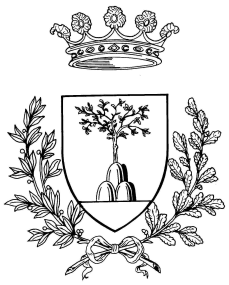


Università degli Studi di Ferrara

Dottorato di ricerca in Matematica e Informatica

Ciclo XXI

A Grid architectural approach applied for backward compatibility to a production system for events simulation

Settore Scientifico Disciplinare INF/01

Dottorando:
Dott. Andreotti Daniele

Tutore:
Prof. Luppi Eleonora

Anni 2006/2008

Università degli Studi di Ferrara

Dottorato di ricerca in Matematica e Informatica

Ciclo XXI

Coordinatore: Prof. Luisa Zanghirati

A Grid architectural approach applied for backward compatibility to a production system for events simulation

Settore Scientifico Disciplinare INF/01

Dottorando:
Dott. Andreotti Daniele

Tutore:
Prof. Luppi Eleonora

Anni 2006/2008

Contents

Abstract

Distributed systems paradigm gained in popularity during the last 15 years, thanks also to the broad diffusion of distributed frameworks proposed for the Internet platform. In the late '90s a new concept started to play a main role in the field of distributed computing: the Grid.

This thesis presents a study related to the integration between the BaBar's framework, an experiment belonging to the High Energy Physics field, and a grid system like the one implemented by the Italian National Institute for Nuclear Physics (INFN), the INFNGrid project, which provides support for several research domains.

The main goal was to succeed in adapt an already well established system, like the one implemented into the BaBar pipeline and based on local centers not interconnected between themselves, to a kind of technology that was not ready by the time the experiment's framework was designed. Despite this new approach was related just to some aspects of the experiment, the production of simulated events by using Monte-Carlo methods, the efforts here described represent an example of how an old experiment can bridge the gap toward the Grid computing, even adopting solutions designed for more recent projects.

The complete evolution of this integration will be explained starting from the earlier stages until the actual development to state the progresses achieved, presenting results that are comparable with production rates gained using the conventional BaBar's approach, in order to examine the potentially benefits and drawbacks on a concrete case study.

Introduction

Grid systems gained in popularity during the last decade as one of the emerging frontier for distributed computing. Benefits for experiments of last generation, which architecture is already organized since the early stages around such a kind of framework, are evident. Going back to that class of projects born before the grid coming, issues concerning feasibility of integration and architecture constrains arise. The BaBar experiment, organized at the Stanford Linear Accelerator Stanford, falls inside this category since its framework was developed in early '90s.

Contributions

In this thesis is illustrated the approach followed to integrate the existing BaBar model for the production of simulated events to the grid environment, in order to verify the degree of adaptation of a traditional and already well consolidated framework to the facilities offered by grid mechanisms. The main goal, therefore, is to obtain a new hybrid system built using components of both worlds and capable to achieve the same results in terms of number of events produced weekly, but providing an easier management in terms of resources and costs. Despite this study is performed on a specific project and considering that different experiments could require different solutions, the work proposed provides some approaches that can be easily adopted by other cases.

Thesis structure

This thesis is organized in five main sections, which core is represented by the range starting from Chapter 3 until Chapter 5. In the first part of Chapter 1 an overview of the distributed computing paradigm is provided to recall the main concepts related to this branch of the computer science. A classification based on different kinds of architecture is then discussed to introduce the second part of the chapter focused on Grid systems, where a taxonomy is given in terms of internal organization.

A concrete example of a grid system is described in Chapter 2, where the INFN-Grid project is presented and discussed in detail. In particular, the main components of this system will be outlined, like data management, bookkeeping and security, in order to give a comprehensive idea of the scenario used as background platform for this project.

Chapter 3 illustrates the BaBar framework implemented at SLAC, describing the evolution of the computing model and the mechanisms adopted by the traditional simulated Monte-Carlo events production. Information provided here help to understand the choices that brought to the definition of a new production system, BaBarGrid, that is explained in detail in Chapter 4.

The new production system is presented pointing out how the constrains of both environments, Grid and BaBar, have been integrated in a full working solution suitable for the simulation production of the experiment. Finally, in Chapter 5, a description about implementation details is given along with a summary of results achieved. In Chapter 6 a general overview of the work done is discussed outlining motivations and open problems.

Chapter 1

Distributed systems

In this chapter will be introduced a general description of what distributed systems are along with a discussion of the most common architectures implemented underneath. Moreover, a topology is then outlined in terms of information flow, taking into account aspects like resource availability and discovering.

In the second part of the chapter grid systems are introduced, outlining a general taxonomy for this category.

1.1 Overview

Distributed systems represents a field of the computer science that rapidly changed in last two decades, especially thanks to the great improvements achieved in two research domains like microprocessors and computer networks. From the mid '80s advance in technology brought the CPU's field to gain an unexpected level of performances, through an evolution from 8/16-bit machines to 32/64-bit architectures that quickly outperformed old mainframes in terms of costs/benefits.

Research in the network field also led to high-speed computer networks, from Local Area Networks able to connect hundreds of machines within a local domain, where fast links of 100 Mbits or more move huge amount of data, to Wide Area Networks which interconnect million of nodes spread over the world by using a variety of links at different speeds [1].

Different typologies of distributed systems have been implemented to achieve that degree of abstraction necessary to present a heterogeneous collection of machines

as it was a single coherent system. In order to support this feature providing at the same time a single-system view, distributed systems are usually organized into logical layers architectures where a key role is played by the middleware layer. The middleware extends over multiple machines hiding the underneath layer represented by operating systems and basics communication facilities, while on the other side provides a common interface for all applications of the above layer. Distributed systems typically have, among other, the following characteristics:

- Allow different vendors and operating systems to inter-operate
- Communication among components are hidden from users
- Provide an architecture which can be expanded easily
- Supply a good level of reliability even though some parts may be unavailable

1.2 Types of distributed systems

Nowadays distributed systems implementations come in a variety of flavors, if considering the kinds of architecture implemented beneath. From the point of view of the information flow, a topology can be defined on the basis of the organization of services and resources and how they can be discovered and accessed. In the following section will be discussed the common solutions adopted both by centralized and decentralized managements, by using graphs where nodes shown represent system's components which exchange information through a network link when connected by an edge [2].

Several proprieties can be discussed when evaluating distributed systems design, therefore the following characteristics will be taken into account in order to give an overview of the most common:

- Manageability: represents the easy of use of a system, including all the fundamental management operations needed to keep the whole system working

- Fault Tolerance: the capacity for a system to deal with different kind of failures maintaining an acceptable level of stability
- Security: this property involves several topics, like users authorization, information integrity, data protection
- Scalability: represents the capacity of the system to grow in size, resulting in an extension of its potentials maintaining at the same time an equal degree of reliability
- Information coherence: is related to the information provided by the system in terms of data consistency, non-repudiation and reliability

1.2.1 Basic distributed systems topologies

Centralized systems represent one of the most well known organization, generally implemented following the classic client/server pattern, where all functionalities are collected on a dedicated server accessed by several clients. Despite this architecture is considered one of the oldest, it is still used in a variety of applications and it was also integrated in the model of the first generation of p2p applications, in relation to data look-up. The ease of management comes at the price of a single point of failure schema which also implies scalability issues.

Hierarchical systems implement an architecture where authority flows down from the root node toward the leaves. The main advantages is represented by the high level of scalability they can achieve, as a concrete example like DNS demonstrated during past years [3]. Despite this typology is more fault-tolerant than centralized systems, it still introduces a single point of failure for the root node and is generally harder to secure compared to a centralized system since when high level nodes are compromised the whole system can be damaged. Coherence is generally achieved by means of cache mechanisms.

Decentralized systems consist of groups of node acting as peers, where each component of the system, in theory, have the same role and communicate symmetrically to each other. A first quite popular implementation of a pure decentralized system

was introduced by the Gnutella network, which demonstrated how easily new nodes can join and leave the system but also highlighted the several drawbacks of such kind of solution. Security is an issue for these systems due to the lack of control on information introduced by nodes, moreover scalability can become an issue if the overly-network management implemented is not able to maintain coherent the whole system, avoiding to carry out too much overhead. However, decentralized systems offer a high degree of fault-tolerance since the failure of one particular node doesn't impact the rest of the system. Nowadays p2p evolution brought to new solutions which allowed to improve the efficiency of fundamental functionalities for a such kind of systems, like datalookup , leading to the development of dedicate protocols like CAN, Chord and Pastry implemented in popular applications for file sharing and communication.

Naturally, other kinds of architectures are implemented as hybrid solutions of those discussed above, like for example the so called brokered systems which can broker specific functionalities [4]. In practice, several factors can determine the nature of a system since even those that are considered heavily decentralized can implement centralized solutions for specific purposes.

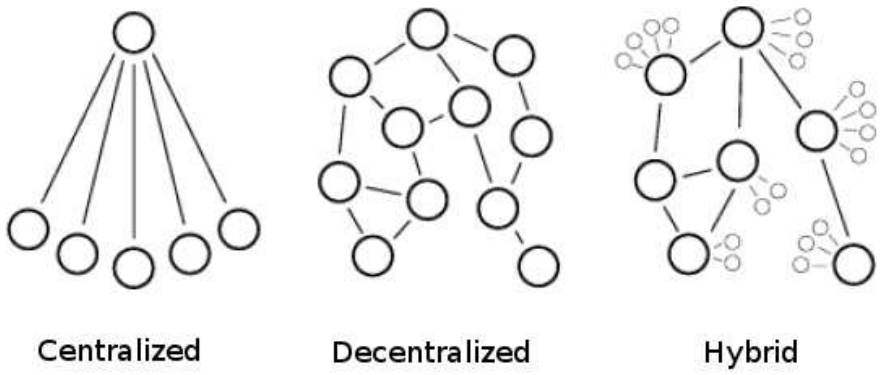


Figure 1.1: Example of distributed systems architectures [3]

In order to determine whether a system is centralized or decentralized, three main areas can be further considered: resource discovery, resource availability and

resource communication. Resource discovery can be provided both centrally or not, depending on the target must be achieved, furthermore it can be subdivided into two other aspects like location information content and information retrieving methods. Information access, for example, can be centralized while methods of how information is retrieved can be decentralized, as a well known example, the DNS, demonstrates. Decentralization plays a more important role in how data are localized than they are accessed due to the need for users to find required information quickly.

Along resource discovery another very closely tied aspect is the availability of resources. In a centralized system resources are collected at a specific site resulting in a single point of failure, while decentralized systems offer many duplicates for supplying the same functionalities.

Resource communication within a distributed system can be considered as two different approaches: point to point or brokered. The first solution involves direct connection between two components, despite the fact the link can be multi-hop. In the latter a central server is placed in the middle of the communication to keep in sync resources that cannot reference to each others directly.

1.2.2 P2P and distributed systems

During last decade p2p systems gained popularity thanks to the implementations supplied for many well known applications. p2p networks allow to connect on large scale transient devices providing robust solutions for different application domains, working using a dedicated overlay network in a complex environment.

P2P systems belong to the full decentralized category, overlapping part of the topologies scenario discussed previously and spreading network and resource demands across the network. Although their decentralized nature, even p2p systems can provide a partial centralized management even if there is no global notion of centralization from the point of view of single peers. More important, the p2p technology led to the development of several applications in different domains, providing new solutions for the distributed systems field.

1.3 Grid systems

The "Grid" concept is based on the idea of sharing different kinds of resources, therefore different typologies of grids can be defined. From the point of view of pure computation, the computational grid category denotes the possibility for a system to aggregate from different sources, even belonging to different domains, a high amount of computational capacity which would be difficult to achieve from any single machine connected to the system.

Depending on the application domain involved, this category can further subdivided into two sub-categories: high performance computing and high throughput computing grids. In the former case, the target requires a strict cooperation between the most part of the pool of machines connected to the system in order to solve a challenging task as quickly as possible. Typical applications are weather modeling that require an intensive computing process to deliver results in time.

On the other hand, many applications don't require to solve a specific problem where time is such a limiting constraint, but instead need to maintain a high throughput rate for completing as many jobs as possible. One of the most diffuse application in this field concerns Monte-Carlo simulations, which production architecture perfectly fits this kind of model.

1.3.1 Grid taxonomy

Scientific experiments led grid architectures to rapidly change in order to adapt to specific requirements, preserving at the same time a standard set of features suitable for being applied to different research domains. A general grid taxonomy involves several aspects depending on the kind of purpose it is focused on, therefore the most common views about data, computation and application will be taken into account in this section, highlighting just those parts directly involved in the aim of this thesis.

Nowadays many research domains are supported by grids which type can be associated to one of these categories since the development of a multi purpose grid, able to provide effective solutions for each of the above fields, still remains an open

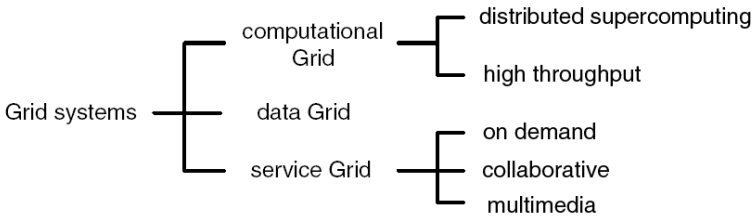


Figure 1.2: Example of grid systems taxonomy [5]

issue. For example, one of the main difference between data and application grid is related to specialized infrastructure the former category must provide for data access and storage management, which requires dedicate solutions not offered by the standard set of services.

1.3.2 Data Grid organization

Data grid can be considered as a branch derived from the evolution of the Web, since both systems were designed to share a huge amount of data to be distributed among domains at different locations. The main difference lies in that pool of instruments the grid offers to elaborate data, giving the possibility to compare information among different domains, providing also those mechanisms to standardize the meanings associated to those information.

Data grid architectures require a complex organization since different aspects must be taken into account. The model implemented to support a data grid infrastructure represents the way data sources are arranged [5], which depends on different factors like data size, the kind of sharing mode adopted and distribution of data sources. A selection of most popular models is shown in figure... and represents a collection of strategy is implemented by different projects to effective solve the organization task:

- Monadic: the main characteristic of this solution consists of a single point for collecting data. Only one main repository is implemented while data can be collected from different sources, even geographically distributed. Repository

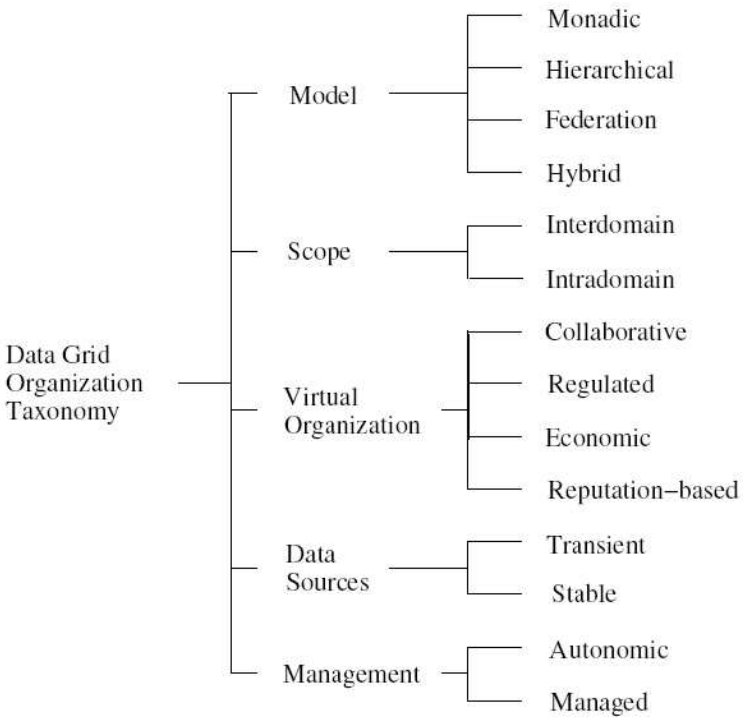


Figure 1.3: Data model organization [5]

replication is allow just to improve fault tolerance, due to the one single point of failure architecture, but data cannot be replicated on other locations to improve data access. Despite all constrains that a such solution presents, it has been adopted in those cases where all accesses were local to a specific area

- Hierarchical: this model implements a distributed approach and spreads data over other centers belonging to the same collaboration. An example is given by the MONARC (Models of Networked Analysis at Regional Centres) [6] group located at CERN that proposed a tiered architecture for data distribution. A main center, at the top of the chain, collects all data generated into the main repository that represents a reference for all smaller centers. Portions of the repository are then subdivided among sub-nodes, classified as lower levels of the chain: these kind of institutions generally don't need the whole set of data

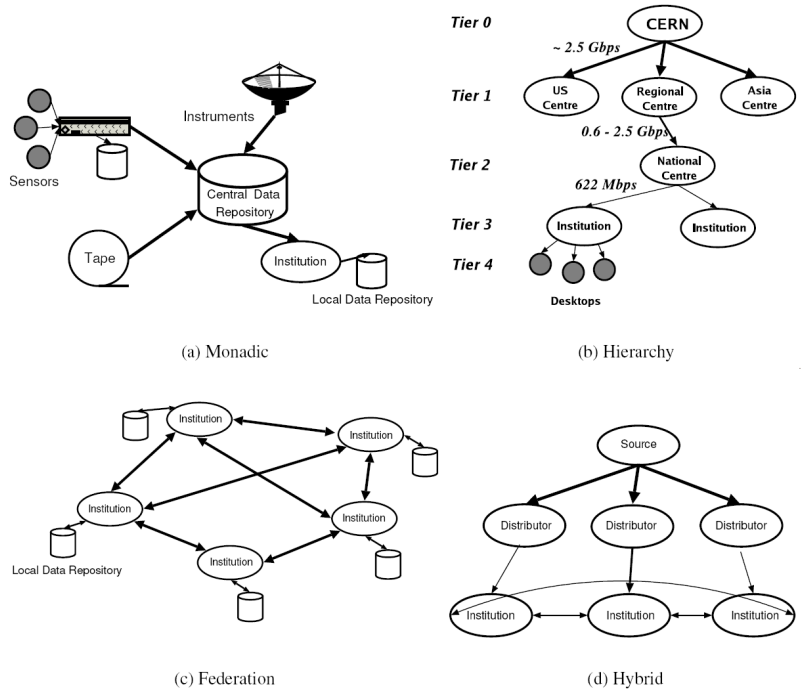
but are interested just to a portion. Walking down toward the bottom of the graph, sites become smaller in terms of requirements like storage capacity and network bandwidth. Several experiments in the field of High Energy Physics, like CMS and BaBar, adopted this kind of model which one advantage is to preserve consistency quite easily due to a single main point where data are collected. Naturally, to improve data access hybrid solution can be used, as for example, to replicate the main repository at those sites that fulfill all requirements to become a mirror of the main center, like in the case of BaBar where, through dedicated tools purpose-made, the central repository has been replicated also to other institutions in Europe.

- Federation: this kind of approach is quite common since is based on the reuse of already installed resources. As example, many institutions like to share information using local databases previously adopted for different tasks, building a data network with other external sites which allow users to demand data from whatever node connected. Each node is responsible for providing mechanisms for accessing data and safely check users credentials. Different kinds of federations are possible depending on sites local policies and suitable mechanisms to define a uniform interface to access data shared on a heterogeneous pool of nodes

- Hybrid: models belonging to this category combine concepts described in the previous architectures and their evolution is driven by the experience gained by users on different application domains to solve common problems. An example of peer-to-peer communication integrated into a hierarchical/federated model is shown in Figure 1.4

Another key element of the general taxonomy is represented by the organizations that operate on the grid, which depending on their internal polity can be classified as follow:

- collaborative: when entities belonging to the same organization work together



(a) Monadic (b) Hierarchy

(c) Federation (d) Hybrid

Figure 1.4: Data Grids taxonomy [5]

sharing resources to reach a common goal, after specific policies about resources usage have been established

- regulated: a single organization establishes rules for accessing resources to all other participants, defining remissions and rights

- economy-based: are based on a profit mechanism where consumers contact producers on the base of quality of services they offer, therefore rules follow the level of agreement among participants

- reputation-based: provide a set of well known services that represents the organizations' credentials by which inviting new entities to participate

A further distinction can be made depending on the scope a data grid is designed for. Intradomain scope means that the whole system is dedicated to just a single

application therefore the infrastructure offer to all participants features and mechanisms specialized for that particular field. On the other hand,interdomain provides a common base for different fields of reaserch, offering a general purpose platform of services.

Data Transport

Within a data grid framework, data transport play a key role since it governs really complex tasks like data access, management, security and transfer of information, providing underneath mechanisms for both low and high communication levels. Since the complexity of this sub-taxonomy, just a general overview will be given about most common topics.

Functions: features provided at this level can be split into three specific tier which cover as many different degrees of communication. The first one, the transfer protocol, manages bits transfer among network nodes and has been largely adopted by grid communities, like the widely used GridFTP [7]. Overlay Network: data routing is performed at this level, providing specific solution over the internet protocol to satisfy a particular purpose. An example is given by p2p network where DHT (Distributed Has Table) are used to achieve a better performance in data access and retrieving [8]. Common features include storage in the network, caching of data transfers for better reliability and the ability for applications to manage transfer of largedatasets. The last level provides file I/O mechanism to hide the complexity and the unreliability of the networks, allowing applications to access remote files as they were locally available.

Security: this is a fundamental requirement for data grids since the integrity of data must be preserved and shared only among trusted users. Several mechanisms for authentication and authorization are possible, ranging from coarse-grained methods like Unix file permissions to fine-grained solutions like public key cryptographic protocols. Moreover, other approaches can be implemented for defining a further stricter access policy, by using instruments like Access Control List (ACLs), while data encryption may also be provided for all information classified as confidential.

Fault Tolerance: when transferring large quantity of data a reliable underneath

mechanism is necessary to grant fault tolerance. Several approaches can be implemented to fulfill this requirements, the most common take care to restarting over or resuming from the point where interruption took place. GridFTP protocols, for example, allow for resuming transfers from the last byte acknowledged.

1.3.3 Service Grid

This category represents one of the most interesting scenario in the field since the target is not just limited to offer a service on demand for a limited period of time, but also provides those services that cannot be offered by any single machine, eventually combining well known functionalities into a new kind of service. Further categories can be outlined from this one: on-demand, collaborative, and multimedia.

- Collaborative grids let interconnected components to collaborate each others, allowing real time interaction between humans and applications
- On-demand grids are able to satisfy on-demand requests performing a dynamic aggregation of different resources to provide new kind of services
- Multimedia grids offer support for realtime multimedia applications, supplying all needed controls for providing the needed quality for data streams over multiple machines [14]

The interest around this kind of systems has arisen during last years, thanks also to the diffusion of web-services paradigm which became very popular also in the business field. Despite this rapid evolution, many efforts are still required to improve these young technologies and ti define solid standards.

1.4 BaBar's events generation systems

In this section will be introduced an overview of the two systems currently used in BaBar for producing simulated events. Although a complete description will be given in the next chapters, a comparison is provided here in terms of common properties for the distributed system paradigm. In spite of the BaBar's traditional

model can be considered more an organization of different sites glued by a common coordination than a real distributed system, some considerations can be made from the point of view of data distribution, therefore, characteristics like the architectural model, fault tolerance, data coherency, security and scalability will be taken into account for both approaches.

1.4.1 The traditional system

As explained before, the BaBar experiment adopted the MONARC model where data are distributed within a tiered architecture. Despite data required for the simulation process are regularly mirrored on secondary sites, providing a natural backup solution when problem occurred at SLAC, the system is still considered a single point of failure since it strongly depends by the efficiency of the central repository, therefore any delay introduced at the main site for regenerating corrupted data heavily prejudices minor sites activities.

Due to the intrinsic centralized nature of the model, data coherency is maintained quite easily by specific tools that take care to import on remote sites the required version of data needed to the simulation. Requests are pointed directly to the main database at SLAC when changes are notified by the central management. Occasionally some sites can run different kinds of simulation in parallel by simply reorganizing a set of specific local links to point to the desired version of input data, previously downloaded. This mechanism grants that all sites involved perform the simulation using the same set of data, reacting promptly when changes are needed and maintaining the required flexibility to handle particular conditions.

Security depends directly on the protection level implemented on each single site involved. Naturally, the root site represented by the SLAC center is crucial and must offer an adequate level of protection since it represents the core of the whole experiment, providing data not only for the simulated events task but also for all other activities. From the point of view of the simulation process, minor sites depends by the root node, therefore security concerns mostly the SLAC main center if considering the flow of data to distribute.

Commonly, centralized systems like the one used in BaBar are not able to scale

as it would be expected since the most of data are distributed by a single source, nevertheless the root site demonstrated to be able to offer the required level of efficiency for managing the workload even when new sites came into play.

1.4.2 The BaBarGrid system

Several aspects of the BaBarGrid system are common to the traditional one, since the grid approach can be considered as a natural evolution of the original schema. Therefore the new system can be thought as a composed design of two distinct parts where the single site, in charge of connecting to SLAC to download data and to require runs for production, hides the underneath grid infrastructure which can be composed of several sub-sites where running the production.

Security, for the BaBarGrid system, can heavily rely on mechanisms implemented within the grid context, where users must be authenticated before accessing system's resources. Moreover, even validity for users' jobs expires after a specific amount of time in order to force the authentication procedure periodically. Whenever a new site joins the grid, it must subscribe security policies required by the system, resulting in a easier management of the security topic.

A grid system should offer a high level of scalability, allowing new sites to join in without affect the overall performances. At present, the BaBarGrid system bottleneck is represented by the external grid resources used for reading input data needed to the simulation process, which amount is limited to few sites compared to the total amount of nodes it could be possible to use as global resources pool.

Considering transparency in data access, it can be split in two different aspects, related to the access to the input data and the access for storing and retrieving results produced. In the former case, the degree of transparency is quite low since both the site manager and simulation jobs must be aware about data location, which is maintained by using hard coded configuration files. This mechanism is necessary since input data for BaBar's simulation are not managed within the standard grid system and therefore a separate service had to be implemented for this purpose. On the other hand, grid facilities greatly help to achieve a very good level of transparency for storing and retrieving output data, hiding low level access/transfer processes

behind a uniform interface.

Chapter 2

Grid computing projects

In this chapter an introduction to the grid computing and its basic concepts is given. Many services will be highlighted along with the components strictly related to the aim of the project described in this thesis. As examples, some international projects will be presented along with an architecture overview of the Italian grid system, INFNGrid, used as the base platform for the development of this work.

2.1 Grid computing

The need to easily access a large quantity of computing resources, like CPUs power and storage facilities, led to research different solutions to face the challenge put by new generation projects, more and more oriented toward a distributed approach. The Grid concept, started in late '90s, represents more than a revolution an evolution of the well known ideas in the field of distributed computing, introducing at the same time a new vision to allow a transparent use of all its potentials [9]. The main difference compared to the World Wide Web consists in what is shared among users: the Web shares information while the grid allows a great flexibility in sharing computing resources of different nature. Many efforts brought to the development of the so called middleware, considered the fundamental component to give a uniform representation of all the different kinds of hardware and software technologies that work together underneath, providing a high level of abstraction that allows users to focus just on the achievement of their own goals, without worrying about details like how to locate best resources for matching their tasks requirements. During last years

several communities, both in academic and business fields, started to embrace the grid philosophy trying to develop their own personal "recipe" to match the best balance between costs and performances, resulting in the birth of different approaches to the same problem. Despite the lack of a real unique standard, these contributes helped to keep alive the research for new solutions to improve the different parts that make up the grid technology.

The grid idea is based on a collaborative approach, since generally a single institution cannot supply all the resources necessary to achieve a specific goal, where a huge amount of data need to be handled or a high level of computational work is required. The possibility to dynamically plug in new resources allows new entities to join the grid environment with little effort, automatically gaining all the benefits provided by the architecture. Nowadays grids can be spread over different countries and are made up different computing centers connected by fast network links. This is a great difference compared to other traditional solutions, where computing resources were usually bounded to a single physical institution or, at most, distributed among few centers. Grid middleware allows to access resources spread worldwide since even the management of services and facilities are distributed, delegating to each single site the responsibility to keep things working to contribute to the efficiency of the whole system. However, the lack of a central management along with the management of institutions that can join an leave the grid dynamically, represents a great challenge that require to design new solutions to efficiently face scalability issues that a such environment implies.

A great contribute for the development and implementation of the Grid come from the open source community which provides well known protocols and opens standards to build the basic services, which are often improved to match those challenging requirements needed to operate on the grid in a reliable and efficient way. Although standardization is not mandatory in grid computing, the need for a common general guideline is necessary to avoid the raise of too many types of grid that would make it much more difficult to combine so different technologies. The main group which is actually working towards standards adoption in grid computing is the Open Grid Forum (OGF) [10] . The path for the definition of a new standard is quite

complicate, moreover is not always easy to find examples of grid implementations where standard solutions have been adopted.

2.1.1 European grid projects

Driven by the need to define common guidelines for a uniform grid environment, during past years the European Union promoted international projects to start building a reliable grid network. The Enabling Grids for E-sciencE (EGEE) project [11], which first part officially ended in 2006, was one of the most important example. The aim of the project was to develop a secure and robust grid network to allow researchers from several countries access major computing resources, independently on their geographical location, bringing together experts from more than 50 countries with the common goal of developing a service Grid infrastructure available to scientists 24 hours-a-day [5]. The two years project, funded by the European Commission with more of 30 million euro, defined two main pilot application domains for testing the performance of the evolving infrastructure: the High Energies Physics field, for supporting the Large Hadron Collider Computing Grid (LHC), and the biomedical research for those communities involved in the bioinformatics field. The next phase of the project, EGEE-II [12], shares its infrastructure with the Worldwide LHC Computing Grid Project (WLCG) [13], providing a total amount of more than 72000 CPUs available, with more than 260 sites connected and at least 7500 registered users, for a total amount of about 20 PB of data store, reinforcing the collaboration among different institutions. Other topics, like training and connection to the industry are taken into account and periodical events are promoted to extend the knowledge about the grid to new users and to discuss new approaches to fulfill business requirements for those companies that would like to invest in this technology.

Several examples show the many potentials the grid has to offer with the different projects started in different fields of research and industry. From the beginning, EGEE formed a strategic alliance with the LHC Computing Grid (LCG) project [13] [14] to provide computing resource to the four applications representing the four different LHC experiments: ALICE (A Large Ion Collider Experiment), ATLAS (A

Toroidal LHC Apparatus), CMS (the Compact Muon Solenoid Experiment) and LHCb (The Large Hadron Collider Beauty Experiment). This support has been extended to several other HEP collaborations that have joined the infrastructure later, such as the US BaBar (the B and B-bar experiment) [15], CDF (Collider Detector at Fermilab) [16] and ZEUS [17]. On the medical field other applications approached the grid philosophy, like the WISDOM Drug Discovery project [18], which aim is to speed up the process of finding new drugs against malaria, or the GPS@ (Grid protein Sequence @nalysis) [19], a bioinformatic portal dedicated to various protein analysis tools, which target is to help biologists and physicians understand the genomes which have now been sequenced.

Looking to other scientific domains, Geocluster is the first industrial application successfully running on the EGEE Grid Production Service on the field of Geophysics, that within the Expanding GEOsciences on DEmand (EGEODE) community explores the composition of the Earth's layers and let researchers to process seismic data, while a project for astrophysics researches, MAGIC [20], investigates the behaviour of air showers in the atmosphere, which are caused by high energetic primary cosmic rays, processing data located on the Canary Islands, to study the origin and the properties of high energy gamma rays.

2.2 The gLite middleware

One of the fundamental part of a grid architecture consists of the middleware component. The aim of the middleware is to act as a glue to interconnect all different kinds of hardware underneath, providing a uniform software platform to the users, in order to allow a transparent usage of all resources available hiding as much of implementation details as possible [21] .The EGEE project actually uses the gLite middleware as the basic platform for its architecture, which was born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers and by the contributions from other projects like LCG and VDT [22].

Middleware is made of several software services and libraries providing function-

alities which are common to different kind of domains, due the general purpose they were designed for. Tasks like finding the right pool of nodes where execute user jobs on a heterogeneous environment or accessing different kinds of storage facilities in a transparent way are standard operations that must be provided to preserve the ease of use of the whole system from the user's prospective. Jobs submission must be integrated with a mechanism able to give the possibility to the user to monitor the full life cycle of the job until its completion, while the underneath architecture takes care to route jobs toward the right place. Another important issue consists in data access, along with data transfer and replication, besides, the status of all available resources must be collected, published and kept updated. Finally, a grid environment must face the security challenge, implementing mechanisms to avoid access and usage of shared resources to unauthorized users.

2.2.1 Workload Management

The Workload Management System (WMS) [23] [24] allows users to submit jobs, and performs all tasks required to execute them, without exposing the user to the complexity of the Grid. The user is responsible to describe his jobs and their requirements, and to retrieve the output when jobs are finished [25]. It consists of several components resulting in a quite complex system in charge of taking decisions about which location on the grid offers the best matching in terms of user's jobs requirements.

Users write job using the Job Description Language (JDL) [26] which codes requirements and properties as attribute-value pairs, including all information needed to perform the job correctly. When submission step takes place, the Information System is contacted to get the list of best candidates CEs that fulfill all the requirements. Several advanced features are included to refine the search of optimal balance among expectations and resources actually available, along with the capability of automatic resubmission in case of errors, the submission of interdependent jobs and a realtime check for jobs output.

2.2.2 Information system

Discovering resources and monitoring related status along with checking how logged in users are using them, is a critical issue for a grid environment and need specific components to be implemented. The Information Service (IS) accomplished these needs providing the necessary feedback about resources availability to permit users to decide which location offers the best environment for jobs execution. Two different types of Information Services are available in gLite: the Globus Monitoring and Discovery Service (MDS) [27] and the Relational Grid Monitoring Architecture (RGMA) [28]. The MDS, based on the LDAP protocol, is capable of handle both static and dynamic resources, like the type and status, and is structured in a hierarchical way. At the top of the chain the Berkeley Database Information Index (BDDI) provides a complete list of all resources available. The list provided by the BDII is published by using the LDAP url format and consists of a set of all gateways, typically CEs, located at remote farms where jobs can be submitted. On each single farm other components take care to collect local information as feedback for the top BDII queries. The Grid Resource Information Server (GRIS), installed on each critical element of the farm like CE, SE or RB, collects all necessary information and then propagates this stream of data to the local site BDII, usually installed on the CE, that communicate with the top level BDII, as depicted in Figure 2.1. Another solution for the same purpose was introduced later by the the OGF, as a relational implementation of the Grid Monitoring Architecture (GMA), based on relational databases. The schema consists of three main components, the Producer, the Consumer and the Registry. The first one provides information registering themselves to the Registry, which replies to the Consumer requests providing the location of Producers holding the information needed. R-GMA represents a more flexible solution compared to MDS enabling users to share information in a virtual database. Indeed, other applications uses this architecture for different purposes like, for example, job monitoring or accounting.

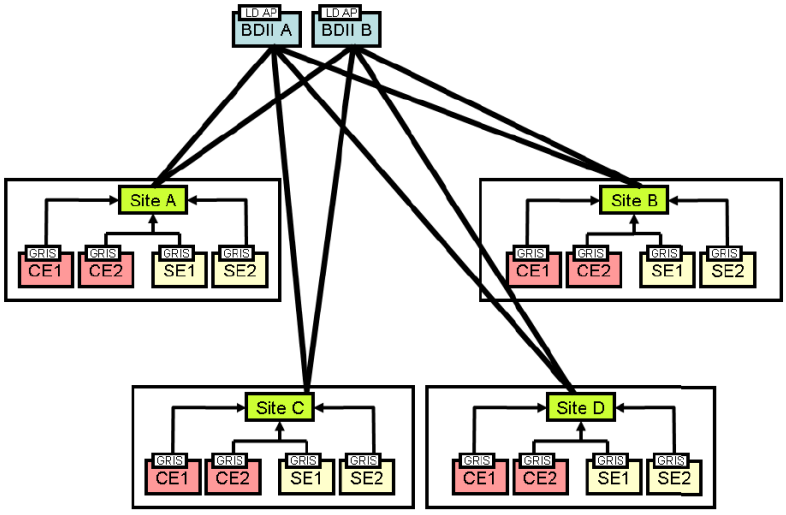


Figure 2.1: MDS Information System architecture

2.2.3 Data Management services

Within a grid environment, a fundamental role is played by the mechanisms delegated to the data handling that must be performed over a distributed architecture composed by heterogeneous kinds of hardware and software components. Several types of data access and transfer protocols are provided to extend the flexibility of the system.The GSIFTP [29] protocol, that includes the characteristics of FTP with support for Grid Security Infrastructure (GSI) [30], allows to transfer data efficiently to and from the SE and along with the GridFTP server represents one of the most common data service installed on WLCG/EGEE SE. The Remote File Input/Output protocol (RFIO) allows to direct access files stored in the SE and was developed for tape archiving systems and both versions, secure and insecure exist. Another protocol for direct remote access is GSI dCache Access Protocol (gsidcap), an enhanced version of the of the dCache [**?**] [31] native access protocol (dcap). In gLite the service for data storage is represented by the Storage Element, where users data are stored for further uses. All recorded information are considered read-only, therefore a file on a SE cannot be modified but just removed and then replaced. Each site can freely choose among different types of SE when deciding to install its

own local data service:

- CASTOR [32]: this solution offers an interface between a tape mass storage system and a disk buffer used as frontend, while all details are hidden by a virtual filesystem. Data transfer is handled by a dedicated process which takes care to migrate files from disk to tape. The RFIO insecure protocol is implemented, allowing access to the SE only from nodes located in the same LAN

- dCache: it can be used either as a disk only storage system and as disk buffer frontend to a other mass storage system like HPSS [33]. A virtual filesystem hides a pool of nodes that can be dynamically added, while a server reperesents the single point of access to the SE. The system adopts the gsidcap protocol for data access

- LCG Disk pool manager: like previous systems, a virtual filesystem hides the complexity of a disk pool architecture, based on the secure RFIO protocol that grants file access even from WAN. The pool consists of a set of disks where new nodes can be added dynamically

In order to provide a uniform interface to every type of storage element, the Storage Resource Manager (SRM) [34] has been designed and is actually implemented to the most of SE installed, hiding the complexity related to the management of disks and tapes beneath. Furthermore, users are unaware about resources setup and can perform several operations like file requesting, space reservation and migration of data from tape mass storage to disk.

In gLite, data management is strictly related to the concept of file as main unit for operations, therefore a reliable mechanism for registering files must be provided. The solution implemented offers different levels for successfully mapping filenames and their location on the grid:

- the Grid Unique IDentifier (GUID) is a 36 bytes unique identifier associated to a file. A such string, composed by a time stamp and a MAC address, has the following format:

`guid:<string>`

- the Logical File Name (LFN) provides a much more human readable way to refer to a file, acting as a alias for it. In this way users can refer to a file being aware about its physical location The typical format is:

  `lfn:<any_string>`

  In the case of the LCG File Catalogue, described later, a logical structure is defined to organize all LFNs, using the format shown below:

  `lfn:/grid/<MyVO>/<MyDirs>/<MyFile>`

- the Storage URL (SURL) represents the Physical File Name (PFN) of the file and identifies a replica in a SE. The format depends on the availability of the SRM interfaces on the SE:

  `<sfn|srm>://<SE_hostname>/<some_string>`

  In the first case the prefix sfn indicates the SE does not support SRM and therefore the path for file location is composed by the following elements:

  `sfn://<SE_hostname><SE_storage_area><VO_path><filename>`

  For SRM-managed SEs a virtual system is used therefore the filename may not be related with its physical location so there is not a any particular format for SURL.

- the Transport URL (TURL) includes information about the protocol supported by a SE and used to access the file:

  `<protocol>://<some_string>`

  where the string after the double slash may have any format that can be understood by the SE serving the file. TURLs can be obtained automatically by the SURL through the Information System or the SRM interface.

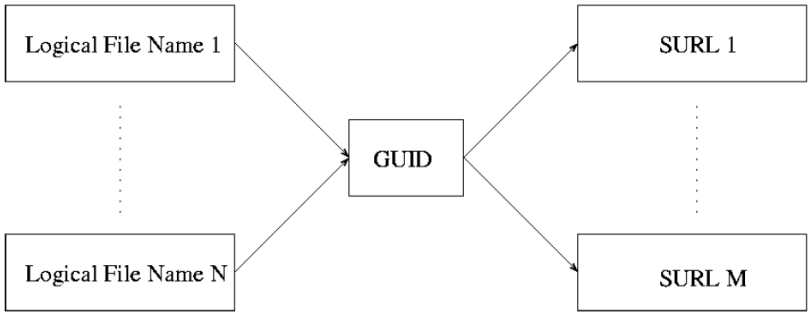


Figure 2.2: File names mapping

To efficiently manage all the associations a single file could produce, a grid catalog has been implemented. In the past the old EDG-RLS catalog was proposed, but due to serious security and scalability issues a new solution was taken into account. The LFC FIle Catalogue (LFC), maintains mappings between GUIDs and LFNs and between GUIDs and SURLs and is implemented using a Oracle database as backend . The catalog publishes its endpoint to the Information System therefore can be discovered and accessed both from users and other services like, for example, the WMS. A pool of lfc commands is provided to let users query the catalog and perform several operations, like create or remove files and directories, in a Unix-like way. Another set of utilities, called *lcg-utils*, is provided to perform operations like creating or removing replicas, moving replicas from a SE to another and retrieve a GUID from a LFN or SURL. Since a file is considered a Grid file only if is both physically present on a SE and registered into the catalog, the use of these high level utilities is strongly recommended to preserve consistency between files in SEs and related entries into the catalog, otherwise a file could be considered corrupted by the system. The typical example occurs when an entry is deleted from the catalog but the related file is still present on the SE. In this case, the file it still on disk but it will result unavailable. High level tools safely remove a file by deleting either the entry in the catalog and all the replicas of the file, independently on their location.

2.3 Security

One of the most critical issue for a Grid environment is related to the security of information exchanged. The system, indeed, must provide a reliable mechanism to protect data against unauthorized users and be able to preserve information integrity in case of potential services failures, due to the violation of communication protocols. As the middleware evolved, new mechanisms have been introduced to make safer operate over the grid platform.

2.3.1 VO

The sharing of resources among structures that collaborate to a common project or are involved in the achievement of the same goal, is a topic which importance is clear in a such distributed environment like the grid. Virtual Organizations were introduced to allow a great flexibility in managing resources within multi-institutional areas, providing instruments to easily define collaborative strategies for problem solving [35]. Specific rules define which resources are available and by who can be accessed, providing a dynamic controlled coordination for sharing services like computing and storage facilities. Generally, each institution defines a VO to share its resources since it is a mandatory step to be allowed to operate on the grid. Users must subscribe to at least one VO before starting work, but further steps are required to be fully authorized. A single user, indeed, must obtain an account on specific machine called User Interface representing the node from where jobs submission starts and copy there the personal certificated previously requested to be authenticated.

2.3.2 Users certificates

In order to operate on the grid, each user must provide his own credentials to other users or to services involved during the communication. The Globus Toolkit [36] defines the Grid Security Infrastructure based on the asymmetric keys mechanism, as described by the Public Key Infrastructure. A couple of keys is generated for each end point of the communication, users or services, creating a private key and

a public key. The private key must be preserved in a safe location while the public key is freely distributed to all the entities interested in the conversation. Messages are encrypted by the sender with the public key of the receiver which is the only one that can decrypt the message using his own private key. Requirements for a secure communication concern the authenticity of the entities involved, each identity must be trusted by participants, the privacy of the message must be guaranteed along with its integrity, its meaning cannot be altered by any external entity and of course the participants must be authorized to interact to each other. TheGlobus Toolkit covers all the described topics through services like X.509 certificates [37] for authentication and proxy certificates for delegation and single-sign on. User certificates are released by trusted entities recognized by EGEE called Certification Authorities (CA). After a user has logged to the User Interface he can use his personal certificate to generate a proxy from it. Proxy is used to delegate the user credentials through to the whole chain of requested authorizations that will follow jobs submission, allowing the execution of all operations needed. In order to reduce security risks, the proxy has a shorter validity than the original certificate, which usually lasts for one year.

2.3.3 VOMS

In order to enable users to access resources provided by VOs and offer support for group membership and roles, the Virtual Organization Membership Service (VOMS) [38] [39] [40] was developed within the DataTag collaboration and then further improved and maintained by the actual European project EGEE. VOMS is a system for managing authorization data within multi-institutional collaborations providing a database of user roles and capabilities and a set of tools for accessing and manipulating data contents to generate Grid credentials for users when needed.

The VOMS database contains authorization data that defines specific capabilities and general roles for specific users. A suite of administrative tools allow administrators to assign roles to users and manipulate capability information, while command-line tools allows users to generate a local proxy credential based on the contents of the VOMS database. This credential includes the basic authentication

information that standard Grid proxy credentials contain, but it also includes role and capability information from the VOMS server. Standard Grid applications can use the credential without using the VOMS data, whereas VOMS-aware applications can use the VOMS data to make authentication decisions regarding user requests.

2.4 The INFN grid project

At the end of '90s the National Institute for Nuclear Physics approved the INFN-GRID project [41] to develop the first Italian Grid Infrastructure, based on GARR, the Italian research network. From the beginning the collaboration with CERN, others European centers and some industries, led the Italian Grid to being part of fundamental project like DataGrid, a milestone towards an infrastructure supporting the common European Research Area (ERA). INFN-Grid has promoted several EGEE projects helping to deploy worldwide grid service to support interoperability among other major grid infrastructures in the world and represents a successful example of collaboration between more than 40 sites supporting several scientific domains like high energy physics, biomedical, earth-observation and industry. The project is based on the standard gLite middleware as core for supported services, providing some custom configurations for specific tasks like resources monitoring. INFNGrid infrastructure represents the grid network used as base for the development of the project described in this thesis and it provides support for the mechanisms concerning all the critical points examined above. In the following sections will be illustrated the typical job submission workflow, highlighting the common components provided by the gLite infrastructure.

2.4.1 Grid elements

In order to join the grid network, the new node must fulfill some basic requirements in terms of components provided. A grid farm to be considered as part of the infrastructure needs at least a local batch system to schedule incoming jobs plus a pool of machine where executing tasks, moreover a storage unit to read and write results is required along with an interface from where logging to the system. The following list gives a brief description for the main elements of the architecture:

- User Interface: this node represents the entry point of the grid farm. Users accounts are created and configured to accomplished all security policies needed. A pool of tools allows to submit jobs to the grid, monitoring and retrieving results after completion

- Computing Element: it is the main gate to access the pool of working machines where jobs are executed. The framework allows to choose among different types of batch systems to manage jobs scheduling. Actually is also used to store the software used by different experiments

- Worker Nodes: are the machines where jobs are actually executed. Generally worker node pools consists in a heterogeneous group of hardware nodes

- Bdii: this critical component belongs to the Information System infrastructure and provides the list of available resources when contacted by the resource broker

- Resources Broker: the purpose of this service is to accept incoming submission requests from the user interface choosing where sending users jobs on the basis of specified requirements and the resources obtained by the bdii

- Storage Elements: is the unit in charge of storing data from job output and input data needed by applications installed by VOs

2.4.2 Jobs submission

The Grid allows users to access and share its computing resources to achieve targets otherwise hard to reach without a collaborative effort. The standard solution actually adopted to provide these kind of features concerned the submission of jobs following a specific workflow. First af all a user must be authorized to use resources belonging to a VO and has to install his personal certificate on the UI, from which will be generated a proxy certificate used as temporary credential ticket for all services will require it for any security reason. From the UI the user submits jobs, written in JDL, to the WMS, where all needed files specified in the Input Sandbox

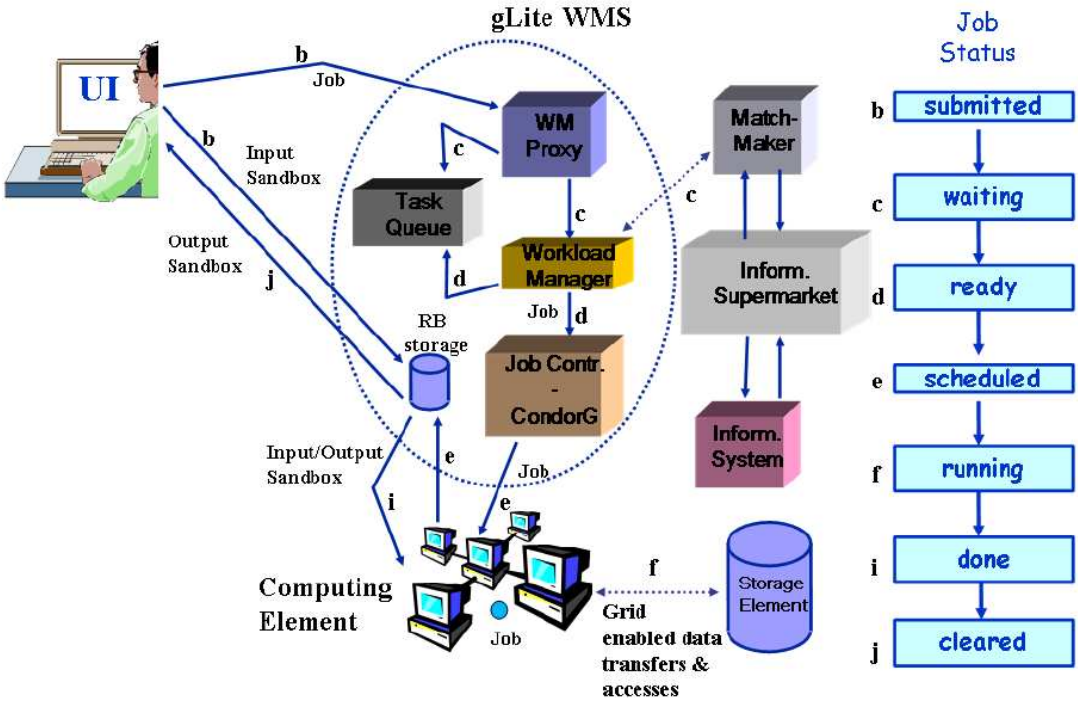


Figure 2.3: Jobs submission workflow

will be copied. The WMS examines information provided by the BDII, to match the best CE where sending jobs, and interrogates the LFC to find input data requested. After creating a wrapper script to prepare the job, it is then transferred to the selected CE where is scheduled on the local batch system. Before jobs execution, input data within the Input Sandbox are copied to the WN where the job will run. During execution the job can use the data management tools to register output files in the grid catalog. When job successful completes, all files eventually contained in the Output Sandbox are transferred to the WMS from where will be retrieved by the user on the UI. Automatically resubmissions are provided in case of failure. The Logging and Bookkeeping services keeps trace of status changing for the job during the whole cycle, providing information on the status of the job when queried by the user. Figure 2.3 shows the logical workflow schema.

Chapter 3

A case study: BaBar

In this chapter we introduce the BaBar experiment in order to focus the attention on the computing model adopted for the production of simulated Monte-Carlo events, known as Simulation Production (SP). The evolution of the BaBar computing model will be highlighted along with the strategy adopted to distribute, locate and retrieve data needed to the simulation process.

3.1 BaBar's computing model evolution

The BaBar experiment investigates the CP Violation phenomenon, related to the subtle matter and anti-matter asymmetry presents in the universe. The collaboration is made up by more than 700 researchers based at 74 institutes spreads over 11 countries in the world, while the main center is located at SLAC, the Linear Accelerator Center of the Stanford University, California, where an electron positron collider is used to study the physics of B-mesons by continuously colliding bunches of high-energy electrons and positrons 250 million times per second. Events simulation is a major task of the experiment, since at least three times as many simulated events are needed as data events, where each event is the result of a collision between an electron and positron. The process simulates the interaction of the particles generated during the physics step with the detector, where particles are created using dedicated programs on the base of specific models and theoretical calculations. The noise produced during the real events generation is also taken into account and mixed together with data for achieving a more realistic result.

BaBar has taken a huge amount of data since 1999, stored in a local database which size now has grown over 800 TB stored in more than 850,000 files. In order to face the challenge that a such amount of data required to be efficiently handled, the framework of the experiment was implemented mainly in C++ as most of the main applications needed for computing processing, for data distribution and to develop database interfaces. The first version of the BaBar computing model was a data-center model [42] where, especially for analysis, main data were available only at SLAC. The increment of the expected luminosity and the relative high cost in terms of computing, led the evolution of the model toward a multi-tiers distributed computing environment, to make use of the computing power and of the storage facilities available at national centers.

BaBar's structure classifies computing centers depending on data set provided, in a MONARC like style. Tier A sites host a copy of all the data and in general they should provide at least 30% of the total data sample, stored on disk or mass storage facilities, in order to make possible to run high level physics analyses, as well as specific detector studies requiring RAW data. Tier B sites serve a region and act as secondary data distribution centers. Tier C are typically individual institutes having access to a small sample of the data corresponding to the direct physics interest of the sites and they are largely used to produce SP events since represents the best solution to accomplished this demanding task.

The storage layer was supported by the Objectivity [43] [44] technology, to provide an object oriented approach to data persistence for the event store of the experiment. In a such as environment, events were stored on the basis of data granularity, from the most detailed information up to the highest level of abstraction suitable for physics analysis, following a hierarchical schema. Moreover, navigation structures were provided in order to access any kind of event at any level of the event store. Data and Monte-Carlo production were written into Objectivity while analysis jobs were read from there [45] [46]. This method offered an optimal compromise for efficiently performing analysis on collections of filtered events but only if the dataset of the eventstore was sufficiently large, indeed, export of these collections to remote sites could result in performance lack due to the large amount of

references to the main database. Besides, other Objectivity limitations related to its internal data representation led to the copy oh hundreds of GBs for each single exported collection. This brought to scaling and maintenance issues and introduced navigational overheads inside theeventstore, increasing latency since many data to be used had to be staged back from external device like tapes. Data compression was not optimized making hard to export parts of the eventstore to other sites [47]. A second event store which was based on ROOT I/O [48] was developed to match this requirements, at least as temporary solution to provide faster access to data, but it was not suitable for reconstruction and Monte-Carlo production.

3.1.1 The Computing Model 2 (CM2)

The high complexity introduced by the Objectivity framework, due to the amount of data growth, required a more reliable ad efficient environment to meet the new requirements of the experiment. The new Computing Model (CM2) [49] resulted in a more efficient architecture, built on a ROOT based system calledKanga [50].

3.2 An overview of the ROOT framework

The ROOT system was developed at CERN (the European Organization for Nuclear Research) [51] to fullfill the modern requirements of performance needed by interactive data analysis tools, offering a set of C++ Object Oriented frameworks. A framework is a collection of cooperating classes that make up a reusable design solution for a given problem domain, it typically includes objects that provide default behaviour programmers can inherit and eventually override in order to let the framework to call application code at the appropriate times. ROOT is being developed for the analysis of Particle Physics data, but can be equally well used in other fields where large amounts of data need to be processed. It consists of about 310 classes grouped in 24 different frameworks, divided in 14 categories [52]. The system offers also an embedded C/C++ interpreter, CINT [53] [54], to allow fast prototyping for applications development. One of the most important component of ROOT consists of the I/O system that provides a convenient way to serialize to disk

data structures, appending them to the internal hierarchical schema. The ROOT framework is also capable to provide remote file access via a TCP/IP data daemon known as rootd. Data, local or remote, can be accessed thanks to the dedicated framework class; a file handle is returned to the client and can be used to access data in a transparent way.

As described in more detail later in this chapter, the BaBar computing team developed a new service for data access in distributed environment, called Xrootd [55]. This service is fully compatible with the standard rootd daemon, but overall its performances matching better the needs of the experiment.

3.2.1 Kanga ROOT

The Kanga eventstore was intended to replace the Bdb/Objy eventstore used in BaBar as part of the original computing model. It was designed to be a scalable, flexible and more reliable replacement for the original eventstore technologies while providing the relevant functionality for analysis and production [56]. It includes, among others, the following facilities:

- access to multiple data components
- multiple access methods
- data borrowing and pointer collections

The user interface to the eventstore is an event "collection". Each collection represents an ordered series of events mapped to one or more files, while the data itself is written to ROOT trees within the files. All of the collections from production will have names beginning with /store, following a schema as shown in the following example:

```
/store/
       SP/
       SP/BkgTriggers/
       PR/
       PRskims/
       SPskims/
              R12/
              R14/
              R16/
```

An event in the Kanga event store consists of components, which correspond to the different levels of detail of the event data. References between objects in the same component as well as references between objects in different components, which may be stored in different files, get persisted using the same custom reference class. The key component of the event is the event header. An event header holds a pointer to each of the event components. Such a pointer basically consists of the logical file name of the ROOT file containing the corresponding component tree, and the number of the entry in that tree that corresponds to the event. The component tree can be in the same ROOT file as the event header, in another file of the same event collection, or in a file of another collection.

The new eventstore structure introduced three concepts for data classification:

- **collections** represent logical site independent names, used to configure users jobs access to multiple data components, perhaps clustered to separate files
- **multiple access** methods
- **data borrowing** and **pointer collections**
- **logical file names (LFN)**: are site-independent names given to all files in the eventstore. Any references within the event data itself must use LFN's so that these remain valid when they are moved from site to site
- **physical file names (PFN)**: file names that will vary from site to site. To create a PFN a prefix is added to the LFN to supply all information needed about how to access data at a specific site

Events are organized in event collections, ordered lists of event headers. An event collection can:

own a component: there's a deep copy of the component data in one of the ROOT files of this collection; the component pointer in the event header points to a file of this collection.

borrow a component: the component pointer in the header points to a file of a different collection. No component data but just the component pointer is stored in this collection.

This feature allows to create very compact pure pointer collections in one case, and more efficient deep copy collections in another case. Because a collection consists of simple ROOT files and because it is relocatable, data import and export was cut down to simple file copy.

The event collection is the user interface to retrieve event data from the event store. If a collection has to be opened for input, the logical file name (LFN) of the first file with the event header tree of the collection gets constructed from the collection name by a simple naming convention. The translation from LFN to physical file name (PFN) also is done without any file catalog or centrally managed database.

The following example of a collection's name translation helps to explain the schema:

/work/users/andreotti/mycollection

When an application needs to read this file, a name translation is performed by the CM2 code to get it to a standard name convention, derived by the collection name by simply append ".01.root" to it:

/work/users/andreotti/mycollection.01.root

The LFN so obtained is a site independent name. In order to find it physical counterpart, a further transformation is required, using a site-specific config file called KanAccess.cfg , described later in this section, that adds the rest of the physical path needed to reach and open the file to the given site. Within the .01.root file there will always be an "event header" which contains information on other LFNs which may need to be opened. Using the same config file, these will be converted to PFNs and opened as needed. The CM2 Kanga eventstore provides two different methods for access files, one based directly on filesystem, locally or remotely by using NFS, or by the Xrootd service. The configuration file which describes the site-specific mapping of LFN's to PFN's is the KanAccess.cfg file:

read /store/PR/* file /nfs/serv1/ read /store/SP/* xrootd serv2:1094/

In this case, the first keyword ("read") simply says that the rule applies to reading data. If the LFN is matched by the second field (e.g. "/store/PR/*") then the rule will be used. The 3rd element in the rule ("file" or "xrootd") chooses an access type and the 4Th element provides the needed information to do the LFN/PFN mapping, indeed providing the prefix required for path completion. As result, PFN is like to:

/nfs/scratch/work/users/andreotti/mycollection.01.root

in case of nfs access, or

root://somehost:1094//work/users/andreotti/14.3.1a/mymooseevents.01.root

if xrootd access is used.

When the main application for simulated events production runs, it automatically looks for this particular file to understand which kind of access is available before establishing a connection with the input data source.

3.3 Data bookkeeping

The BaBar experiment needed a bookeeping mechanism to efficient access the huge amount of data generated. Since the early stages the design adopted led to a poor implementation due most to the kind of technology beneath, resulting in an almost useless tool difficult to use and where improvements were hard to achieve. The scenario drastically changed when the new computer model was adopted, fixing performance lacks introduced by the old system.

3.3.1 New BaBar's bookkeeping

At the end of 2003, BaBar switched to the new computing model with an eventstore based on structured flat files coded in the ROOT format. The old system based on Objectivity technology allowed just poor control of the event data to file association. This made the navigation from event data of low level of detail to the corresponding data of higher detail practically impossible if the more detailed data had to be staged back from tape [57]. Another key element in support of the change was the

implementation of a metadata catalog that allowed users to easily locate and retrieve data needed for analysis. The requirements highlighted the need for a scalable and flexible solution to provide users with data even in a distributed environment, leading to the development of a system based on a relational database and a set of tools to interact with it.

Data from the detector is divided into runs, each run needs to be processed and there can be several versions of processing. The output of each processing needs to be in the bookkeeping and runs stored form the units data of the eventstore , called collections, which name is unique. A collection can contain events from any number of runs; a list of all the collections is the core of the bookkeeping. More specifically, in order to provide different parts of the eventstore to users, collections can be grouped into lists, based on well defined criteria, called dataset [58]. These lists are pre -created and saved in the database, providing fast access and ease of use. Since the BaBar experiment is a large collaboration, the whole bookeeping was designed to be fully mirrored to other sites, keeping changes in sync. Along with distributing the metadata, the bookkeeping includes data import and export tools, to distribute the data on the base on the defined dataset [59]. The aim of the BaBar's bookkeeping is to to keep track of data produced that have successfully passed a chain of checks and were declared good to be used by users. It was implemented in order to build up a workflow stream were the state of the production could be updated during the different steps involved into the process.

Concerning the production of simulated events, each site sends request to the SP coordinator at SLAC to get new runs to execute. The coordinator fulfills user requests and updates the bookkeeping to keep trace of the whole productions status. After runs completion, on each site automatic tools take care to update the central database, providing information about local production status [60].

3.4 BaBar software releases

BaBar software is based on releases, where each release is a set of stable packages and library suitable for different tasks supported by the experiment. Periodically, when

new features come into play, a new release is built and deployed to the collaboration, resulting in an improvement for the capabilities of the software. The BaBar software release [61] tools consist of a set of scripts which work within a specific directory structure providing an easy way to build the software of the experiment by using the GNUmakefiles mechanism. The whole release system is often referred to as SRT, referring to the SoftRelTools (Software Release Tools) package which is itself used to install and maintain releases.

A package is a self contained piece of software intended to perform a well defined task, identified by a unique name and includes its own library and files. Some complex projects, like for example the Geant [62] simulation of BaBar, require the integration of different packages, while some others packages can be used on their own. This part of the BaBar software is maintained by a package coordinator, who is responsible for testing the code and releasing new versions when appropriate, including the source code and documentation for stable and self consistent versions of all packages within their own subtrees. All the files required for a package should then be contained within the single package directory which will also include a GNU makefile which implements some standard targets and definitions.

A software release is made of a consistent set of packages together with the libraries and binaries created for various machine architectures. All created releases are grouped inside a specific directory where several symbolic links are used to refer to particular releases, defined as follows:

- newest: the most recently built release. The only quality requirement is that the release (mostly) compiled. The release coordinator determines which release this should point to

- test: the most recent release which has passed basic tests, such as running the full reconstruction on a small number of events. The reconstruction coordinator determines which release this should point to

- current: this release is the most recent to have reached an acceptable level of quality. Acceptable is determined by a combination of code and physics checks

3.4.1 Software releases management

The most of BaBar software and documentation is maintained using the Concurrent Versions System, CVS [63]. This is a widely used public domain tool for maintaining software. CVS allows different people to modify software, keeping track of the modifications and allowing specific copies to be tagged. These tagged versions can then easily be accessed at any later time, even though many changes may have been implemented since the original tag. All packages created are tagged when considered stable and new versions are defined when improvements take place, while files and subdirectories of a BaBar package can be maintained as a whole, identified uniquely by the related tag. The repository consists of a single copy of the master sources and contains all the information to permit extracting previous software releases or packages at any time based on either a symbolic revision tag.

3.4.2 Import of software releases

Since all BaBar software is maintained in AFS (Andrew File System) [64] directories, users at remote sites have to create a personal AFS account to access the CVS repository to import latest releases and packages. A set of specific tools are provided to easily import each part of software required, re-creating the correct release directory structure at the remote site. The procedure is split in two steps, the first one imports only those parts that are operating system independent, while the other imports any operating system specific files. Users must take care of setting the local environment following the BaBar defined policies, eventually making some specific changes to configuration files as needed, before running configuration scripts to complete the release installation. After the completion of this task at remote sites, a complete version of the release is available as independent copy from the main repository.

3.5 The traditional Simulation Production workflow

The BaBar detector produces signals that after several stages are translated into a collection of data in the BaBar event store [65]. Simulated data consists of generated particles which leave signals in a simulated-BaBar detector having the same format as the signals left by real data in a real detector. The aim of simulation production is to create simulated collections that mimic real data collections as closely as possible, therefore several stages are required to achieve this result:

- Generation of the underlying physics event
- Particle transport and calculation of the idealized energy deposits in the detector
- Overlaying of backgrounds and digitization of the energy deposits
- Reconstruction of the event

After particles generation, provided by specific events generators modules, data are propagated to a detector simulation layer over GEANT4 [66] running in the BaBar framework. The output is stored in a data structure which lists the idealized energy deposited by the particles passing through the detector, and the location of each energy deposit. The next stage transforms the acquired data into signals which look like the real data, including also backgrounds to represents detector's noise, then the reconstruction stage is performed retrieving for each event, data produced in the previous stage and combined into candidate events consisting of particle tracks, energy clusters, and probable particle identifications. Simulation jobs come in runs composed by at least 2000 events, where each simulated event takes about 8 seconds on a modern processor and results in 20kB of storage.

In the early years of the experiment, the simulation events process was split into three distinct phases, where the output of each step provided the input for the next one, requiring the intermediate data to be retrieved from and stored to the database between stages. At the end of 2002, SLAC could provide a considerable

computing power but since the most was dedicated for other tasks, like data reconstruction and analysis, this led to increase the distributed computing efforts among the several institutions belonging to the collaboration, so the simulation production process became the ideal candidate: from SLAC runs were distributed to remote sites and executed to local batch systems. During the following years, in order to improve production efficiency, a new simulation executable called Moose (Monolithic Object-Oriented Simulation Executable) replaced the three-stages workflow and was able to perform all steps on each event before producing the final output. The new architecture brought several benefits on management, the number of jobs to handle decreased to one third and the servers load was reduced since there was no output from each stage. The human resources management played a main role since the whole procedure required three people at SLAC working in shift. Since such constraint was not acceptable for remote sites, the agreement was to offer just one person, a reliable toolkit was developed to provide a suite of services to handle the distributed production. The toolkit was called ProdTools [67], a set of PERL command line tools and libraries designed to assist the production manager on each site, providing an interface between the central production database at SLAC and the local batch system used at remote site. The whole system, indeed, was developed around a single Oracle database which provided the global coordination for runs requests and information configuration for each job to submit on remote sites.

In order to support the wide range of different batch systems used by external farms, like PBS [68] or LSF [69], a layer was introduced to let sites create the appropriate interface following the template provided by the SLAC central management. In this way all remote sites could set up and improve their own custom interface without the central management had to take care to test all the different remote batch systems. Before the new computing model took place, all jobs produced were written into the Objectivity database each site had to install locally, so a dedicate utility was designed, MocaEspresso [70]. Its main goal was to execute two tasks in parallel, in order to recognize data completed and ready to be extracted and the relative transfer to SLAC through multiple streams. This application introduced a great improvement for speed within the production chain and quickly became the

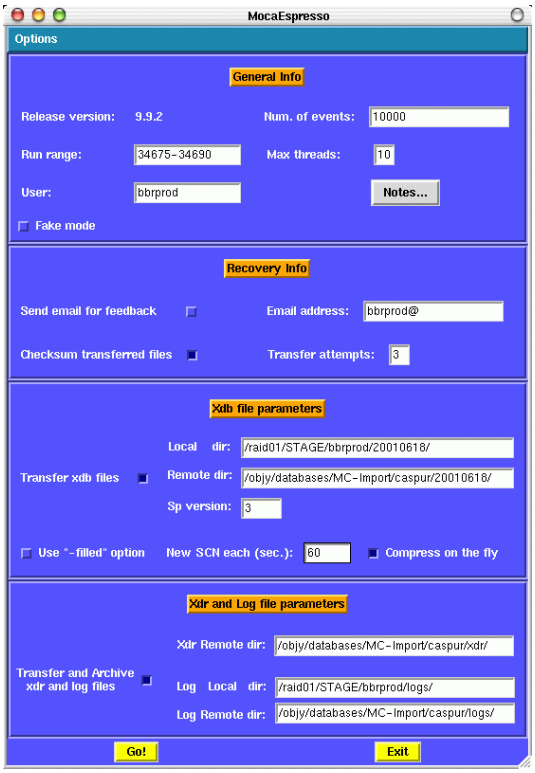


Figure 3.1: MocaEspresso GUI provided an easy way to handle the large amount of parameters needed for simulation

main tool for data export to SLAC, where all produced events were staged before further uses [71]. After the introduction of the new computing model, a new set of tools were developed to better interconnect data transfer and bookkeeping updates, leading to the actual configuration that was used as base model for the purpose of this work.

3.5.1 Software utilities: ProdTools

The evolution of the computing model in BaBar and the related distributed approach for SP, introduced the need for a set of tools able to provide the required support to easily manage the different steps during the production of simulated events. The ProdTools suite was designed to achieve this purpose and consisted of a set of PERL scripts and libraries grouped in two conceptually software layers. At the top of the

chain the main tools covered all the critical steps, like jobs creation submission on the local batch system, check of submitted runs, merging of completed jobs into collections and update of the bookkeeping at SLAC to keep general status in sync. Besides the main tools, other modules were provided to allow each site to build its own interface to achieve the best configuration for the production without modifying the standard tools, which maintenance was under the responsibility of the development team at SLAC, eventually adding new features. This flexible approach, already used in BaBar to uniform the access to the different kind of local batch systems supported at remote sites, allowed to write a software layer to use the Grid as the standard batch system for the BaBar SP schema.

ProdTools were regularly updated at SLAC and users could download new versions via the CVS system, while each upgrade introduced at local sites could be submitted into the repository for further usage. The most of operations performed by ProdTools kept also updated the main database at SLAC, delegating each remote site to keep information in sync without any extra effort from the main center. This distribute approach allowed to keep trace of the whole life-cycle of each job, from the build stage until the recording of data produced.

3.5.2 Data access for Simulation Production

The new computing model design represented an improvement even for the SP workflow, increasing the deployment of new data needed for the simulation process producing output data directly into ROOT format, easier to handle. During this transition step through to a full Kanga-ROOT system, condition data were still read from the Objectivity database while the background trigger, representing the detector's noise, were indeed converted into ROOT format too. A key element during this phase was still played by the Advanced Multithreaded Server (AMS) [72], a server component acting like an interface between the OO database and clients requests. The AMS server was designed to grants data access in different ways, including the possibility to fulfill requests coming from remote clients. Unfortunately, due to some weakness in its design, the standard version didn't perform as expected so a new modified version was written at SLAC to try to enhance productivity.

In order to understand the actual usability of the system, we implemented a testbed in Italy for both versions, using dedicated machines to simulate the typical stress conditions could happen at the server side when reached by an increasing amount of remote connections generated by real jobs. The aim of this test was to verify the robustness of such a system, focusing on its remote access mechanism in case of further implementation within a heavy distributed environment like the grid. Infact, inside the BaBar community, AMS server was used most locally within farms, providing access just to local machines without giving any kind of performance evaluation about an intensive use in a more distributed context.

3.5.3 Ams performance tests

Tests were performed using a dedicated data server in Ferrara, with Objectivity installed and configured to distribute typical condition data through an AMS server, while simulation jobs run on clients located at Ferrara, Naples, Trieste, Catania, Bari and Padova. Comparative measurements of performances were performed on both versions of AMS, standard and enhanced, configured in the same way: 32 threads used to handle client requests, for a total amount of 100 to 250 jobs of 2000 events each. Standard AMS reached its limits after just 30 concurrent simulation jobs, with a failure rate of almost 50% as load increased up to 80 parallel clients, while performances improved using the SLAC AMS, able to handle about 75 clients before jobs failure occurred. Limiting the number of connections for a single client increased the execution time for some jobs while allowing more jobs to contact the AMS in parallel. A good balance between efficiency and stability was achieved, setting the maximum number of connections to 20 for each client. In Figure 3.2 are reported tests performed on both servers.

A different test was performed to investigate the effect of connection speeds between different sites. The aim of this test was to verify the effective influence of distance on data access, in order to take into account the relation between data and clients locations. Two AMS servers were set up, the first at Ferrara and the second at Naples, while jobs run on clients locate at Bari, Ferrara, Napoli and Padova. Each test consisted of 200 jobs, each with 2000 events in order to simulate a real

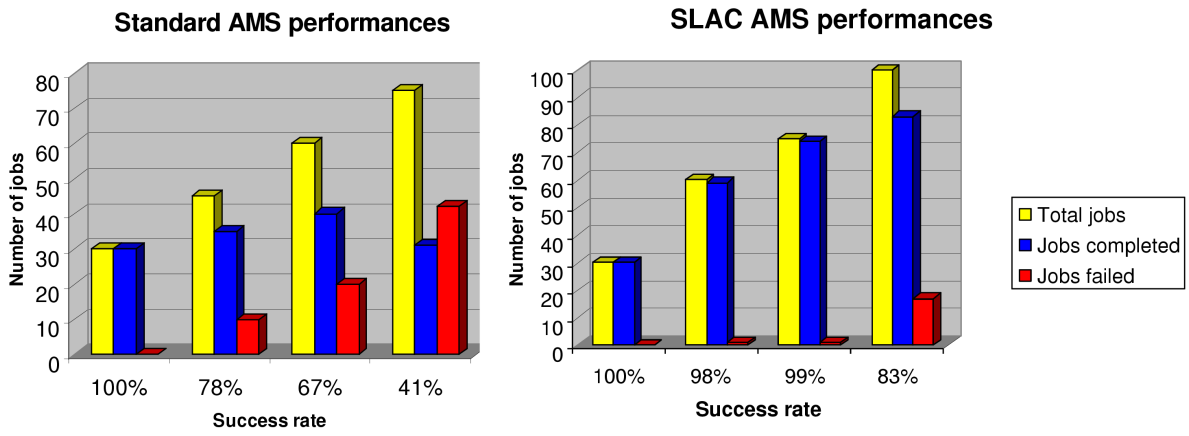


Figure 3.2: AMS servers comparison

production flow. The time spent to complete the jobs on each site was measured along with the maximum number of concurrent clients connected during production. The link speeds to Bari, Ferrara, Napoli and Padova were 15, 16, 32 and 160 Mb/s respectively. In the first part of the test, data were read using the AMS server installed in Ferrara, while production jobs run on clients located in Ferrara, Padova and Bari, successful completing the 99% of the submission. In the second part, the production jobs read the data remotely from the AMS installed at site in Naples. Due to a faster network link, Padova completed jobs more quickly than other sites, confirming the quality of network access and data location played a basic role in turn around times and is an important parameter in choosing the optimal running conditions. The whole system produced more than 3 million events, successfully completing the whole production cycle for 98% of the jobs [73]. The mean elapsed time for jobs accessing the two different servers is reported in Figure 3.3.

Despite the adoption of the enhanced version of AMS dramatically improved the stability of the system for distributed jobs access, it represented just a temporary solution. Scalability issues resulted in a bottleneck for large job submissions, moreover the setup required to achieve an acceptable level of production was not feasible to be deployed to all computing centers involved in a grid system.

However, AMS provided the required reliability for those sites of the collabora-

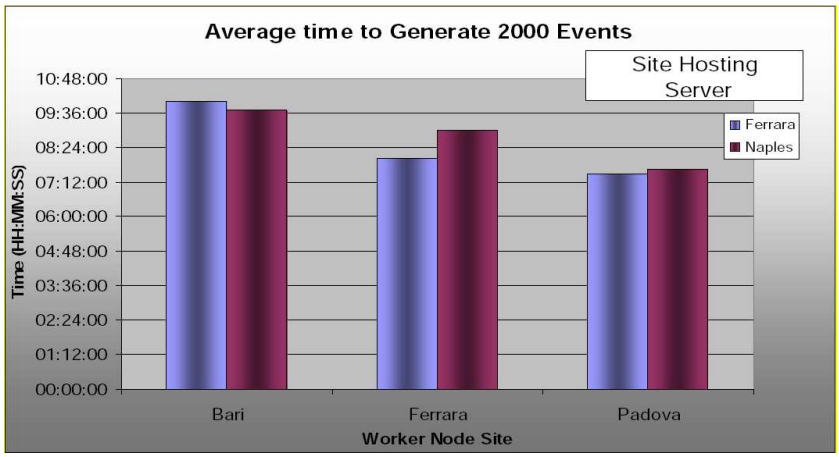


Figure 3.3: Access times with local and remote AMS servers

tion where more local servers were able to run at the same time, while for managing the events simulation task over a grid system other solutions needed to be introduced to implement an equivalent mechanism suitable for real production.

3.5.4 Xrootd

When BaBar decided to switch from a data eventstore based on database technology to the new computing model, a new approach in data access was required, in order to built a reliable system able to efficiently work within a distributed environment to retrieve information spread over different locations, and also supporting the ROOT protocol, chose as default data format.

Despite the standard file access mechanism provided by ROOT via a TCP/IP based data server daemon known as rootd, already capable of transparent data access by a plugin manager layer, Figure 3.4, this solution was not suitable for the purpose, resulting in a waste of system resources when multiple requests needed to be handled concurrently.

This new scenario pushed the efforts toward the data distributing field, leading to the development of the XrootD suite [74] which main components consist of server daemons and clients that communicate via a dedicated protocol, maintaining a backward compatibility with the original rootd server. The main goal was to design

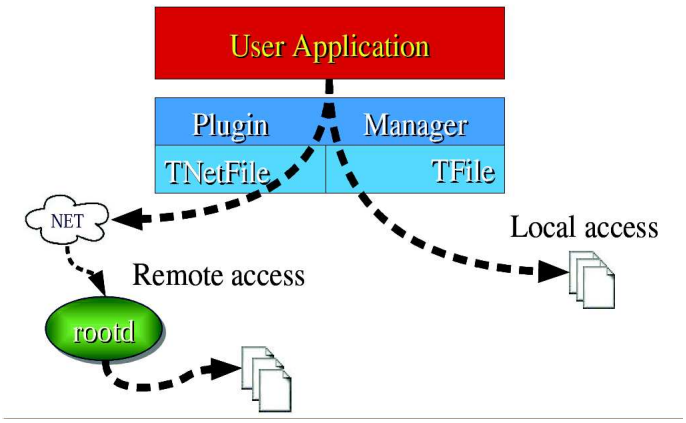


Figure 3.4: Mechanism for transparent data access in ROOT

a new architecture in order to achieve transparency in data access for clients, that had to be unaware of servers underlying file system type, while providing cooperation among different servers to balance and distribute the workload. Moreover, many efforts were requested to optimize system resources to gain the best performance on both server and client sides, taking into account the need for a high level of fault tolerance behaviour to face connection problems could occur during client-server communications. This new system outclassed the standard rootd service, and fulfilled the need of a real production environment, where thousands of clients could send requests to a large number of servers. Due to this new set of features, the system is considered an extension of the classic rootd (eXtended rootd) [55]. The main component consists of the xrootd file server capable of high scalability over a P2P network, implemented as an architecture plug-in based. Moreover the protocol developed provides for a generalized security framework, while both the client-server paradigm and p2p models are embedded in the same file access protocol, resulting in a high level for scalability.

Multiple parallel requests per client are allowed reducing system resources consumption thanks also to the connection multiplexing system adopted to let many clients to share the same TCP connection. Different kind of components, like data servers and redirectors cooperate to export a unique namespace. The client is developed both as a generic Unix implementation and as an extension of the classic

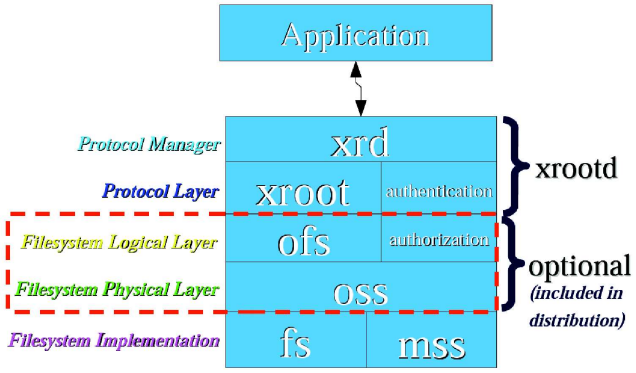


Figure 3.5: Xrootd server architecture

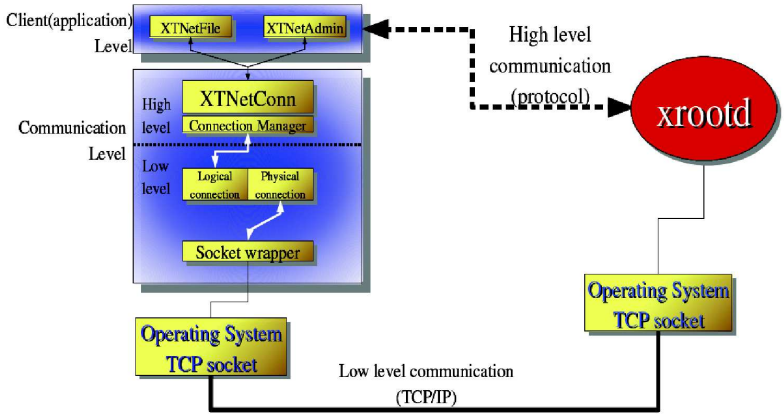


Figure 3.6: Xrootd client architecture

ROOT framework, called TXNetFile, implementing the fault tolerant and reliable behaviour defined in the communication protocol along with strategies for data caching.

Xroot/TXNetFile provided very good performances in distributed file access, playing a key role in the development of the simulation production over the Grid.

3.5.5 Output storage

After jobs completion, users can retrieve output data by using ProdTools scripts, that take care to automatically collect jobs results. A parallel task provides to merge all data into collections of about 1.5 GB each, ready to be exported to the

main center at SLAC. Here collections are handled by automatic import tools that store them both into the local mass storage and into the High Performance Storage System (HPSS) [75] for further usage. HPSS software manages petabytes of data on disk and robotic tape libraries providing a highly flexible and scalable hierarchical storage management that keeps recently used data on disk and less recently used data on tape. HPSS uses cluster, LAN and/or SAN [76] technology to aggregate the capacity and performance of many computers, disks, and tape drives into a single virtual file system. In order to have HPSS work efficiently it is important to have large files be put into HPSS, since the time to mount a tape and to position it for file reading are significant (ten's of seconds) so files sizes of 1GB and larger are desirable. When import of collections is completed, the related status is updated in the bookkeeping an data are marked as ready for further analysis [77].

Chapter 4

BaBarGrid system

In the previous chapters two different computing environments have been described, the INFN-Grid model and the BaBar architecture, represented as standalone entities. In the following sections it will be highlighted the strategy adopted to join together the common components of the two environments to design a suitable system to efficiently perform simulation production on the grid.

4.1 Simulation production requirements

When the grid paradigm was proposed as a new alternative way for the BaBar SP production [78] [79], just few implementations took place at the beginning. The main issue concerned that the traditional production model was already working at full speed achieving very good results, and despite the new approach was considered promising due to the several benefits it could offer, it neither provided an easy way to quickly join the framework of the experiment nor offered the required flexibility for easily setup a parallel environment capable of the same level of productivity. By the time the grid model was introduced, the average rate of simulated events was of about 40 million per week, run over more than 2000 CPUs spread among different sites. A such huge amount of resources became an issue since BaBar was not involved in any grid project capable of the same potential, so a detailed plan had to be designed to achieve this goal and allow a fast prototyping for a new design to deploy over a grid platform. A new testbed was created in Italy based on the INFN-Grid project, while the main center for tests was setup by the local INFN

section based at the Ferrara University [80].

The need for a broad platform where running the simulation led to an evaluation of the resources offered by the Italian grid, in order to achieve the best compromise between the maximum number of effectively usable sites and performances. As a matter of fact, different aspects had to be taken into account, concerning both practical and technical issues like:

- agreements policy for sites access and utilization
- local resources management and problem solving
- network link quality and data distribution

The Italian grid shares its computing resources among different kind of experiments, so the first step to log in was to define a Virtual Organization for BaBar to allow a full access to any kind of facilities provided and exporting a uniform interface for users utilization. As general rule, a site can decide which VOs to support and since many institutes connected to the INFN grid were also involved in BaBar, a first testbed was built considering that intersection as policy.

Another critical issue concerned the manpower required to handle those resources at candidate sites, as just standard services were provided without any support for specific BaBar requirements, like reliable solutions for input data access needed for the production of simulated events. The general agreement in BaBar was to allocate at most a couple of experts to manage production at remote sites, so the new approach had to fulfill or eventually improve this requirement.

Besides, to reach the optimal balance, data had to be distributed in a such way to allow good performances, favouring high speed connections to decrease latency for input reading, especially for those sites that required remote access. The most obvious choice would have been to prefer just sites with the higher number of computing slots available, but in practice this was not always the best solution since many different type of jobs were generally routed to farms belonging to this kind of typology, resulting in an increment of the length of the queued jobs list. A first testbed was built on a group of 8 sites (Padova, Ferrara, Bologna, Pisa, Perugia,

Napoli, Bari and Catania) selected over a set of 20 for a total amount of about 500 CPUs, paying attention to cover the most of the national area [81] [82].

4.1.1 Data access

Data distribution and access was a critical point to investigate since its fundamental role played in the production process. The main drawback concerned the lack of a full integration between the BaBar data services and the grid storage facilities, moreover the main executable for the simulation needed to read a large quantity of data which was not opportune to move closer, indeed the typical size of a full conditions dataset was of about 60 GB, while a background triggers set was of about 15 GB. Technically speaking, move the whole set of input data closer to the main application before starting the simulation it would have been an easy step, but due the way data were read, it could not be accepted as good compromise. The executable was able to read data both locally and remotely, but in the first case it would have been necessary to copy almost the whole set of data to the disk where jobs would have taken place for execution and where it would have been automatically removed by the system after jobs completion. On the other hand, to access data remotely, dedicated servers were required to store and distribute data via thexrootd service. In order to understand which methods provided the best solution, a comparison between different access methods was made within a real production environment, reading the background data ROOT files as follows:

- Direct access via a local Xrootd server using its protocol
- Directly out of a local dCache Storage Element using the DCAP protocol
- Copying data from a dCache Storage Element to a local disk, by using the LCG data management tools via the GridFTP protocol, where files could be locally read by the job

A set of 100 jobs composed of 2000 events each were run once for each different method working on the same hardware set, while the average time to complete the task was calculated, providing as input data set a collection of background trigger

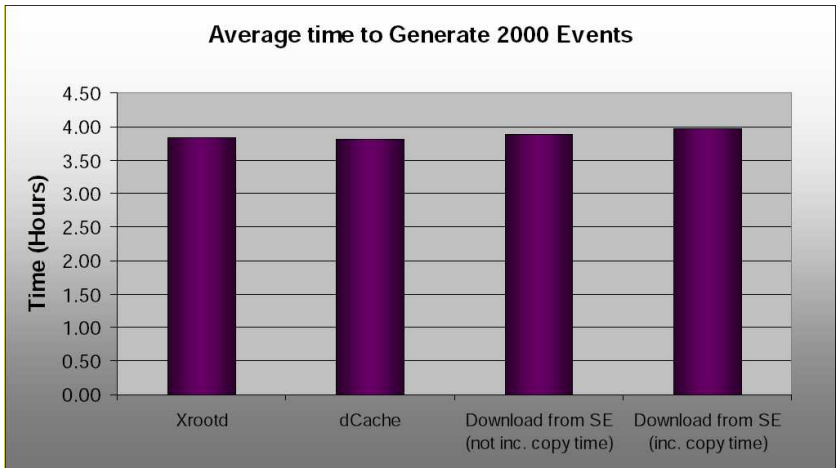


Figure 4.1: Mean elapsed time for different access methods

consisted of a set of ROOT files for a total amount of 2.5 GB. The second and third methods tested were included because considered good candidates for providing data in a scenario where BABAR specific services were not required [83]. As shown in Figure 4.1 , the difference among input methods is almost irrelevant since the time taken to read the data is low enough compared to the computing time, even considering the time taken to copy the files to the local disk.

Due to these slight differences and taking into account the limit introduced by copying data to each job location every time a new simulation process starts, the final decision considered the xrootd access method the most appropriate for providing data, in order to uniform the access interface for input data reading on the whole set of grid sites. Since the above approach required the installation of dedicated machines external to the grid framework, three xrootd servers were configured and placed on different farms to provide the best coverage for clients demand, trying to achieve a balance to minimize the network latency for those sites that had to require data remotely over a wan connection. Despite it was not possible to install an xrootd server at each site due to management/permissions problems, the usage of three servers really improved the production rate, thanks also to the features provided by the xrootd suite.

4.1.2 Storage facilities

The introduction of the new BaBar computing model allowed for a more flexible data management since results could be written directly to single flat files instead of interact with a database interface like Objectivity. As general improvement related to the simulation production, it was not necessary to install dedicated AMS servers at remote sites since this procedure have been replaced by the installation of Xrootd servers which setup provided a much smoother configuration, resulting in a reduction of the number of people involved for remote site maintenance. More specifically, when new condition/background data were required to be imported at local Xrootd server site, just a simple copy transfer of data took place from SLAC or from an already updated server, dramatically reducing update timing.

Particular attention has been dedicated in the design of the output storage mechanism, since the critical role played in the production workflow. Requirements for this step were quite strict since the simulation output took several hours to be completed, asking for a reliable procedure to store results safely. Output data were also written into flat ROOT files so it became quite easy integrate the common storage facilities provided by the grid middleware with the production workflow. In order to built a safe and robust mechanism for preserving output results, jobs had to communicate with the standard LFC catalog to associate output data name to a unique logical id, easily accessible from the whole grid environment as the default bookkeeping service, besides different levels of data transfer recovery from worker nodes to storage elements were introduced, as backup solutions in case of failures during the registration into the catalog.

4.1.3 Submission jobs grid compliance

A critical point that needed an accurate planning, during the migration to the traditional production model of BaBar to the grid system, concerned how to build and submit jobs in a grid-like way. All single farms belonging to the BaBar collaboration already have installed local batch systems, like PBS or LSF, required to schedule incoming jobs over the computing resources availability. The idea was to create an

interface layer between the job creation and the job submission steps, in order to allow the reuse of the code already written for BaBar's purposes, specializing it for the grid paradigm. More specifically, the SP software provided the ProdTools suite written to fulfill every production requirements that had to be modified for achieving the target. The integration had to handle two different aspects of the workflow: the first was related to the setup of a grid job complaint with the grid submission system, taking care of creating a specific JDL wrapper for each single job where encapsulating all needed information for a successful submission. This step required also to design a mechanism to deal with input data needed for the simulation, along with the managing of the output produced and more generally to define an environment to safely perform the job execution. On the other side, integration had to face the incompatibility between the grid monitor tools and the mechanism provided by BaBar for the same purpose, resulting in the development of a new mechanism to translate information produced by the BaBar submission workflow to a stream compatible with the grid environment. This process was critical to maximize the reuse of the already written code developed for BaBar providing at the same time a correct management on the grid of each single simulation production issue. Moreover, specific new tools needed to be planned to provide mechanism like a realtime monitor for the retrieve and check of submitted jobs.

4.2 Software distribution

In order to efficiently run SP jobs on the grid, a re-organization of the BaBar software was required to take advantage of all benefits offered by the new environment. Software for simulation production needed to be easily installable on grid nodes resulting in independent package ready to run everywhere without any external reference.

4.2.1 Simulation Production software

BaBar software is based on releases, where each release is a set of stable packages and library suitable for different tasks of the experiment. Periodically, when new features come into play, a new release is built and deployed to the collaboration,

resulting in an improvement for the capabilities of the software. In the traditional BaBar framework the whole release has to be installed at remote sites which required it, introducing a considerable delay due to the fact a single release is about 2 GB in size, and must be downloaded and then fully configured before using. This limit was not compatible with the grid approach, where the need for a capillary distribution requires to package all the needed software in a format easily installable everywhere without any extra adjustment by the local site manager. Moreover, one of the benefit introduced by this system would have been to cut the amount of persons involved in the management for external sites not directly accessible by the production manager. In order to fulfill these requirements, a dedicated module was developed to extract from the main release just the portion of software dedicated to the production of simulated events, creating an independent component ready to be installed at any computer element via the standard grid submission. The process uses the grid bookkeeping catalog to make the software package available everywhere on the grid, while a specific grid job, sent to the target site, takes care of downloading the package and to install it. Following this procedure allows a fast distribution of the package without involving managers at remote sites. In order to allow the simulation production manager to install his own software to an external site, a special grid user was defined to gain the right privileges to perform this kind of operation.

4.2.2 Resources pool selection

The SP manager is responsible for selecting the pool of grid farms where jobs will be executed. This choice is performed before starting the production workflow, but new sites can be added to the pool even later, and it depends on several factors like the total amount of slots available, site location and local support provided in case of problems. Generally, a group of 6-8 sites is considered the standard pool where submitting jobs since, as the practice demonstrated, such solution provides the right balance between resources needed and ease of management.

As general rule, sites are selected considering how far they are from data servers to improve access to input data required for the simulation, along with the level of support provided to fix local problems on selected farms. Due to the dynamic man-

agement of the grid resources and to some weaknesses in the Information System, is not possible to delegate this choice to an automatic mechanism. Indeed, local policies could be adopted at remote sites to distribute free resources on the basis of specific criteria, allocating, for example, the most of free slots for jobs execution to a specific experiment and usually these kind of information are not published by the Information System.

After several tests it was possible to select a pool of nodes which demonstrated to fulfill all requirements, offering a good level of support for troubleshooting and able to provide a stable platform for getting into a real production flow. Special tags are used to mark selected CEs where software is installed correctly to allow the Resource Broker to easily identify nodes belonging to the pool of suitable resources.

4.3 ProdTools changes

Since the traditional SP workflow for jobs is not suitable for the grid environment, the ProdTools suite have been modified accordingly. The following section describes for which tools changes were required while all implementation details will be given in the next chapter.

4.3.1 spbuild

The build process maintains almost the same schema provided for the traditional model, but important differences have been introduced to prepare the job structure for the submission over the grid. In particular, a new software library has been implemented: when the spbuild script is run, the information regarding the characteristics of the job are obtained by the bookkeeping at SLAC as usual, but during the initialization the build process interacts with the new code to create any additional files needed to wrap the job, providing the compatibility with the standard grid submission instruments.

4.3.2 spsub

In order to control jobs running over the grid, a dedicated mechanism has been integrated during the submission step. The spsub script has been modified to directly

use the standard grid submission tool, providing all information needed for checking the whole life cycle of each production job to the custom monitoring application specifically developed. As for the previous step, all new features provided by the library are implemented in a transparent way from production manager's prospective and maintained on a separate level to maintain an high grade of flexibility.

4.3.3 spmerge

Collections are aggregated using the same mechanism provided by the old workflow, since this operation is not performed over the grid. The main difference consists to execute this step on a local machine instead of using the same pool of nodes that perform the simulation in order to optimize the workload. The reasons that lead to that choice along with details about its implementation will be discussed in the next chapter.

4.4 General workflow

The main idea about SP on grid was to allow one single site manager to efficiently manage the whole production cycle by his own, resulting in a resources saving through a central coordination of all sites involved. After completing all tests on SPGrid feasibility, the whole project has been installed at INFN CNAF (Italian National Center for Research and Development about Information and Data transmission Technologies) which became the main center for SPGrid in Italy.

4.4.1 Environment setup

All software needed for BaBar SP, has been installed on the User Interface from where jobs can be submitted over the grid. This machine plays a key role since from a logical point of view is in the middle of the two environments, supplying configuration settings for both BaBar and Grid software. As first step, the production manager logs in to the UI and all required settings for the BaBar SP are executed. This step allows to identify locations where storing new jobs before submission and provides links to other important parts of the software of the experiment, like the ProdTools repository and the main BaBar software releases installation. The latter

location is used by those components within the ProdTools involved in operations that take place before and after the simulation over the grid, which related packages versions must be in sync with the current release developed at SLAC. After the completion of this step, which for the most is transparent from the user's prospective, the UI presents to the SP manager the same kind of environment it would be provided by a standard BaBar machine, while the standard settings already supplied by the UI offers all necessary functionalities to communicate with the grid.

4.4.2 Run allocation management

Once logged in, the SP manager queries the central database at SLAC to ask for new runs to produce. Typically a run consists of a production job made of 2000 events at least and is provided with all configuration and data files needed for computing the simulation correctly. The SP coordinator in charge at SLAC arranges new bunches for all sites belonging to the collaboration to fulfill clients request automatically when they run out of runs. Different types of runs can be served to remote clients, but generally the most of sites combine efforts to work on a common goal.

The infrastructure implemented at CNAF has been identified at SLAC with the name "infngrid" to be included in the list of sites capable to give a contribute to the simulation production, hiding details about the grid-based implementation in order to provide a transparent interface from the SP coordinator's prospective.

4.4.3 Services

In order to complete the initial setup, the user starts a set of services assigned to specific purposes. Scripts for merging collections produced by simulation, along with the one that provides to transfer them to SLAC must be activated, moreover a dedicated monitor application is run by the SP manager to control jobs over the grid. After these operations, the whole system can run automatically, while the only manual operation required is to query for new runs to build. Due to the distributed nature of the grid this operation was performed manually to avoid any kind of problems could occur at remote sites and that could not be easily fixed remotely by the production manager once runs have been allocated. Merged collections are

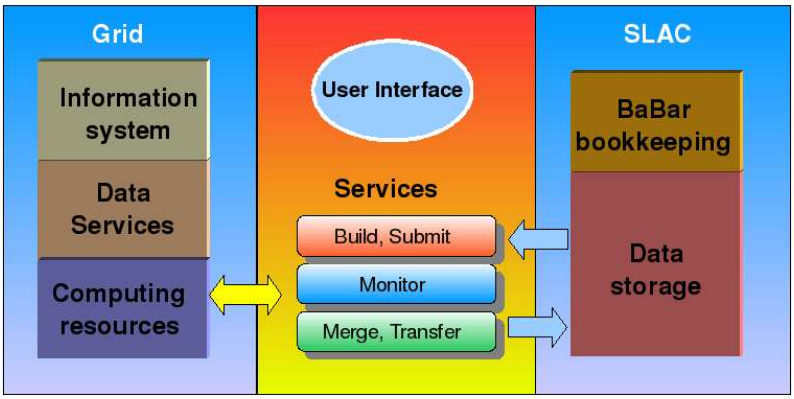


Figure 4.2: SPGrid schema

continuously exported to SLAC where automatic tools take care to import them in the main repository, while the BaBar bookkeeping is updating accordingly.

Basically, the SPGrid system is designed over two layers to organize both operations performed over the grid and those executed locally on dedicated machines. From this prospective the UI links local BaBar services to the grid through a monitor application that takes care to check production status. Monitoring the simulation over the grid represents the core service since it was designed to perform different tasks in order to coordinate the production workflow. The SPGrid monitor application gets as input data the references generated by the submission step to trace jobs over the grid, providing to retrieve and check results which are then passed to the next steps, executed locally on the User Interface. A web interface is also supplied as visual feedback about the ongoing production. Tasks performed over the grid can be further subdivided depending on functionalities provided, like for example pure computing, data transfer or information system operations.

4.4.4 Execution environment setup

When a job reaches a Worker Node a sequence of steps is performed before running the simulation process in order to set the execution environment accordingly. In particular the right version of condition and configuration data, representing the input domain for the simulation, must be set along with the location which offers the

best access for reading those data. New conditions and background data versions are periodically provided from SLAC and installed on each Xrootd server to be accessed by grid jobs. A map for the right input data to read is dynamically generated on the WN before running the simulation, besides the most convenient server is chosen on the base of job's location. This mapping takes into account where job is physically located and select the server using a built-in schema where a list of target machines is defined. The schema provided is set statically by the SP manager inside a specific file read during the job initialization and can be modified as needed. Due to the limited amount of external resources, actually just three Xrootd servers are installed within the SPGrid system, jobs routing toward input data machines is defined considering factors like distance and the amount of bandwith for location where a server is installed.

Chapter 5

BaBarGrid: implementation and results

The following sections outlined the hardware and software organization required to set up the SPGrid infrastructure, covering all aspects involved. The second part of the chapter discusses results achieved through a comparison between the other sites involved in the collaboration, where the old production system is still adopted as the main approach.

5.1 Organization of hardware resources

Several hardware components are required to build a reliable infrastructure able to connect SP functionalities with services provided by the grid middleware and the User Interface represents the gate among these two systems. As described later, the most of the BaBar software is accessible trough this machine, which mounts different partitions data using the Network File System (NFS). Since it also provides the common middleware services for interacting with the grid, is recommendable to preserve its standard configuration as much as possible, to prevent serious system overloads that could compromise fundamental functionalities. Therefore custom applications developed for managing the SPGrid workflow and installed on the UI have been designed to respect this directive.

Another fundamental component of the infrastructure is the Computing Element, where SPGrid software is installed. Like for other experiments, specific software package are installed in predetermined disk area on CEs, which provide to export

its contents to all the underneath Worker Nodes of the related farm through NFS.

Next sections covers in more detail custom setups performed to adapt the SP architecture to the grid system used, taking into account the I/O data management for the simulation.

5.1.1 Input data setup

Since from the beginning one of the most critical part for the interoperability among SP and the grid was represented by the need to find the good balance between the constrains defined by both environments without limiting performances. Hardware setup had to take into account three different levels of service, like jobs submission, I/O data management/bookkeeping and results publishing. Data handling required a specific customization since input data needed for the simulation process were installed on dedicated machines and made available via Xrootd servers. Each server has been installed and accordingly configured, while ROOT files for conditions and background triggers have been copied on dedicated partitions on servers.

The main access for SPGrid submission is provided by the UI, where the ProdTools suite has been installed and configured, since most of the critical steps of simulation production are managed from this middleware component. Besides jobs submission over the grid, other processes installed on the UI provide features like jobs monitoring and retrieving, while even the merge step is performed locally on this machine.

Within the ProdTools directory has been created the repository where is possible to modify templates provided for local configurations, grouped into sub-directories for each different site. At this level the infngrid directory includes three fundamental components written to achieve the SP-INFNGrid interoperability: batchutils.pl, Moose.bash and local-grid-setup.

The first one is the common template that comes with any new version of ProdTools and where is possible to add new features for a specific site, like the generation of a grid skeleton for the job when it is built. Moreover, it supplies a mechanism to support the submission of two different kinds of job, those related to the real production and those performed for merging output collections.

The local-grid-setup file contains information to point the job toward the right Xrootd server, providing the access endpoint on the basis of job's location. This file also defines some environment settings like the default Storage Element to use for data storage, the entry point in the LFC catalog where registering metadata information and the description of which version of condition and configuration data using for the current production, to allow the job to reach the correct information on the Xrootd server assigned. When a job runs on a Worker Node, the local-grid-setup file is executed within the main job's wrapper. After assigning the most appropriate Xrootd server, it provides to write a specific KanAccess.cfg file which will be used when the job will attempt to contact the server, and where is written a set of rules that describe how to access data on that server. Moreover, some special files are created to describe which conditions and configurations data need to be read from the Xrootd server assigned. Before starting the simulation, the job will read those files to get the right version of input data requested and will try to access them on the server. The local-grid-setup file is fully customizable, so when conditions and configurations need to be update, generally after on average one month, it's considerably easy to change the old versions with the new ones. Naturally, the setup provided by this file is applied to all the jobs involved in the simulation production for a specific version of input data.

The Moose.bash script represents the main wrapper in charge of setting the environment of the WN where job will be sent, therefore it inspects all configuration files supplied with the job by the central database at SLAC, adding information to accomplish the right execution on the grid, including those defined by the local-grid-setup file. When the job arrives on a WN , the Moose.bash is run and takes care to execute the following tasks:

- setting of the job's environment after reading all configuration files supplied with the job
- running the simulation according to the directives provided by the local-grid-setup file for input data access
- checking the collection produced using the standard BaBar's procedure for

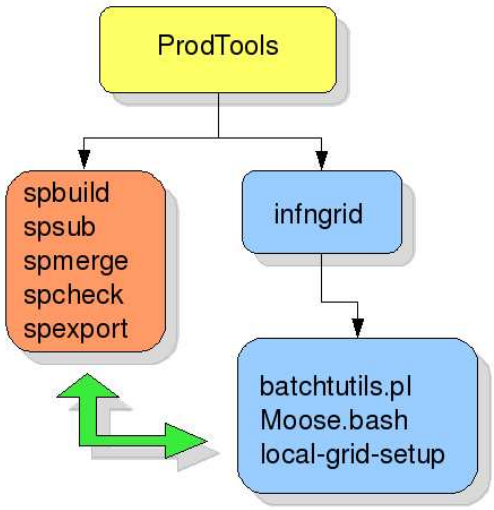


Figure 5.1: ProdTools repository tree

inspecting SP collections

- transferring output files to the Storage Element selected

5.1.2 Output data handling

A critical part within the production workflow is related to the validation process for collections created during the simulation. Since inspecting a collection can take several minutes, the choice was to delegate this operation to the node were execution started and not to the User Interface where the monitor tool takes care of retrieving data, therefore, after job completion, the following actions take place on the Worker Node:

- inspection of the collection to find corrupted parts
- attempt to register the checked file into the LFC catalog, using as storage unit the closest SE for the WN where job run

- if errors occur, the file is transferred to the same SE but using low level data transfers utility like GridFTP
- if the closest SE is down, a GridFTP transfer is performed to the default SE, typically the one installed at CNAF

The transfer step is implemented as a three levels approach to minimize problem with the grid bookkeeping service, that could be not available, trying to safely store output data. As practice demonstrated, this solution provided a pretty robust mechanism to avoid data loss, thanks also to the redundancy offered by the INFNGrid infrastructure in terms of services availability. Checking collections on WNs allows a concurrent control for results obtained without overloading the monitor application on the UI.

5.2 Software setup

Dedicated software components were developed as support to the SPGrid project, representing the effective core of the system. Several aspects have been taken into account to achieved the necessary interoperability and to build a reliable and efficient system in order to monitor the SP activity on the grid.

5.2.1 SP software packaging

In order to provide an efficient distribution of the simulation software over the grid, a specific tool written in PERL was designed to extract from the main software release of BaBar just those components related to the production of simulated events. Due to its size, about 2 GB against about 200 MB of code really used for SP, a complete release is not suitable for being installed to all selected CE, besides new versions are released quite frequently resulting in a slow management for removing and re-installing the software.

The extractor-tool executes the main application, performing a short simulation to match the same conditions of a real production job. The test is performed on the UI using configuration files for a real job while input data are reading by one of the Xrootd server available. Indeed, a copy of the KanAccess.cfg file used in the real

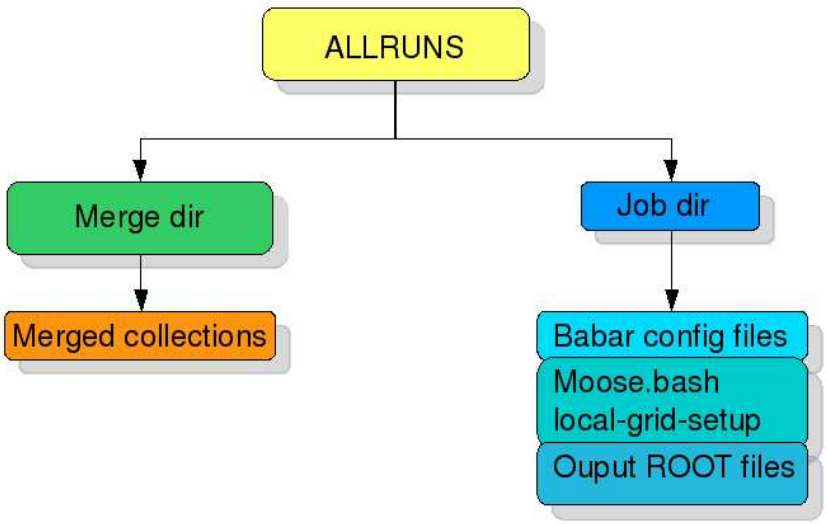


Figure 5.2: Jobs directory tree on UI

production process is maintained on the UI. When simulation is over the resulting output log contains all the system calls performed, by means of, all libraries and data files are included in a single tar archive ready to be used as self consistent unit. The next step consists to register the package on the LFC catalog to make it available on the grid, trough a standard job that provides to download and install it on the target CE, where is shared between the beneath WN.

5.2.2 Jobs creation and submission

When jobs are built by the spbuild tool, the main database at SLAC is queried to obtain information related to jobs. The directory ALLRUNS is created on the UI along with a sub-directory for each single job as described by the structure shown in Figure 5.2:

Inside each job's directory several files are generated when the run is built, representing the complete description of the job in terms of kind and total amount of events to simulate using a specific decay mode for particles involved, along with the

type of background triggers to use. All these information are stored at SLAC for each single job and distributed to remote sites when the SP manager decide to built the allocation given. From the simulation production's point of view, these files come into play after the submission step, when the job reaches the Worker Node, to be used both just before it starts to simulate the bunch of pre-assigned events to correctly setup its working environment, and during the real production stage. Moreover, each time a new job is built, two fundamental files are also copied from the infngrid repository to the job directory: the "Moose.bash" and the "local-grid-setup". These two files are basically the same for all jobs: the first never changes since it contains the consolidated list of steps to perform a single SP simulation, while the second is updated only when new conditions are provided by SLAC.

The need for the capability of submitting jobs directly over the grid required the introduction of some changes into the standard process used by BaBar to build jobs. In particular, the "spbuild" script has been modified to automatically generated a JDL file for each new job created, thanks to a new library developed to provide grid functionalities to SP jobs. This software layer overrides the standard template provided by ProdTools, the "batchUtils .pl" file, which has been modified to offer a set of functionalities for using the grid. In this case, a check on the input parameters of the job is performed and integrated with other information to make the job suitable for a grid submission. Specific requirements are defined into the JDL wrapper for routing the job toward CEs with particular characteristics, including all configuration files needed for SP. At the end of this procedure a run is ready and can be submitted using the standard grid procedure (Figure 5.3).

The library implemented supplies also other functionalities for the submission step since different use cases could occur according to the kind of jobs could be submitted. The "spsub" tool is actually used both to perform a regular submission over the grid and for processing a merge job when collections produced need to be aggregate into a single file.

In the case of a standard job, the submission can be related either to a real production activity or to the validation process, which consists to send a well defined range of jobs to a particular CE to validate input data installed at that site. This in

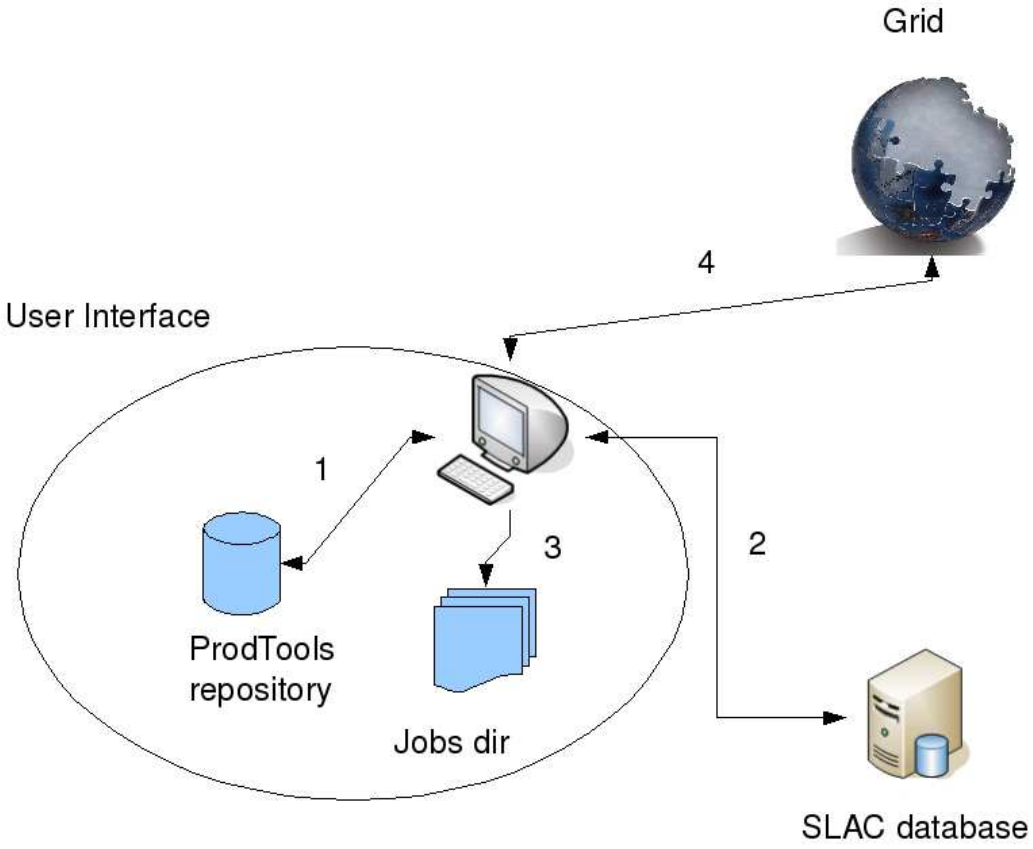


Figure 5.3: Jobs submission schema

a mandatory step before starting a new run allocation in order to be sure to produce correct results. Generally, at traditional BaBar sites the same batch system is used both for production and merging purposes, but this solution was not suitable for a such grid environment due to the imposed constrains, therefore was necessary to handle the two cases separately.

Merging jobs is a very intensive task in terms of CPU load and requires to install the ProdTool suite locally to run, moreover merged collections are stored within the ALLRUNS dir, making difficult to distribute the whole process over grid nodes. More important, this process was designed to run over a dedicated pool

of resources, where each merge is performed independently by others, but in the case of the grid it would have meant that two different kinds of task (simulation and merge) had to compete using an already shared pool of resources, probably resulting in a extreme time consumption process. To solve the problem, merging processes are executed directly on the UI, accordingly configured to support the load, implementing a special wrapper to let the spmerge script to fork a new child process for each single merge, performing the task as a parallel operations. As the practical experience demonstrated, the maximum number of parallel merge jobs depends on system resources availability in terms of memory and CPU power. Despite is not possible to achieve the same efficiency provided by a dedicated batch system, this solution allowed to reach a good compromise with at most 8-10 concurrent jobs for daily production peaks of about 15 million of simulated events.

5.2.3 Monitor

A fundamental feature for the management of jobs submitted to a batch system is represented by the possibility to check the status of each single activity, in order to have a feedback about the behavior of the whole process. Simulation Production already provided a set of scripts for this purpose, but the design proposed concerned just the usage with a traditional local batch system. Moreover, some implementation details put too many constrains for achieving an efficient deal with a distributed environment like the grid, therefore a more suitable solution was planned to build a custom tool for monitoring grid jobs for SP.

The monitor was designed as a two layers concept, a part acting as backend which provides the core functionalities of the application, and another layer represented by a set of tools that provides the frontend for the user, where results can be monitored through a web interface, implemented as a classic producer-consumer logic. The main idea was to develop a tool able not just to report jobs status, but also to decide which kind of action take for each single case could occurred during jobs life-cycle. The resulting application takes care of checking the status of each single job, retrieving and validating results or eventually resubmitting failed jobs automatically.

5.2.4 Monitor backend

The SP-Grid monitor runs as a Unix daemon that periodically checks status for already submitted jobs and also verifies if any new job has been recently submitted. This check is performed reading the list created during the first submission step, saved in a dedicated file, where for each job is maintained a link between the original SP run number and the relative grid id. Besides, a job can be submitted to the grid more than once in case of errors, therefore this list could contain a new grid id for the same job (run) id.

The first time a job is submitted after being built, its status is updated in the main database at SLAC, following the standard SP procedure. As mandatory rule of BaBar, when a job fails is necessary to rebuild it using the same run number but with a different version tag before trying to resubmit it again, in order to preserve the integrity of the main database and to trace its whole history, marking the old version as failed when simulation did not end successfully. On the other hand, this kind of approach could become quite inconvenient to handle when jobs fail frequently for problems due just to the grid system underneath. Rebuilding a large amount of jobs slows down performances and increase the overhead between jobs submission. To avoid this problem, further resubmissions for a single job are performed without interacting again with the central database at SLAC, but simply resubmitting it over the grid. Job's status is then updated to "resubmitted" while the new job's entry is added to the list as it was a new re-built submission from the SP system' perspective. This procedure can be considered safe from the BaBar management's point of view since the most of failures are not generally related to simulation errors but just to common grid drawbacks, like software not properly configured at remote sites. However, simulation errors can occur sometime and in this case the job is just dropped after a specific timeout letting the garbage system, provided by ProdTools, to update accordingly the BaBar bookkeeping.

The monitor daemon maintains a data structure for saving jobs status, which is periodically updated taking into account only all valid jobs actually active on the grid. This new set is both saved on a file, to be further compared with the submission

list, and stored in a table in memory as to be used as a catalog for retrieving jobs information. A comparison between the monitor's table and the submission list is performed each time monitor awakes (generally after 10 or 15 minutes on the basis of user customization), in order to check if new jobs have been submitted and to verify also if any new versions for old jobs have been supplied. The operation is performed on the basis of the submission date: if a new run is found into the list of submitted jobs and it is not already present in memory, it is simply inserted as a new element into the table, otherwise a check is performed on the submission date to consider just the newest version of the run found. The old version states that some kind of problem occurred and that job is not running anymore, therefore it can be ignored.

In the main table stored in memory are kept all information needed to trace jobs behavior, including both the SP and grid ids along with the location where jobs have been sent for execution and, of course, their status (Figure 5.4).

SP job id	Job dir	Grid job id	Status	Start time	Site
14944901	A24.3.5V01x67F	https://gridrb.fe.infn.it:9000/VGHQ8z8yCY8XZlPnmF_ZHQ	Archived	1222937641	CNAF
14945602	A24.3.5V01x67F	https://gridrb.fe.infn.it:9000/Ky5svYSPcGlvRss8yqhEpw	Running	1223369224	CNAF
14945603	A24.3.5V01x67F	https://gridrb.fe.infn.it:9000/Htw9TFneJLWOz6-jAUbIFw	Running	1223369296	CNAF
14945604	A24.3.5V01x67F	https://gridrb.fe.infn.it:9000/x-swSp4olufmyrLCaXb_Gg	Running	1223369318	FE
14945605	A24.3.5V01x67F	https://gridrb.fe.infn.it:9000/NqYYVtuR9mG6_uuVOYqzHw	Merged	1223398210	NA
14945606	A24.3.5V01x67F	https://gridrb.fe.infn.it:9000/YpIoV3xV5vGzwhw9lPXa4A	Submitted	1223369424	BA

Figure 5.4: Table for jobs information

From the monitor point of view, the pool of possible states a job can walk through includes all standard grid stages plus others related to the operations performed off-grid, like building and merging.

Pre-Grid	Post-Grid
Built	SPChecked
	Merged
	Archived

Table 5.1: Monitor off-Grid status

Check for jobs status is performed incrementally, considering only those jobs that have not yet reached a terminal state, excluding all completed ones, along with those marked as failed or that need to be resubmitted or have been moved toward to other off-grid steps like merging or transferring. For all active jobs a temporary list of their grid ids is collected and used to perform the status check as a unique bulk request, through the standard tool provided by the gLite middleware. The output is parsed and the main jobs status table is accordingly updated with new information. Failures for jobs submitted could occur for several reasons, due both to a problem on grid sites and to errors related to the simulation process. In such cases a list of failed jobs is kept to allow an automatic resubmission. Jobs resubmission is also automatically performed for those jobs that waited on queue for more than six hours at remote CEs, in order to optimize the distribution over all the available resources and gain better production performance.

Each job completed is then retrieved, copying output log files from the RB to the UI, while the ROOT file containing simulation results is copying back following the same schema used when data are transferred from the WN to the SE: first the LFC catalog is queried and in case of errors the file is retrieved directly using low level data management tools like GridFTP .

Despite a job could be considered as "Done" by the grid system perspective, it does not mean it is valid for BaBar, since problems could be generated from different sources, like for example simulation errors or transfer problems after results completion. In order to verify the quality of the ROOT file produced, the following procedure is applied:

- the existence of the file is checked within the job directory on the UI
- the standard BaBar check is performed to investigate the number of events effectively contained into the file. This number must be the same of that specified in the configuration file for the job, created when run was built and stored within the job's directory along with all needed files. The classical approach for validating files was split in two steps, the first and more computing consuming is performed directly by the WN, while the second one, a shallow and

faster inspection of the file, is executed by the monitor at this stage taking also into account result generated by the first check

- any noticed error is considered as a failure and job is marked as "bad"

This process is critical since corrupted collections of data can bring to serious problems during the merging stage, resulting in data loss. Jobs that pass this step are marked as "good" and are considered ready for the next steps.

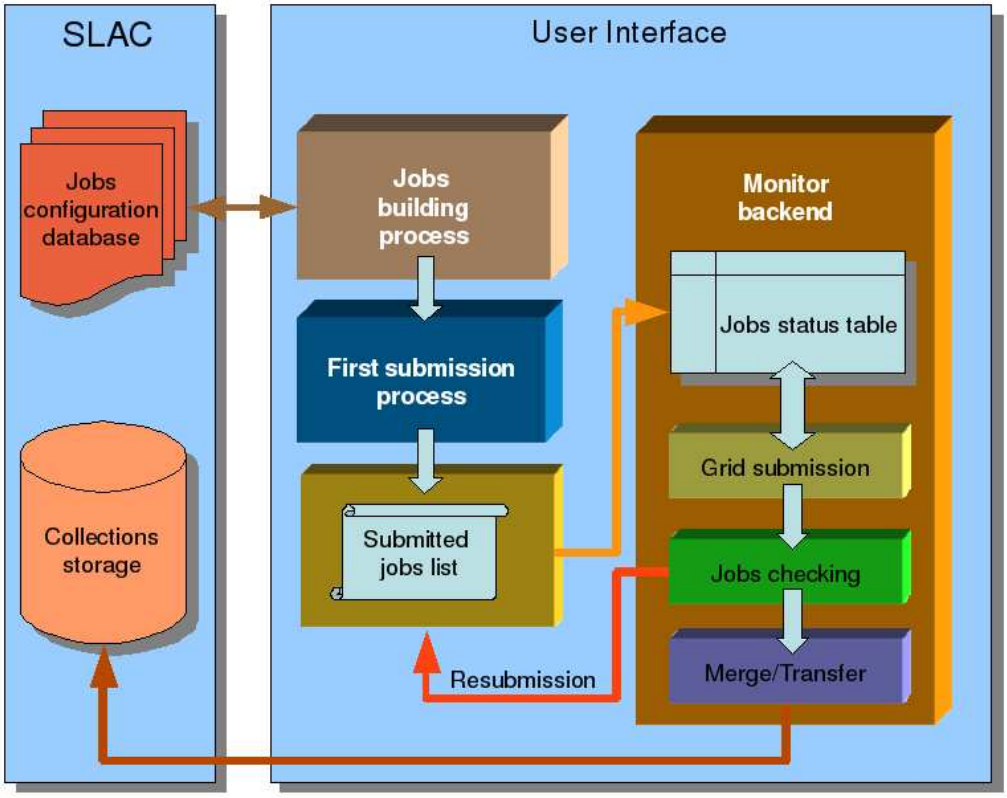


Figure 5.5: Backend logical view

Information about jobs status are periodically saved to a file to minimize loss of data in case of system crashes, which is also used at each new restart of the application to populate the table kept in memory with last available information about jobs activities. Another feature supported by the application is to preserve

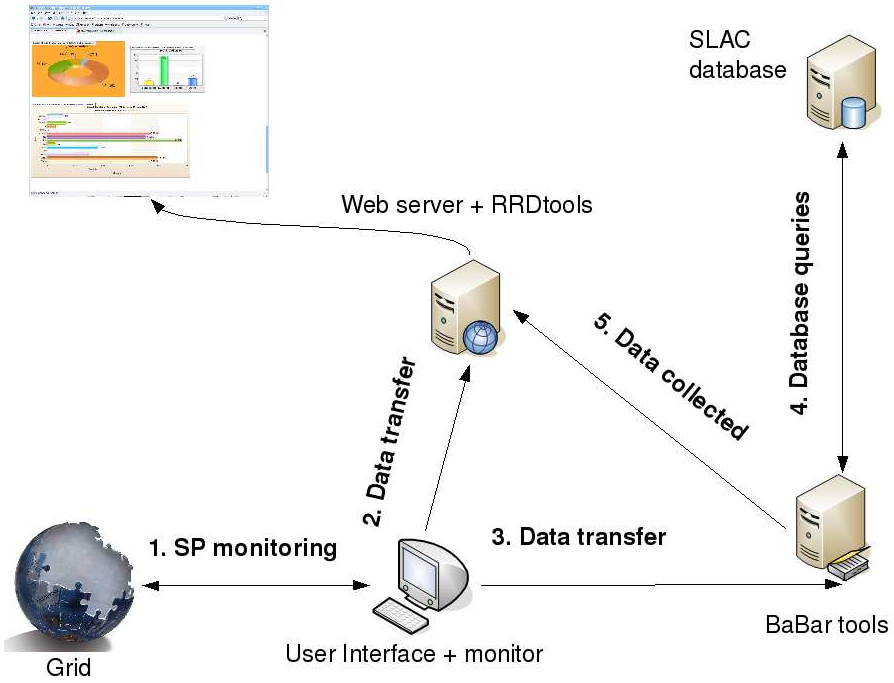


Figure 5.6: Monitor backend schema

all data collected on a persistent support that can be analyzed for statistical uses and that is sent, with a frequency of five minutes, to the web server in charge of publishing results, as shown in Figure 5.6.

5.2.5 Monitor frontend

Driven by the need to provide an effective feedback for data monitoring, a web interface was designed to allow to check the simulation production on the grid. The idea was to give the most interesting information grouped by topics, maintaining both an historical database for the general workflow and supplying details about the status of production for sites involved. In order to fulfill these requirements, a new set of tools was developed using a modular approach to handle the different kind of information through a parallel pipeline.

On the official web server of INFN was installed the set of scripts in charge

of preserving a history for the production using the Round Robin Database suite (RRDtool) [84], a toolset for high performance data logging that also offers a graphing system for time series data and that provides flexible solution for customize user application. This tool was chosen also for the ease of use and maintenance since it is capable of acquiring data from several kind of sources, automatically handling data storage as function of the time, allowing the user to choose the desired granularity. A script takes care to parse the file transferred by the monitor application from the UI to extract information about jobs status, accordingly formatted and inserted into the database and also plots a chart for each specific time interval. For the purposes of this work, data are preserved for at most one year while the range for granularity varies from monthly to hourly views. Moreover, the general production workload is taking in account and a plot for locations where jobs run is provided. The result is an overview of the production workflow in terms of general status for submitted jobs, which contributes to simplify the management and help to schedule the operations to perform. Further features allow to aggregate data for each single site in terms of jobs activities, providing information about the distribution of jobs over the grid nodes through a dedicated chart, which supplies details about workload within single sites (FIgure 5.7). On a dedicated machine, where BaBar software release was previously installed, another application queries the main database at SLAC to extract other useful information from the bookkeeping, like the amount of simulated events produced by INFN -Grid each single day. These data are also aggregated into a plot that shows the weekly trend, besides the total amount of events produced by each site involved in BaBar is showed, offering a comparison between results achieved by different institutions. The queries to the main database are performed asking for all the completed collections exported to SLAC from the beginning of the BaBar production. All charts shown in the web page are refreshed every five minutes, except for the ones generated by the queries to the SP bookkeeping, while another process controls when new updates for jobs status come into play to keep the monitor frontend in sync with the backend counterpart.

Figure 5.7: Web interface for monitor frontend

5.2.6 Monitor utilities

To coordinate the activities of all components described above, a set of scripts have been developed to run as Unix daemons on the User Interface. The main functionalities concern the following features:

- automatic submission of already built jobs

- check of the monitor backend activity
- automatic stop of the backend in case the process hangs and restart of the service

The main purpose of these utilities is to keep the production process always up and running in order to avoid system downtime. Indeed, to complete a full cycle, for each single run must be taken into account the delay introduced by each step, like building, submission, queuing/running, merging and transfer, plus the time required by the automatic import tool at SLAC to mark last imported collections as valid and update the bookkeeping, which usually takes 12 hours as safety constrain against data loss. Therefore is mandatory to react promptly, for example when new runs are ready to be submitted or when new merged collections are ready for export. Moreover, a periodic check is performed to ensure the monitor backend is active and to prevent unexpected blocks of the production system, that occasionally can occur when the UI is overloaded, restarting the application if necessary. In this case a special interrupt signal is sent to allow the monitor backend to conveniently save all data in memory to persistent structures before exiting and start over again.

5.3 Results

In this section is reported a discussion about results achieved at the end of this study. Despite the BaBar experiment has already reached the end, data collected need to be analysed therefore simulation will be performed during next months. The system implemented is still ongoing and collaborates within the SP community to produce a relevant part of the daily production rate, providing the expected level of support to the experiment.

5.3.1 Productions comparison

After more than one year and half of intensive production, we can say the new system stepped into the production pipeline quite smoothly, providing even better results than expected. A comparison among the last three production cycles of the

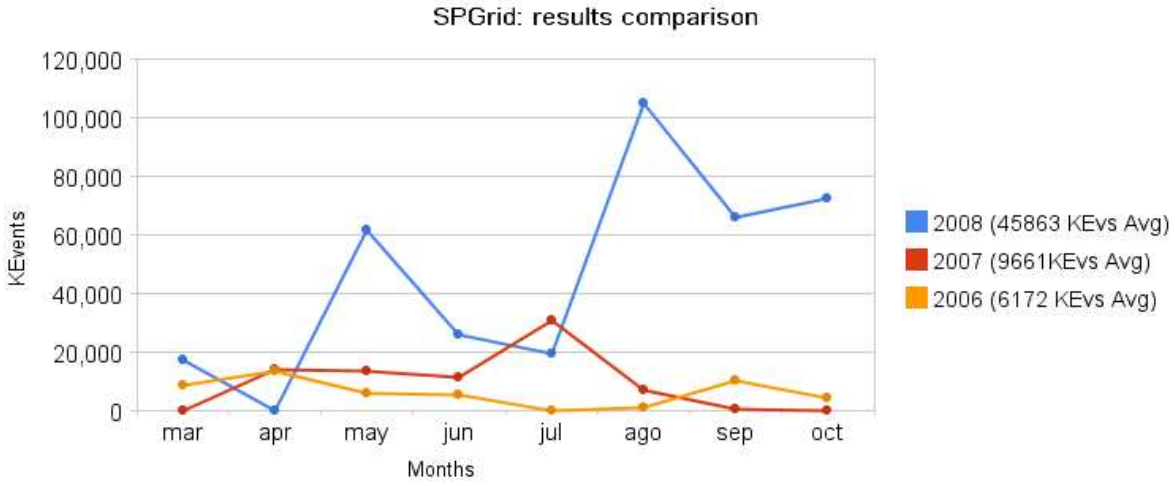


Figure 5.8: SPGrid: a three years comparison

experiment is shown is Figure 5.8. The SP activity in BaBar lasts about one year before a new cycle come into play, besides, two production cycles could in general be overlapped for few weeks before all sites move to the new one. Therefore, the period taken into account represents a significant interval since during the months selected just one cycle at a time was running, allowing a better understanding of the information obtained. The improvement gained during the last cycle is evident, in terms of total amount of simulated events produced. Of course this result depends directly on the increment of computing resources, which availability has grown significantly in the last period, but it also depends on the stability and reliability achieved by the SPGrid system, which allowed to sustained a high and continuous rate for the whole production pipeline.

As explained above, a high reactive system is fundamental to rapidly respond to every kind of event can occur within the pipeline, along with a good fault tolerance to extend the uptime as long as possible, to achieve the highest production rate over the unit of time, which in general is considered to be a working day. Improvements to the code and a better organization for sites belonging to the SP pool, in terms of quality of services, allowed to achieve better performances resulting in a more efficient and stable system, as reported in Figure 5.9. In 2008 it was able to stay up

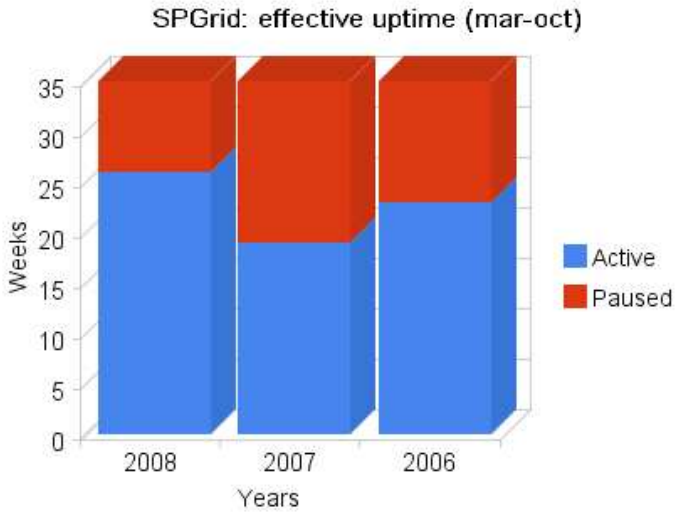


Figure 5.9: SPGrid: uptime improvements

for the 75% of the time, against the 55% and the 65% of the second and first year respectively. In some occasions the SP process stopped at SLAC for different kinds of problems or just because a quick reorganization was required, for example to keep the bookkeeping neat or to reprocessing some corrupted conditions data, therefore the evident inactivity shown by the plot has to be considered conform to the BaBar SP policy.

In order to evaluate the behavior of nodes where SP jobs run, an efficiency index has been calculated for each site as the ratio between the total amount of submitted jobs and the ones successfully completed daily. In practice, the monitor backend keeps trace for each site of the productivity efficiency, calculating this index and considering all jobs managed within the 24 hours. This index gives information about the reliability of a specific site, helping to decide where it could be more convenient to perform further submissions and outlining possibly failures at specific locations.

The most important contributes to SPGrid are given by sites like CNAF and Pisa, thanks to the considerable amount of computing resources available, although other sites with a large quantity of job slots, like for example Catania, are often taken into account. Figure 5.10 shows resources allocated for the BaBar VO within

the INFNGrid system. In spite of so many sites offer support for the BaBar's VO, it doesn't mean that all resources listed are freely accessible by SP jobs, indeed, many sites adopt private policies to preserve a certain amount of resources for favoring particular activities, depending on the experiments they are involved in. Therefore, a such list of resources is just indicative: the production manager must take care to add or remove a specific site in the pool of nodes used for production, depending on circumstances that can occur in a dynamic environment like the grid.

Sites like CNAF and Pisa allocate a certain number of slots for BaBar thanks to special agreements involving institutions where these farms are installed. In particular, at CNAF no limit was set in terms of number of slots usable, but the local fairshare policy allows to utilize just a specific amount of computing power per month for each VO. This value is directly proportioned to the investment made by each group for supporting the center. At Pisa, instead, just the 10% of computing slots is available for BaBar, amount shared with other experiments, assuring, however, the required level of resources.

General | Job | Charts | XML

VO Name ▼	Q#	MinSlotFree	MaxSlotFree	RunJob	WaitJob	Storage Available	Storage Total	Storage Load
DTEAM	0	-	-	-	-	-	-	-
INFNGRID	0	-	-	-	-	-	-	-
LHCB	0	-	-	-	-	-	-	-
OPS	0	-	-	-	-	-	-	-
alice	30	3170	3270	4	3	36.3 TB	94.4 TB	62%
argo	22	3091	3091	-	-	35.6 TB	91.4 TB	61%
astro	6	11	11	-	-	-	-	-
atlas	40	3473	3506	4	22	21.7 PB	25.8 PB	16%
auger	7	266	266	-	-	-	-	-
babar	25	3113	3113	-	-	36.4 TB	94.6 TB	62%
bio	19	2732	2732	0	82	35.9 TB	92.1 TB	61%
biomed	21	2950	2950	30	404	38 TB	100.1 TB	62%
cdf	23	3125	3125	-	-	36.5 TB	94.1 TB	61%
cms	38	3401	3469	189	2	120.3 TB	297.6 TB	60%
cometa	1	1	1	-	-	-	-	-
compass	1	0	0	-	-	24.8 TB	70 TB	65%
compassit	11	2620	2620	-	-	32.6 TB	90.1 TB	64%
compchem	28	2754	2754	-	-	35.9 TB	92.1 TB	61%
cresco	1	1	1	-	-	-	-	-
cybersar	1	1	1	-	-	-	-	-
cyclops	13	2543	2543	-	-	32.3 TB	87.2 TB	63%

Figure 5.10: Resources allocated for the BaBar VO

5.3.2 Comparison with other sites

INFNGrid was the first site where the grid approach for SP was implemented, and despite other few attempts have been tried by other institutes, no other grid imple-

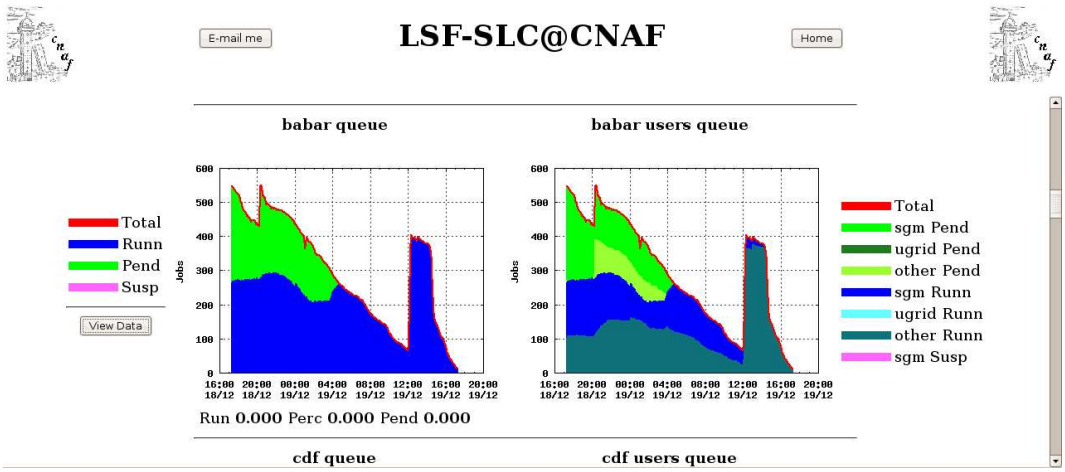


Figure 5.11: CNAF resources monitor for BaBar

mentation succeed in reach the same level of performances. As depicted in Figure 5.12, SPGrid stepped into the effective production workflow about 3 years ago, increasing the total amount of simulated events at every new cycle. For the first years its contribution was quite marginal, due to the overall instability of the system that relegated the INFNGrid site at a low rank, if compared to other sites much more competitive in terms of number of events produced. After passing the test phase, rate increased until reaching the standard level required to be qualified as a reliable site.

Considering the last production cycle, which is currently ongoing, results achieved are pretty interesting if compared wide world to ones obtained by other sites involved in BaBar SP, the most of which still adopt the traditional system of production. Figure 5.13 presents the total amount produced for SP in the last year, taking into account contributes for each site belonging to the collaboration. As reported by the chart, INFNGrid is one of the most important source of simulated events.

Started as an experimental project to evaluate the possibility to run the BaBar SP on grid, at the beginning it was introduced to get behind the production already performed in Italy and based on the traditional system. Actually the SPGrid approach covers about the 85% of the total amount of simulated events produced in Italy, a result that brought to consider this method as the official one for the

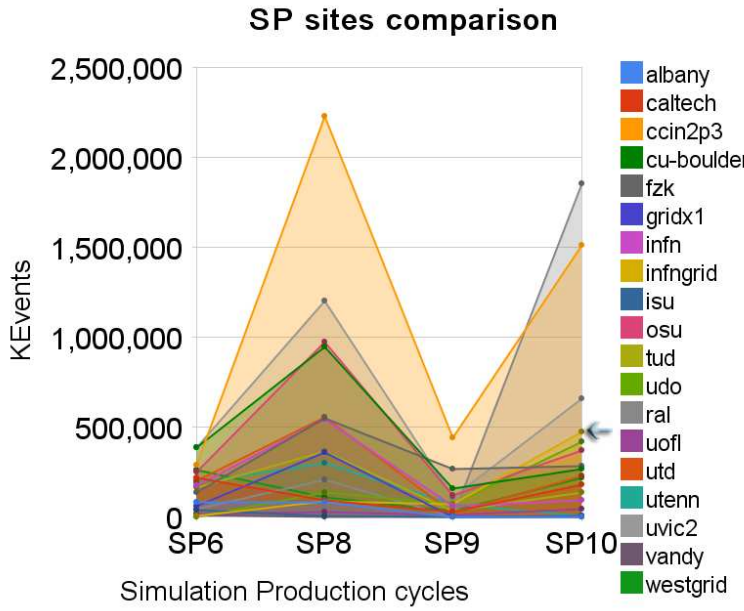


Figure 5.12: SP: cycles comparison by sytes

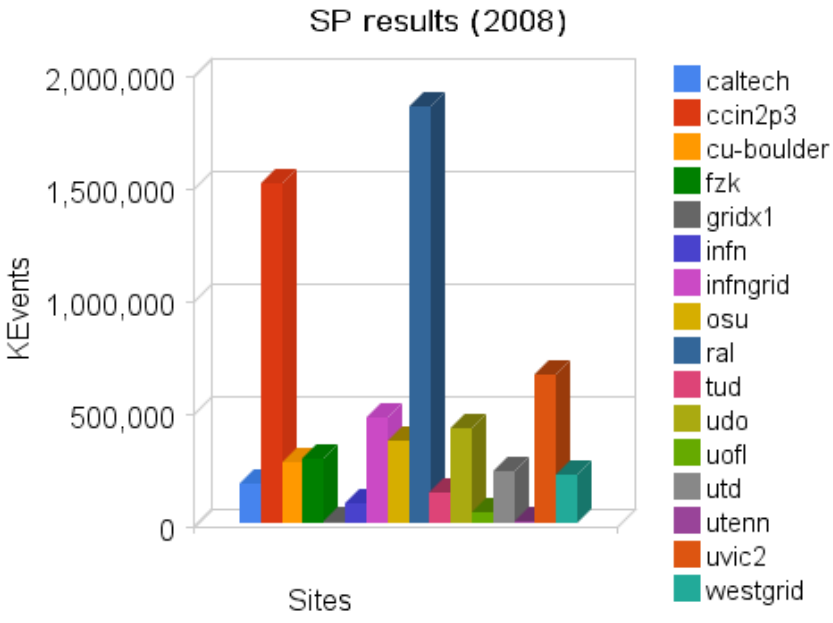


Figure 5.13: SPGrid: global production

next years, replacing also the small part that is still performed using the traditional method.

Chapter 6
Conclusions

Grid systems are becoming a strong reality within the distributed domain, although the evolution of this branch of computer science is still quite young. Nevertheless, important steps forward have been successfully done to define common standards and reliable solutions for building a solid platform for a variety of different applications. In a such promising scenario is also interesting to think how to apply these mechanisms retroactively to those projects that are still ongoing, but which frameworks were designed before the grid concept came into play. Despite the constrains imposed by their architectures, for some specific topics is feasible to adopt the grid way. A typical example is given by the batch processing task that in the grid paradigm finds a natural evolution.

On the basis of these considerations, in this thesis we introduced as case study the BaBar experiment, a well known project developed at the Stanford Linear Accelerator Center in the field of High Energies Physics. BaBar's framework was designed in the early '90s therefore was an ideal candidate for the purpose of this work, which aim was to successfully apply the grid paradigm to such a traditional model, gaining at least the same level of performances in terms of simulated events produced weekly. Different aspects of BaBar's model were taken into account, like data management and information bookkeeping, outlining those characteristics considered as consonant to the grid.

After an initial period which required a preliminary study about the feasibility of the SPGrid process, the whole system has been implemented at CNAF, establishing the main center for producing simulated events on the grid in Italy. In spite at the

beginning the new approach has not been considered reliable enough to sustain a complete production cycle [70], the scenario improved considerably when the integration of the different components of the SP model reached its maturity, thanks both to software and infrastructure improvements.

At the present time, results achieved state the SPGrid approach can really coexist with the traditional one, providing on average the same level of production rate reached by the most of sites involved in the collaboration, but requiring a less intensive resources management from the point of view of the final user. This assumption is especially true in terms of human resources, since just a single person can control all the whole production workflow on several farms when the traditional approach requires at least one person for each single site. Moreover, costs can be reduced as obvious consequence of the intrinsic nature of the grid, that allows to access resources of different domains transparently. Therefore the SPGrid manager can focus just to those grid sites stated as appropriate for the production, without worry to manage any external resources except for the ones specifically dedicated to the experiment.

6.1 Open problems

Approaching a new vision is rarely painless for a project with an already well established background. Although target was achieved in terms of expectations, some aspects could not be further improved due to both BaBar's framework limitations and grid policies. In particular, a more fine distribution of input data needed for the simulation process would have been required. The management of non grid nodes to host this kind of data has as direct consequence a serious scalability issue that limited the SPGrid pool nodes to grow in size. At the time the SPGrid approach was implemented, the Xrootd service was not included into any official middleware release, while, from the SPGrid's prospective, this was the only feasible way to access input data in such a system like the grid. The best solution would have been represented by the deployment of the Xrootd service toward the Storage Element units, allowing the widest possible distribution and, at the same time, to

fully supporting this service at each grid farm.

Lack of such a kind of flexibility imposed rigid constrains to the interoperability with other grid systems ready for supporting the BaBar VO. The impossibility for those systems to provide a reliable and continuous support to maintain dedicate external servers for input data, along with the absence of a global coordination, limited the cooperation among grids of other countries, although some tests were also performed toward that direction, but never for real production purposes.

On the other hand, this work was driven by the idea to connect these two different environments respecting the basic characteristics of both worlds, therefore avoiding any kind of spurious solutions that would have led to a hard maintainability of the whole system. From this point of view, and considering the natural constrains imposed since the beginning, the project revealed the required level of flexibility to accomplish the task.

6.2 Future trends

The SPGrid approach has been started to supply the required amount of computing resources, otherwise difficult to collect for a single Italian site at the time the project began, to compete with other international research centers for the production of simulated events. Despite the fact that new experiments nowadays are able to add more innovative technologies within their workflow, the BaBar SPGrid experience demonstrated it worth to invest in this new approach although the early scepticism about the flexibility it could offer when applied to an already established model. Moreover, within the SPGrid design some new kind of services were adopted, like the Xrootd suite, which was intensive used and promoted, offering to the grid community an example that has been taken into account even by other and more recent experiments.

Appendix

Main job's configuration files

local-grid-setup

```
1  ##########################################################################
2  #                                                                        #
3  #     Xrootd server designation on the basis of job's location           #
4  #                                                                        #
5  ##########################################################################
6
7
8  #!/bin/bash
9
10 s=$VO_BABAR_DEFAULT_SE
11
12
13 if [ ! -z `echo $s|grep "\.fe\."` ]; then
14   export XROOTD_HOST=babarbecue.fe.infn.it
15 else
16   if [ ! -z `echo $s|grep "\.na\."` ] || [ ! -z `echo $s|grep "\.ba\."`
       ] ; then
17     export XROOTD_HOST=babarbecue.fe.infn.it
18   else
19     if [ ! -z `echo $s|grep "\.pd\."` ]; then
20        export XROOTD_HOST=bbr-serv08.cr.cnaf.infn.it
21     else
22       if [ ! -z `echo $s|grep "\.cnaf\."` ]  ; then
23         export DATA_PATH=/storage/gpfs_01/babar/SP_cdb_cfg_bkg/data/
24       else
25           if [ ! -z `echo $s|grep "\.pi\."` ]  ; then
26               export XROOTD_HOST=babarxrd.pi.infn.it
27           else
28               export XROOTD_HOST=bbr-serv08.cr.cnaf.infn.it
29           fi
30       fi
31     fi
32   fi
33 fi
34
35
36 ### Default Storage Element and LFC path definition ###
37
38 export OUTPUT_SE=storm02.cr.cnaf.infn.it
39 export OUTPUT_SE_PATH=/storage/gpfs_01/babar
```

```
40 export LFC_LFN_PATH=/grid/babar/SPGrid/infngrid
41
42
43
44 #SP10
45 export CDB_ROO_BOOT="kanga::/cond24boot/full/cdb_boot.root"
46
47 export CFG_DEFAULT_IMPL=ROOT
48 unset OO_FD_BOOT
49
50
51 ### Dynamic generation of the KanAccess.cfg file ###
52
53 mkdir -p $PWD/kanga/config
54
55
56 if [ -z `echo $DATA_PATH` ] ; then
57   echo "rootenv Root.XTNetFileAllowWanConnect 1">$PWD/kanga/config/
       KanAccess.cfg
58   echo "rootenv Root.XTNetFileAllowWanRedirect 1">>$PWD/kanga/config/
       KanAccess.cfg
59   echo "rootenv XNet.RedirDomainAllowRE *.infn.it">>$PWD/kanga/config/
       KanAccess.cfg
60   echo "rootenv XNet.ConnectDomainAllowRE *.infn.it">>$PWD/kanga/config
       /KanAccess.cfg
61   echo "rootenv XNet.RequestTimeout 1200">>$PWD/kanga/config/KanAccess.
       cfg
62 fi
63
64 echo "">>$PWD/kanga/config/KanAccess.cfg
65
66 echo "# CfgDB">>$PWD/kanga/config/KanAccess.cfg
67
68 if [ -z `echo $DATA_PATH` ] ; then
69   echo "read /store/cfg/* xrootd $XROOTD_HOST:1094/">>$PWD/kanga/config
       /KanAccess.cfg
70 else
71   echo "read /store/cfg/* file /storage/gpfs_01/babar/SP_cdb_cfg_bkg/
       data/">>$PWD/kanga/config/KanAccess.cfg
72 fi
73
74 echo "write /store/cfg/* error">>$PWD/kanga/config/KanAccess.cfg
75
76 echo "# CDB">>$PWD/kanga/config/KanAccess.cfg
77
78 if [ -z `echo $DATA_PATH` ] ; then
79   echo "read /store/cdb/* xrootd $XROOTD_HOST:1094/">>$PWD/kanga/config
       /KanAccess.cfg
80 else
81   echo "read /store/cdb/* file /storage/gpfs_01/babar/SP_cdb_cfg_bkg/
       data/">>$PWD/kanga/config/KanAccess.cfg
82 fi
83
84 echo "write /store/cdb/* error">>$PWD/kanga/config/KanAccess.cfg
85
86
87 if [ -z `echo $DATA_PATH` ] ; then
88   echo "read /store/* xrootd $XROOTD_HOST:1094/">>$PWD/kanga/config/
       KanAccess.cfg
89 else
```

```
90    echo "read /store/* file /storage/gpfs_01/babar/SP_cdb_cfg_bkg/data
          /">>$PWD/kanga/config/KanAccess.cfg
91  fi
92
93
94  ### Writing of config files for defining the last version of conditions
        and configurations data to use ###
95
96  #SP10 CFG
97  export CFGDB="/store/cfg/2008/05/CfgDB-20080514T224248.root"
98
99  #SP10 CDB
100 export CDB="/store/cdb/cond24boot/full/2008/09/20080915T142817/CDB
        -20080915T142817-cdb_boot.root"
101
102
103 mkdir -p $PWD/kanga/config/cfgdb/
104 mkdir -p $PWD/kanga/config/cdb/
105
106 echo "write /cfg/CfgDB\.root error">>$PWD/kanga/config/cfgdb/
        CfgDBNameRules.cfg
107 echo "-include kanga/config/cfgdb/CfgDBNameRules-latest.cfg">>$PWD/
        kanga/config/cfgdb/CfgDBNameRules.cfg
108
109 echo "read /cfg/CfgDB\.root $CFGDB">$PWD/kanga/config/cfgdb/
        CfgDBNameRules-latest.cfg
110
111 echo "write /cdb/cdb_boot\.root error">$PWD/kanga/config/cdb/
        CdbNameRules.cfg
112 echo "-include kanga/config/cdb/CdbNameRules-latest.cfg">>$PWD/kanga/
        config/cdb/CdbNameRules.cfg
113
114
115 #SP10
116 echo "read /cond24boot/full/cdb_boot\.root  $CDB">$PWD/kanga/config/cdb
        /CdbNameRules-latest.cfg
117
118
119 ### Setting the current dir on WN where job runs as the root dir for
        the simulation ###
120
121 export BFROOT=$PWD
```

Moose.bash

```
1  ##########################################################################
2  #                                                                        #
3  #                  Wrapper for SP jobs over the Grid                     #
4  #                                                                        #
5  ##########################################################################
6  #!/bin/bash
7
8  if [ -a config.sh ] ; then
9          . config.sh
10         echo "Sourcing config.sh from current directory $PWD" >>
               $TYPE$JOBNUM.batchout
```

```
11 fi
12
13 HOST=`hostname | cut -d. -f1`
14
15 #set status file
16 STATUSFILE=`pwd`"/status.txt"
17
18
19 ### Redefine settings for the Grid environment ###
20
21 export SITE=$HOSTNAME
22 export REL=Moose-$JOBSRT
23 export SW_DIR=$VO_BABAR_SW_DIR/SP
24 export SP_ROOT=$SW_DIR/$REL
25 export PARENT=$SP_ROOT/src
26 export LD_LIBRARY_PATH=$SP_ROOT/shlib
27 export PATH=${PATH}:$SP_ROOT/bin
28 export OO_CONNECT_RETRIES=4
29 export OO_RPC_TIMEOUT=50
30 export BDB_STANDALONE_MODE="yes"
31 export OO_CACHE_MAX=2500
32 export OO_CACHE_INIT=1000
33 export OO_FD_LIMIT=20
34 export MooseHBookFile=framework.hbook
35 export SPCHECKLOG=spcheck.log
36 export LCG_CR_TIMEOUT=900
37 export ROOTSYS=$SP_ROOT/root
38
39
40 #tell status file that script has started
41 echo "Job.bash - $PROCSPEC - Starting job on $HOST - "`date` >>
      $STATUSFILE
42
43 #run the site commands
44 #. $SITEDIR/local-$TYPE-batch >> $TYPE$JOBNUM.batchout
45
46 /bin/ln -f -s $PARENT PARENT
47 apppath=$SP_ROOT/bin/
48
49
50 #run local grid setup
51 . local-grid-setup
52
53
54 #determine platform
55 SPPLATFORM=`uname`
56
57 # write out information to log file
58 echo Run $JOBNUM $TYPE version $JOBSRT on $HOST >> $TYPE$JOBNUM.log
59
60 # check log file was successfully created (avoids core dumps later on)
61 if [ ! -s  $TYPE$JOBNUM.log ] ; then
62   errmsg "ERROR: Cannot write to $TYPE$JOBNUM.log. Has the disk filled
        up?"
63   exit 2
64 fi
65
66 if [ "$USER" = "bbrprod" ] ; then
67         echo USER: bbrprod >> $TYPE$JOBNUM.log
68         echo JOB SETUP BY: $userlog >> $TYPE$JOBNUM.log
```

```
69  else
70          echo USER: `whoami` >> $TYPE$JOBNUM.log
71  fi
72
73  echo DATE: `date` >> $TYPE$JOBNUM.log
74  echo RELEASE: $RELEASE >> $TYPE$JOBNUM.log
75  echo PARENT:  $PARENT >> $TYPE$JOBNUM.log
76  echo BOOTFILE: $OO_FD_BOOT >> $TYPE$JOBNUM.log
77  echo ================ >> $TYPE$JOBNUM.log
78  printenv >> $TYPE$JOBNUM.log
79  echo ================ >> $TYPE$JOBNUM.log
80  ulimit -a >> $TYPE$JOBNUM.log
81  echo ================ >> $TYPE$JOBNUM.log
82  echo PRODDECAYFILES: $SPDFTAG >> $TYPE$JOBNUM.log
83  echo ================ >> $TYPE$JOBNUM.log
84  if [ "$SPPLATFORM" = "SunOS" ] ; then
85          psrinfo -v >> $TYPE$JOBNUM.log
86  fi
87  if [ "$SPPLATFORM" = "Linux" ] ; then
88          cat /proc/cpuinfo >> $TYPE$JOBNUM.log
89  fi
90  echo ================ >> $TYPE$JOBNUM.log
91  #cat $OO_FD_BOOT >> $TYPE$JOBNUM.log
92  echo ================ >> $TYPE$JOBNUM.log
93
94  if [ -z "$PWD" ] ; then PWD=`/bin/pwd` ; fi
95  export PWD
96
97  if [ -r $tclfile ] ; then
98    echo "Using specified tcl file $tclfile" >> $TYPE$JOBNUM.log
99  else
100   errmsg "ERROR: Cannot find $tclfile"
101   exit 2
102 fi
103
104 # write things out to the logs
105 echo '======== Application ========' >> $TYPE$JOBNUM.log
106 ls -lL $apppath >> $TYPE$JOBNUM.log
107 echo '==== workdir listing ========' >> $TYPE$JOBNUM.log
108 ls -l >> $TYPE$JOBNUM.log
109 echo '==== symlink listing ========' >> $TYPE$JOBNUM.log
110 find . -type l -exec ls -ldL {} \; >> $TYPE$JOBNUM.log
111 if [ -f user1.dec ] ; then
112   echo '======== user.dec ==========' >> $TYPE$JOBNUM.log
113   cat user*.dec >> $TYPE$JOBNUM.log
114 fi
115 echo '==========================================' >> $TYPE$JOBNUM.log
116
117
118 #tell status file that application is started
119 echo "Job.bash - $PROCSPEC - starting $apppath - "`date` >> $STATUSFILE
120
121 # run the job and time it
122         $TIMEBIN $apppath/$appname $tclfile >> $TYPE$JOBNUM.log 2>&1
123
124 jobstat=$?
125 echo $appname finished at `date` >> $TYPE$JOBNUM.log
126
127 #tell status file that application is finished
```

```
128 echo "Job.bash - $PROCSPEC - $appname finished, signal $jobstat - "`
    date` >> $STATUSFILE
129
130
131
132 #check root output
133
134 $apppath/KanCopyUtil -r ${JOBNUM}.moose  &>$SPCHECKLOG
135
136
137 # Source local-$TYPE-wrapup
138 #. $PRODTOOLS/site/$BFSITE/local-$TYPE-wrapup $JOBNUM $JOBSRT >>
    $TYPE$JOBNUM.batchout
139
140 echo "gzipping framework.root" 2>&1 >> $TYPE$JOBNUM.batchout
141 gzip framework.root
142
143 gtar -czf run${JOBNUM}${TARGET}.tar.gz *.root 2>&1 >> $TYPE$JOBNUM.
    batchout
144
145
146 s=$HOSTNAME
147 if [ ! -z `echo $s|grep .pi.` ]; then
148
149 VO_BABAR_DEFAULT_SE=lcg-se01.cr.cnaf.infn.it
150 export VO_BABAR_DEFAULT_SE
151
152 fi
153
154 ### Start output transfer
155
156 echo "starting lcg-cr" 2>&1 >> $TYPE$JOBNUM.batchout
157
158 #the lcg-cr can fails for lfc server down or for ClosestSE server down
159 echo "lcg-cr -v -t $LCG_CR_TIMEOUT --vo babar -l lfn:/grid/babar/SPGrid
    /infngrid/run${JOBNUM}${TARGET}.tar.gz file:$PWD/run${JOBNUM}${
    TARGET}.tar.gz"
160 lcg-cr -v -t $LCG_CR_TIMEOUT --vo babar -l lfn:$LFC_LFN_PATH/run${
    JOBNUM}${TARGET}.tar.gz file:$PWD/run${JOBNUM}${TARGET}.tar.gz
    2>&1 >> $TYPE$JOBNUM.batchout
161
162 #Check if the ClosestSE server is up
163 if [ $? -ne 0 ];then
164     echo "lcg-cr failed for Closest SE, try with lcg-cr on ${OUTPUT_SE
        }" 2>&1 >> $TYPE$JOBNUM.batchout
165
166     lcg-cr -v -t $LCG_CR_TIMEOUT --vo babar -l lfn:$LFC_LFN_PATH/run${
        JOBNUM}${TARGET}.tar.gz -d ${OUTPUT_SE} file:$PWD/run${JOBNUM}$
        {TARGET}.tar.gz  2>&1 >> $TYPE$JOBNUM.batchout
167
168 #Check if the OUTPUT_SE server is up
169     if [ $? -ne 0 ];then
170         echo "lcg-cr fails for default OUTPUT_SE ${OUTPUT_SE}, file not
            transferred" 2>&1 >> $TYPE$JOBNUM.batchout
171
172 #Use gridftp as backup transfer method on the $OUTPUT_SE (in local-grid
    -setup)
173         echo "starting globus url copy" 2>&1 >> $TYPE$JOBNUM.batchout
174         echo "globus-url-copy -vb file://$PWD/run${JOBNUM}${TARGET}.tar
            .gz gsiftp://$OUTPUT_SE/$OUTPUT_SE_PATH/run${JOBNUM}${
```

```
                TARGET}.tar.gz" 2>&1 >> $TYPE$JOBNUM.batchout
175
176             globus-url-copy -vb file:///`pwd`/run${JOBNUM}${TARGET}.tar.gz
                    gsiftp://$OUTPUT_SE/$OUTPUT_SE_PATH/run${JOBNUM}${TARGET}.
                    tar.gz 2>&1 >> $TYPE$JOBNUM.batchout
177
178             if [ $? -ne 0 ]
179                 then
180                 echo "globus-url-copy on $OUTPUT_SE failed" 2>&1 >>
                        $TYPE$JOBNUM.batchout
181             else
182                 echo "globus-url-copy on $OUTPUT_SE succeeded" 2>&1 >>
                        $TYPE$JOBNUM.batchout
183             fi
184         else
185             echo lcg-cr succeeded on $OUTPUT_SE 2>&1 >> $TYPE$JOBNUM.
                    batchout
186         fi
187
188     else
189         echo lcg-cr succeeded for Closest SE >> $TYPE$JOBNUM.batchout
190     fi
```

Glossary

A

AFS Andrew File System;
ALICE A Large Ion Collider Experiment;
AMS Advanced Multi-threaded Server;
ATLAS A Toroidal LHC ApparatuS;

B

BaBar B and B-bar experiment;
BDII Berkeley Database Information Index;
BOGUS Babar Object-oriented Geant-4-based Unified Simulation;

C

CA Certification Authority;
CE Computing Element;
CERN European Laboratory for Particle Physics;
CM Computer Model;
CMS Compact Muon Solenoid;
CNAF INFNs National Center for Telematics and Informatics;
CP Charge Parity;
CVS Concurrent Version System;

E

EDG European DataGrid;

F

FTP File Transfer Protocol;

G

GeV Giga electron Volt;
GIIS Grid Index Information Server;
GLUE Grid Laboratory for a Uniform Environment;
GRAM Globus Resource Allocation Manager;
GRIS Grid Resource Information Service;
GSI Grid Security Infrastructure;
GUI Graphical User Interface;
GUID Grid Unique ID;

H

HEP High Energy Physics;

I

ID Identifier;
IN2P3 Institut Nacional de Physique Nuclèaire et de Physique des Particules;
INFN Istituto Nazionale di Fisica Nucleare;
IS Information Service;

J

JA Job Adapter;
JC Job Controller;
JCS Job Control Service;
JDL Job Description Language;

L

LB Logging and Bookkeeping Service;
LCG LHC Computing Grid;
LDAP Lightweight Directory Access Protocol;
LFN Logical File Name;
LHC Large Hadron Collider;
LHCb Large Hadron Collider beauty experiment;
LM Log Monitor;
LSF Load Sharing Facility;

M

MDS Monitoring and Discovery Service;
MOOSE Monolithic Object Oriented Simulation Executable;

N

NFS Network File System ;
NS Network Server;
NTP Network Time Protocol;

P

PBS Portable Batch System;
PERL Pratical Extraction and Report Language;
PEP-II Positron Electron Project II;
PFN Physical File name;
PID Process IDentifier;

R

RA Registration Authority;
RAL Rutherford Appleton Laboratory;
RB Resource Broker;
RC Replica Catalog;
RLS Replica Location Service;
RM Replica Manager;
ROOT An Object-Oriented Data Analysis Framework;

S

SDK Software Development Kit;
SE Storage Element;
SimApp Simulation Application;
SLAC Stanford Linear Accelerator Center;
SP Simulation Production;

U

UI User Interface;

V

VO Virtual Organization;
VOMS Virtual Organization Management Team;

W

WM Workload Manager;
WMS Workload Management System;
WN Worker Node;

X

XROOTD eXtended Root Daemon;

List of Tables

List of Figures

Bibliography

[1] A.S. Tanenbaum and M. Steen. *Distributed Systems: Principles and Paradigms*. Pearson, 2007.

[2] N. Minar. Distributed Systems Topologies: Part 1. *Retrieved November*, 15:7, 2001.

[3] N. Minar. Distributed Systems Topologies: Part 2. *Retrieved January*, 15:8, 2002.

[4] I.J. Taylor. *From P2P to Web Services and Grids: Peers in a Client/Server World*. Springer, 2009.

[5] S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of Data Grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR)*, 38(1), 2006.

[6] The MONARC project. Website, 2000. `http://monarc.web.cern.ch/MONARC/`.

[7] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol extensions to FTP for the Grid. *Global Grid ForumGFD-RP*, 20, 2003.

[8] F. Dabek, J. Li, E. Sit, J. Robertson, M.F. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput.

[9] J. Joseph, M. Ernest, and C. Fellenstein. Evolution of grid computing architecture and grid adoption models. *IBM SYSTEMS JOURNAL*, 43(4):624, 2004.

[10] Open Grid Forum. Website, 2008. `www.ogf.org/`.

[11] EGEE Home Page. Website, 2006. http://egee1.eu-egee.org/.

[12] EGEE Home Page. Website, 2008. http://www.eu-egee.org/.

[13] WLCG. Website, 2008. http://lcg.web.cern.ch/LCG/.

[14] P. Lebrun. The Large Hadron Collider, A Megascience Project. In *Proceedings of the 38th INFN Eloisatron Project Workshop on Superconducting Materials for High Energy Colliders*, 1999.

[15] The BaBar Home Page. Website, 2008. http://www.slac.stanford.edu/BFROOT/.

[16] The Collider Detector at Fermilab. Website, 2008. http://www-cdf.fnal.gov/.

[17] Zeus Homepage. Website, 2007. http://www-zeus.desy.de/.

[18] Wisdom. Website, 2007. http://wiki.healthgrid.org/Wisdom:web_site.

[19] Gpsa web portal. Website, 2005. http://gpsa.ibcp.fr/.

[20] The MAGIC telescope. Website, 2008. http://wwwmagic.mppmu.mpg.de/.

[21] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems: Concepts and Design*. Addison-Wesley Reading, Mass, 2001.

[22] Virtual Data Toolkit. Website, 2008. http://vdt.cs.wisc.edu/.

[23] G. Avellino, S. Beco, B. Cantalupo, A. Maraschini, F. Pacini, M. Sottilaro, A. Terracina, D. Colling, F. Giacomini, E. Ronchieri, et al. The DataGrid Workload Management System: Challenges and Results. *Journal of Grid Computing*, 2(4):353–367, 2004.

[24] E. Laure, SM Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, et al. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

[25] S. Burke, S. Campana, A.D. Peris, F. Donno, P.M. Lorenzo, R. Santinelli, and A. Sciaba. gLite 3.1 User Guide, 2008. `https://edms.cern.ch/file/722398//gLite-3-UserGuide.pdf`.

[26] F. Pacini. Job Description Language HowTo, 2003. `http://www.infn.it/workload-grid/docs/`.

[27] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *10th IEEE International Symposium on High Performance Distributed Computing*, volume 184. San Francisco, 2001.

[28] A. Cooke, A.J.G. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, et al. R-GMA: An Information Integration System for Grid Monitoring. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 462–481, 2003.

[29] G. Aloisio, M. Cafaro, and I. Epicoco. Early experiences with the GridFTP protocol using the GRB-GSIFTP library. *Future Generation Computer Systems*, 18(8):1053–1059, 2002.

[30] Carnegie Mellon University. Overview of the Grid Security Infrastructure. Website, 2008. `http://www.globus.org/security/overview.html`.

[31] P. Fuhrmann. dCache, the commodity cache. In *proceedings of the Twelfth NASA Goddard and Twenty First IEEE Conference on Mass Storage Systems and Technologies, Washington DC*, 2004.

[32] J.P. Baud, B. Couturier, C. Curran, J.D. Durand, E. Knezo, S. Occhetti, and O. Barring. CASTOR status and evolution. *Arxiv preprint cs.OH/0305047*, 2003.

[33] RA Coyne, H. Hulen, R. Watson, I.B.M.F.S. Co, and TX Houston. The High Performance Storage System. In *Supercomputing'93. Proceedings*, pages 83–92, 1993.

[34] A. Shoshani, A. Sim, and J. Gu. Storage Resource Managers: Middleware Components for Grid Storage. In *Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems*.

[35] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Gianoli, F. Spataro, F. Bonnassieux, P. Broadfoot, G. Lowe, L. Cornwall, et al. Managing Dynamic User Communities in a Grid of Autonomous Resources. *Arxiv preprint cs/0306004*, 2003.

[36] I. Foster and C. Kesselman. The Globus project: a status report. In *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*, pages 4–18, 1998.

[37] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2002.

[38] R. Alfieri, R. Cecchini, V. Ciaschini, L. dellAgnello, Á. Frohner, K. Lőrentey, and F. Spataro. From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Generation Computer Systems*, 21(4):549–558, 2005.

[39] Web Documentation Voms. Website, 2008. `https://twiki.cnaf.infn.it/cgi-bin/twiki/view/VOMS/WebDocumentation`.

[40] Grid ecosystem - voms. Website, 2008. `http://www.globus.org/grid_software/security/voms.php`.

[41] R. Alfieri, R. Barbera, P. Belluomo, A. Cavalli, R. Cecchini, A. Chierici, V. Ciaschini, L. DellAgnello, F. Donno, E. Ferro, et al. The INFN-Grid Testbed. *Future Generation Computer Systems*, 21(2):249–258, 2005.

[42] Boutigny D. The BaBar experiments distributed computing model. *Computing in High Energy Physics and Nuclear Physics (CHEP 2001)*, 2001.

[43] J. Becla. The BABAR Database: Challenges, Trends and Projections. Technical report, SLAC-PUB-9179, Stanford Linear Accelerator Center, Menlo Park, CA (US), 2002.

[44] A. Adesanya, T. Azemoon, J. Becla, A. Hanushevsky, A. Hasan, W. Kroeger, A. Trunov, D. Wang, I. Gaponenko, S. Patton, et al. On the Verge of One Petabyte-the Story Behind the BaBar Database System. *Arxiv preprint cs.DB/0306020*, 2003.

[45] I. Gaponenko, D. Brown, D. Quarrie, E. Frank, and S. Gowdy. An Overview of the BaBar Conditions Database. In *Proceedings of the International Conference on Computing in High Energy Physics*.

[46] A. Hasan, A. Trunov. Managing the BaBar Object Oriented Database. In *CHEP Conference, Beijing, China, September 2001*.

[47] I.A. Gaponenko and DN Brown. CDB–Distributed Conditions Database of the BaBar Experiment. *CHEP proceedings, Interlaken, Switzerland, September*, 2004.

[48] R. Brun, F. Rademakers, et al. ROOT-An Object Oriented Data Analysis Framework. In *Proceedings AIHENP*, volume 96, pages 81–86, 1997.

[49] Peter Elmer. Cm2 - an introduction. Website, 2004. `http://www.slac.stanford.edu/BFROOT/www/Computing/Documentation/CM2/intro/`.

[50] Peter Elmer. Howto setup up a cm2 kanga eventstore. Website, 2004. `http://www.slac.stanford.edu/BFROOT/dist/releases/nightly/HOWTO/HOWTO-Setup-a-CM2-Kanga-Eventstore`.

[51] CERN. Website, 2008. `http://public.web.cern.ch/public/`.

[52] Rademakers F. Brun R. Architectural overview. Website, 1996. `http://root.cern.ch/root/Architecture.html`.

[53] M. Goto. Concept and application of cint c++ interpreter. *Interface Magazine*, 4, 1996.

[54] M. Goto. Cint-embedding tcl/tk graphics. *Interface magazine*, 4, 1997.

[55] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky. XROOTD-A Highly scalable architecture for data access. *WSEAS Transactions on Computers*, 1(4.3), 2005.

[56] P. Elmer. The new BaBar Computing and Analysis Model. Technical report, 2003.

[57] M. Steinke and P. Elmer. How to build an event storeThe new kanga event store for BaBar. In *Computing in High Energy Physics Conference (CHEP04), Interlaken, Switzerland*.

[58] D. Smith. BaBar Book Keeping Project–a Distributed Meta-Data Catalog of the BaBar Event Store. In *Computing in High Energy Physics Conference (CHEP04), Interlaken, Switzerland*.

[59] D. Smith. Bookkeeping, An Overview of some issues, May 2005. BaBar Collaboration Meeting. Technical report.

[60] Bookkeeping documentation. Website, 2007. `http://www.slac.stanford.edu/BFROOT/www/Computing/Distributed/Bookkeeping/Documentation/`.

[61] Srt: Software release tools. Website, 2004. `http://www.slac.stanford.edu/BFROOT/www/Computing/Environment/Tools/SRT/`.

[62] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, et al. Geant4a simulation toolkit. *Nuclear Inst. and Methods in Physics Research, A*, 506(3):250–303, 2003.

[63] Brian Berliner. Concurrent Versions System. Website, 2006. `http://www.nongnu.org/cvs/`.

[64] Carnegie Mellon University. Andrew Fyle System. Website, 2008. `http://www.openafs.org/`.

[65] Overview of the babar simulation. Website, 2002. `http://www.slac.stanford.edu/BFROOT/www/Computing/Offline/Simulation/web/simover/simover_main.html`.

[66] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce Dubois, M. Asai, G. Barrand, R. Capra, S. Chauvie, R. Chytracek, et al. Geant4 developments and applications. *Nuclear Science, IEEE Transactions on*, 53(1 Part 2):270–278, 2006.

[67] Douglas Smith. Prodtools man pages. Website, 2008. `http://www.slac.stanford.edu/BFROOT/www/Computing/Offline/Production/prodtools.pdf`.

[68] About Open Pbs. Website, 2008. `https://secure.altair.com/openpbs.html`.

[69] Load Sharing Facility. Website, 2008. `http://www.platform.com/Products/platform-lsf`.

[70] DA Smith, F. Blanc, C. Bozzi, A. Khan, S.L.A. Center, and CA Menlo Park. BaBar simulation Production-a millennium of work in under a year. *Nuclear Science, IEEE Transactions on*, 53(3 Part 3):1299–1303, 2006.

[71] Daniele Andreotti. Mocaespresso Home Page. Website, 2001. `http://hepunx.rl.ac.uk/BFROOT/www/Computing/Offline/Production/MocaEspresso/index.htm`.

[72] Setting up ams. Website, 2002. `http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/experts/setupAMS.shtml`.

[73] CAJ Brew, FF Wilson, G. Castelli, T. Adye, E. Luppi, and D. Andreotti. BABAR Experience of Large Scale Production on the Grid. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing table of contents*, page 151, 2006.

[74] A. Dorigo, P. Elmer, and F. Furano. XROOTD/TXNetFile: a highly scalable architecture for data access in the ROOT environment. In *Proceedings of the 4th WSEAS International Conference on Telecommunications and Informatics table of contents*. World Scientific and Engineering Academy and Society (WSEAS) Stevens Point, Wisconsin, USA, 2005.

[75] High Performance Storage System Portal. Website, 2008. `http://www.hpss-collaboration.org/hpss/index.jsp`.

[76] D. Publishing, D. Applications, E. Computing, M. Languages, and V.A. Titles. Introduction to Storage Area Networks by Jon Tate; Fabiano Lucchese; Richard Moore.

[77] A. Trunov, T. Azemoon, A. Hasan, and W. Kroger. Production Data Export And Archiving System For New Data Format Of The BaBar Experiment. *CHEP proceedings, Interlaken, Switzerland, September*, 2004.

[78] D. Boutigny, DH Smith, E. Antonioli, C. Bozzi, E. Luppi, P.V.G. Grosdidier, D. Colling, J. Martyniak, R. Walker, R. Barlow, et al. Use of the European Data Grid software in the framework of the BaBar distributed computing model. *Arxiv preprint physics/0306082*, 2003.

[79] T. Adye, D. Andreotti, R. Barlow, B. Bense, C. Bozzi, CAJ Brew, RD Cowles, E. Feltresi, A. Forti, G. Grosdidier, et al. Grid Applications for High Energy Physics Experiments. Grid, 2005.

[80] Infn Ferrara. Website, 2008. `http://www.fe.infn.it/`.

[81] C. Bozzi, T. Adye, D. Andreotti, E. Antonioli, R. Barlow, B. Bense, D. Boutigny, CAJ Brew, D. Colling, RD Cowles, et al. Using the grid for the BaBar experiment. *Nuclear Science, IEEE Transactions on*, 51(5 Part 1):2045–2049, 2004.

[82] D. Andreotti, E. Antonioli, C. Bozzi, E. Luppi, M. Melani, P. Veronesi, I.N. di Fisica Nucl, and I. Ferrara. Production of simulated events for the BaBar experiment by using LCG. In *Nuclear Science Symposium Conference Record, 2004 IEEE*, volume 3, 2004.

[83] CAJ Brew, FF Wilson, G. Castelli, E. Luppi, and D. Andreotti. BaBar SPGrid - Putting BaBar's Simulation Production On The Grid. In *Proceedings of the*

Second IEEE International Conference on e-Science and Grid Computing table of contents, 2006.

[84] Tobias Oetiker. About RRDtool. Website, 2008. `http://oss.oetiker.ch/rrdtool/`.