completely transparent to the VO.

User management

User authentication is based on X.509 certificates, ensuring a fully compatibility with the required safety standards in a Grid Computing environment. Each user action is performed using its personal grid certificate. VOMS service is completely supported. Robot certificates authentication was successfully tested by Biomed VO. Each user can be member of more groups, each group with its own permissions and VOMS roles.

Workload management

The Dirac Central Task Queue (CTQ) collects all jobs submitted by users. According to jobs in CTQ, pilot jobs are launched on the available computing resources. The pilot job starts executing a routine that inspects the environment (hardware and software). Once the inspection is completed, the pilot jobs ask to CTQ for a payload to execute, according to characteristics of Worker Node. CTQ takes into account of jobs priority before assigning a payload to pilot job. This mechanism ensures the implementation of VO's priority policies.

Data management

EGI Storage Elements are fully supported, including access via SRM and GridFTP. Dirac provides a custom (DISET-based) Storage Element service, useful to simply add new Storage Elements, avoiding the complexity of installation and maintenance of a standard gLite storage service. Dirac storage elements can be used by jobs in a transparent way, like a normal gLite SE. Data transfer can be managed both using standard gLite copy commands (eg. lcg-cp) as well via File Transfer Service (FTS) for big data movement. Dirac is fully compatible with LCG File Catalog (LFC), but in addiction provides a custom File Catalog, named Dirac File Catalog (DFC). Such a catalog is SRMv2 compatible and provides advanced functionality like structured metadata management and advanced replica management. LHCb experiment is using DFC as default system and LFC as backup solution.

Monitoring

The Resource Status System (RSS) is a Dirac component that provides advanced monitoring capabilities of available resources. Monitoring can be configured with high granularity and allow the disabling of resources in according to established metrics. The RSS can be interfaced with third party monitoring systems (eg. Nagios).

Interfaces

Dirac provides a Command Line Interface (CLI) and a Web-based interface. Both CLI and Web-portal can be extended and customized. Webportal offers a graphical interface to File Catalog: using DFC even metadata can be managed. Dirac is equipped with a complete and extensible python API set: a GangaDirac layer based on this API is available. A REST interface is available, allowing Dirac integration with every programming language and Dirac access via mobile devices (eg. smartphone, tablets). Dirac REST interface is active in Barcelona Dirac instance.

Dirac architecture

Dirac has a modular structure: several components (Systems) interact each other via a custom secure protocol named DISET. Dirac is distributed: every "System" implements a set of functionality, different Systems can be installed on one or different servers, every System can be replicated on several machines. Dirac is fault tolerant due to Systems replication and distribution across geographically distributed sites. "Setup" is a set of cooperating Systems in order to provide a set of functionality: Dirac can manage several concurrent Setups (eg. a production Setup, a testing Setup, a development Setup). Dirac allows to configure and manage a distributed computing infrastructure, but simplifying management and minimizing human effort. In Dirac systems there are three kind of components: Agent, Service, Database.

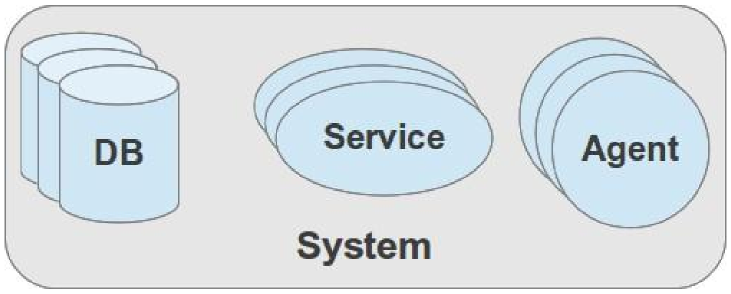


Figure 4.1: Dirac Sytems are made of Agents, Services and DBs.

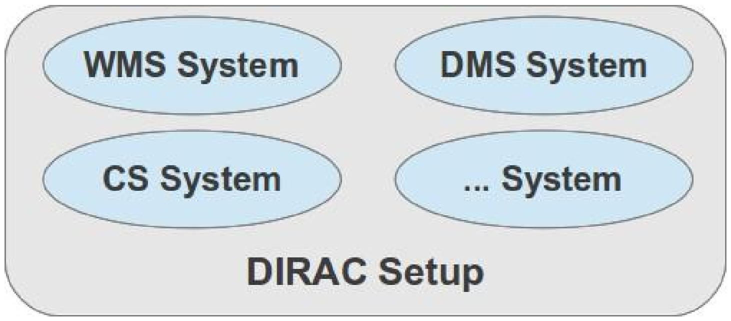


Figure 4.2: A Dirac Setup is a set of cooperating Systems.

An "Agent" is a stateless component, executed periodically according to configured polling time. Agent is used to perform active actions like script execution, job submission, file transfers, etc. A "Service" manage information, taking data from databases or other sources (eg. configuration files) and exposing methods to retrieve them. A "Database" stores information needed by system. Database back-end can be only MySQL.

Dirac community

Dirac community is made up of several VOs and computing centers. Below a list updated to December 2012.

Table 4.1: Entities belonging by a Session in SBK4

Subject	**Type**
LHCb	collaboration/experiment
Belle II	collaboration/experiment
BEPC	collaboration/experiment
BES III	collaboration/experiment
BYOMED @ CREATIS - Lyon	topical community
Lyon Computing Center	regional community
CESGA	regional community
CTA	collaboration/experiment
GISELA	regional community
GLAST	collaboration/experiment
ILC/CLIC	collaboration/experiment
IOIT @ Hanoi	regional community

4.2 Framework integration with Dirac system

4.2.1 Dirac capability check

The first step of the integration process was the Dirac capabilities check and comparative tests with our project in order to measure the use case target compatibility.

A comparative test between the framework, object of this dissertation, and Dirac system was made in order to give a quantitative performances measurement. There where several tasks to keep under control during the test: assure that the sites involved in the test were correctly configured, store the input files and the applications in DFC, replicate these files on the Storage Element of the sites, prepare the JDL for job submission, test Severus script in charge of running the simulation application on the Worker Node. A total of 1000 jobs were submitted, 100 for each of the 10 sites involved. Each job simulated 10000 events in FastSim session for a running time of about 3 hours per job. The input data were 5 files, for a total of 3GB, previously uploaded in DFC and replicated on several Storage Elements. The output data, were saved at INFN-T1 and registered in DFC, saving the output of each job in a specific folder. Test results about Dirac after 48 hours:

- 87,8% jobs successfully executed
- 8,5% jobs failed
- 3,7% jobs still in waiting
- up to 753 jobs simultaneously running
- filling mode not enabled

Table 4.2: General-framework vs Dirac comparative test results

	General framework				**Dirac**			
Site	done	expired	failed	ratio	done	waiting	failed	ratio
CYFRONET	100	0	0	100%	100	0	0	100%
IN2P3-CC	100	0	0	100%	20	0	80	20%
IN2P3-LPSC	33	67	0	33%	100	0	0	100%
INFN-BARI	97	0	3	97%	100	0	0	100%
INFN-FRASCATI	99	0	1	99%	59	37	4	59%
INFN-LNL-2	99	0	1	99%	100	0	0	100%
INFN-MILANO	100	0	0	100%	100	0	0	100%
INFN-PISA	100	0	0	100%	100	0	0	100%
INFN-T1	97	3	0	97%	100	0	0	100%
UKI-LT2-QMUL	0	100	0	0%	99	0	1	99%
TOTAL	825	170	5	82,5%	878	37	85	87,8%

Dirac and the General-framework performance are comparable. The framework has been optimized for test purpose, while Dirac has been used below of his best performances due to a lack of a fine-tuning caused by a non complete knowledge of the product at the time of the test. Test results show that Dirac is able to manage simulation productions as well as the General-framework, but in order to support an integration step of Dirac with the developed framework, Dirac should demonstrate to have some additional features. In fact other two major Dirac features have been tested: massive data management (FTS transfers) and efficiency gain enabling the filling mode.

The Dirac capability in managing data transfers via FTS have been tested using the routes (FTS channels): INFN-T1, INFN-Bari, IN2P3-CC(Lyon, France), RAL-LCG2(Manchester, UK). Dirac was able to perform test transfer successfully.

FTS management via Dirac offers two pros with respect to FTS standard usage via gLite:

- performing FTS actions are easier using Dirac with respect to standard gLite tools: gLite needs the full path (SURL), while Dirac needs only source and destination site
  - **gLite glite-transfer-submit -m <proxy_server> -s <fts_server> <source_file> <destination_file>**
  - **Dirac dirac-dms-fts-submit <lfn> <sourceSE> <destinationSE>**
- Dirac automatically updates the File Catalog inserting information of new available copy of transferred files, while using gLite this operation on File Catalog must be manually performed

Filling mode is the pilot job strategy allowing the exploitation of site WN until the pilot job proxy certificate is not expired. As we described in the previous section the pilot job once it is scheduled on the WN makes a request to the Task Queue Director for a user job (called payload) and run it. If the Filling Mode is active the pilot job can exploit all its proxy life-time to execute jobs/payloads as soon as the Central Queue can provide them.

The advantage provided by this technique consists in:

- make the most of the time available on the WN to perform many more payload as possible
- perform the matchmaking (the match between available resources and the resource requirements for a payload) only once for a set of payloads, rather than having to performed it once for each payload

The figure 4.3 shows the Pilot job submission cycle in Dirac. The blue boxes perform the task queue manager, the scheduler and the submitter modules, the pilot job running in the worker node is on duty for payload retrieval from Task queue (green line).

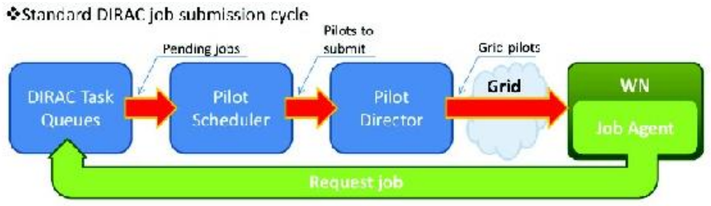


Figure 4.3: Dirac job submission cycle: Pilot job strategy.

A test was carried out in order to measure the actual gain in the performances offered by this operating mode. The test consisted in the submission of a test payload (the calculation of the Fibonacci series up to the 40th digit), submitting at INFN-T1 a series of 100, 1000 and 10000 job simultaneously. The test was first carried out with the filling mode disabled, then was repeated with the filling mode activated.

Graphs 4.4, 4.5 and 4.6 summarize the results of this test, from which it can be seen how the filling mode involves a significant decrease in the total execution time of the payloads[2], but also a more rapid emptying of the tail of payloads to execute (related to the area under the lines in graphs, where the number of jobs to be executed is projected against time). This effect becomes more evident as the number of submitted payloads increases.

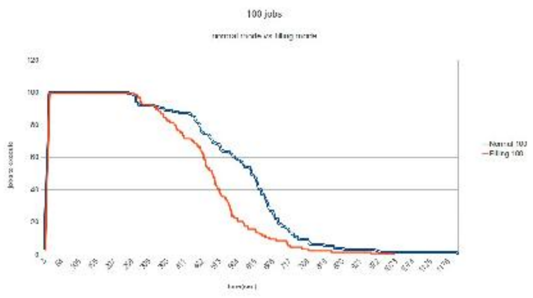


Figure 4.4: Filling mode test running 100 jobs. Pilot jobs run in 85,2% of normal jobs time.

[2]The time the last payload is completed

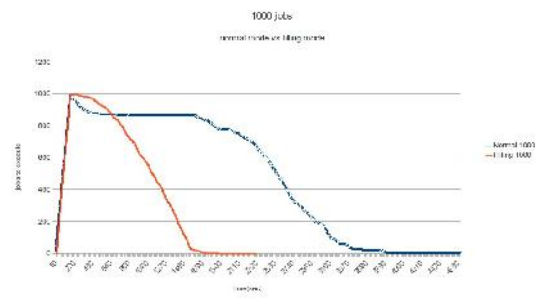


Figure 4.5: Filling mode test running 100 jobs. Pilot jobs run in 44,7% of normal jobs time.

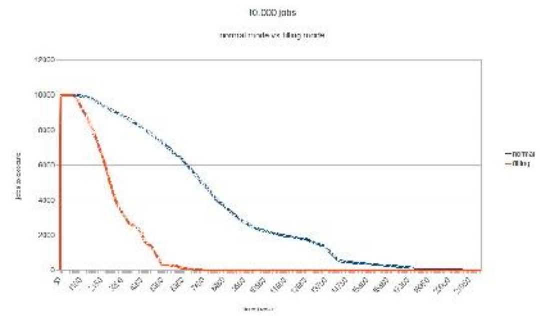


Figure 4.6: Filling mode test running 100 jobs. Pilot jobs run in 36,1% of normal jobs time..

4.2.2 Dirac merging project

The goal of this work is to obtain a Dirac system flavor tightly integrated with specific VO operations. Such integration regards the concept of Session and its own components: Production, Request and Submission; the typical VO computing model including Dirac system at present time includes also a disconnected bookkeeping DB and a logic layer acting as a bridge between the two environments. The pure workload management (Dirac) and the VO operations in terms of application and metadata management can interact directly in Dirac. That guarantee the highest level of synergy that subsequently brings the system to an increased level of usability and robustness.

The results of the Dirac evaluation phase described in the previous section have been evaluated taking into account the following factors:

- Dirac advanced features (mass data transfer, pilot job management, cloud interface and Dirac plug-able architecture)
- the developers and community support: very responsive
- the results of tests
- expertise gained inside the collaboration in managing and extending Dirac

We took the decision of starting the merging of our project peculiarities in the Dirac core architecture. The development of a new Dirac module was started. It has been organized in three sub-projects:

1. Create an interface between Dirac and Bookkeeping Database.
2. Extend Dirac web-portal creating new web-sections to allow the following actions:
    - Configuration and management of available sites for Production executions.
    - Production definition and initialization.
    - Productions monitoring.
    - Creation and monitoring of Requests.
    - Submission and monitoring of jobs belonging to a given request.
3. Execution of Severus job wrapper in the context of Dirac pilot job strategy.

The modular structure of Dirac can be used to extend its functionality by adding new modules. The initial goal of the new Dirac module is to bring together Dirac with the Bookkeeping DB and the Job Wrapper 4.7 in order to

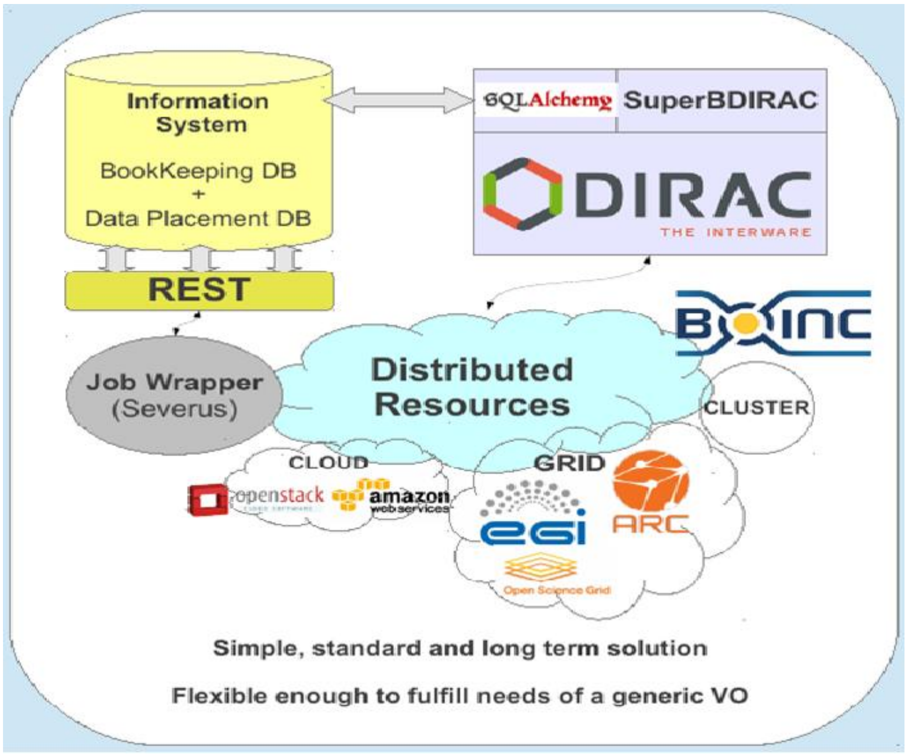


Figure 4.7: Bird's eye view on the Dirac merging project including Dirac system on the right-up corner, the framework elements (bookkeeping DB, REST communication layer and Severus job wrapper) on the left side and all the enabled distributed resources at the bottom.

allow the creation, monitoring and execution of the jobs for which metadata are structured in a Session architecture typical of our project.

One of the key constraint the design of the final Dirac flavor need to respect is the separation between cores functions and the bookkeeping DB. In fact the VO have to maintain the capacity of modify/upgrade the DB with no limitation in terms of dependence with Dirac. This requirement have to be taken into account during the integration with Dirac. On the other side Dirac doesn't provide a generic interface to SQL database. Therefore the first problem to face was the decision on how to interface Dirac with bookkeeping DB. Several solutions are available in Python allowing SQL database interface. Two tools has been evaluated for such a goal: psycopg [74] and SQLAlchemy [75]. Both psycopg and SQLAlchemy are released under an open source license (LGPL3 [76] license for psycopg, MIT [77] license for SQLAlchemy) so they are immediately usable. Psycopg relies on official Python DB API, can interface a PostgreSQL database and is able to map DB in python objects (Tuples in records, lists in array, dictionaries in hstore (Postgresql data type)). SQLAlchemy is an SQL toolkit and Object Relational Mapper (ORM) for Python. It is based on a data mapper pattern, so classes can be mapped to the database in multiple ways. SQLAlchemy natively support many of SQL dialects, so it's able to interact with following SQL database: Drizzle, Firebird, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SQLite and Sybase. Psycopg use raw SQL code to perform queries. SQLAlchemy by default maps database entities to python objects. In this way the interaction with database is just like interacting with a normal python object, even though also raw SQL code can be used. SQLAlchemy is the most elegant and powerful solution since the ORM[3] allows to write pure python code, avoiding use of raw SQL commands. The possibility to change DB backend without requiring any modification to source code but only modifying the connected DB is very important in the attempt to develop a code usable for a long time. Adopting psycopg allows to reuse all SQL queries just available, while using SQLAlchemy all queries must be coded using ORM paradigm. The great flexibility provided by SQLAlchemy with respect to psycopg was considered very important. The final decision was to adopt SQLAlchemy.

[3]Object Relational Mapping (ORM) is a programming technique in which a metadata descriptor is used to connect object code to a relational database. ORM converts data between type systems that are unable to coexist within relational databases and Object Oriented Programming (OOP) languages.

Job wrapper updates in Dirac

Since Severus was developed before Dirac evaluation test, it did not use any Dirac features. For this reason, some modifications have been necessary to allow its execution inside Dirac architecture based on pilot jobs. In our project jobs have planned to be launched from the same site where the VO application and its input files reside. As a consequence of this, input data files and software were supposed to be always locally accessible. Configuration file for Severus and job submission plan are supposed to be generated by web operational portal.

In order to exploit Severus from Dirac the file manager class has been modified to update the above mentioned characteristics: file access method needs to include standard gLite command use at central site too, the VO application need to be uploaded in the central Storage Elements and registered on DFC (not just in LFC) to allow the access via Grid tools in place of a local access only. The modified Severus version was successfully used via Dirac to perform a functionality test.

Summarizing, the job wrapper has been modified to meet Dirac system requirements for the following aspects:

- stage-in and stage-out managed via Dirac or via other protocols (eg. ftp, scp, curl etc.) as fallback options
- central framework site has been uniformed with all other sites with respect to the methods for input and application files access

The new Dirac module

The main component of the new module is the SimulationSystem, which is made of two components: the SBKService and the Dirac Web portal extension.

SBKService A generic SQL DB connector has been developed as a Dirac service, it is called SBKService. As we discussed in the previous section the

SQLAlchemy technology has been adopted to map the PostgreSQL rDBMS, the implementation of bookkeeping DB. SBKService exposes only Dirac compliant methods, so interactions with different databases is simply an interaction with another Dirac service. The figure 4.8 shows the abstraction on DB interaction implemented by SQLAlchemy technology: the new Dirac service makes use of the Managers software layer to perform read/write requests to the entity mapper layer, SQLA Mappers.

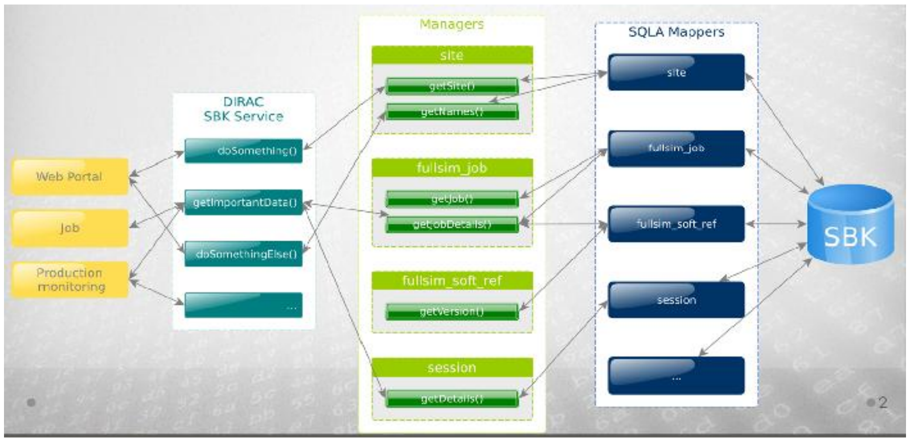


Figure 4.8: Mapping layers for bookkeeping DB in Dirac using SBKService and SQLAlchemy technology.

Dirac web portal extension A Dirac web portal extension has been developed to replicate operational web portal functionality, using the same technologies adopted in Dirac: Pylons [78] and ExtJS[4]. A site monitoring page is available (see figure 4.9), showing data from bookkeeping DB: site description (OS, grid flavor, architecture), site status in terms of software availability and certification, job submission thresholds, queue occupancy, site availability. Simulation jobs monitoring page (see figure 4.10) shows job status as reported in bookkeeping DB.

[4]ExtJS is a pure JavaScript application framework for building interactive web applications using modern techniques such as Ajax, DHTML and DOM scripting.

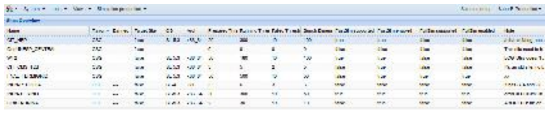


Figure 4.9: Web interface reporting the distributed sites metadata using the new Dirac module.

Figure 4.10: Web interface reporting jobs metadata using the new Dirac module.

Functional test The goal of functional test is proving the correct integration of bookkeeping Db and Severus job wrapper in Dirac system. In particular we need to verify the functioning of job wrapper communications with bookkeeping DB and the correctness of metadata information inclusion in Dirac web portal. Dirac job monitoring and distributed resource monitor have to be compliant with DB Session schema in terms of contents and entities relation-ships. Correct job execution is out of test scope.

The Production and Request parameters have been defined in bookkeeping DB via operations web portal. Such information will be retrieved by a specific Python script encapsulating the Dirac client functions. It uses Dirac API to prepare and submit jobs. Dirac server receives job submission commands from the Dirac client, then it starts the normal procedure for job management in Dirac: scheduling, pilot submission, payload retrieval and execution and stage-out. Job status updating is performed as usual by Severus script (the Dirac compliant) via REST communication with bookkeeping DB. Jobs were submitted to the Grid resources by Dirac server. Submission site was INFN-T1. FastSim session has been chosen as the more representative session for this test. Every job simulated 3000 events: this value was chosen to have an execution time longer than 10 minutes. Three main bunch submissions of 400 jobs were performed at INFN-T1 to obtain a total of 1200 simulation jobs. Two different job failures have been reproduced to verify the correct interactions between job and DB or Dirac web portal and DB.

Bookkeeping database has been properly updated for every job in every test. All status updates have been promptly displayed in operational web portal as well in Dirac web portal, with no appreciable delay between two portals. SQLAlchemy didn't introduce any appreciable delay or information loss, at least in this functionality test. Success rate was established as ratio between status saved in bookkeeping DB and status displayed in Dirac web portal: its value was 100% in all submissions. The new Dirac module could be considered a success in integrating bookkeeping DB and monitoring specific requirements in Dirac architecture.

Chapter 5

A real use case: description and results

The general-purpose platform for Grid resources exploitation has been validated and tested in a real use-case back in the context that was its original incubator, the SuperB experiment. As we discussed in Chapter 2 the SuperB project needed two types of Monte Carlo applications, the Full- and the Fast-simulations. Both processes can produce several types of events, depending on a set of parameters given to the application at run time, and may use a few files as input.

The INFN-Tier 1 site at CNAF in Bologna, Italy, has been chosen as the central EGI service site; it provides the User Interface to the Grid on which and the core functionality of the framework have been deployed. The distributed computing infrastructure used for these simulation productions includes 15 sites in Italy, France, UK, USA and Canada, which deploy three different Grid middleware (EGI/gLite, OSG/Condor, Westgrid/gLite). Each site has been carefully configured by enabling the SuperB VO, installing the applications software and registering the required input files in their Storage Elements.

The two sessions related to FastSim and FullSim use cases have been configured in bookkeeping DB, subsequently the productions and the requests have been initialized.

The framework has been successfully used for intense production cycles of both Full- and Fast Simulation. More than 11 billion simulated events have been produced. Over an effective period of 4 weeks, approximately 180000 jobs were completed with a 8% failure rate., mainly due to executable errors (0.5%), site misconfigurations (2%), proxy expiration (4%), and temporary overloading of the machine used to receive the data transfers from the remote sites (2%). The peak rate reached 7000 simultaneous jobs with an average of 3500. The total wall clock time spent by the Monte Carlo applications is 195 years.

In the next paragraphs the results from two production cycles will be reported in details.

5.1 First simulation production cycle

During this production phase, over 1.7 billion simulation events, equivalent to $\sim 0.2ab^{-1}$(Inverse attobarn[1]), have been produced using the distributed computing environment.

The Full Simulation production has been divided in two categories: simulation of background frames for Fast Simulation (bgframes) and full simulation for background studies (bgstudies).

The background frames production consists in just one Production_Series, named 'cycle1_bgframes'. The full simulation for background studies consists in four Production_Series depending on geometry, physics list and generator's parameter MinDeltaEnergy: cycle1_full_HP, cycle1_full_DeltaE_0.002, cycle1_full_DeltaE_0.05, and cycle1_full.

The entire Full Simulation production has been run at CNAF in about 12 days. Table 5.1 shows the production summary.

[1]Inverse attobarn: is a unit of area $10^{-46}m^2$ originally used in nuclear physics for expressing the cross sectional area of nuclei and nuclear reactions, today it is also used in all fields of high energy physics to express the cross sections of any scattering process, and is best understood as a measure of the probability of interaction between small particles.

Table 5.1: Cycle 1: Full Simulation Production Summary

cycle1_bgframes

Geometry	Status	Jobs	Events
SuperB	done	4000	10^6
SuperB	failed	1	250
SuperB	sys-failed	1	250

cycle1_bgstudies

Geometry	Status	Jobs	Events
shielded	done	4840	604000
shielded	sys-failed	1	100
unshielded	done	785	196250
unshielded	failed	2	500
unshielded	sys-failed	13	3250

The Fast Simulation production has been divided in two types (requests): simulation of "Generics" and "Signal Mode" events.

For Generics, four generators (B0B0bar, B+B-, ccbar and uds) and three geometry configurations have been used. Events with- and without- background mixing have been produced. In the "Signal Mode" production four physics channels events have been simulated (BtoKNuNu, KplusNuNu, BtoKstarNuNu, and BtoTauNu) with background mixing.

Results are summarized in table 5.2.

Fast Simulation production has involved nine remote sites; about 82% of submitted jobs have used the Grid infrastructure and/or exploited remote resources, as illustrated in figure 5.1 for the Generics production.

Over a period of 2 weeks, approximately 20000 jobs were completed with an average $\sim$ 8% failure rate, mainly due to site misconfigurations (2.6%), proxy expiration (2.0%), and temporary overloading of the machine used to receive the data transfers from the remote sites at CNAF (3%). The peak rate reached 3200 simultaneous jobs with an average of 500.

Table 5.2: Cycle 1: Fast Simulation Production Summary

cycle1_Generics

Bkg	Jobs	Events
Y	4508	104.340×10^6
N	14672	1472.100×10^6
Total	19180	$\approx 1.58 \times 10^9$

cycle1_Signal

Signal	Jobs	Events
BtoTauNu	30	3×10^6
BtoKNuNu	60	6×10^6
BtoKstarNuNu	60	6×10^6
BtoKplusNuNu	688	68.8×10^6
Total	838	83.8×10^6

5.2 Second simulation production cycle

During this production cycle over 10.3 billion simulated events have been produced.

The Full Simulation production has been devoted to produce 1M of RadBhaBha background[2] events to be then used by the Fast Simulation. Only CNAF resources have been used for 8 days. Fast Simulation has been used then to produce Pairs background events at CNAF for a couple of days.

After this preliminary background production, efforts have been devoted to the effective Fast Simulation production in the distributed environment.

The Fast Simulation production has been divided in several 'analyses', each consisting of several Requests. The analysis groups have simulated events that will be used for a specific physics analysis and requires different generators and geometries. In addition, RadBhaBha background mixing has

[2]RadBhaBha is a generator of simulated background frames. Such background events need to be taken in consideration during the complete collision simulation in FastSim methodology

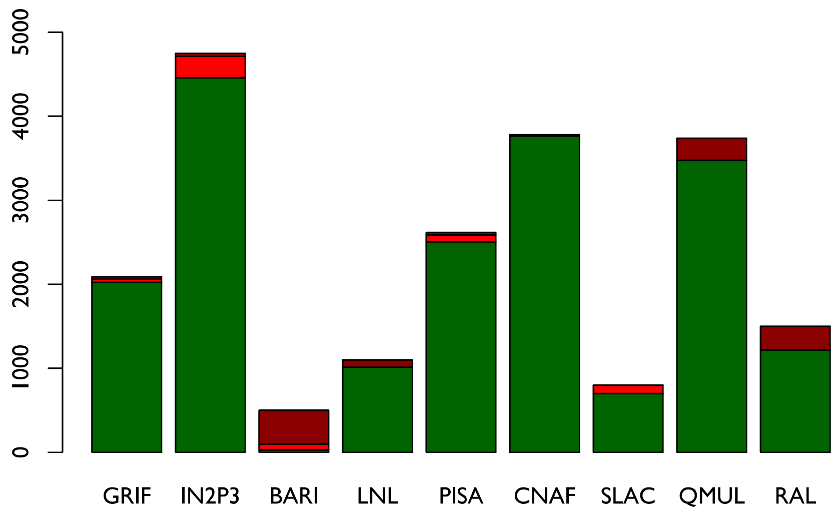


Figure 5.1: Jobs submitted to remote sites for the Generics Production.

been always used while pairs mixing has been used for selected datasets.

Table 5.3 summarizes the produced analyses and requests (all possible combinations of generator and geometry and mixing).

Fast Simulation production has involved 15 remote site; figure 5.2 shows the distribution of produced events among remote sites. About 69% of submitted jobs have used the Grid infrastructure and/or exploited remote resources. This value is lower than the first cycle because the CNAF farm has been upgraded allowing the exploitation of more slots and also because at the end of the cycle jobs have been submitted to the local CNAF queue only.

Over an effective period of 4 weeks, approximately 162000 jobs were completed with a $\sim$ 10% failure rate, mainly due to executable errors (0.5%), site misconfigurations (2%), proxy expiration (5.0%), and temporary overloading

Table 5.3: Cycle 2: Fast Simulation Analysis and Requests Summary

Analysis	Generator	Geometry	Bkg
	B+B-*	DG_4	All
HadRecoilCocktail	B0B0bar*	DG_4a	NP
		DG_Babar	
	B+B-*	DG_4	All
SLRecoilCocktail	B0B0bar*	DG_4a	NP
		DG_Babar	
Generics	B+B-*	DG_4	All
	B0B0bar*	DG_4a	NP
BtoKNuNu	B+B-_K+*	DG_4	All
	B0B0bar_K0*	DG_4a	NP
BtoKstarNuNu	B+B-_Kstar+*	DG_4	All
	B0B0bar_Kstar0*	DG_4a	NP
	B+B-_K+*		
	B+B-_Kstar+*	DG_4	All
KplusNuNu	B0B0bar_K0*	DG_4a	NP
	B0B0bar_Kstar0*		
	B+B-_tau*		

of the machine used to receive the data transfers from the remote sites at CNAF (2%). The peak rate reached 6000 simultaneous jobs with an average of 3200. The total WCT spent by the simulation executables is $\sim$ 195 years.

It is important to notice that in the second production cycle, most of the requests have been run several times with different implementations of the background mixing code for pairs, which have different performances and levels of precision. Figure 5.3 shows the histograms (bins of 1 hour) of jobs status with respect to time for all the production period (above) and for the last 2 weeks (below); the most reliable events are those of the last production period, and correspond to $\sim 40\ ab^{-1}$.

In this last period the failure rate is relatively smaller ($\sim$ 6%) than the average over the entire production period because site misconfigurations and proxy-related issues were solved.

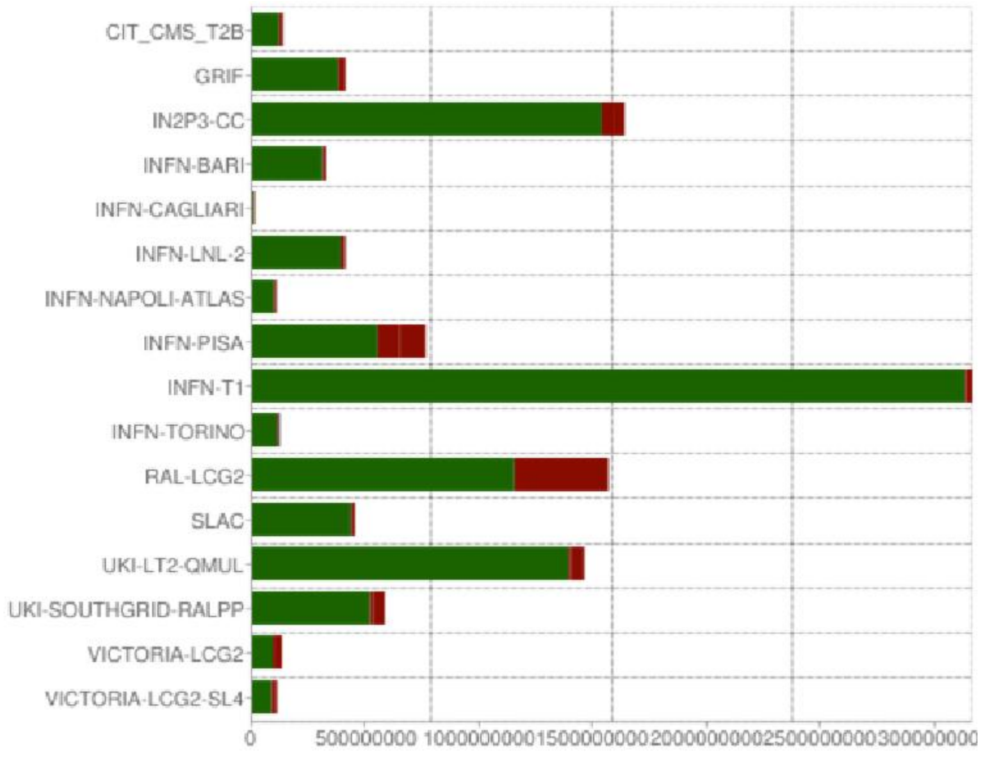


Figure 5.2: Jobs submitted to remote sites for the Fast Simulation production during second cycle.

5.3 Summary of results and conclusions

Table 5.4 summarizes the results of the cycle 1 and 2 simulation productions; a test production which has been performed only locally at CNAF is reported for comparison.

The system has proven to be reliable and efficient although it still required a careful initial configuration in terms of Session definition and environment of distributed computing sites.

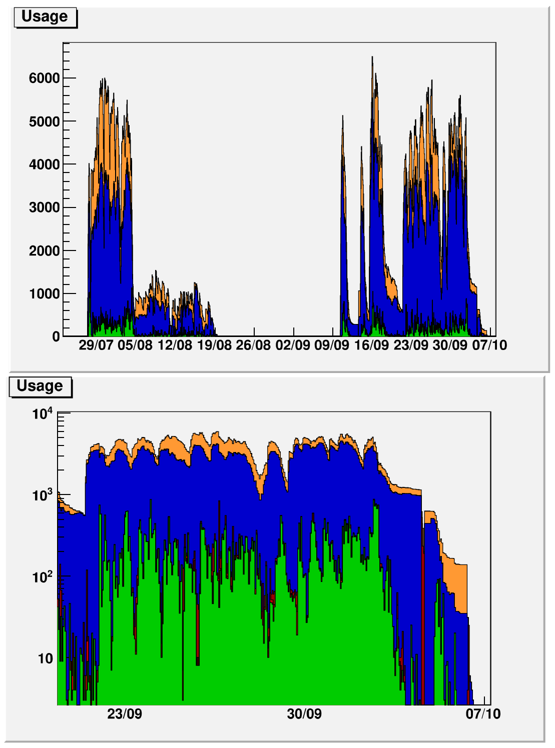


Figure 5.3: Histograms of jobs submitted (orange), running (blue), done (green) and failed (red) during the entire second cycle Production (above) and detail of the last 2 weeks (below). 1 hour binning.

Table 5.4: Summary of the SuperB simulation productions

	Test prod	Cycle 1	Cycle 2
Statistic (ab^{-1})	0.1	1	> 40
Analysis Streams	2	5	6
Jobs done	5K	20K	160K
Failure rate	10%	8%	10%
Events	2.25×10^8	1.6×10^9	1.03×10^{10}
Sites	1	9	15
WCT (years)	6	19	195
Disk (TB)	0.5	5	25

Chapter 6

Conclusion

The distributed computing paradigm has evolved in the last 10 years under the push, also, of High Energy Physics communities, becoming the computational model in several research and private projects. Nowadays, small and medium size communities are still approaching Grid and Cloud infrastructures asking for tools capable to provide a simplified resources exploitation and monitor.

This work described the design and implementation of a lightweight framework for automatic Grid submission to the Grid science infrastructure. It provides a set of services allowing the management of job submissions and monitor. Moreover, it provides a bookkeeping database layer that can be dynamically adapted to fulfill VO requirements. These characteristics are distinctive of the project. During the dissertation a detailed description of the framework workload has been provided, it includes the job workflow, the submission strategy and the job wrapper functionality. A conspicuous part of the document has been dedicated to the data and metadata management explanation as for the bookkeeping DB which is the key element allowing the tool suite to be completely independent by the VO specificity. The job wrapper and the operations web portal, besides the bookkeeping DB completes the set of main framework bricks providing an efficient distributed resource exploitation, advanced monitor functions and a user friendly VO interface.

Integration of general-purpose framework with Dirac provided a new Dirac system flavor capable to tight together VO specific operations and Grid job management in a single framework. The Dirac workload management and the VO operations in terms of application and metadata management can now interact directly in the same system. This design guarantees the highest level of synergy and subsequently an increased level of usability and robustness. At present time Dirac provides several Cloud interfaces such as the Amazon Elastic Compute Cloud (EC2), OpenStack, OpenNebula and CloudStack.

Other projects with similar objectives can be found in literature. For instance, WS-PGrade/gUSE [79] is a complete web portal solution for the development, execution and monitoring of workflows and workflow based parameter studies on various Grid platforms. It hides low-level Grid access mechanisms and is used by other projects as their base framework. ComputER [80] and SHIWA [81], for example, offer Grid access to scientific communities by customizing and developing new features on top of WS-PGrade.

The framework has been successfully used in a 15 Grid site environment, producing 35TB of simulated data during one month of operations. Furthermore, the suite has engendered the interests of different groups from biology to astrophysics fields.

Acknowledgments

I want to thank my family, who allowed me to carve out time to write this thesis. Especially my wife Ilaria and the grandparents who have bent over backwards to handle children, Margherita almost three years old and Mattia only 10 months old.

Appendix A

Glossary

BDII: Berkeley Database Information Index

CA: Certification Authority

CE: Computing Element

CERN: European Laboratory for Particle Physics

dcap: dCache Access Protocol

DIT: Directory Information Tree

DN: Distinguished Name

GAS: GRID Access Service

GIIS: GRID Index Information Server

GMA: GRID Monitoring Architecture

GRIS: GRID Resource Information Service

GSI: GRID Security Infrastructure

gsidcap: GSI-enabled version of the dCache Access Protocol

gsirfio: GSI-enabled version of the Remote File Input/Output protocol

GUID: GRID Unique ID

INFN: Istituto Nazionale di Fisica Nucleare

IP: Information Provider

IS: Information Service

ISM: Information Super Market

JDL: Job Description Language

LB: Logging and Bookkeeping Service

LDAP: Lightweight Directory Access Protocol

LFC: LCG File Catalog

LFN: Logical File Name

LHC: Large Hadron Collider

LCG: LHC Computing GRID

LRC: Local Replica Catalog

LRMS: Local Resource Management System

MDS: Monitoring and Discovery Service

MPI: Message Passing Interface

MSS: Mass Storage System

PFN: Physical File name

PID: Process IDentifier

RB: Resource Broker

RFIO: Remote File Input/Output

R-GMA: Relational GRID Monitoring Architecture

RLI: Replica Location Index

RLS: Replica Location Service

RM: Replica Manager

RMC: Replica Metadata Catalog

RMS: Replica Management System

SE: Storage Element

SMP: Symmetric Multi Processor

SURL: Storage URL

TQ: Task Queue

TURL: Transport URL

UI: User Interface

URI: Uniform Resource Identifier

URL: Universal Resource Locator

UUID: Universal Unique ID

VO: Virtual Organization

VOMS:	Virtual Organization Membership Service
WMS:	Workload Management System
WN:	Worker Node

Bibliography

[1] F. D. Paoli, "Cdf way to the grid", Journal of Physics: Conference Series, vol. 53, no. 1, p. 413, 2006.

[2] C. Brew, F. Wilson, G. Castelli, T. Adye, E. Luppi and D. Andreotti, "Babar experience of large scale production on the grid," e-Science and Grid Computing, International Conference on, vol. 0, p. 151, 2006.

[3] M. C. Vistoli et al., "Prototyping production and analysis frameworks for lhc experiments based on lcg/egee/infn-grid middleware," in Computing in High Energy and Nuclear Physics, 2006.

[4] The Large Hadron Collider "http://home.web.cern.ch/about/accelerators/large-hadron-collider"

[5] LHC Computing Grid home page "https://twiki.cern.ch/twiki/bin/view/lcg/."

[6] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," Int. J. High Perform. Comput. Appl., vol. 15, pp. 200-222, August 2001.

[7] National Institute of Standards and Technology Special Publication 800-145, 7 pages (September 2011)

[8] M. Bona et al., "SuperB: A High-Luminosity Asymmetric e+ e- Super flavor Factory. Conceptual Design Report," 2007.

[9] Dirac project home page, "http://diracgrid.org."

[10] The LCG TDR Editorial Board, *LHC Computing Grid, Technical Design Report* LCG-TDR-001, CERN-LHCC-2005-024 (2005).

[11] T. Dimitrakos et al, "Towards a Trust and Contract Management Framework for Dynamic Virtual Organizations". In Proceeding of the e-Challanges

[12] Worldwide LHC Computing Grid, "http://wlcg.web.cern.ch/"

[13] EGEE project home page, "http://www.eu-egee.org".

[14] OSG project home page, "http://www.opensciencegrid.org"

[15] Nordic Data Grid Facility (NDGF), "http://www.ndgf.org"

[16] gLite middleware, "http://glite.web.cern.ch/glite/."

[17] SRM working group, "http://sdm.lbl.gov/srm-wg/"

[18] dCache project home, "http://www.dcache.org/"

[19] DPM project home page
"https://svnweb.cern.ch/trac/lcgdm/wiki/dpm"

[20] VOMS project home page
"http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html."

[21] FTS project fundations
"https://edms.cern.ch/document/591792/1"

[22] M. flechl and L. field, "Grid interoperability: Joining grid information systems," J. Phys. Conf. Ser., vol. 119, p. 062030, 2008.

[23] LCG File Catalog admin's guide
"https://twiki.cern.ch/twiki/bin/view/LCG/LfcAdminGuide"

[24] User's Guides for the DGAS Services
https://edms.cern.ch/document/571271"

[25] User's Guide for the VOMS Core Services "https://edms.cern.ch/document/571991/1"

[26] VOMS admin user's guide, "https://edms.cern.ch/document/572406/1"

[27] JDL Attributes Specification, "https://edms.cern.ch/document/555796/1"

[28] WMS User's Guide, "https://edms.cern.ch/document/572489/1"

[29] LB Service User's Guide, "https://edms.cern.ch/document/571273/1"

[30] User Guide For Edg Replica Manager 1.5.4, "http://cern.ch/edg-wp2/replication/docu/r2.1/edg-replica-manager-userguide.pdf"

[31] Developer Guide For Edg Replica Manager 1.5.4, "http://cern.ch/edg-wp2/replication/docu/r2.1/edg-replica-manager-devguide.pdf"

[32] Sergio Andreozzi. Glue schema. "http://glueschema.forge.cnaf.infn.it/"

[33] LB Service User's Guide, "https://edms.cern.ch/document/571273"

[34] EGI-Inspire home, "https://www.egi.eu/about/egi-inspire/"

[35] Herr, W.; Muratori, B. (2006). "Concept of luminosity". In Brandt, D. CERN Accelerator School: Intermediate Course on Accelerator Physics, Zeuthen, Germany, 15-26 Sep 2003. CERN. pp. 361-378. doi:10.5170/CERN-2006-002. ISBN 978-92-9083-267-6.

[36] CabibboLab home page, "http://www.cabibbolab.it/"

[37] Fishman, G. S. (1995). Monte Carlo: Concepts, Algorithms, and Applications. New York: Springer. ISBN 0-387-94527-X.gdml

[38] Full Simulation application: Bruno "http://appdb.egi.eu/store/software/bruno"

[39] J. Apostolakis et al., GEANT4: Status and recent developments, Proceedings of CHEP'04, Interlaken, Switzerland, 24 Sep-1 Oct 2004, CERN-2005-02, Vol 1, p.199

[40] R. Andreassen et al., "Fastsim: fast simulation of the SuperB detector," in Proceeding of the IEEE NSS-MIC 2010 conference, 2010.

[41] The BaBar Detector "http://www.slac.stanford.edu/BFROOT/www/doc/workbook /detector/detector.html"

[42] Geometry Description Markup Language (GDML) "http://gdml.web.cern.ch/GDML"

[43] R. Brun and F. Rademakers, ROOT-An Object Oriented Data Analysis Framework, Nucl. Inst.&Meth. in Phys.Res.A389(1997)81-86.

[44] NAGIOS home page, "http://nagios.org"

[45] Service Availability Monitoring system "https://wiki.egi.eu/wiki/SAM"

[46] LDAP system, "http://www.openldap.org"

[47] F. Brochu, U. Egede, J. Elmsheuser, K. Harrison, R. W. L. Jones, H. C. Lee, D. Liko, A. Maier, J. T. Moscicki, A. Muraru, G. N. Patrick, K. Pajchel, W. Reece, B. H. Samset, M. W. Slater, A. Soroko, C. L. Tan, and D. C. Vanderster, "Ganga: a tool for computational-task management and easy access to grid resources," CoRR, vol. abs/0902.2685, 2009.

[48] E. Corso, S. Cozzini, A. Forti, A. Ghiselli, L. Magnoni, A. Messina, A. Nobile, A. Terpin, V. Vagnoni, and R. Zappi, "StoRM: A SRM Solution on Disk Based Storage System," in Proceedings of the Cracow Grid Workshop 2006 (CGW2006), Cracow, Poland, 2006.

[49] Hadoop apache project, "http://hadoop.apache.org/"

[50] Condor Project Homepage, "http://www.cs.wisc.edu/condor/"

[51] D. Erwin (Ed.), UNICORE Plus final Report - Uniform Interface to Computing Resources, Forschungszentrum JÃ$\frac{1}{4}$lich, 2003.

[52] Lustre file system home page, "http://www.lustre.org"

[53] Blomer J *et al* 2012 Status and future perspectives of CernVM-FS *J. Phys.: Conf. Ser.* **396** 052013

[54] U. Egede and T. M. Jakub, Working with GANGA 5.0. 2009.

[55] Koziol, Conrad (12 September 2005). "Introducing IPython"

[56] fielding R.T., *Architectural Styles and the Design of Network-based Software Architectures* , PhD. Thesis, University of California Irvine, 2000.

[57] Aiftimiei C, Andreetto P, Bertocco S, Dalla Fina S, Dorigo A, Frizziero E, Gianelle A, Marzolla M, Mazzucato M, Sgaravatto M, Traldi S, Zangrando L 2010 Design and implementation of the gLite CREAM job management service *Future Generation Computer Systems* Volume **26** Issue 4 pp 654-667, doi: 10.1016/j.future.2009.12.006.

[58] Phprestsql software layer, "http://phprestsql.sourceforge.net/"

[59] Gridsite community "https://www.jiscmail.ac.uk/cgi-bin/webadmin?A0=GRIDSITE-DISCUSS"

[60] Grid site gacl specification "http://www.hep.man.ac.uk/website/admin.cgi?cmd=print&file=config.html"

[61] Grid File Access Library, "https://svnweb.cern.ch/trac/lcgutil/wiki/gfal2"

[62] "jQuery: The write less, do more, JavaScript library". The jQuery Project. Retrieved 29 April 2010.

[63] A. Jardine, "datatables jquery plug-in.", "http://datatables.net"

[64] The LHCb collaboration ”http://lhcb.web.cern.ch/lhcb”

[65] VMDirac project, ”https://github.com/DIRACGrid/VMDIRAC/wiki”

[66] Belle2 Collaboration web portal, ”http://belle2.kek.jp/”

[67] Amazon Elastic Compute Cloud (Amazon EC2) ”http://aws.amazon.com/ec2”

[68] Open Cloud Computing Interface, ”http://occi-wg.org/”

[69] OpenStack Open Source Cloud Computing Software ”http://www.openstack.org/”

[70] OpenNebula project, ”http://opennebula.org/”

[71] Apache CloudStack, ”http://cloudstack.apache.org/”

[72] BOINC project, http://boinc.berkeley.edu/

[73] VirtualBox project, ”https://www.virtualbox.org/”

[74] Psycopg project, ”http://initd.org/psycopg/”

[75] SQLAlchemy project, the Python SQL Toolkit and Object Relational Mapper, ”http://www.sqlalchemy.org/”

[76] The GNU Lesser General Public License, version 3.0 (LGPL-3.0), ”http://opensource.org/licenses/lgpl-3.0.html”

[77] The MIT License (MIT), ”http://opensource.org/licenses/MIT”

[78] Pylons project, ”http://www.pylonsproject.org/”

[79] Liferay technology ”https://guse.sztaki.hu/liferay-portal-6.0.5/welcome”

[80] P.Veronesi et al., ”Multidisciplinary approach for computing in Emilia Romagna (Italy)”, EGI User Forum, 11-14 april 2011, Vilnius, Lithuania.

[81] Shiva project, ”http://www.shiwa-workflow.eu/home”