Università degli Studi di Perugia

Dottorato di Ricerca in Matematica ed Informatica per la elaborazione e la rappresentazione dell'informazione e della conoscenza

XXIV Ciclo

Design optimization of the Grid data analysis workflow in CMS

Candidato:
Hassen Riahi

Relatori:
Dott. Daniele Spiga
Dott. Attilio Santocchia

Coordinatore del corso di Dottorato:
Prof. Giulianella Coletti

A.A. 2010/11

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface, 25
BDII	Berkeley Database Information Index, 41
CAF	CERN Analysis Facility, 47
CE	Computing Element, 40
CMS	Compact Muon Solenoid, 43
CMSSW	CMS physics SoftWare, 59
CORBA	Common Object Request Broker Architecture, 25
CPU	Central Processing Unit, 11
CRAB	CMS Remote Analysis Builder, 58
DBS	Dataset Bookkeeping System, 53
DMWM	Data Management/Workload Management, 94
DNS	Domain Name System, 14
FTS	File Transfer System, 57
FWJR	FrameWorkJobReport, 98
GRAM	Grid Resource Allocation Management, 33
GRIS	Grid Resource Information Service, 30

GSI	Globus Security Infrastructure, 37
HPC	High Performance Computing, 29
IDL	Interface Definition Language, 12
IS	Information Service, 40
LAN	Local Area Network, 15
LFN	Logical File Name, 55
LHC	Large Hadron Collider, 39
LRM	Local Resource Manager, 35
LSF	Local Sharing Facility, 28
MC	Monte Carlo, 44
MPI	Message Passing Interface, 29
PBS	Portable Batch System, 28
PFN	Physical File Name, 55
PhEDEx	Physics Experiment Data Export, 55
RB	Ressource Broker, 30
RMI	Remote Procedure Invocation, 34
RPC	Remote procedure call, 18
RSI	Remote Service Invocation, 18
SE	Storage Element, 40
SSL	Secure Socket Layer, 37
SSO	Single Sign-On, 36
TCP	Transport Communication Protocol, 20

TMDB	Transfer Management DataBase, 57
UDP	User Datagram Protocol, 19
UI	User Interface, 41
WAN	Wide Area Network, 15
WLCG	Worldwide LHC Computing Grid, 39
WMS	Workload Management System, 41
WN	Worker Node, 40

Introduction

The main goal of the Large Hadron Collider (LHC) is to investigate the symmetry breaking mechanism at the base of the Standard Model of fundamental interactions, allowing to search the yet undiscovered Higgs boson in all the mass range allowed by the theory. It produces several Pbytes of data each year, distributing them over many computing sites around the world and enabling data access at those centers for analysis. Data should be accessed efficiently, namely with minimal delay and waste of computing resources.

Distributed systems, such as Grid computing, are becoming a standard choice for implementing scalable computing systems. Being able to provide a highly reliable and performance infrastructure allowing to exchange information and sharing resources, the distributed system was a natural choice to maintain and analyze the data accumulated by the LHC.

The Compact Muon Solenoid (CMS), one of the two LHC experiments aimed to a wide range of physics results, has identified the distributed sites as the primary location for physics analysis to support a wide community of users, potentially as many as 3000 users. This represents an unprecedented challenge of distributed computing systems scalability in term of resources and number of users.

The chain of the typical CMS analysis workflow execution starts once configured and submitted by the physicist and ends when the outputs become available in the physicist storage area. During the execution, the workflow interacts with CMS computing infrastructure and services. These workflows have shown several bottlenecks through the years introducing delays in the execution of the end-user analysis and consequently differing the availability of physics results to the collaboration and squandering often the distributed resources.

This thesis focuses on the study aiming to optimize the design of data analysis workflow of CMS, executing over LHC distributed computing infrastructure. The main delays are coming from how the analysis workflow is searching data to run over and how the final analysis results are made available to users. The latter has shown also a bad impact on the use of the distributed resources.

This thesis is organized as follows: in the first chapter the distributed systems is introduced, providing a detailed description of the distributed computing.

The second chapter describes the CMS computing infrastructure and frameworks allowing the distributed data analysis.

In the third chapter, CMS data analysis workflow is described, as well as, the parameters acting significantly on its performance.

The Automation tool design and implementation are discussed in chapter 4. The first use-case supported by this tool is the highest priority analysis workflow of CMS.

The fifth chapter describes the asynchronous stage-out system design and the firsts functional and latency tests of this tool.

The sixth chapter shows the performance of the Automation tool in production and the performance of the asynchronous stage-out tool in pre-production.

Chapter 1

Distributed systems

The World Wide Web is used by millions of people everyday for various purposes. Using a standard Web browser, the user can access information stored on Web servers distributed world wide. This gives the illusion that all this information is located on the users computer. So, the Web represents a huge distributed system that appears to users as a single resource.

Distributed systems are becoming a standard choice for implementing modern computing systems. Next is a non-exhaustive list of the more active research areas in the distributed system community:

- local and wide area networks,
- distributed operating systems,
- distributed databases,
- distributed file servers,
- distributed and concurrent programming languages,
- techniques for cooperative problem solving,
- distributed simulation.

There are also some fields, where already addressed problems are newly discussed when the distributed system is required, for instance, the naming service or the process scheduler of an operating system.

The choice of a distributed system allows resource sharing, which reduces costs, load balancing on resources and high availability of data through replication.

A common misconception when discussing distributed systems is that it is just another name for a network of computers. A distributed system is built on top of a network and tries to hide the existence of multiple autonomous computers. It appears as a single entity providing the user with whatever services. A network is a medium for interconnecting explicitly addressable entities enabling the exchange of messages between them using well-known standard protocols that comprise together the Open System Interconnection (OSI) protocol suite. The OSI model is shown in Figure 1.1.

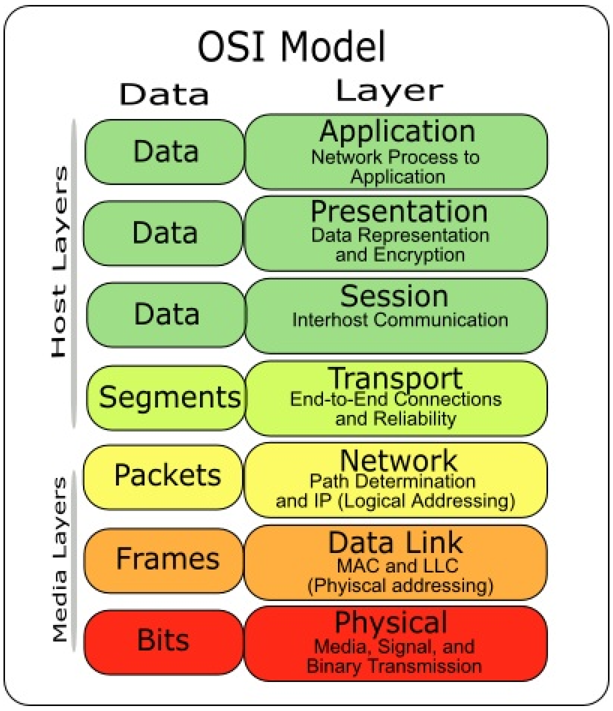

OSI Model
Data
Layer
Host Layers
Media Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections and Reliability
Packets
Network
Path Determination and IP (Logical Addressing)
Frames
Data Link
MAC and LLC (Phyiscal addressing)
Bits
Physical
Media, Signal, and Binary Transmission

Figure 1.1: OSI model

There are several definitions and points of view on what distributed systems are:

- ”A system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing" [1];

- "A collection of independent computers that appear to the users of the system as a single computer"[2];
- "A distributed system is one on which I cannot get any work done because some machine I have never heard of has crashed"[8].

The several number of definitions reflects the huge number of challenges faced by distributed system designers. Despite these challenges, the benefits of distributed systems and applications are many, making it useful to follow up. Figure 1.2 shows a general structure of a distributed system.

1.1 Goal of distributed systems

There are various types of distributed systems, such as Cluster [9], Distributed Storage Systems (e.g [3], [5]), Distributed Information systems [6], Wireless Sensor Networks [7] and so on.

In this section we discuss four important goals that should be met to build a distributed system:

- Make remote resources accessible;
- Hide the fact that resources are distributed across a network;
- Openness;
- Scalability.

1.1.1 Connecting resources and users

The main goal of a distributed system is to make easy for the users and their applications to access remote resources, like printers, computers, storage facilities, data, files, Web pages, and networks, and to share them in a controlled and efficient way. There are many reasons to share resources:

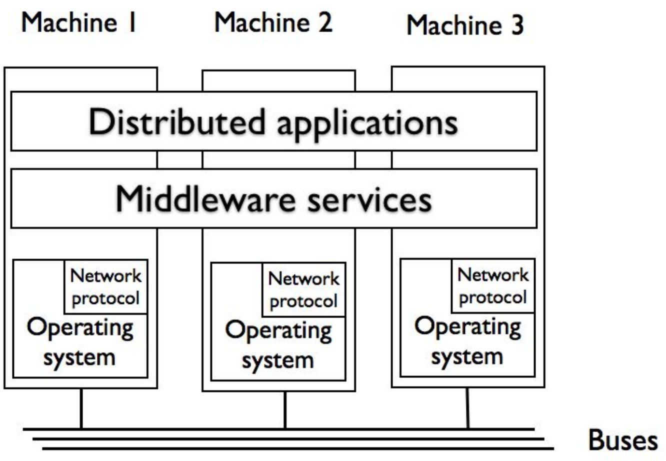


Figure 1.2: General structure of a distributed system

- Economics: for example, it is cheaper to make a printer shared by several users in an office than having to buy and maintain a separate printer for each user. Likewise, it makes sense to share costly resources such as supercomputers, high-performance storage systems, and other expensive peripherals.
- Connecting resources makes the exchange of information easier to the collaboration. This is clearly illustrated by the Internet where groups of people, geographically widely dispersed, work together by means of Groupware such as email and newsgroup applications.

However, proportionally to the increase of connectivity and sharing, security becomes increasingly important. Of course, few protections against eavesdropping or intrusion on communication are provided by the systems. Users passwords and other sensitive information are often sent as cleartext through the network, or stored at servers that we should trust.

Another security problem is related to the communications tracking. Such intrusion can easily end up by explicitly violating privacy. A related problem is that increased connectivity can also lead to unwanted communication, such as electronic junk mail, called spam. In such cases, to protect ourself, the messages filters can be used to analyzes the contents of messages and detect the undesired ones.

1.1.2 Distributed resources transparency

As previously said, an important goal of a distributed system is to hide that their processes and resources are physically distributed across multiple computers. The resources transparency has various forms:

- **Access transparency** : it has to do with hiding differences in data representation and the way that resources are accessed by users. The basic objective is to hide differences in machine architectures, but more is to reach agreement on how data can be represented by different machines and operating systems.

- **Location transparency** : an important group of transparency types deals with the location of a resource. Location transparency refers to the fact that users cannot tell where a resource is physically located in the system. Naming plays a crucial role to address the location transparency. Particularly, location transparency can be achieved by assigning only logical names to resources, giving no clue about the location of machines like http://www.google.com.

- **Relocation transparency** : distributed systems in which resources can be moved or replaced without affecting the access to those resources are said to provide migration transparency. More important is the situation in which resources can be relocated while they are being accessed by the users applications without noticing anything. In such case, the system is said to support relocation transparency. An example of relocation transparency is when users can continue to use their wireless laptops while moving from one place to another one without ever being disconnected.

- **Replication transparency** : replication plays a very important role in distributed systems. The resources may be replicated to increase availability or to improve performance by placing a replica close to the place where it is accessed. Replication transparency has to do with hiding the fact that several copies of a resource exist. To hide replication from users, it is necessary that all replicas have the same name, content, and seem to be in the same location. Consequently, a system that supports replication transparency should generally support location transparency as well.

- **Concurrency transparency** : as already mentioned, a foremost goal of distributed systems is to allow sharing of resources. In many cases, sharing resources is done in a cooperative way through communication. However. there are also the competitive sharing of resources. As example, two users may each wants to run an application using the same Central Processing Unit (CPU). In such cases, it is important that each user does not notice that the other is using the same resource. This occurrence is called concurrency transparency. A crucial issue is that parallel access to a shared resource should leaves that resource in a consistent state. Consistency can be achieved by using correctly the locking mechanisms when giving the users the exclusive access to the desired resource by means of this mechanism.

- **Failure transparency** : a system is said failure transparent when a user does not notice that a resource fails to work properly. Hiding failures is one of the hardest issues in distributed systems and is even impossible in some complicated cases.

Despite distribution transparency is generally considered important for any distributed system, there are situations in which attempting to completely hide all distribution aspects from users is not possible. As an example, a wide-area distributed system that connects a process in New York to a process in Amsterdam cannot be expected to hide the fact that a message is sent from one process to another one in more than hundreds of milliseconds. Signal transmission is not only limited by the speed of light, but also by limited processing capacities of the intermediate cables, switches and routers.

There is also a trade-off between a high degree of transparency and the performance of a system. As example, many network applications repeatedly try to contact a server before finally giving up and may be trying another one. Consequently, attempting to mask a server failure may slow down the system as a whole, it may have been better to give up earlier, or at least let the user cancel the server contact attempts.

As conclusion, looking for distribution transparency may be a nice aim when designing and implementing distributed systems, but system performance and architecture should be considered too.

1.1.3 Openness

Another goal of distributed systems is openness. An open distributed system is a system:

- **that inherent services are accessed through protocols**: The distributed systems services should be provided according to standard rules that describe the syntax and semantics of those services, so called protocols.

- **Portable and Interoperable**: In distributed systems, software components are generally specified through interfaces, which are often described in an Interface Definition Language (IDL). The interfaces definitions in the distributed systems enable the communication between the software components and an arbitrary external application. An interface should be interoperable and portable. Interoperability characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by relying on each other's services, while Portability characterizes the application, developed for a distributed system SysA, can be executed, without modification, on a different distributed system SysB that implements the same interfaces as SysA.

- **Extensible**: Another important goal for an open distributed system is that it should be easy to configure by different developers. Also, it should be easy to add new components or replace existing ones without affecting the other components.

- **Flexible**: To reach the flexibility in open distributed systems, it is crucial to organize the system as a collection of relatively small, configurable and easily replaceable or adaptable components. This implies that not only definitions for the highest-level interfaces seen by users and applications should be provided, but also definitions for interfaces for internal components of the system and the description of the interactions between them. Many systems are constructed using a monolithic approach in which components are only logically separated but implemented as one huge program. This approach makes hard to replace or adapt a component without affecting the entire system, which makes the trend of monolithic systems to be close.

  To achieve flexibility, a separation between system policy and mechanism is also needed. In practice, this can be implemented by offering a rich set of parameters that the user can set dynamically.

1.1.4 Scalability

The scalability can be defined as the ease with which a system or component can be modified to fit with the problem area. Another definition of the scalability, in telecommunication and software engineering, indicates the capability of a system to increase performance under an increased load when resources are added. Scalability of a system can be measured along at least three axis:

1. A system can be scalable in size, meaning that we can easily add more users and resources to the system.

2. A geographically scalable system is one that maintains its usability without being affected by how far its users or resources are.

3. A system can be administratively scalable if it can continue to be easy to

manage even if it spreads over many independent administrative organizations.

Unfortunately, a system that is scalable in one or more of these dimensions often shows some loss of performance as the system scales up.

1.1.4.1 Scalability Problems

When a system needs to scale, very different types of problems need to be solved.

- **Scalability in size**: In the case where the system scales in size, by supporting more users or resources, we are often confronted with the limitations of:
  - Centralized services: many services are centralized in the sense that they are implemented by a single server running on a specific machine in the distributed system. The obvious problem with that is the fact the server can become a bottleneck as the number of users and applications is growing.

    Even if we have unlimited processing and storage capacity, the communication with that server will eventually disallow further growth at some point. Unfortunately, using a single server to host a service is sometimes unavoidable, for example if we have a service to manage a highly confidential information such bank accounts. In such cases, it may be better to implement this service in a single server running in highly secured separate place, accessible only across a special network by the other components of the distributed system. Replicating this service in several places by another servers will increase, of course, the performance but in the same time will make the service less secure.
  - Centralizing data: it is another bad approach in term of performance. We can imagine how the Internet would work if its Domain Name System (DNS) was implemented as a single table by a single server. DNS maintains information on millions of computers worldwide and it is an essential service for locating Web servers. If each request to

    resolve an Uniform Resource Locator (URL) is forwarded to a single DNS server, it is obvious that at some point no one would be able to use the Web.

  - Centralized algorithms: they are also bad ideas. In a large distributed system, an enormous number of messages have to be routed over many lines. From a theoretical point of view, the optimal way to do this is to collect complete information about the load on all machines and lines, and then run an algorithm to compute the optimal routes. This information can be thus spread around the system to route the packets of messages. The issue is that collecting all the input and routing all the output will cause the overload of some part of the network. In fact, all algorithms based on the collect of information from all sites, the process of these information and sending then the results obtained over the network should generally be avoided. So, only decentralized algorithms should be used.

- **Geographical scalability**: Geographical scalability has its proper problems. Synchronous communication adopted by Local Area Network (LAN) distributed systems makes hard to scale with those systems in Wide Area Network (WAN). In a LAN, a client blocks until a reply is sent back. This procedure takes at worst a few hundred microseconds in a LAN and it will reach hundreds of milliseconds in WAN. Another problem related to geographical scalability is the unreliability of the WAN, In contrast with the highly reliable LAN. Geographical scalability is strongly related to the problems of centralized solutions. If we have a system with many centralized components, it is clear that geographical scalability will be limited by the performance and reliability problems resulting from wide-area communication.

- **Administrative scalability**: A major problem, that needs to be solved, in a distributed systems across multiple and independent administrative domains, is the conflict policies with respect to resource usage and payment, management, and security.

1.1.4.2 Scaling Techniques

Discussing some of the scalability problems drives us to the question how to resolve them. In general, scalability problems will lead to performance problems. There are basically three techniques for scaling:

- **Hiding communication latencies**: This technique is important to resolve geographical problems. Basically, in this technique, it is tried, as much as possible, to avoid waiting for reply from the distant service. Practically, the idea is to submit a request for a service and therefore continue other works in pending or begins new ones. When a reply comes in, the application is interrupted and a special handler is called to complete the previous request. Such approach defines the asynchronous communication. However, there are many applications that cannot make use of the asynchronous communication. In such cases, a much better solution is to try to reduce the overall communication, for example, by moving part of the computation that is normally done at the server to the client process requesting the service.
- **Distribution**: It means taking a component of the system, splitting it into smaller part, and therefore spreading those parts across the system. An example of distribution is the Internet DNS.
- **Replication**: Considering that scalability problems can be seen across the performance degradation, it is generally a good idea to replicate components across a distributed system. Replication not only increases availability, but also helps to balance the load between components leading to better performance. Also, in geographically widely-dispersed systems, having a copy nearby can hide much of the communication latency problems mentioned before. Caching is a special form of replication, although the distinction between both techniques is often hard to make. There is one serious drawback to caching and replication that may affect scalability. By having multiple copies of a resource, the modification of one copy makes that copy different from the others and, consequently, can lead to consistency problems.

1.1.5 Requirements of a distributed system

Distributed systems differ from other ones by the dispersion of their components across the network. Not taking this dispersion into account during the design time will make the system complex and include mistakes that need to be patched later on. These mistakes can be formulated as the following false assumptions that everyone makes when developing a distributed application for the first time [4]:

1. The network is reliable.
2. The network is secure.
3. The network is homogeneous.
4. The topology does not change.
5. Latency is zero.
6. Bandwidth is infinite.
7. Transport cost is zero.
8. There is one administrator.

When developing a non-distributed applications, many of these issues will not be faced.

1.2 Distributed systems architecture

Many researchers agree on the fact that reasoning in terms of clients that request services from servers helps on understanding and managing the complexity of distributed systems.

1.2.1 Processes communication forms

Processes are pieces of work with a state and role. The state consists of the data that is managed by the process while the role corresponds to the implementation of the applications logic. The processes cooperate with one or more process through messages exchange. A message consists of a sequence of bytes that are transported between two processes. When exchanging message, one process assumes the role of sender and one or more processes will be receivers.

A process is either in an active or a passive state. A process can carry out some works during an active state. Various events, such as the receipt of a message, can change the state of a process. A simple classification of communication mechanisms can be as follow:

- Message-oriented communication: the sender transmits a message to the receiver but does not wait for a reply. Message exchange takes place in only one direction.
- Request-oriented communication: the receiver responds to the sender with a reply message. The communications process is not complete until the sender has received a reply to his request.

The synchronicity of a communication mechanism depends on communication patterns. The synchronicity describes the time separation between sender and receiver. In a synchronous communication, the sender is passive during the communication process until the message has arrived at the receiver. Conversely, during asynchronous communication the sender remains active after a message has been sent. Within the latter, a sender can transmit messages more quickly than the receiver is able to accept them, and thus the transport system must be capable of buffering messages. Although the buffering of messages in the transport system is not necessary in the case of synchronous communication, sender and receiver are coupled together. However, asynchronous communication is better at supporting the parallelism of processes. As example of communication mechanisms:

- synchronous request-oriented communication is the Remote procedure call (RPC). The sender sends a request to the receiver and stays in passive mode until the receiver delivers the results of the request. An example of asynchronous request-oriented communication is Remote Service Invocation (RSI). During this type of communication, the sender remains active while the receiver is processing the request. Although RSI makes better use of the parallelism offered in distributed systems, while RPC is based on popular programming paradigms and is therefore intuitive on its interpretation.

- asynchronous message-oriented communication is the Datagram service, on which is based the Internet protocol User Datagram Protocol (UDP). The sender transmits a message to the receiver without waiting for a reply or changing to the passive state.

- synchronous message-oriented communication is the rendez-vous protocol. It is used in the synchronization of two processes.

1.2.2 Client-Server model

In the Client-Server model, the server is a process implementing a specific service as a file system service or a database service. The client is a process that requests a service from a server by sending it a request and then waiting for the server's reply. This Client-Server interaction, also known as request-reply behavior is shown in Figure 1.3.

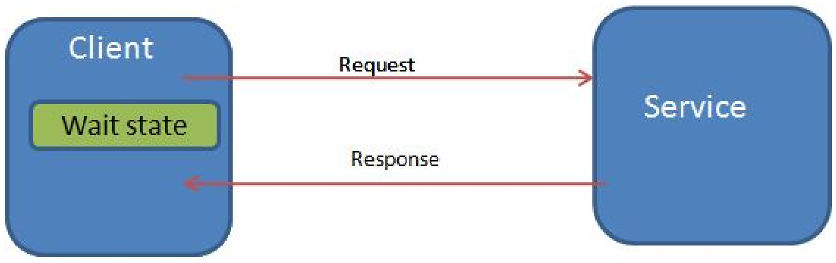


Figure 1.3: Request-Reply behavior in Client-Server architecture

Communication between a client and a server can be implemented by means of a simple connectionless protocol when the underlying network is fairly reliable as in many LANs. In these cases, when a client requests a service, it simply packages a message for the server, identifying the service it wants, with the necessary input data. The message is then sent to the server. The server, in turn, will always wait for an incoming request to precess it. Once the message is received and the request is processed, the server sends the reply message with packaged results to the client.

The use of a connectionless protocol has the advantage of being efficient. As long as messages do not get lost or corrupted, the request/reply protocol just works fine. Unfortunately, making the protocol resistant to occasional transmission failures is not trivial. The only thing we can do is possibly let the client resend the request when no reply message comes in. However, the problem is that the client is not able to distinguish if the request message was lost or if the transmission of the reply, by the server, failed. In the latter, resending the request by the client can result in performing the operation twice. If that operation was something like bank transfer then it is clear that it would be better that we simply reported an error instead. On the other hand, for idempotent operations, it would be perfectly acceptable to resend the request.

As an alternative, many Client-Server systems use a reliable connection-oriented protocol. While this solution is not entirely appropriate in a LAN due to relatively low performance, it works perfectly in wide-area systems, as has demonstrated the Internet protocol Transport Communication Protocol (TCP). In that kind of communication, the client sets up a connection to the server before sending the request. This connection is also used by the server to send the reply message. This connection is then torn down. The issue is that setting up and tearing down a connection is expensive in term of performance, especially when the request and reply are small.

It is clear that Client-Server systems can easily scale up in size more than centralized solutions since the services provided by the server can be distributed across several server computers as the number of clients increases. Server processes can thus run in parallel, each process serving its own set of clients. However, when there are multiple servers, there must be some coordination mechanism to update information in order to avoid inconsistencies.

The drawbacks of the Client-Server model are that:

- security is more difficult to ensure in a distributed environment than in a centralized one,
- the administration of distributed equipment can be much more expensive

than the maintenance of a centralized system,

- data distributed across servers needs to be kept consistent, and that the failure of one server can make a large Client-Server system unavailable, unless the system is designed to be fault-tolerant.

The computer network can also become a performance or reliability bottleneck for the Client-Server model in a distributed environment: if the network fails, all servers become unreachable or if one client produces high network traffic, consequently, all clients may suffer from long response times.

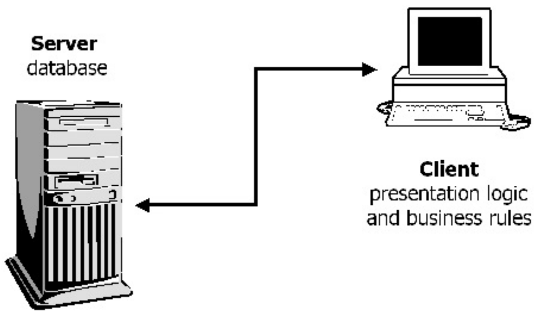


Figure 1.4: Typical two-tier architecture.

1.2.2.1 Design considerations in the Client-Server architectures

An important design consideration for large Client-Server systems is whether a client talks directly to the server, namely a two-tier architecture, or whether an intermediary process is introduced between the client and the server, defined as a three-tier architecture and also known as a multi-tier architecture.

The two-tier architecture is easier to implement and is typically used in small environments. However, a two-tier architecture is less scalable than a three-tier architecture. Typically, in the two-tier architecture, there is an application running in the client machine which interacts with the server, generally, a database management system (see Figure 1.4). The client application, also known as a fat client, contained some or all of the presentation logic, the application, the business rules and the database access. Every time the business rules is modified, the client application had to be changed, tested and redistributed, even when the user interface remained the same. In order to minimize the impact of business logic alteration within the client, the presentation logic must be separated from the business rules. This separation becomes the fundamental principle in the three-tier architecture. In a three-tier architecture, there are three or more interacting tiers, each with its own specific responsibilities (see Figure 1.5):

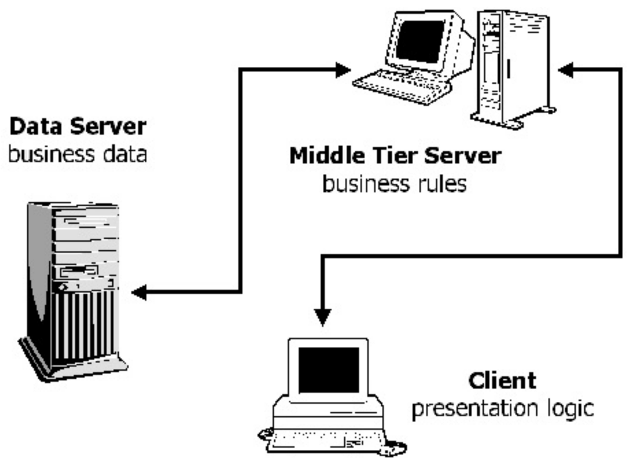


Figure 1.5: Three-tier architecture.

- Tier 1: the client contains the presentation logic, including simple control and user input validation. This application is also known as a thin client.
- Tier 2: the middle tier is also known as the application server, which provides the business processes logic and the data access.
- Tier 3: the data server provides the business data.

The followings are some of the advantages of a three-tier architecture:

- Ease in modifying or replacing any tier without affecting the other ones.
- Better load balancing by separating the application and database functionality.

- Security policy can be enforced in server tiers level without involving the clients.

When designing a Client-Server system, there are a lot of considerations to take into account to benefit from the advantages of the model. The followings are some of these considerations to have a consistent system, secure and reliable.

- Stateful vs. stateless: The fact is whether a server should be stateful or stateless. A stateless server retains no information about the data that clients are using. Client requests are fully self-contained and do not depend on the server. The advantage of the stateless model is that it is easier to implement and that the failure of a server or client is easier to handle. However, applications where clients need to acquire and release locks on the records stored at a database server usually require a stateful model, because locking information is maintained by the server for each individual client.

- Authentication: For security purposes, servers must also address the problem of authentication. In a networked environment, an unauthorized client may attempt to access data stored on a server. Authentication of clients can be handled by using cryptographic techniques such as public key encryption. In public key encryption, the client application signs requests with its private cryptographic key, and encrypts the data in the request with a secret session key known only to the server and to the client. On receipt of the request, the server validates the signature of the client and decrypts the request only if the client is authorized to access the server.

- Fault-tolerance: Applications such as bank-transactions systems and real-time market must be fault-tolerant. This means that important services should remain available despite the failure of part of the computer system, and that no information is lost or corrupted when the failure occurs. To address the high availability, critical servers can be replicated, which means their services are provided on multiple computers. If one replica fails then the other replicas still remain accessible by the clients. In other hand, the database consistency should be ensured.

1.2.3 Distributed Object Computing

The concepts of an object model are applied in structuring complex problem domains. The result of the modeling is a representation that is used as the point of departure for studies and discussions.

The main concept of object models is the object. The decomposition of a problem domain using the concepts in the object model results in a set of objects. An object is a thing of our imagination that can be abstract or even concrete in nature. Although the choice of granularity for an object is relative to the observer, an object distinguishes itself from other objects only on the basis of its definition. An object is a self-contained problem area.

Objects are owners of an identity. All objects are differentiated on the basis of their unique identity. An object is characterized also by its state and its behavior. The behavior is usually influenced by the state. State and behavior define the functionality of an object, which can be used by other objects. The implementation realizes the functionality of an object by a programming language. Each object corresponds to a component, and these components communicate through messages.

Distributed object computing is a generalization of the Client-Server model. Object-oriented modeling and programming are applied to the development of Client-Server systems. In this context, objects are pieces of software that encapsulate an internal state and make it accessible through a well-defined interface. In such model, the interface consists of object operations and attributes that are remotely accessible. Client applications can connect to a remote instance of the interface, with the help of a naming service, and invoke the operations on the remote object. The remote object acts as service provider in the server.

This use of objects naturally favors heterogeneity and autonomy. It supports heterogeneity since requests sent to server objects depend only on their interfaces and not on their internal interactions. It permits autonomy because object implementations can change easily and in a transparent way, provided they maintain their interfaces.

Client-Server objects of given systems have to be able to interact with Client-Server objects of other Client-Server systems even if they are written in different programming languages and run on different hardware and operating system platforms.

Standards are needed for objects to interact in heterogeneous environments. One of the most adopted standards is the Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) specification.

1.3 Middleware

Middleware is a class of software designed to manage the complexity and heterogeneity in distributed systems. It is defined as a layer of software working above the operating system and networking softwares but below the application program. Middleware enables the interaction and communication between different applications through Application Programming Interfaces (API) over distributed components. As the APIs allow to applications to access operating systems and other services, the middleware can be seen as a common development environment to enable the connection of components written in different languages and running on different operating systems.

The classical definition of an operating system is the software that makes the hardware useable. Similarly, middleware can be considered to be the software that allow to implement a distributed system.

1.3.1 Categories of Middleware

There are four main different kinds of middleware that have been developed, namely transactional, procedural, message-oriented and object-oriented middleware. These vary in terms of the programming abstractions they provide, their designs, and the level of heterogeneity supported.

- **Transactional Middleware (TM) or Transaction Processing (TP) monitors**: they were designed to support distributed synchronous transactions. The main function of a TP monitor is a coordination between the clients and servers to execute the transactions that the clients ask. TM uses grouping of the service requests into transactions. A transaction must support ACID properties, namely Atomic, Consistent, Isolated, and Durable.

Atomic means that a given transaction either completes or it does not. Consistency implied that the system should be kept in a consistent state, independent of the status of a transaction. Isolation is ability of one transaction to work independently from other transactions. Durability means the ability of a transaction to survive system failures. Client and server components can reside on different hosts and therefore requests are transported via the network transparently to client and server components. TM supports software and hardware heterogeneity, since the components can be deployed on different hardware and operating system platforms. But TM does not support data heterogeneity very well, because it cannot translate complex data structures and therefore cannot marshall these structures.

- **Procedural Middleware (PM)**: RPCs were developed by Sun Microsystems in 1980. RPCs can be implemented on different operating systems, including most Unix and MS Windows systems. RPCs define server components as RPC programs. An RPC program contains parameterized procedures. Remote clients can invoke these procedures across the network using low-level network protocol, such as TCP/IP or UDP. Typical RPC-based communication is synchronous. The heterogeneity in RPCs is addressed via IDL. It defines the interfaces that represent relations between servers and clients. IDL is programming language independent, which means that client does not need to know the language that server supports, since IDL compiler can translate the client's request to a server.

- **Message Oriented Middleware (MOM)**: it allows a communication through messages. There are two different types of MOM: message queuing and message passing. Message queuing is defined as indirect communication model, where communication happens via a queue. Message from one program is sent to a specific queue, identified by name. After the message is stored in queue, it will be sent to a receiver. The message passing is a direct communication model where the information is sent to the interested parties. One important subgroup of message passing approach is publish-subscribe model. In this model, clients have the ability to subscribe to the interested messages. After subscribing, the client will receive only messages corresponding to a subscribed subject. MOM supports both synchronous via message passing and asynchronous via message queuing communication. In comparison to RPCs, most messaging products support for many additional communication protocols. While the support of data heterogeneity is limited in MOM, because is not automatically generated, as in RPCs, and needs to be implemented by programmers

- **Object Oriented Middleware (OOM)**: it is an evolution from RPCs, extending them by adding object-oriented concepts. These concepts are mainly inheritance and object references. It delivers all advantages provided by RPC-based middleware such as automatic marshalling and unmarshlling, and the use of IDL to allow the communication between heterogeneous platforms. In addition to the synchronous communication, some OOM products, such as CORBA 3.0, support the asynchronous communication.

1.4 Distributed computing systems

An important class of distributed systems is the one used for the high-performance computing. Two important subgroups of this class are Cluster and Grid computing. In Cluster computing the underlying hardware consists of a collection of workstations or PCs, closely connected by means of a high-speed local-area network. In addition, each node runs the same operating system. The situation becomes different in the case of Grid computing. This subgroup consists of distributed systems that are often build as a federation of computer systems, where each system may belong to different administrative domain, and may be deployed in a very different environment from hardware, software, and network technology points of view.

1.4.1 Cluster Computing Systems

Cluster computing is not a new area of computing, it was a natural consequence of several trends in computing in the 1990s. However, it is evident that there is a growing interest in its usage in all areas where applications are using parallel or distributed computing platforms. The growing interest has been stimulated mainly by:

- the availability of powerful microprocessors and high-speed networks as off-the-shelf commodity components,
- the rapidly maturing software components available to support high performance and high availability applications,
- the improvement of the price/performance ratio of personal computers and workstations.

All computers in the Cluster are attached together with middleware support for collective usage as a single computing resource.

The most common system categories for building a Cluster of computers are:

- Centralized Job Management Systems:

  The systems of this category include at least 3 major modules:

  - User server: It is the interface of users for the interactions with the Cluster. It allows the user to know the status of the queues, and to create and manage their jobs. Furthermore, it places a job into execution after being scheduled.
  - Job scheduler: It performs job queueing and scheduling based on job type and requirements, resource availability, and scheduling policy,
  - Resource manager: It allocates and monitor resources inside the Cluster.

  Portable Batch System (PBS)[69] and Local Sharing Facility (LSF)[68] are examples of Centralized Job Management Systems.

- Distributed Operating Systems:

  - One well-known example of a Cluster computer is formed by Linux-based Beowulf Clusters. Each Cluster consists of a collection of compute nodes that are controlled and accessed by means of a single master node. The master typically handles the allocation of nodes to a particular parallel program, maintains a batch queue of submitted jobs, and provides an interface to users to manage their jobs. Actually, the master implements the middleware needed for the execution of programs and management of the Cluster, while the compute nodes often need nothing else but a standard operating system. An important part of this middleware is formed by standard communication libraries such as Message Passing Interface (MPI) or Parallel Virtual Machine (PVM). Many of these libraries effectively provide only advanced message-based communication facilities.

– As an alternative to this hierarchical organization, a symmetric approach is followed in the MOSIX system. In the case of MOSIX, it is attempted to provide a single-system image, as the other Cluster systems, offering the ultimate distribution transparency by appearing to be a single computer unlike some other Cluster systems. The high degree of transparency is provided by allowing processes to dynamically and preemptively migrate between the nodes, that make up the Cluster, to make efficient use of resources.

1.4.2 Grid computing model

Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach a common goal. What distinguishes Grid computing from conventional High Performance Computing (HPC) systems such as Cluster computing is that Grids tend to be more loosely coupled, heterogeneous, and geographically dispersed.

A key design in a Grid computing system is that resources from different organizations are made available together to the collaboration of a group of people or institutions, by means of virtual organization. The people belonging to the same virtual organization has access rights to the resources that are provided to that organization. Typically, resources consist of computer servers (including supercomputers, possibly implemented as Cluster computers), storage facilities, and databases. In addition, special and expensive devices, such as telescopes and sensors, can be provided as well.

1.4.2.1 Layered architecture

The Grid architecture consists of four layers:

- The lowest one is the fabric layer that provides interfaces to local resources at a specific site. These interfaces, typically, will provide functions for querying the state and capabilities of a resource, as well as functions for resource management, to allow sharing of resources within a virtual organization.

- The connectivity layer consists of communication protocols for supporting Grid transactions across multiple resources, like protocols to transfer data between resources, or to simply access a resource from a remote location. In addition, this layer includes the security protocols to authenticate users and resources. The resource protocols are responsible for managing a single resource. It uses the functions provided by the connectivity layer and calls directly the interfaces made available by the fabric layer. This layer can provide functions for obtaining configuration information on a specific resource or to perform specific operations such as creating a process or reading data. The resource layer is thus seen to be responsible for access control, relying on the authentication performed as part of the connectivity layer.

- The collective layer offers system-wide services such as Resource Brokers (RB) or Grid information services.

  The Grid information service collects and combines the information from the individual Grid Resource Information Services (GRIS) of the Grid sites.

  Resource brokers provide resource discovery and allocation. If, for example, an application should be executed on a parallel system with at least 1000 processors, the resource broker can consult the Grid information service to identify candidate resources and decides on the resource to be used for the job.

  Other services on this layer are workflows management, co-reservation and co-allocation, and replication management.

- The application layer consists of the applications of a given virtual organization and making use of the Grid computing environment. Typically the collective, and connectivity and resource layers provide jointly access management of resources that are dispersed across several sites.

Grid layers described above are illustrated in Figure 1.6.

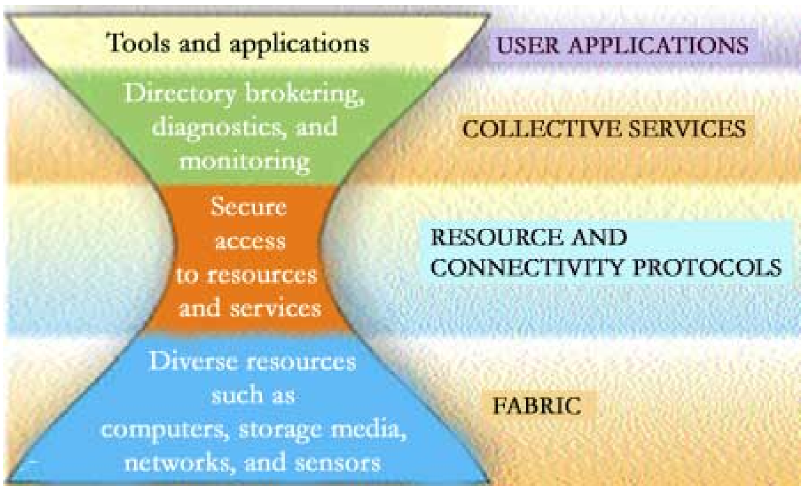

Tools and applications
Directory brokering, diagnostics, and monitoring
Secure access to resources and services
Diverse resources such as computers, storage media, networks, and sensors
USER APPLICATIONS
COLLECTIVE SERVICES
RESOURCE AND CONNECTIVITY PROTOCOLS
FABRIC

Figure 1.6: Conceptual structure of a Grid

1.4.2.2 Grid resources management

As the Grid couples multiple resources owned by different individuals or organisations, the choice of the right model for resources management architecture plays a major role in its eventual success. There are a number of approaches that one can follow when developing Grid resource management systems. The followings are the most used ones:

- Hierarchical Model: This model (shown in Figure 1.7) was proposed in the Grid Forums second meeting [13]. A possible scenario that makes use of such model can be:

  1. A user submits a job to a job control agent, which calls an admission agent.

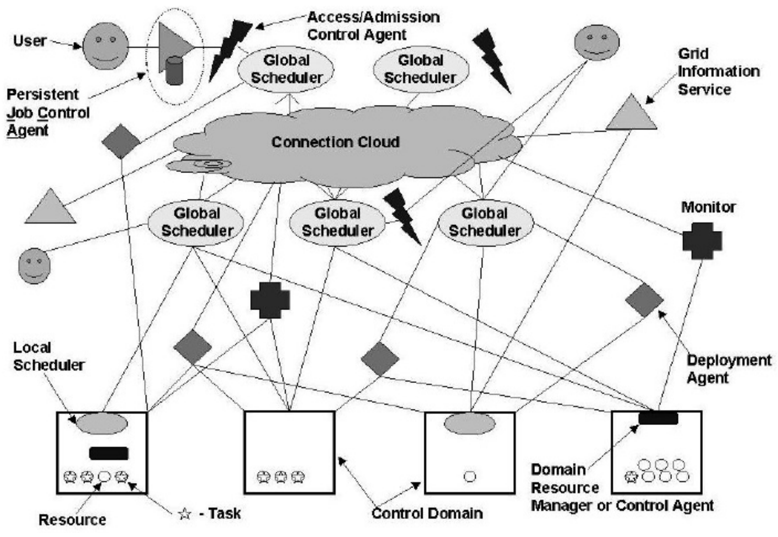


Figure 1.7: Hierarchical Model for Grid resources management.

2. The admission agent examines the job requirements in term of resource, interacting with a Grid information system, and evaluate whether it is safe to inject the job to the system work pool. If it is the case, the admission agent passes the job to a scheduler.
3. The scheduler performs resources discovery using the Grid information system and then consults with domain control agents to determine the current state and availability of resources. The scheduler then computes a set of mappings and passes these mappings to a deployment agent.
4. The deployment agent negotiates with the domain control agents for the resources indicated in the schedule, and obtains reservations for the resources. These reservations are passed to the job control agent.
5. The job control agent works with a different deployment agent, and the deployment agent coordinates with the appropriate domain control agents to start the tasks running.
6. A monitor tracks progress of the job and performs outcalls to Job Control Agents and Schedulers to effect remapping, job status, extending lifetime, etc.

This scenario is shown schematically in Figure 1.8. Grid Resource Allocation Management (GRAM) in Globus toolkit [15] and Legion project [14] are examples of applications implementing this model.

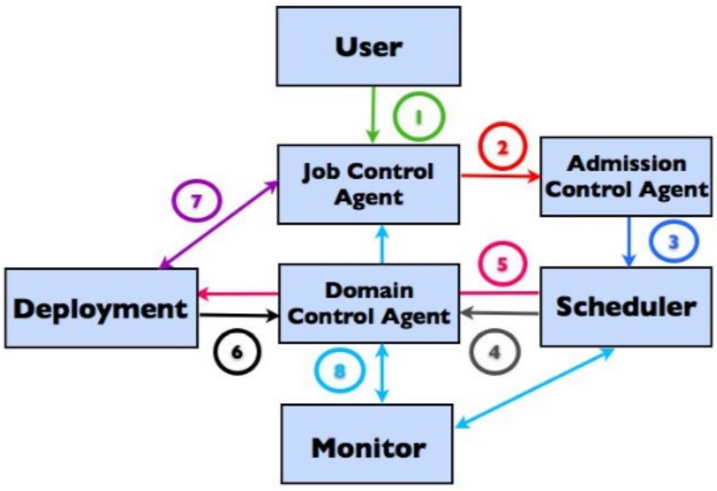


Figure 1.8: An example of scenario in the hierarchical Model.

- Abstract Owner (AO) Model: The principle of this model is that users look for the ability to make an agreement with some entity regarding the conditions under which the resources can be used, the mechanisms for using the resources, the cost of the resources, and the means of payment, and don't want to care who owns the resources. It is the same approach used for the Internet where the user deals with the Internet Service Provider (ISP) to access Internet resources. ISP is not the owner of Internet resources but the user can think of it that way abstractly.

  In this model, to acquire access to a resource from an AO, the client (which may be another AO) negotiates with that AO through its Order Window. These negotiations may include asking how soon the resource may become available, how much it might cost, etc. If client is not happy with the results of the negotiations, it may just terminate negotiations, or might actually place an order. After being ordered, the resources are delivered from the

AO to the client through the Pickup Window. The precise protocol to be used for acquiring the resources is flexible and may also be negotiated at order time, for e.g, the AO may alert the client when the resource is ready.

Order/Pickup windows should be accessible remotely, and must support a standard interface such as Remote Procedure Invocation (RMI) interface. The AO might be an owner of resources and/or broker.

The general structure of this model is shown in Figure 1.9.

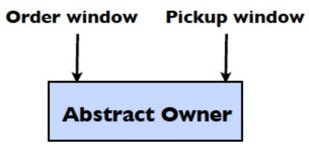


Figure 1.9: General structure of AO model.

The AO performs only part of the standard job scheduling process, namely resources acquisition, leaving the rest of the process to the client such as tasks assignation to resources and the monitoring. As alternative is to consider a new layer "Job Shop" atop of the AO to handle the job scheduling and uses AOs only to acquire resources, as shown in Figure 1.10 .

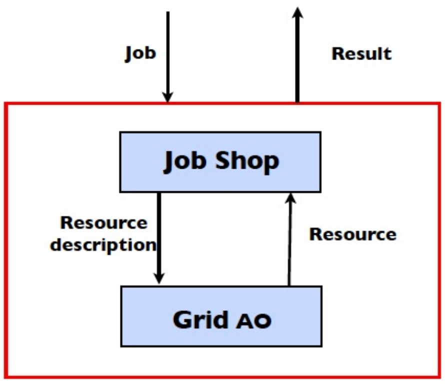


Figure 1.10: Job Scheduling.

The AO model is still a proposal and there are many remaining gaps in the model.

- Computational Market/Economy Model: Grid resources are geographically distributed and belong to different organization. Each organization has it own resources management policy and mechanism and may different charges for different groups of users, necessitating the need of a market-driven resource management to minimize the user expenses and maximize the owner return-on-investment.

  The key components in the Market model are:

  - User Applications.
  - Grid Resource Broker: it acts as mediator between users and Grid resources using middleware services. It takes care of the resource discovery, resource selection, matching data, applications, and hardware resources, updating the user view to the Grid with the changes of resources.
  - Grid Middleware: it provides Grid core services such remote process management, storage access, and authentication and security management. In addition, it provides specific economy-driven model services such as :
    - Trade Server: it is a resource owner agent aims to maximize the resource utility and profits for its owner.
    - Pricing algorithms: defines the price that resource owners would like to charge users.
    - Accounting System: records resource usage and responsible for users billing.
  - Domain Resource Manager: the Local Resource Manager (LRM) are responsible for managing and scheduling works, such as computations, access to storage devices, or special scientific instruments, across local resources.

These models are described in detail and discussed in this paper [16].

1.4.2.3 Security

Grid applications are in particular distinguished by their simultaneous use of large numbers of resources, dynamic resource requirements, use of resources from multiple administrative domains, complex communication structures, and stringent performance requirements. The characteristics of computational Grids lead to security problems that are not addressed by existing security technologies for distributed systems. The followings are some considerations to take into account when developing a Grid security infrastructure:

- Since the resources are shared among multiple organizations, the local system protection should be enforced against the external use.
- Data and applications should be protected against the remote system where the computation runs.
- The large and dynamic number of resources and users.
- The processes constituting a computation may communicate by using a variety of mechanisms, where the need to support secure group communication and multiple communication mechanism implementations.
- Resources may require different authentication and authorization mechanisms and policies, furthermore, the resources and users may be located in different countries, where the need for an uniform credential and certification infrastructure using defined standards.
- A computation may require, during its execution, several starts of processes which necessitate the dynamic acquire and release of resources. So, a user should be able to authenticate once and then initiate computations using Grid resources without further authentication, namely Single Sign-On (SSO) mechanism.
- User credentials must be protected.
- Proxy / agent delegation to avoid the need to communicate user credentials across the network.

Globus Security Infrastructure (GSI)[17] is an example of such infrastructure and was developed as part of the Globus project [17]. The GSI is based on public key cryptography and the Secure Socket Layer (SSL)[18] mechanism, and is designed to address Grid security problems such as delegation and SSO.

1.4.3 Cloud computing

The Cloud computing is a new computing model, inspired by the idea of utility computing [19] from 1960s, to turn computation into something like an electric Grid, where every user can just plug in and use it. In other words, it aims to allow the access to an HPC level infrastructure from a Web browser:

- without initial high investments to buy the hardware,
- no additional costs to rent or buy server rooms, cooling, energy, etc,
- no hiring personnel to maintain Network and Systems,
- pay just for the time you use it,
- scale up or down as needed,
- Focus on the core interest.

Despite the number of products, white papers, publications, seminars, workshops, and more focused on such model, there are still many ambiguities and only few concrete terms. The National Institute of Standards and Technology (NIST)[20] definition of the Cloud computing is the most commonly used one: ”Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management.”. The Cloud computing model is composed of three main service models:

- Software as a Service (SaaS): Easy Access to hosted applications over the network from various client devices such as a Web browser and providing the API to these applications.

- Platform as a Service (PaaS): Deploy onto the Cloud infrastructure new applications created using supported programming languages and tools.
- Infrastructure as a Service (IaaS): Provision computation, storage, networks, and other fundamental computing resources by providing access to hardware resources.

These services are shown in Figure 1.11.

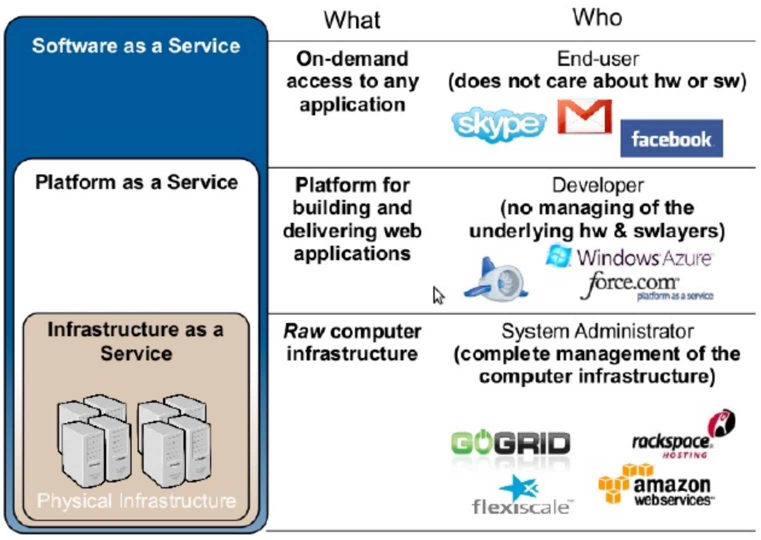

Software as a Service
Platform as a Service
Infrastructure as a Service
Physical Infrastructure
What
Who
On-demand access to any application
End-user (does not care about hw or sw)
Platform for building and delivering web applications
Developer (no managing of the underlying hw & swlayers)
Raw computer infrastructure
System Administrator (complete management of the computer infrastructure)

Figure 1.11: Cloud service models

This new model becomes the trend in HPC since few years for the following reasons:

- The continuous growing of massive scale data to access,
- A growing interest to new programming frameworks (such as MapReduce [21]), data models (such as NoSQL [22]), file systems (e.g HDFS [23]), tools developed "in house", etc,
- The massive amount of resources all across the planet (more than 500,000 cores only for Google),

- Pervasive broadband Internet,
- The cheap cost of very fast interconnection (10GbE),
- A growing interest to virtualization techniques developments and improvements,
- Higher needs to consolidate (e.g the virtualization and the energy costs),
- Continuous emergence of new devices such as iPhone,
- A truly inter-connected planet.

However, it comes with an entirely new set of problems.

1.5 Worldwide LHC Computing Grid distributed system

The Worldwide Large Hadron Collider (LHC)[51] Computing Grid (WLCG)[26] is a global collaboration of more than 140 computing centres in 34 countries, the 4 LHC experiments, and several national and international Grid projects. Launched in 2002, the mission of the WLCG project is to build and maintain a distributed data storage and analysis infrastructure, of 15 Petabytes (15 million Gigabytes) of data annually generated by the LHC, for the entire high energy physics community that will use the LHC at CERN[46]. The computing centers, contributing in such infrastructure, are inter-connected in a tiered architecture, act nevertheless as a single coherent system, with different resources and services provided by each Tier level. The project serves currently a community of more than 8,000 physicists around the world with near real-time access to LHC data.

The WLCG is now the world's largest computing Grid. It is based on the two main global Grids currently in operation:

- European Grid Infrastructure (EGI)[28] in Europe
- Open Science Grid (OSG)[29] in the United States

Regional or national Grids are associated to these projects. Grid infrastructures in Asia also contribute to the WLCG, such as Taiwan Grid (TWGrid)[30] and EU-IndiaGrid[31], amongst others.

Although WLCG depends on the underlying Internet, the middleware is essential to handle the underlying Grid infrastructure. The Grid middleware used for the WLCG Grid includes components that have been developed by a number of projects and organisations. The most important stacks in WLCG Grid middleware are:

- European Middleware Initiative (EMI)[32], which has the aim to implement a standard solution of the major European middleware stacks (ARC[33], gLite[34] and UNICORE[35]).
- Globus Toolkit developed by the Globus Alliance [36].
- Open Middleware Infrastructure Institute (OMII) [37].

This middleware is called "EGEE/WLCG". The most important services, that provides, are:

- Computing Element (CE) : provides an interface for the use of site computing resources. Its main functionality is the jobs management. A CE includes: Grid Gate (GG), which is a generic interface to the Cluster, the batch system, and the Cluster, which is a collection of Worker Nodes (WN).
- Storage Element (SE) : provides uniform access to data storage resources. It may manage a simple disk servers, large disk arrays or tape-based Mass Storage Systems (MSS). Storage Elements can support different data access protocols and interfaces, such as GridFTP[38] and DCAP[39].
- Information Service (IS) : provides information about WLCG Grid resources and their status. Most middleware stacks rely on information from Globus Monitoring and Discovery Service (MDS)[40], nevertheless Relational Grid Monitoring Architecture (R-GMA)[41] service is also in use for other purposes. The MDS is formed by different levels of servers from which the

information can be obtained, namely resource Berkeley Database Information Index (BDII)[43], which provides specific information for a particular service, site BDII, as data collector of all resource BDII at the site, and top-level BDII, which collects all information coming from site BDIIs. The BDII is essentially an LDAP[24] server updated by an external process.

- User Interface (UI) : provides an entry point to WLCG infrastructure and environment.
- Ressource Broker: takes care to match resources with jobs requirements. It is the responsibility of the user to describe the jobs and their requirements. In WLCG, the Workload Management System (WMS)[44] component plays the role of the Resource Broker.

Figure 1.12 shows the job flow in the computational WLCG / EGEE Grid.

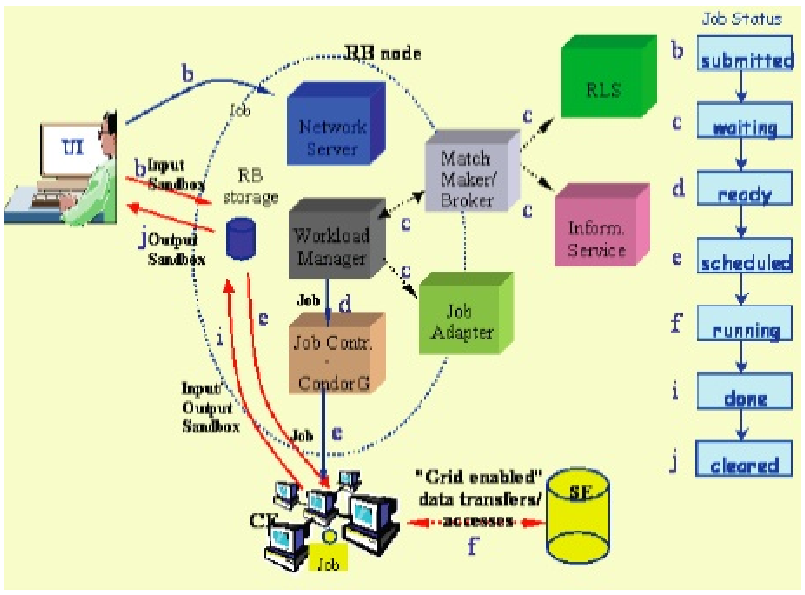


Figure 1.12: Job flow in the WLCG / EGEE Grid.

The main steps are as follows:

a) The user gets a digital certificate from the Certification Authority, enrolls in a VO and gets an account in an UI. At this point, he can create, in the UI, a proxy certificate that allows him to interact with the Grid;

b) A job is submitted to the RB from the UI, specifying the files to be copied to the Worker Nodes (Input Sandbox); the job status is SUBMITTED;

c) Workload Management System searches the best available Computing Element according to the requirements specified by the user, by querying the BDII; the state of the job is WAITING;

d) RB prepares the job for the submission and sends the necessary information to the CE selected; the job status is READY;

e) CE receives the request; the job status is SCHEDULED;

f) The local resource manager of the Cluster sends the job to a local WN for the execution and user files are copied from the RB to this WN; the job status is RUNNING;

g) During its execution, the job can read files from the Storage Element;

h) The job can produce output file which can be stored in the SE and accessed by other users;

i) If the execution of the job is completed successfully, the output (Output Sandbox) is copied to the Resource Broker; the status of the job is DONE;

j) At this point, the user can, from the UI, retrieve the output of its jobs; the status of the job is CLEARED;

k) Requests to track the status of the job and resources can be made from the UI;

l) The job can be passed to another CE in case where the site to which the job has been sent is not able to accept and execute it; after several retries of resubmission, the job is marked as ABORTED.

Chapter 2

CMS computing model and distributed data analysis

Recent collider experiments such as the Large Electron Positron Collider (LEP)[45] at CERN[46] and the Tevatron[47] at Fermilab[48] have provided remarkable results and precision tests of the Standard Model of Particle Physics [49]. However, a number of questions remain unanswered.

A principal concern is the lack of any direct evidence for the Higgs Boson[50], the particle resulting from the Higgs mechanism should provide an explanation for the masses of elementary particles. Other questions include uncertainties in the mathematical behavior of the Standard Model at high energies, the lack of any particle physics explanation for dark matter and the reasons for the imbalance of matter and antimatter observed in the Universe. The LHC and the associated experiments are designed to address a number of these questions.

Four main experiments benefit of the high energy collisions delivered by the LHC. A Toroidal LHC Apparatus (Atlas)[52] and Compact Muon Solenoid (CMS)[53] are large multi-purpose experiments which will collect data from proton-proton (p-p) collisions; LHC beauty (LHCb)[54] is an experiment dedicated to the study of Charge Parity (CP) violation and rare decays in the range of b-quark physics arising from p-p collisions; A large Ion Collider Experiment (Alice)[55] will analyze data produced by heavy ions collisions.

To satisfy their computing and storage requirements, the Grid approach is being used, by means of the WLCG project already described in 1.5, for interconnecting the regional computing centers.

In this chapter, CMS computing environment is presented, introducing briefly the LHC accelerator and the CMS experiment. The architectures and roles of most important CMS data management/workload management frameworks are also described.

2.1 LHC and CMS experiment

The LHC is the largest and most powerful particle accelerator of the world. LHC is a proton-proton and ion-ion collider, based at CERN across the Swiss-French border. It is being built into the tunnel of the former LEP accelerator located about 100 m underground, so that most of the infrastructure could be recycled and only small civil engineering modifications were necessary. Figure 2.1 shows geographically the 4 LHC experiments location. The LHC will be able to accelerate two beams of protons in opposite direction in a 27 km long ring up to beam energy of 14 TeV (1 TeV = 10^{12} of electron volt). The LHC project proposal was approved in December 1994 and achieved the first proton-proton collisions in November 2009 with a centre of mass energies of 0.9 and 2.36 TeV. During 2010, the LHC delivered proton collisions with a centre of mass energy of 7TeV.

The main goals of the experiment are:

- to explore physics at the TeV scale;
- to discover the Higgs Boson;
- to look for evidence of physics beyond the standard model, such as supersymmetry, or extra dimensions;
- to study aspects of heavy ion collisions.

The CMS experiment is one of the two experiments aimed to a wide range of physics results which started operation in 2009. The scientific analysis of data taken by the detector and Monte Carlo (MC)[58] events simulation require

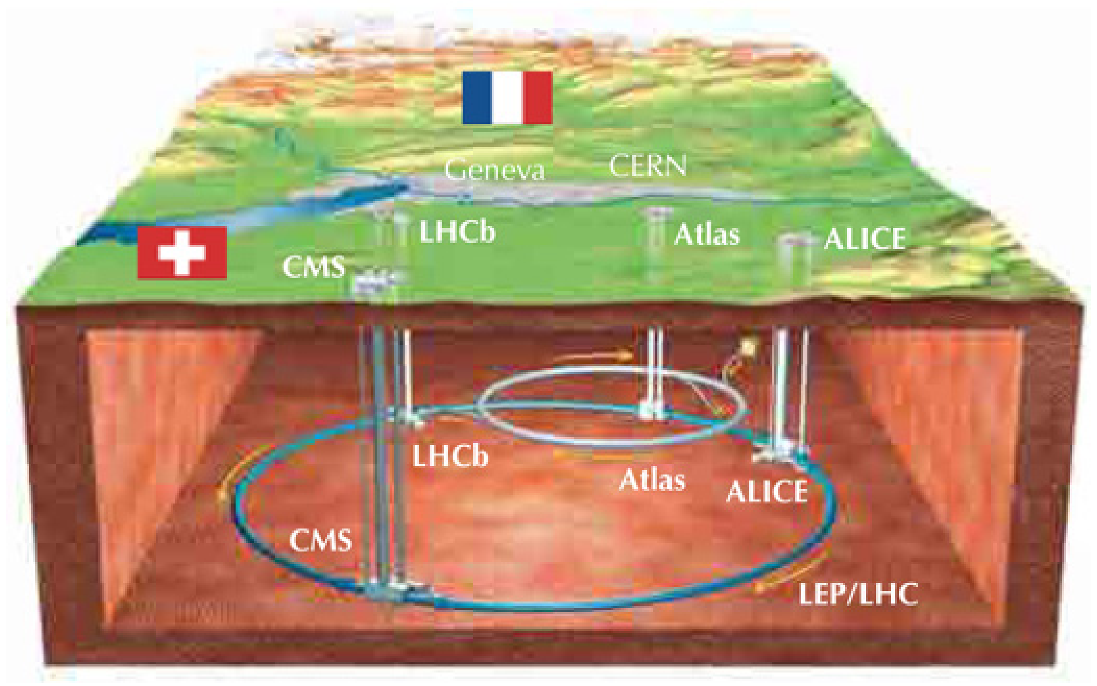


Figure 2.1: LHC experiments

a large amount of well organized computing resources, to guarantee more than 3000 CMS collaborators located in 40 countries around the world to carry out their physics analysis with minimal geographical and processing constraints. The CMS experiment has a worldwide distributed computing model based on Grid technology.

The CMS detector (Figure 2.2) at the LHC is a very complex apparatus with more than 70 million acquisition channels. The experimental apparatus contains sub-detectors capable of measuring the energy and momentum of photons, electrons, muons and other particles created in the collisions. It is geometrically divided into several cylindrical layers coaxial to the beam direction, referred to as barrel layers, closed at both ends by detectors disks orthogonal to the beam pipe, the endcaps, to ensure detector hermeticity. The innermost layer is a silicon-based tracker. Surrounding it is a scintillating crystal electromagnetic calorimeter, which is itself surrounded with a sampling calorimeter for hadrons. The tracker and the calorimetry are compact enough to fit inside the CMS solenoid which generates a powerful magnetic field of 4 Tesla. Outside the magnet are the large muon detectors, which are inside the return yoke of the magnet. The entire experimental setup of CMS has a diameter of 15 m, is 21 m long and has a total weight of over 12,000 tons.

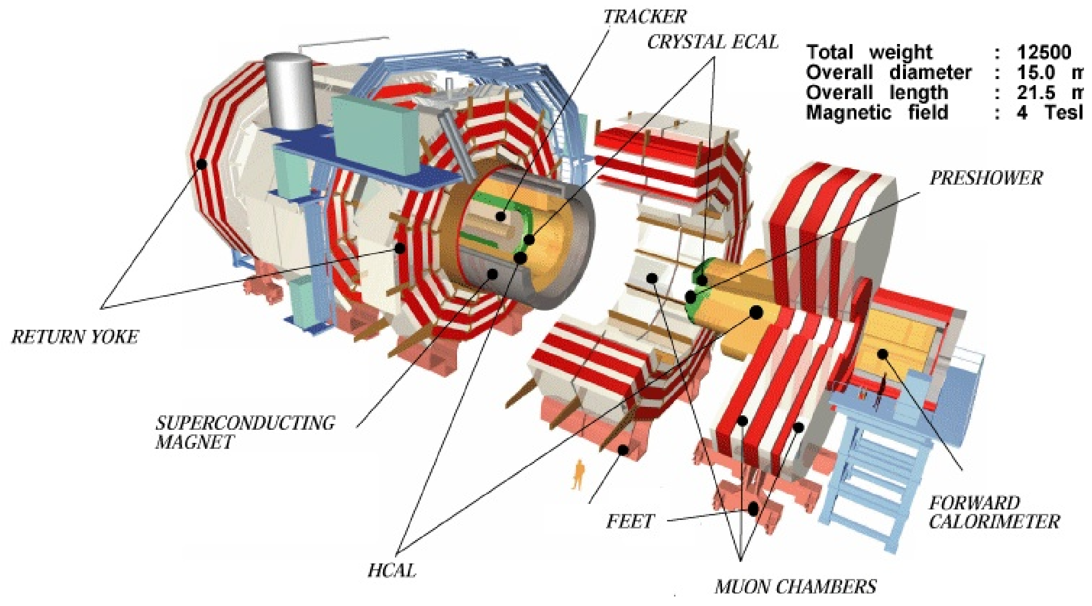


Figure 2.2: CMS detector

At a rate of 40,000,000 times per second, the LHC accelerator lets two bunches of particles coming from opposite directions cross through each other inside the CMS detector. In every bunch crossing in the detector, on average 20 collisions occur between two protons from opposite bunches. The collision phenomena that occurs during a single bunch crossing are called an event.

CMS uses a number of event data formats with varying degrees of detail, size, and refinement. The first level (L1) of the CMS trigger system runs synchronously with the LHC bunch crossing frequency of 40MHz. It is designed to reduce the incoming data rate to a maximum of 100 kHz by selecting events with interesting signatures. This is further reduced by a factor of 1000 using a high-level trigger (HLT) to produce RAW data. The event size of RAW event is nearly 1 MB. Each year, there is approximately 100 days ($\sim 10^7$ seconds) of data taken. So, at 100 Mb/s of RAW data production rate, $\sim$ 1 PB of events are taken and processed.

RAW events are processed at CERN, creating new higher level physics objects, RECO events. AOD are derived from the RECO information to provide data for physics analysis in a convenient and compact format.

2.2 The CMS Computing Model

The computing environment of the CMS experiment at the LHC has been constructed as a distributed system of computing resources and services relying on Grid technologies. A hierarchy of computing Tiers has been proposed since the Models of Networked Analysis at Regional Centers for the LHC Experiments (MONARC)[56] model, yielding to a structured set of resources consisting of a Tier-0 center plus the CERN Analysis Facility (CAF)[57] located at CERN, 7 Tier-1 centers located at large regional computing sites, and about 50 Tier-2 centers plus few Tier-3 centers.

The resources, made available to CMS community in the LHC tiered centres, comprise mainly the computing and storage systems that CMS uses to perform data processing, data archiving, Monte Carlo event simulations, physics analysis and all computing-related activities. By following such approach, CMS not only gains access to the valuable resources and expertise which exist at collaborating institutes, but also benefits from improvements in robustness and data security, through redundancy amongst multiple centres.

2.2.1 Tier-0 centre

The Tier-0 is by definition a purely production facility, dedicated to the storage and reconstruction of RAW data. Reconstruction is the process of reconstructing data that originates from the collisions in the detector to provide access to reconstructed physics objects for physics analysis in a convenient format. During CMS data-taking periods, the centre accepts and buffers data from the online systems at the CMS experimental site. To release buffer space to the online system, a temporary copy of the data is immediately made at the Tier-0. After generation of corresponding RECO data, the RAW and RECO components are stored together as FEVT data to simplicity the later access. The FEVT is copied to permanent mass storage in both the Tier-0 and a Tier-1 centre.

CMS requires that the scheduling and completion of reconstruction tasks on the Tier-0 resources are independent of other CMS or non-CMS activities at CERN, and for this reason the Tier-0 is not sharing computing resources with any other activity.

The final role of the Tier-0 is the distribution of FEVT data to external Tier-1 centres. Reliable distribution of FEVT for both data safety and analysis is among the highest priority tasks for the experiment, and a robust mechanism is employed to ensure that all data is copied to at least two secure archives before being deleted from the buffers at the Tier-0.

The Tier-0 operations are subject to Quality of Service requirements. During data taking periods, Tier-0 is operating 24/7 by means of a fail-over architecture to ensure this.

The Tier-0 does not directly provide user-oriented services to CMS collaborators. In the aim to allow the later analysis of CMS data, it provides the following high-level services:

- Acceptance of RAW data: The RAW data is transferred, using CMS tools, from the online systems with guaranteed integrity and low latency.

- Reconstruction of RAW data: The centre performs in pseudo-real time the first pass reconstruction of the RAW data, stream the output into physics datasets and secures both RAW and RECO data onto tape as FEVT format. The reconstruction process problems are detected quickly by the Data Quality Monitoring (DQM) system. Reconstruction jobs are steered using a batch system.

- Mass storage: The mass storage system which accepts FEVT data from the Tier-0 facility provides guaranteed throughput for writing new files.

- Distribution of RAW data: Reliable transfer of the FEVT data is ensured to external Tier-1 centres, with guaranteed bandwidth available to avoid significant backlog or latency.

- Prioritisation: CMS priorities are implemented in the Tier-0 for the reconstruction and data distribution.

2.2.2 Tier-1 centres

The Tier-0 distributes RAW and processed data to a set of large Tier-1 centres located in France, Germany, Italy, Spain, Taiwan, the United Kingdom and the United States. The CMS Tier-1 centres, based at large sites around the world, provide a wide range of high-throughput high-reliability computing services for the collaboration.

The tasks carried out at Tier-1 centres principally relate to organised processing of data and extraction of datasets to be further analysed at Tier-2 centres. A given Tier-1 centre may have the only available copy of some data samples, and must therefore allow access to CMS users in accordance with the policies and priorities set by the collaboration. Different priorities can be assigned by the collaboration to users, or groups of users.

Most CMS users use a Tier-2 or a local Tier-3 centre to perform their physics analysis, although some special types of analysis, like the ones related to calibration work, or requiring very large statistics, are carried out at Tier-1 centres. Tier-1 centres accept also data from Tier-2s. This includes simulated data samples generated at Tier-2 and Tier-3 sites.

Tier-1 centres available to CMS are shared between experiments, and thus many WLCG services at the centres are common between activities. However, each Tier-1 centre must provide specific services to CMS collaboration when required.

A crucial function of Tier-1 centres is to provide a large portion of RAW and simulated event data storage. CMS makes two custodial copies of all RAW event data. The first will be stored at the CERN Tier-0 centre, and the second distributed between Tier-1 centres. Both of these copies can be used for reprocessing when required. The acceptance of a portion of the RAW data by a Tier-1 site implies that the centre undertakes to be responsible of damage that can occur to this data in the long term. This involves ensuring that the underlying mass storage system is protected against risks such as hardware or media failures, environmental hazards, or breaches of security.

In addition to storing CMS data, the Tier-1 centres take responsibility for distributing data, providing suitable high-performance I/O and network infrastructure. The Tier-1 centres serve FEVT data to Tier-2 centres for analysis, and Tier-1 centres for replication, following the priorities set by CMS. Connections between Tier-0 and Tier-1, and between Tier-1s, forms the backbone of the CMS data distribution system.

Each Tier-1 centre has the following roles in CMS:

- Securing and making available to the collaboration the second copy of the RAW data and RECO data.
- Securing and distributing Monte Carlo simulated samples produced in the Tier-2.
- Participating, with the Tier-0, to the calibration and providing feedbacks to the running experiment.
- Receiving and making available a copy of the full CMS Analysis Object Data (AOD) dataset.
- provide computing resources for further re-processing (re-reconstruction, skimming, etc) of Monte Carlo samples and datasets, and for high priority analysis as well.
- Serving datasets to the Tier-2 and other regional or institute computing facilities.

2.2.3 Tier-2 centres

Each Tier-2 centre provides CMS with a flexible resource with large processing power, but with looser storage availability and connectivity than required in a Tier-1. CMS operates more than 40 Tier-2 sites typically located at universities and research institutes in 22 countries. Tier-2 centers are a very important resources for CMS computing since they form together more than 50% of the total CMS computing power and around 40% of the global collaboration disk storage capacity.

The basic functions supported by a Tier-2 include:

- Fast and detailed Monte Carlo event generation.

- Data processing for physics analysis.
- Data processing for calibration and alignment tasks, and detector studies.

The storage capacity provided by Tier-2s to the collaboration should be big enough to handle the datasets needed for the user analysis and a smaller fraction for the simulated data. Medium or long term storage of required data samples is needed. These will be mostly AOD, with some fraction of RECO. RAW data may be required for calibration and detector studies.

Moreover, The Tier-2 sites should guaranty the management of the local working areas for analysis results, plus, individual users spaces. Remote and local batch job submission is also supported by this Tier level with prioritization mechanisms configured for accessing resources by remote and local users, and this in accordance with both CMS and local policies.

Given the important role of data in the CMS computing model, the whole design is very dependent on the way data is transferred. An efficient and flexible data transfer system, named PhEDEx [64], has been built by the computing project. This system will be described later in this chapter.

2.2.4 Tier-3 centres

The Tier-3 level is special by mean that it is not a pledged resource of the experiment, but rather voluntarily provided to CMS. A Tier-2 must have sufficient CPU and disk space to produce Monte Carlo simulations and to support CMS analysis activities, while a Tier-3 does not have these requirements. CMS can use Tier-3 resources, for opportunistic purposes, as best effort computing capacity for the collaboration where it cannot rely on their availability.

2.2.5 CMS-CAF

CAF plays a unique role for latency-critical functions at the source of the CMS analysis chain, with no dependence on the Grid infrastructure. Such workflows are based on a highly selected subset of the full production data, so called Calibration and Express Streams (ES), and are made available to restricted analysis groups soon after data taking occurrence. They can be classified, in order of priority, as follows:

- alignment and calibration;
- trigger and detector diagnostics, monitoring and performance analysis;
- physics monitoring, analysis of express stream, fast-turnaround high-priority analysis.

The key hardware components for addressing the low-latency and high-priority requirements are a large dedicated processing farm and a highly accessible disk-only storage system. The limitation in available resources is addressed via selectiveness in jobs scheduling and in data storage.

Dataflow between CMS Computing Centres is schematically illustrated in Figure 2.3.

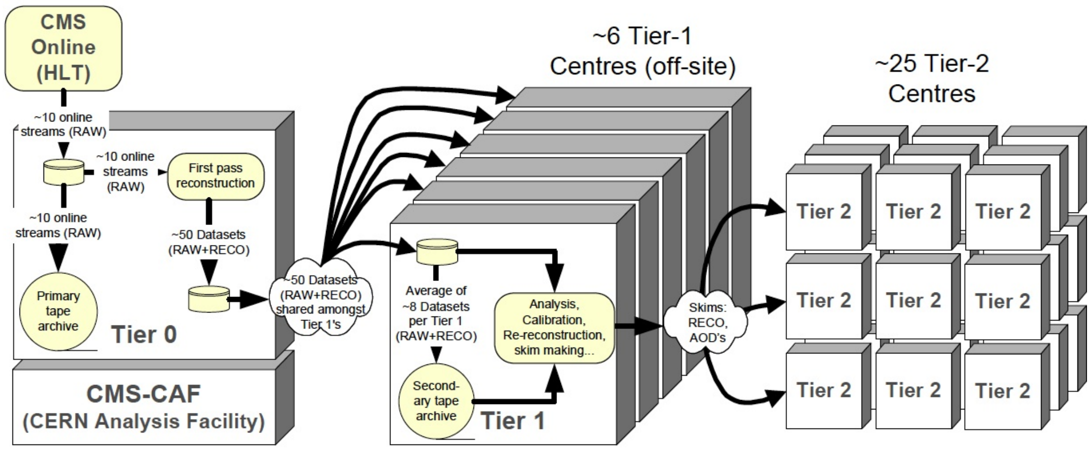

CMS Online (HLT)
~10 online streams (RAW)
~10 online streams (RAW)
First pass reconstruction
~10 online streams (RAW)
~50 Datasets (RAW+RECO)
Primary tape archive
Tier 0
~50 Datasets (RAW+RECO) shared amongst Tier 1's
CMS-CAF (CERN Analysis Facility)
~6 Tier-1 Centres (off-site)
Average of ~8 Datasets per Tier 1 (RAW+RECO)
Analysis, Calibration, Re-reconstruction, skim making...
Second-ary tape archive
Tier 1
Skims: RECO, AOD's
~25 Tier-2 Centres
Tier 2

Figure 2.3: Dataflow between CMS computing Centres in the LHC model

2.3 Framework for CMS Distributed Analysis

The CMS analysis model supports data location driven activities to minimize network occupancy. Data are distributed over many computing centers according to CMS data placement policies. They are processed at the computing centre where they reside, typically in Tier-2s. In order to enable distributed analysis, a set of Workload and Data Management tools have been designed in the aim to build specific CMS tools on top of existing Grid services.

2.3.1 Data Management

The CMS Data Management system provides a set of complementary tools to manage the large amount of data produced, processed and analysed in a distributed environment. These tools represent independent components building together the CMS Data Management system.

2.3.1.1 Dataset Bookkeeping System

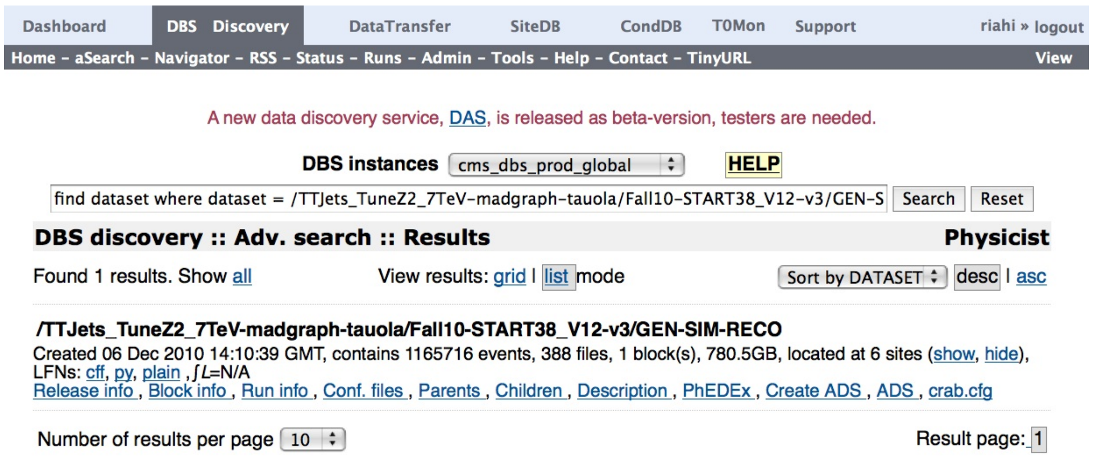
The Dataset Bookkeeping System (DBS)[60] provides the means to describe, discover and use CMS event data. The main features that DBS provides are:

- Data description: The DBS classifies attributes that characterise data such as run number, the algorithms and configurations used to process the data, and the type of content (RAW,RECO...). This description is specific to CMS data structures.

- Data Discovery: The DBS stores information about real and simulated CMS data in a queryable format. The results of DBS queries allow users to know what data is available and how it is organized logically in term of packaging units (files and file-blocks).

The use cases for DBS include Monte Carlo generation, detector data, large scale production processing and user data analysis. The DBS instances are usable fundamentally in two scopes: Global and Local. The Global scope is a single DBS instance available to all CMS. It is the official source of data information for the collaboration. Regarding the Local scope DBS, there are more than one instance, installed by users, to describe data produced by MC production. In general, it is possible to migrate data, which can interest the entire collaboration, from the Local scope instance to the Global one.

The DBS is used for data discovery and job configuration by production and analysis systems. Information about the data being produced and processed is registered into DBS by the production system. Files resulting from users analysis can also be published into DBS in the aim to make them available to other CMS users. Details of data published are available to users by means of DBS Web interface, or Command Line through DBS API. Figure 2.4 shows the discovery result of a dataset using the DBS Web interface.


Dashboard
DBS Discovery
DataTransfer
SiteDB
CondDB
T0Mon
Support
riahi » logout
Home - aSearch - Navigator - RSS - Status - Runs - Admin - Tools - Help - Contact - TinyURL
View
HELP
Search
Reset

A new data discovery service, DAS, is released as beta-version, testers are needed.

DBS instances cms_dbs_prod_global

find dataset where dataset = /TTJets_TuneZ2_7TeV-madgraph-tauola/Fall10-START38_V12-v3/GEN-S

DBS discovery :: Adv. search :: Results — Physicist

Found 1 results. Show all — View results: grid | list mode — Sort by DATASET — desc | asc

/TTJets_TuneZ2_7TeV-madgraph-tauola/Fall10-START38_V12-v3/GEN-SIM-RECO
Created 06 Dec 2010 14:10:39 GMT, contains 1165716 events, 388 files, 1 block(s), 780.5GB, located at 6 sites (show, hide),
LFNs: cff, py, plain , ∫L=N/A
Release info , Block info , Run info , Conf. files , Parents , Children , Description , PhEDEx , Create ADS , ADS , crab.cfg

Number of results per page 10 — Result page: 1

Figure 2.4: A discovery result using the DBS Web interface

The DBS system is a Web based application, designed in a modular approach to make it in particular easily adaptable to multiple database technologies (ORACLE[59], MySQL[61], and SQLite[62]). This enables DBS to be deployed in a range of environments from large installations to specific personal installations. Given the large number of users accessing the Global scope DBS, this instance was deployed in a multi-tier architecture, hosted in the CERN Oracle RAC (Real Application Cluster)[59] at CERN. The Local scope DBS instances are installed at CERN and at other sites for private use. The DBS interacts with the Client by means of HTTP payloads exchanged in XML format.

2.3.1.2 Local Data Catalogue

A CMS applications are aware only of the logical files distributed across sites and relies on a site local catalogue service to access to the physical files. Every

CMS site has a Trivial File Catalogue, written in Extensible Markup Language (XML)[77] format, installed to look up a Physical File Name (PFN) from a Logical File Name (LFN) based on site-specific physical file paths. The input of such service consists on the LFN and the access protocol.

2.3.1.3 Conditions data

The CMS Experiment is using a multi-tier Web approach to deliver the alignments and calibrations of the detector, known as conditions data, to a worldwide community of distributed processing and analysis Clients. Since the same condition data needs to be accessed by many jobs running worldwide, a caching technics, in the site of the processing activity, is deployed in order to gain in term of performance. The CMS system, which provides conditions data, is called Frontier and it is composed of:

- Frontier Client converts database queries into HTTP and sends it over the network to Frontier server.
- Frontier server looks up the results in a central database at CERN. The results are then packed into an HTTP formatted stream and sent back to the Client.
- Squid proxy/caching servers, between the Frontier server and Client, are deployed at each site in which conditions data details are needed.

2.3.1.4 Physics Experiment Data Export

In order to meet the data distribution requirements of the CMS experiment at the LHC, the Physics Experiment Data Export (PhEDEx) project was designed to facilitate and manage global data transfers over the WLCG Grid. Since its conception in 2004, it evolved considerably and has shown to be a robust, reliable, and scalable system.

In the period of the first realistic test of the CMS Computing infrastructure, namely the Computing, Software and Analysis Challenge that took place in the fall of 2006 (CSA06), Phedex has shown that it is able to support the transfer of large amount of data. It succeeded to transfer $\sim$ 50 TB of data in that period without major bottlenecks. Another important CMS challenge was the Common Computing Readiness Challenge of 2008 (CCRC08) phase, an LHC-wide computing challenge. In this period, Phedex has transferred more than 160 TB per month, which demonstrated again that this tool is able to scale. Since the beginning of the 2010 LHC physics run in March, it has guided CMS data transfers with peaks in global speed between all sites exceeding 2.8 GB/s, distributing 21 PB of replicas over all sites. Figure 2.5 shows the average daily transfer volume per month of data managed by PhEDEx from mid-2004, when the project began. Reliability is ensured by the consistency check of each file transferred, while robustness is achieved by intelligently backing off when large numbers of failures are encountered.

PhEDEx provides a simple Web interface to define, execute decision of, and monitor data movement. The PhEDEx Web interface provides to users a full picture of data movement in particular the details of on-going transfers, data location, which copies are custodial, to which physics analysis group the data is allocated for, and transfers rates.

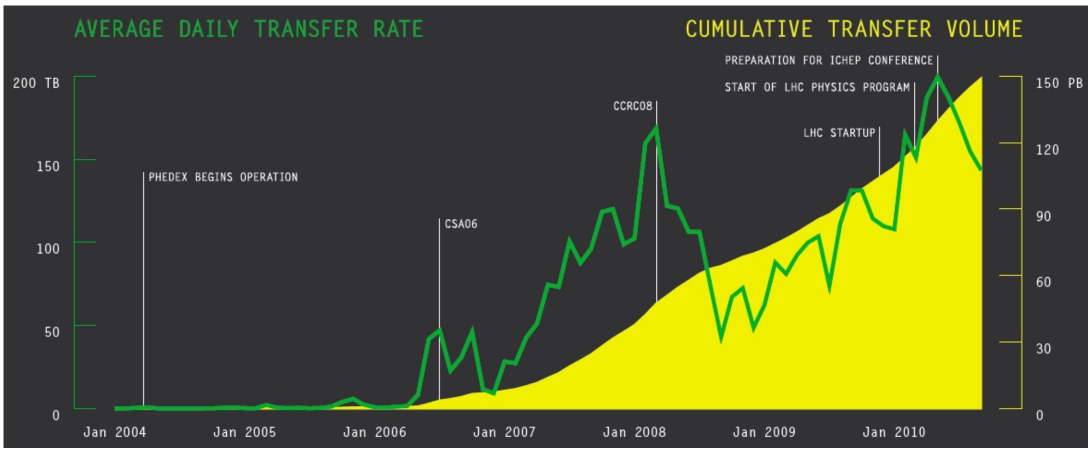


Figure 2.5: Average daily transfer volume per month and the cumulative transfer volume of data managed by PhEDEx from mid-2004

In PhEDEx, CMS sites or disk/tape area within a site are represented by nodes. Links between nodes define the transfer topology. The workflow begins when user makes a transfer request via a simple Web form indicating the destination nodes, the datasets he wishes to transfer and his CMS group. The node's Data Manager receives a notification mail in order to make decision by approving or refusing the request. Once approved, the optimal source node among the available file replicas of the request is determined. To achieve this, PhEDEx uses Dijkstras algorithm)[92] to calculate the path of least cost, where cost of transfer for each link is determined by the recent transfer rate and the size of the current queue over that link. Applying such method, the transfers performance are improved by balancing the load among multiple sources when the same data is requested to multiple nodes. Within this infrastructure, data transfers are also fault-tolerant by switching to another link when one link fails to perform the transfer.

PhEDEx software is based on agents storing their states and communicating via a central black board database, known as the Transfer Management DataBase (TMDB), hosted in the CERN Oracle RAC. A set of service agents run centrally at CERN managing the intelligence of data routing and transfer task creation. The download agents running at the site receive these tasks and initiate the data transfers using a technology-specific backend to the site, interacting with the Grid middleware. However, the typical download agent submits jobs to File Transfer System (FTS)[79], an underlying Phedex service, and manages therefore the transfers using FTS backend. After a transfer attempt, the file is verified using a site-specific script, and the result of that verification is used to indicate success or failure of the transfer. Files deletions is also provided by agents running at the site, in a similar manner of file transfers but with less complication since file deletion operation involves only one site. The Web data service provided by Phedex makes its interaction with other CMS computing components possible by formatting the results got from the TMDB in XML or JavaScript Object Notation (JSON)[87] data. Phedex keeps track of data locations. CMS processing tools rely on its Web data service to get in particular the location of the data when submitting jobs.

2.3.2 Workload Management

The role of CMS Workload Managed system is to manage the user processing requests in a local or distributed environment hiding the complexity of interactions with the Grid and CMS services. It includes requests interpretation, jobs creations and submissions, jobs executions, and jobs monitoring. The Production Agent (ProdAgent)[65] is the tool developed to perform this within Monte Carlo, reconstruction and skimming jobs on Tier-1 sites, while CMS Remote Analysis Builder (CRAB)[66] tool is optimized to do the same within user analysis jobs in a more distributed environment composed mainly by CMS Tier-2s.

2.3.2.1 ProdAgent

The ProdAgent software has been initially developed as a tool for Monte Carlo production. It has become the primary tool for all CMS's organized event processing.

ProdAgent is designed in a modular approach in order to be extensible. Scheduler and Storage specific interfaces are implemented in specific plug-ins, loaded at run-time. It works with a variety of batch systems, Grid schedulers, multiple storage systems and a number of job and workflow types. It aims to automate as much as possible the work done on behalf of users by reducing the human effort required to operate it.

The ProdAgent creates jobs and sends them to run at sites using the configured submission method. When running, these jobs interact with only few sites services, namely sites SEs and conditions data cache.

In general, each job produces in site's SE data in unsuitable size for the long-term storage on tape. To address that, the product of several jobs are merged together through subsequent merge jobs. The intermediate data products are deleted therefore by another subsequent jobs. Logs are copied into site storage for further inspection by experts. They are removed and achieved later on tape at CERN. Once data has been produced, it is registered into DBS and Phedex. At this point, it can then be transferred to become available for analysis at CMS Tier-2 sites.

The production system architecture consists of three components:

- The Request system (ProdRequest), acting as a frontend application for production request submitted by users into the production system.
- The Production Manager (ProdManager) will contact ProdRequest to obtain the requested work. It will take care therefore to distribute it between ProdAgent instances for execution.
- A ProdAgent itself is composed of autonomous components, written in Python[83], interacting together by means of asynchronous and persistent messaging system using MySQL database back-end. Components are event driven and each one has a specific role. When receiving an event, a component carries out its work and may, in turn, trigger actions in other components by sending messages.

The most important concept in ProdAgent is a workflow. It specifies the steps, that must be executed to complete a production or processing request, and their configurations. Workflows are created from a CMS physics SoftWare (CMSSW) configuration and request specific details such as detector conditions, and may contain multiple processing steps. Once a workflow is given to a ProdAgent, a set of independent jobs are created and then submitted to the Grid if the required resources are available. If the data merging is required further jobs are created and submitted to perform this work. The workflow is complete when all jobs have finished and the outputs datasets have been produced and registered into DBS and Phedex.

2.3.2.2 CRAB

CRAB was the first analysis tool in CMS which interfaces physics analysis operations with the LHC Grid infrastructure. It helps users in configuring CMS applications for discovering the location of remote datasets, submitting and executing the analysis jobs.

CRAB has progressed from a limited initial prototype (nearly 5 years ago), named CRAB 1, to a fully validated system, named CRAB 2, that is, actually, the reference framework for CMS data analysis.

- The Analysis Workflow:

  CMS data are organized in large datasets; their locations drive the CMS Analysis activities. The usage of Tier-2s resources, such as computing power, network and storage, is optimized by load balancing data, and consequently also the analysis activities is balanced between sites. Data transfers to Tier-2s are performed basing on the geographical aspects of sites and also Physics Group interests.

  The typical analysis workflow executed by users is:

  - Analysis program is developed by the physicist and tested locally on a small amount of data;
  - In general, there is more than one replica of a dataset. Thus, the user can decide on which location of datasets he wants to run his code. This allows users to choose the closest Tier-2 facility;
  - User's code is shipped to the destination site for execution;
  - Results are made available to users for further processing.

  The analysis scenario described above requires knowledges of CMS services, such as Phedex to know where data is located or the CMS Sites information (siteDB)[67] which provides the details to allow the interactions with remote storage systems. Grid technology knowledges are also needed, as the Job Description language (JDL), to define the analysis task. CRAB has been developed to address this complexity.

  CRAB translates the analysis workflow into Task and Job objects. A Job is a piece of work, requiring Grid computing power for execution. Task is a set of Jobs, that have the same configuration and running over different pieces of data.

- Functionalities:

  CRAB needs to be installed on an UI. It supports CMSSW based analysis programs. The followings are the functionalities that CRAB provides:

  - Data discovery and location: This is achieved querying specific CMS services, respectively DBS and Phedex, to find how and where data can be accessed.
  - Job preparation: Pack local user code and the environment to be sent for execution in the remote site.
  - Job splitting: A set of data splitting algorithms, such as split by event or split by files, are available to users. Thus, a physicist can choose the suitable one.
  - Job submission: Submit job to remote sites hosting the required data.
  - Job monitoring: Monitor the status of the submitted jobs by querying Grid services.
  - Handling output data: Copy produced outputs to the remote Tier-2 specified by the user or return them back to him. In general, users specify the associated Tier-2 as destinations of their outputs.
  - Publish data: User can publish their outputs, in case that interest the entire collaboration or a particular group, into DBS for further analysis.

- Design:

  CRAB is coded in Python. The interface to the Grid middlewares and local batch systems is provided by BossLite[70], a Python library developed to interface the Workload Management tools to the Grid middleware and local batch systems. It relies on a relational database to track and store information about the user requested task. For extensibility purposes, the standard operations in BossLite, such as job submission, tracking, cancellation and output retrieval, are performed via a generic abstract interface. Scheduler (batch system or Grid) specific interfaces are implemented in plug-ins, loaded at run-time. Presently, plug-ins for several Grid middleware such as gLite, OSG, and ARC are implemented. In addition, batch systems plug-ins such as LSF and PBS are provided.

CRAB 1 was based on a standalone approach, requiring the user intervention for all operations of interaction with the Grid. After a long experience using the standalone CRAB setup, the natural evolution was a more efficient and modular approach based on a Client-Server architecture. The guidelines of this evolution, from CRAB 1 to CRAB 2, were aimed to keep CRAB interface, which is already familiar to physicists, invariant and reduce as much as possible the user intervention on the management of their jobs.

The Client is a command line application installed on the UI. When invoking the task creation command, CRAB takes care of the local environment interaction, plus packages private user library and code, in the aim to be able to replicate the same local configuration in the remote WN where the jobs will run; it performs the data discovery by means of the DBS and locates data through Phedex; It splits Jobs over data discovered using the algorithm selected by user. At this point, user can invoke the submit command. If the server mode is selected by the users in the configuration, the Client interacts with the server using the Web service technology, Simple Object Access Protocol (SOAP)[71], to delegate the task management to it. Otherwise, the task is submitted directly to the Grid. The Client uses SQLite database for logging purpose. The interaction with the database is performed using BossLite API.

The CRABServer implements a modular architecture, based on independent agents communicating asynchronously through a persistent messaging system, based on MySQL database. In such architecture, each component achieves a specific role. The whole CRABServer functionalities can be extended by adding easily and in a transparent way a new component. The architecture of CRABServer and the interactions between CRAB components are shown in Figure 2.6. In order to manage many tasks in parallel, most of the component implement a multithreading approach, using safe sessions and connection pools. CRABServer relies on an external storage server to host users inputs and outputs data. It interacts with that service using the plugin-based storage API (SEAPI), library hiding the complexities to interact with Grid SEs.

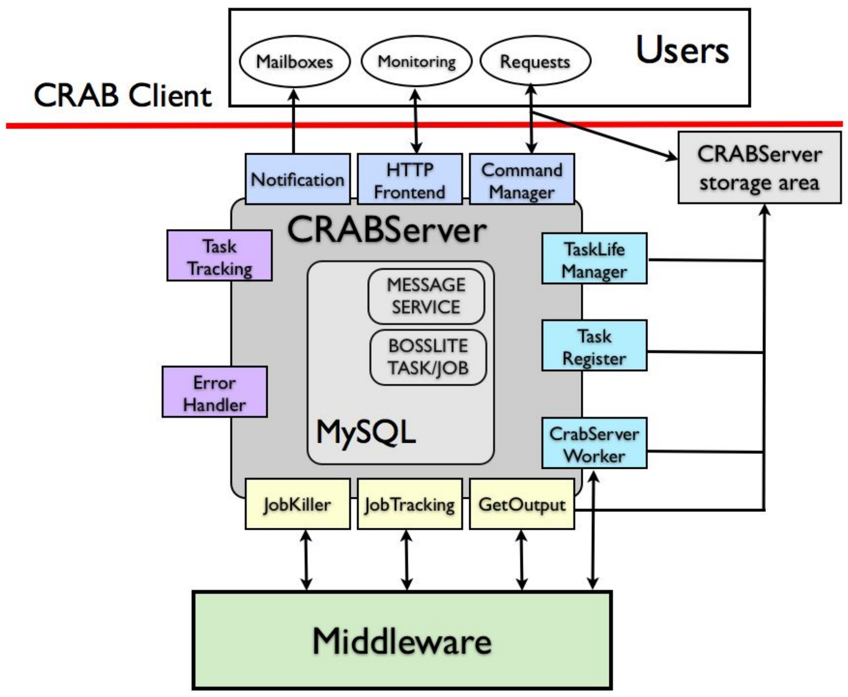


Figure 2.6: Interactions between CRAB components and the architecture of CRABServer

2.3.2.3 CMS Dashboard

The implementation of CMS Dashboard[72] is based on a database running on Oracle server, a set of information collectors feeding from various sources, and Web interfaces providing access with different levels of detail and aggregation. The job wrappers generated by the job submission tools of the CMS are instrumented to report meta-information about the user's task and user's job at the worker node to a central collector. Within the CMS Dashboard Web interface, it is possible to navigate from one view to another. Such feature allows a fast access to the desired information. A set of monitoring applications are provided to CMS users, including:

- History view: It presents the history in time of relevant metrics to highlight overall trends of CMS computing infrastructure in term of jobs.
- Task monitoring for analysis user: It exposes a user-centric set of information regarding submitted tasks. It shows analysis tasks details, such as reason of failure, distribution by site and over time, consumed time and efficiency. Figure 2.7 shows graphical views of a given task status as provided by Task monitoring application of CMS dashboard.
- Interactive interface : It is a job-centric view giving details to understand and debug current jobs issues.

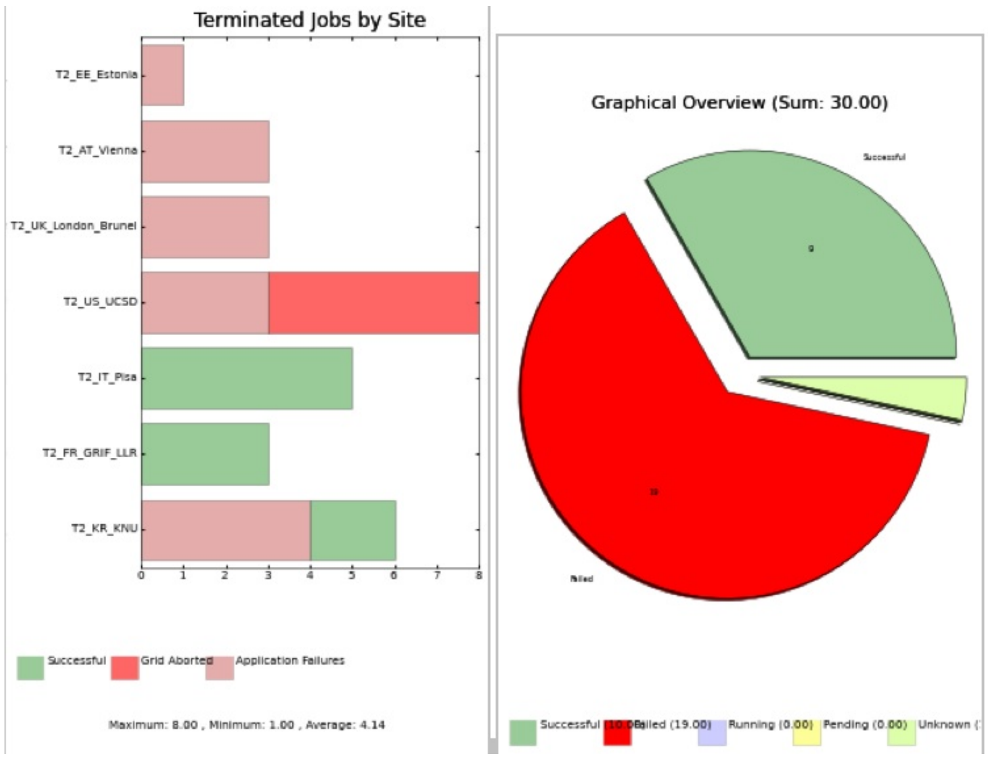


Figure 2.7: Graphical views of a given task status as provided by the Task monitoring application of the CMS dashbaord

2.4 Analysis Organization at CMS Sites

Physicists are organized in Physics Groups each with its own internal organization. Each Physics Group is associated to one or more Tier-2 which takes care to provide it the storage space, processing power and support. Depending on its size, a Tier-2 can be associated to one or more Physics Group. The Tier-2s are tasked to transfer and then store data needed by the associated groups. The groups leaders decide which data should be hosted in the Tier-2 for their community. Additionally, each user in CMS has an assigned Tier-2 where he has 1 TB of personal storage space. These assignments are based on geography, so a user may not have personal space at the Tier-2 where the official data he is using resides.

Within the CMS computing model, Tier-2 sites are the facilities dedicated to support the analysis activity. However, a few special analysis workflows are hosted at Tier-1s. Additionally to re-processing workflows, a set of analysis passes are performed promptly at Tier-1, as data arrive from the Tier-0 in case of RAW data and from the Tier-2 in case of simulated data. This step, known as skimming, is performed in agreement with the Physics Groups and aims to reduce the data in term of event size and number of events. The outputs produced by this step are made available to physicists at Tier-2s.

Normally only the data operation team, who takes care of data processing activity, has access to the Tier-1 but for some specific analysis activities some physicists need to access to large samples of RAW data, which cannot be hosted in Tier-2 sites. These activities include the detector calibration or the Data Quality Monitor validation. A special Grid credential has been created to grant access to the Tier-1 facilities to physicists achieving these activities.

Chapter 3

Study of CMS data analysis workflow performance

CMS data analysis is performed in a distributed environment. It is a complicated task since it assumes to know which data are available, where they are stored, and how to access them. In addition, it is needed to understand which resources are able to comply with user analysis requirements and interact with them if match. To achieve that, CRAB interacts with Grid and CMS dedicated services, and CMS computing infrastructure.

Physicists need to analyze available data promptly, which requires a minimal delay in the execution of the end-user analysis workflows, and in particular, they should be able to run over data once available at the various computing sites. From the computing system performance point of view, it is important that these workflows use as efficiently as possible the Grid resources. For this reason it is crucial to optimize data access and outputs management strategies in the current analysis workflow.

While CRAB tool and the Grid/CMS services are described in 2.3, in this chapter, the data analysis workflow using CRAB is detailed and the performance of its execution is analyzed.

3.1 Data analysis workflow in CMS

In order to simplify bulk data management, files are grouped together into file-blocks of a convenient size for data transfer. File-blocks are in turn grouped in datasets whose size is driven by physics. The tracking of data location is file-block based and it provides the name of sites hosting the data. So, when a dataset (or a set of blocks of the dataset) is transferred, the list of sites hosting the dataset is updated in Phedex only when the transfer of an entire block transfer is achieved. The file-block contains files that can be processed and analyzed together. The packaging of events into files is done so that the average file size is kept reasonably large (e.g. at least 1GB), in order to avoid scaling issues with storage and tape systems and to optimize data transfer. This is achieved merging small outputs files produced, in general in the Tier-1s, by individual jobs in fewer larger files.

As data is organized in large datasets, analysis activities are data location driven to minimize network occupancy. The usage of resources is optimized with the association of Tier 2s based on geography or Physics Groups of interest.

The typical end-user analysis workflow is performed as follows:

- An analysis program is developed and tested locally by the physicist on a small amount of events;
- The user prepares the CRAB configuration file where he specifies on which data he wants to run his code and the storage site, in general a CMS Tier-2, where he wants to have their outputs. He should make sure, by consulting Phedex or DBS, that the data is already available in a Tier-2;
- User creates their jobs by executing crab -create
- By the means of CRABServer, the crab -submit command ships user's code to the destination site where data is residing;
- User's code executes in a worker node;
- Outputs are copied from the worker node to the remote storage selected once the user's code execution completes successfully.

- If, for some reasons, the user's code fails when executing or the outputs could not be copied to the final storage element, CRABServer retries four times the resubmission of the job before marking it as failed. End-users should resubmit failed jobs manually for further retries.

Figure 3.1 shows the job flow and data flow in the typical analysis workflow.

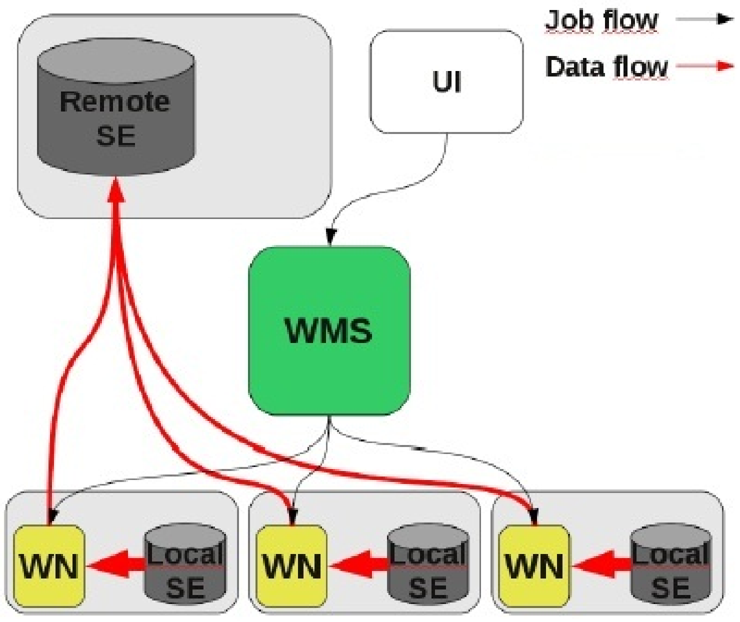


Figure 3.1: Job flow and data flow in the typical analysis workflow.

3.2 Limits of the current Workload management strategies

Within the current CMS analysis workflow, data access and stage-out strategies have played the key role in the bad performance, noticed during 2010 and 2011, within the analysis workflow.

3.2.1 Data access

The Tier-0 centre at CERN accepts data from the CMS online system, archives the data, performs prompt first pass reconstruction. Special data are then injected into Phedex and transferred to CAF to run over them the time-critical end-user analysis workflows, named prompt alignment and calibration. Reconstructed data at the Tier-0 together with the corresponding RAW data are distributed to Tier-1s which take custodial responsibility. At the Tier-1 centres data is re-processed based on improved detector conditions and selected on the interesting physics channels. The analysis datasets resulting are then transferred to Tier-2s using Phedex, basing on Physics Groups associated, for the end-user analysis. User can also make a Phedex request to transfer analysis data from a Tier-1 or Tier-2 to the home computing site or associated Tier-2.

The followings represent main data access strategy limits in the current analysis workflow already described in the previous section:

- Data in transfer becomes available by file in the computing site and it is published as available by file-block in Phedex and DBS. So, users can submit their analysis workflows only when at least an entire file-block from requested data was transferred to the destination computing site. Depending on the continent and the Tier of sites involved in the transfer, such latency can vary from 1 hour to more than 41 days as shown in Figure 3.2.

- Users need to check the DBS, and configure, submit and track a new workflow at each time when new data from the dataset of interest is published available by the DBS.

Those limits make the execution of the user workflows possible only after a latency time (order of hours/days) since the data is available in the computing site.

3.2.2 User data output management

Currently, CRAB uses a synchronous stage-out approach. The stage-out operation is performed by the job wrapper on the WN once the output is produced

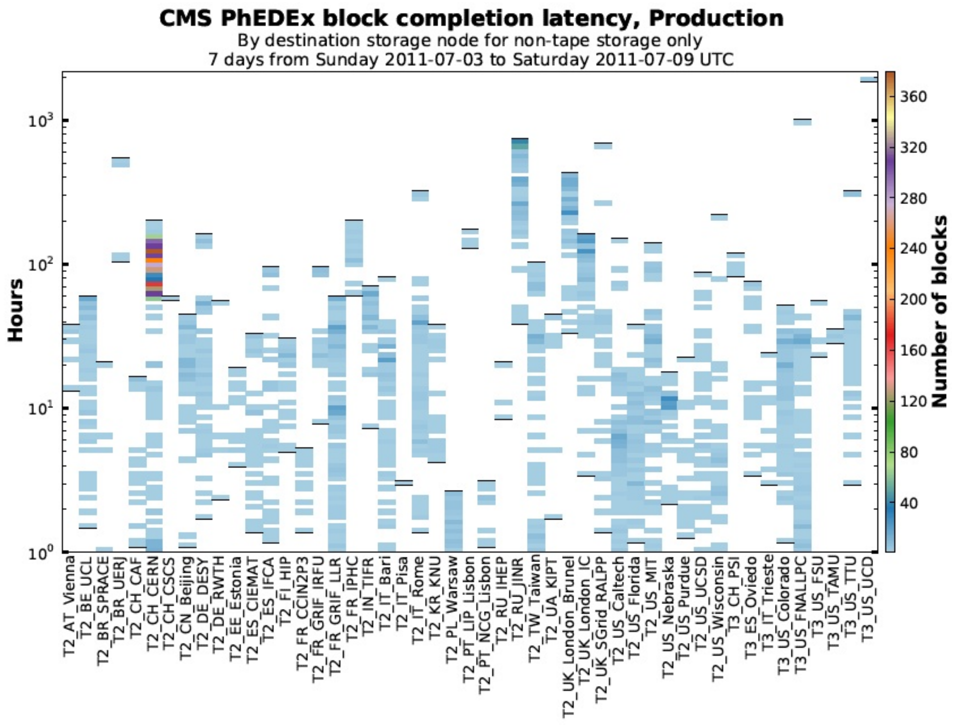


Figure 3.2: The data transfer latency using Phedex as a function of CMS site name for the week from 03-07-2011 to 09-07-2011.

locally. Actually, the job wrapper takes care to copy the output of the job once available in the WN to the remote SE. In case where the copy fails, the job wrapper tries the operation again after a short time, however, it is not efficient to occupy the CPU for a long period in attempting to copy the output. In general, it is unlikely that an infrastructure problem can be resolved in a short interval of time. If each attempt fails, the job wrapper continues its execution and clears the output of that job loosing it effectively. It is therefore needed to resubmit the job and the resources used by the last job are effectively wasted.

The remote stage-out approach used affects the efficiency with which the computing resources are exploited. Since the outputs can be copied only once the execution of job's code is finished, it is important that the copy process succeeds at the first attempt, to avoid having to resubmit the job and occupying further resources.

The failure of the stage-out may be due to network problems, the data storage system or authentication problems on the sites involved in the transfer of the output. Most are temporary problems, and another attempt at a different time might succeed. The failure of the stage-out is particularly inefficient since not only it occurs after the use of computing resources required to execute the job, including the queues of Grid and local schedulers, but also because the resubmission of the job is responsible for a delay in the completion of the task, forcing users to wait a longer time before being able to analyze their outputs.

For the CMS Tier-2s, this strategy has shown that results often in a kind of Deny of Service [82] attack to the Tier-2 storage systems where a huge number of WNs dispersed widely trying to stage-out there. So, it introduces a lot of issues to CMS Tier-2s, and in particular inefficient storage and network overload with long Gridftp queues to serve.

Such defect in the design of the analysis workflow causes often the failure of the users analysis jobs after reaching the timeout of the Grid copy command. Figure 3.3 shows in (a) the fraction of the jobs failed in 2010 and, among the jobs failed for application error, (b) shows the percentage of failure caused by the remote stage-out approach.

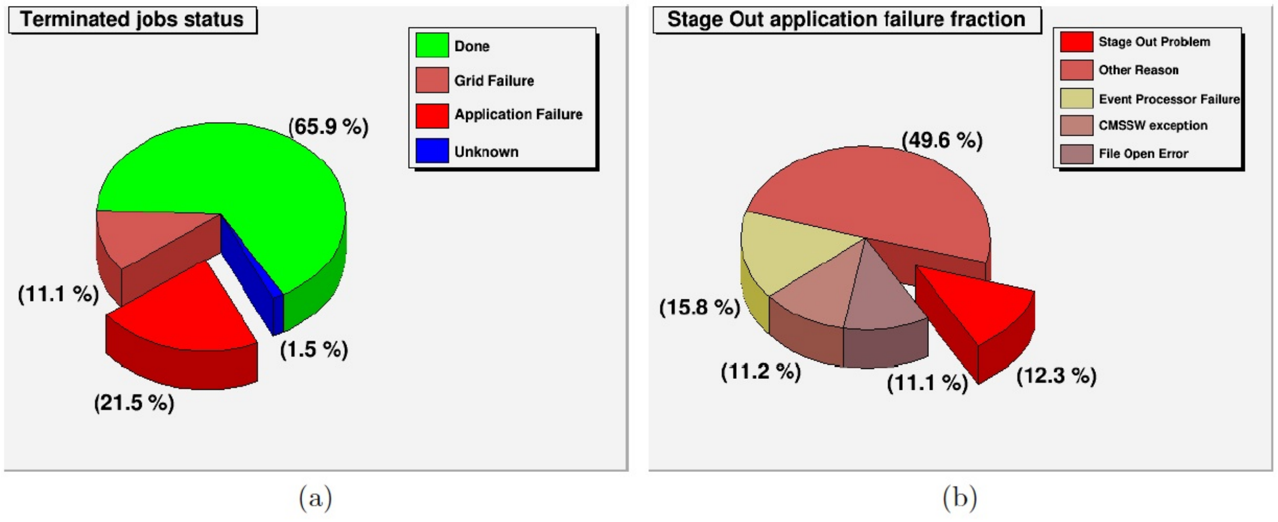


Figure 3.3: Fraction of the jobs failed in 2010 and, among the jobs failed for application error, the percentage of failure caused by the synchronous remote stage-out.

During July 2011, 4.766.902 analysis jobs have been submitted, where 4.067.588 has finished successfully and 14.391 has failed for the remote stage-out.

The distribution of the number of succeeded jobs over CPU wall-clock time intervals spent for the remote stage-out is shown in Figure 3.5. The 95% of succeeded jobs spent between 0 and 500 seconds in performing the remote stage-out which means that they consumed between 0 and ∼22.362 days of CPU wall-clock time and the collaboration had to wait that time to have results of their analysis jobs.

Regarding failed jobs, Figure 3.4 shows in (a) the distribution of the number of jobs failed for the remote stage-out over the CPU wall-clock time intervals spent and in (b) the distribution of the same number of failed jobs over CPU wall-clock time intervals spent for the remote stage-out. The manual or automatic resubmission of a failed job can happen only after the end of the execution of the current job. In (a), Figure 3.4 shows that 69% of jobs consumed between 0 - 20.000 seconds of CPU wall-clock time for execution. So, between 0 - ∼2.299 days of CPU wall-clock time were effectively wasted by ∼9.930 jobs. In addition, the collaboration had to wait at least that time before the resubmission of their jobs. In particular, Figure 3.4 shows in (b) that the remote stage-out has consumed, for 53% of jobs, between 0 - 500 seconds of CPU wall-clock time and, for 31% of jobs, between 3400 - 4000 seconds. This means that the remote stage-out step delayed the resubmission of analysis jobs submitted during July 2011 by 0 - ∼44 days for 7.626 jobs and by ∼175 - ∼206 days for 4.461 jobs .

As summary, the issues introduced by the synchronous stage-out strategy are:

- Delays in the execution of analysis workflows;
- Inefficient Tier-2s storage and network, overloaded by long GridFTP queues to serve;
- Unreliability of Tier-2s in the Grid;
- Resources wasting.

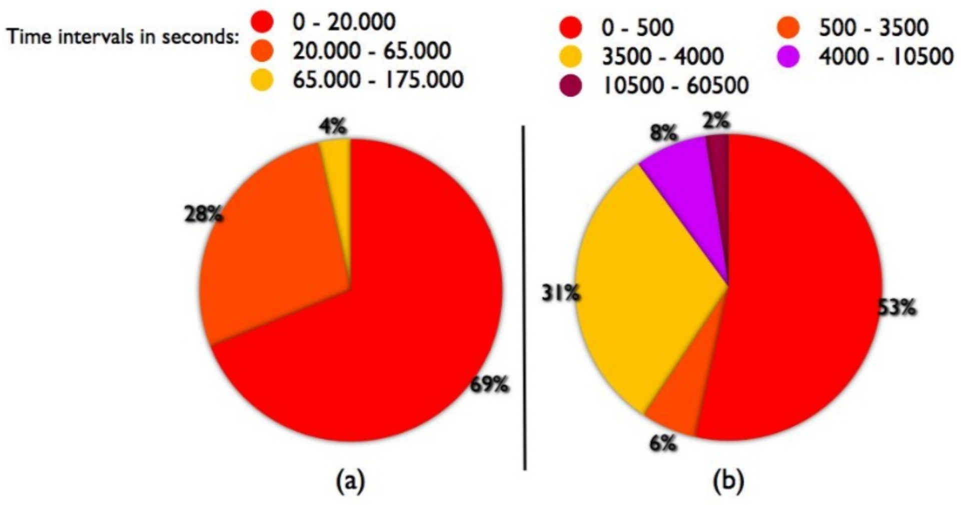


Figure 3.4: (a) The distribution of the number of jobs failed for the remote stage-out during July 2011 over CPU wall-clock time intervals spent. (b) The distribution of the number of jobs failed for the remote stage-out during July 2011 over CPU wall-clock time intervals spent by the remote stage-out step.

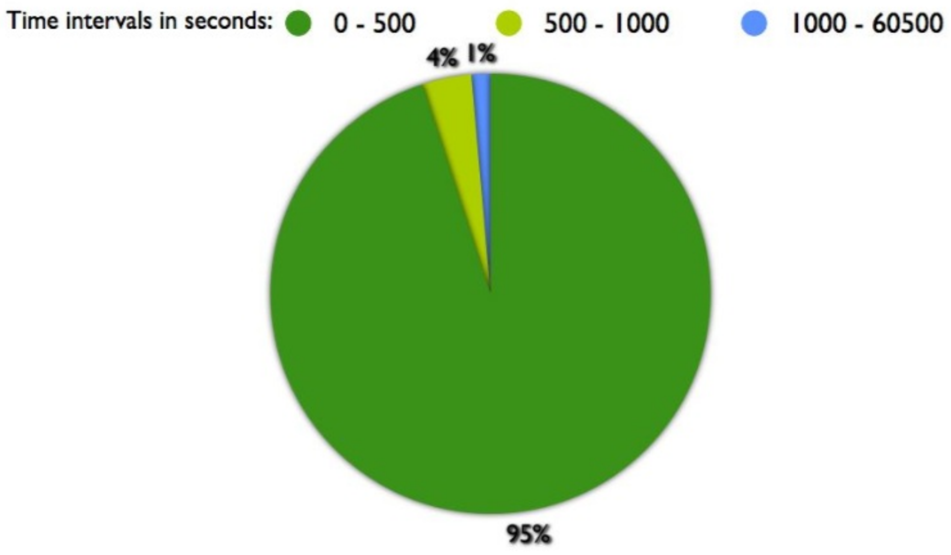


Figure 3.5: The distribution of the number of jobs done during July 2011 over CPU wall-clock time intervals spent for the remote stage-out.

Chapter 4

Design and development of the Automation tool

To address issues described in 3.2.1, an Automation tool is designed to cache and prioritise user requests, feeding information to components that take care of the distributed data bookkeeping, and then triggering the submission of jobs once user conditions are satisfied.

In this chapter, the keys design of the new Workload Management system are detailed. Next, the design of the Automation system is described. It is also presented the implementation and integration strategies of the tool. Finally, The CMS alignment and calibration activity is presented, as a case of study, and the low latency facility hosting the main workflows of this activity are described. In particular, it is shown the Automation system tuned over the first use case, the calibration of the Beam Spot measurement[73].

4.1 New generation of Workload Management tools

After more than 4 years of experience acquired with CMS jobs submission tools, CRAB, ProdAgent and Tier-0, CMS collaboration has decided to design and develop a new common system for managing their workloads for the following reasons:

- increase common code;
- reduce maintenance and support costs;
- improve the system stability, quality and usability;
- share benefits and learning from experiences acquired by different CMS Workload Management tools developers to enforce the system sustainability;
- and have more flexibility for new use cases.

This new system, named WMCore/WMAgent, is designed to support three main application areas: data processing and reconstruction, large scale production of Monte Carlo simulated events and data analysis. In other words, it will form the basis of the existing and any new Workload Management Services. It is in this context that it is decided to extend CRABServer to support the automation of the end-user analysis workflow.

One among most important core libraries of WMCore framework is Workload Management Bookeeping system (WMBS) since it defines the "Job entities". A WMBS Workflow instance can be seen as a set of jobs owned by a user. An important feature provided by WMBS is its capability to handle inter Workflow jobs dependencies. Thus, it allows the chaining of workflows to compile and execute complex workflows with no human intervention. The following is a descriptions of few WMBS objects and their main methods:

- **File** is a simple object representing a file in WMBS. The attributes of this object that can be set when it is instantiated are: lfn, id, size, events, checksums, parents, locations, first_event, and last_event. Within File object, the following operations are permitted:
  - addChild(lfn): set an existing file (lfn) as a child of this file;
  - addParent(lfn): set an existing file (lfn) as a parent of this file;
  - create(): create a file;

  - delete(): remove a file from WMBS;
  - exists(): check the existence of the File object in WMBS;
  - load(): load any meta data that is associated with a file. This includes id, lfn, size, events and checksum;
  - updateLocations(): write any new locations to the database. Therefore, all locations will be reloaded from the database.

- **Fileset** is a simple object representing a Fileset in WMBS. A fileset is a collection of files for processing. This could be a complete block, a block in transfer, some defined dataset by user, etc. The arguments of this class are: name, id, is_open (is_open parameter is a bool representing whether all files of a fileset are already associated), files, parents, parents_open (it is a bool representing whether all files of the parent filesets are already associated), source (It is the data source. It can be DBS, Phedex, etc) and sourceUrl. The methods of this class are:
  - addFile(file): add the file object to the set of files in the fileset, but dont commit to the database in order to enable bulk operation. commit() methods call updates the database;
  - create(): add the new fileset to WMBS, and commit the files;
  - delete(): remove this fileset from WMBS;
  - markOpen(isOpen): change the open status of this fileset. The isOpen parameter is a bool representing whether or not the fileset is open.

- A **Job** is owned by a Jobgroup (which is associated to a workflow) and is associated to a (set of) file(s). The Job interacts with its Jobgroup to acquire/complete/fail files. A Job knows its Workflow via its Jobgroup. A Job is meaningless without a Jobgroup. Jobs are added to the WMBS database by their parent Jobgroup, but are responsible for updating their states. The attributes of this class include name, files and id. The followings are few methods of this class:

  - associateFiles(): update the wmbs_job_assoc table with the files in the inputFiles for the job.
  - getFiles(type = list): retrieve a list of files that are associated with the job.
  - getState(): retrieve the state that the job is currently in.
  - load(): load the jobs name, id, jobgroup, state, state_time, retry_count, couch_record, location and outcome from the database. Either the ID or the name must be set before this is called.
  - save(): flush all changes that have been made to the job to the database.

The design of WMCore framework is purely object-oriented in order to :

- provide a clear modular structure for programs which makes it good for defining abstract data types where implementation details are hidden,
- make the framework easy to develop, maintain and modify existing code,
- provide a good framework for core libraries where supplied software components can be easily adapted and modified by the programmer,
- improves productivity: code developed faster, more reliable, natural management of development tasks in a large team of developers.

The Data Access Object (DAO) design pattern is also used in this framework to separate low-level data access logic from high-level business logic. Adopting this approach, modifications in the business logic can rely on the same DAO interface, while provides flexibility to change an application's persistence mechanism over time without the need to re-engineer application logic that interacts with the DAO tier. The loosely dependency between the connectivity and data access code, and the business logic makes easy to migrate the application from one type of data source to another. Within WMCore, SQL interactions are defined in objects packaged under the SQL dialect. The DAOFactory object is used to dynamically load these SQL objects. In general, business objects hold a DAOFactory object and use it to load up the various required SQL objects. An example of an instantiation of the DAOFactory object and the subsequent use is shown in Figure 4.1. The argument package of the DAOFactory object sets the base for all subsequent dynamic class loads. In this example, "newWorkflow" would load WMCore.WMBS.Workflow.New.

```
daofactory = DAOFactory( package = 'WMCore.WMBS' , logger =
self.logger , dbinterface = self.dbfactory.connect() )
newWorkflow = daofactory( classname = 'Workflow.New' )
```

Figure 4.1: An example of an instantiation of the DAOFactory object and the subsequent use.

Presently, WMCore support MySQL, SQLite and Oracle databases as backends. Additionally, it can interact with the Document database "CouchDB".

The CMS Workload Management system is developed using Python programming language. Being a highly dynamic, pure object-oriented, and allowing to develop reliable program rapidly prototyped, Python is widely used in CMS.

4.2 Objective of the automated processing of analysis workflows

The goal of this work is to provide an automatic way of managing CMS end-user analysis workflows. The basic concept of the automation is that workflows should be executed in a data driven manner. It foresees to take actions automatically when a sufficient amount of suitable new data are ready. The actions typically involve submitting jobs as soon as sufficient new data are accumulated, which can be defined as a number of new files or new luminosity sections in a run, the end of a run, or as a given number of events in a file, etc. As already mentioned, the alignment and calibration workflows can be complex since the outputs of one execution step serves as input of the next and thus can be represented symbolically via a Direct Acyclic Graph (DAG) model (shown in Figure 4.2).

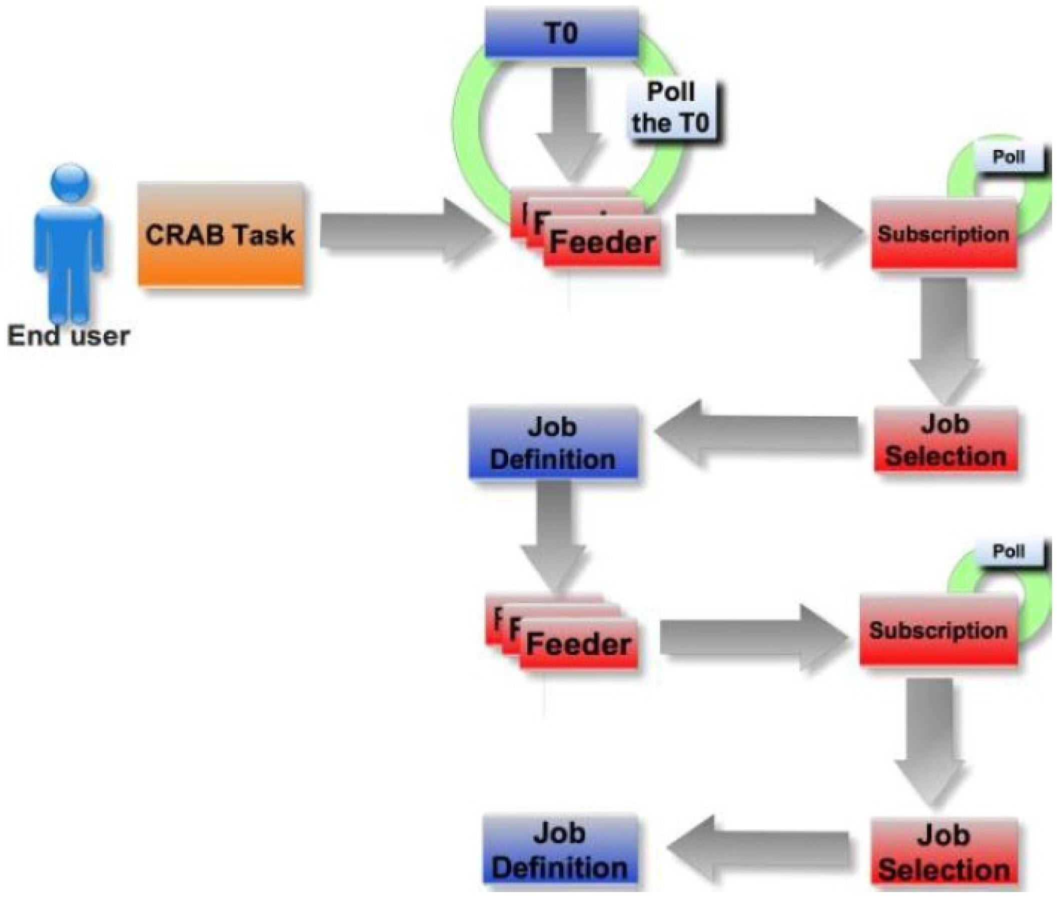


Figure 4.2: Complex automated workflows

The foremost objective is to automate alignment and calibration workflows for the following reasons:

- during the initial data-taking, the workflows are not mature enough to be handed over to a central data operations team;
- the alignment and calibration team need to be able to monitor the execution of their workflows very closely and apply correctness to the methods themselves and their configurations;
- and with less priority, to reduce the delays supported by typical analysis workflows.

The general aim is to automate more the work done on behalf of users reducing the efforts that they invest to manage their analysis workflows and decreasing, of course, the delays in their execution.

4.3 Implementation strategy

The WMBS system, described previously in this chapter, provides the ability to manage the workflows at the file level granularity, making possible to handle complex workflows as the automated processing of analysis workflows. In order to build the infrastructure needed to support automated workflows, an interface has been developed in the current CRABServer framework to interact with WMBS.

The use of CRAB allows to benefit from the simplicity of its existing Grid interface and also to achieve the first step required to build the next generation of CRAB framework (based on WMCore) . Figure 4.3 shows a simple automated workflow using CRABServer and WMBS.

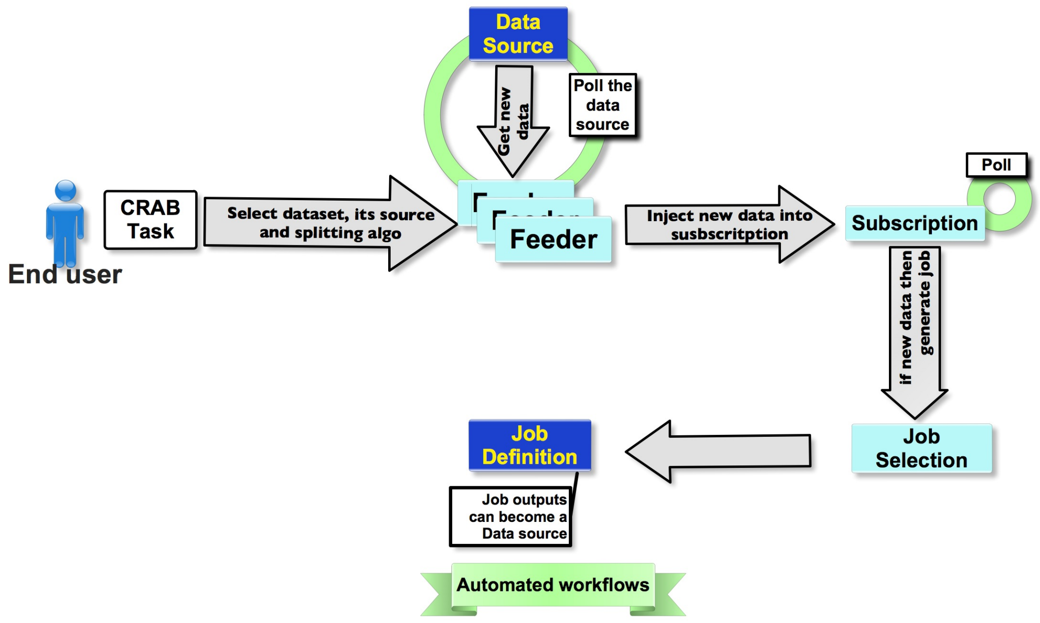

Data Source
Get new data
Poll the data source
End user
CRAB Task
Select dataset, its source and splitting algo
Feeder
Inject new data into susbscritption
Subscription
Poll
If new data then generate job
Job Selection
Job Definition
Job outputs can become a Data source
Automated workflows

Figure 4.3: Simple automated workflows using CRABServer and WMBS

The most important objects of the WMBS system, used to automate workflows, are WMBS instance, WMBS Subscription, and WMBS Feeder:

- WMBS instance is a workflow dependency manager that can be used to drive subsequent processing steps on the output of jobs as they run, such as merging.
- An instance of WMBS Feeder watches a data source and injects new data into the Subscription. A Feeder can be DBS, T0AST, Phedex, UserPlugin, etc.
- WMBS Subscription is an association of a WMBS Workflow, where a splitting algorithm is defined, and a WMBS Fileset. WMBS Subscription watches new slices of data that accumulates over the time. Active subscriptions are checked for new files and if conditions are met, new jobs are created according to the algorithm defined in the WMBS Workflow.

4.4 CRABServer extension

As described in 2.3.2.2, CRABServer implements a modular architecture, based on independent agents communicating through a persistent message system, based on a database. Within the Automation system, modules from CRABServer are involved to provide a flexible WMBS Feeder based infrastructure. In this way, requirements of different kinds of workflows can be fulfilled by customizing dedicated Feeders.

CRAB Client provides a simple user interface for the submission of workflows. From the workflow point of view, it transmits task specifications encoded in XML [77] to CRABServer for processing.

Once the task specification reaches CRABServer, the TaskRegister component starts the processing. It adopts a thread-based architecture to split the work in order to satisfy efficiently the large number of tasks sent by users. It creates BossLite objects and therefore contacts the FeederManager and WorkflowManager through the WMBS MsgService object. It sends an "AddDatasetWatch" message to the FeederManager to instantiate WMBS objects, namely WMBS

Feeder and WMBS Fileset, and an "AddWorkflowToManage" message to the WorkflowManager to create a WMBS Workflow object.

The FeederManager associates WMBS Fileset and WMBS Feeder objects. A new thread is handled in this component to create an empty Fileset once the "AddDatasetWatch" message is received from the TaskRegister. Furthermore, it polls regularly, in a main thread, the data source across the WMBS Feeders to complete file information of managed Filesets. The Feeder closes the Fileset when all its files are there. The format of messages coded in Python and the details of the "AddDatasetWatch" message are shown in Figure 4.4.

```
{'name':'AddDatasetWatch', 'payload':{'FeederType':feeder,
'dataset':dataset, 'FileType':processing, 'StartRun':startRun}}
```

Figure 4.4: Messages format.

The WorkflowManager associates WMBS Workflow and WMBS Fileset objects. In the same way as the FeederManager, this agent polls the WMBS database regularly to check unsubscribed Fileset to be matched by the managed-Workflows and vice versa. The WMBS Workflow name is a regular expression specified by the user in the CRAB configuration to specify the data that he is interested in. The WorkflowManager creates a Subscription if it matches a Fileset with a Workflow. A user can also remove Workflows from CRABServer whenever they wish.

At this point in the workflow, all information needed to create new BossLite jobs is available. CrabJobCreator is the component developed in CRABServer to support the extension and the Automation system. The general design of the CrabJobCreator component, using the Unified Modeling Language (UML)[78], is shown in Figure 4.5. CrabJobCreator polls existing Subscription objects to check new data. Subscriptions acquiring new files are picked up to take the following actions:

1. Applies the WMBS splitting algorithm to the Subscription to create WMBS Jobs.
2. Completes the BossLite Task filling BossLite Job and running Job tables.

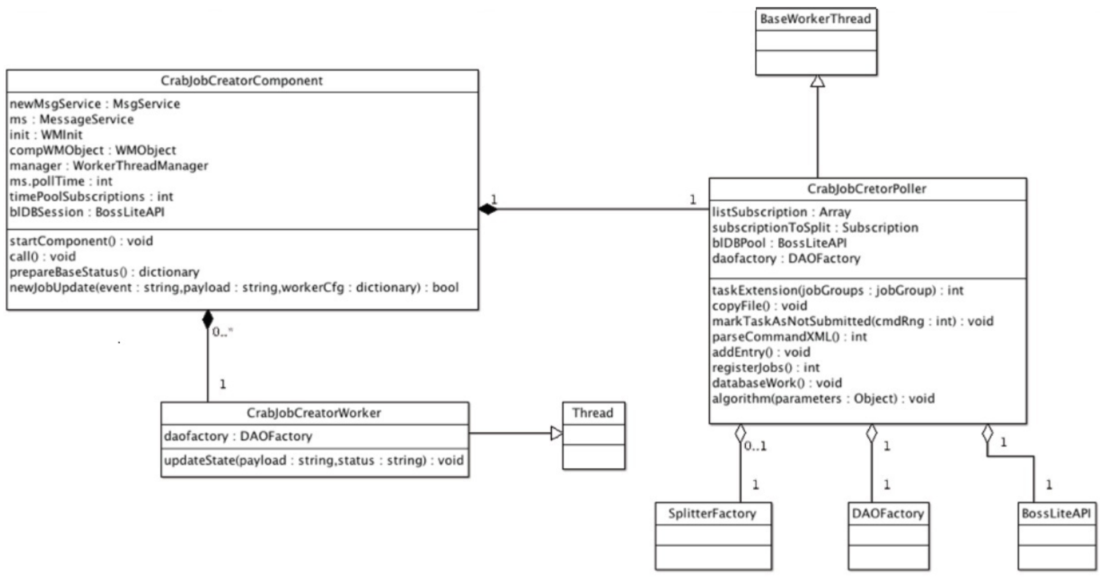


Figure 4.5: General Class diagram of the CrabJobCreator

3. Contacts CrabServerWorker for jobs submission.

The interaction between these modules is shown schematically in Figure 4.6.

4.5 Case of study: alignment and calibration workflows

Some calibrations are performed online at the CMS detector site, the offline workflows for alignment and calibration are performed mainly at the Tier-0 site and CAF, while a few others are performed at the Tier-2 level. They are used to compute and improve the alignment and calibration constants.

Event information relevant for alignment and calibration is streamed from the CMS detector site via the standard physics event stream and a special calibration stream, as well as streams with special event content named express. The reconstructed data are then skimmed to create a series of datasets that contain only the minimal amount of information required by the alignment and calibration workflows (AlCaReco). These datasets are transferred to CAF where they

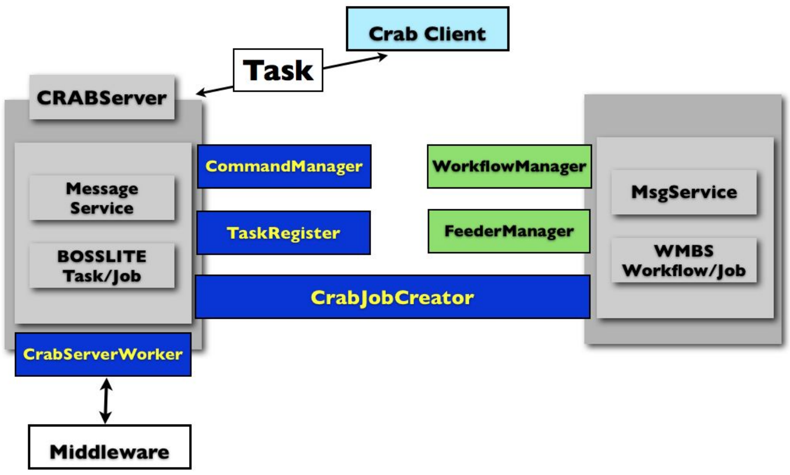


Figure 4.6: CRABServer extension

are processed by the alignment and calibration algorithms. Figure 4.7 shows the alignment and calibration activity embedded in the overall CMS offline workflow.

The compact skims, AlCaReco, are produced automatically both from the express stream and the bulk processing of all physics data. The express stream corresponds to about 10% of the physics stream bandwidth and it requires a low latency processing. The calibration streams contain information from laser systems and dedicated calibration triggers.

The alignment and calibration workflows, performed at the CAF, use the AlCaReco datasets to generate alignment and calibration constants that are validated and uploaded to the Conditions Database. They rely on an intentional delay of prompt reconstruction during which time RAW data are buffered on disk, while specific calibration workflows determine these constants on the fly from the express stream data, such that prompt reconstruction will proceed with the updated constants.

Re-reconstruction at the Tier-1 sites, using the new constants, also generates new AlCaReco datasets that are used in turn as input to the next series of improvements on alignment and calibration constants.

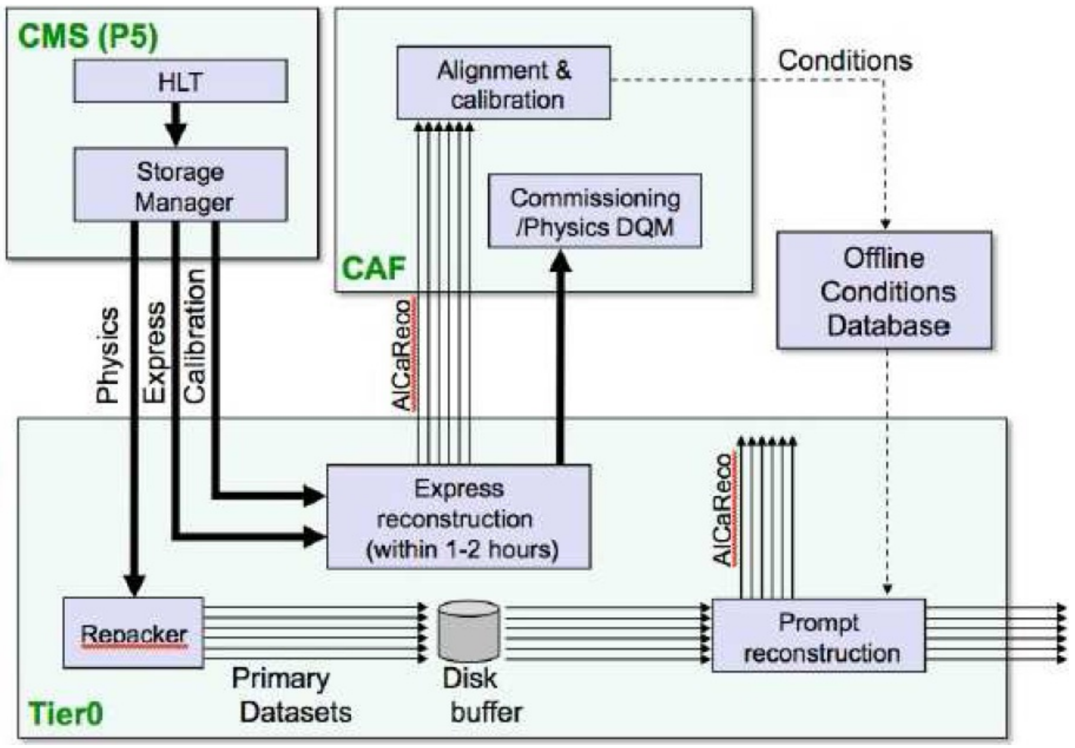


Figure 4.7: The overall CMS offline workflow

4.5.1 Low latency analysis facility

The technical requirements on the CAF are based on the capability to satisfy high-priority and low latency workfows, by means of enforcing controlled and prioritised access policies to hundreds of users. The CAF systems need to be setup such that particular jobs can start with very short latency in parallel to many other concurrent jobs and with highly efficient access to the CMS production data.

A dedicated processing farm has been installed in various steps since 2008, reaching a total of 876 cores. Access, scheduling and sharing is handled by the Platform LSF system. The batch system comprises a set of queues (788 job slots) available to all CAF users and a special queue (54 reserved job slots) dedicated to very high-priority alignment and calibration workfows.

A large disk storage system based on CASTOR technology [75] has been installed in various steps since 2008, reaching a capacity of 2.1 PB. Highly efficient and asynchronous access to the data coming from the Tier-0 is ensured by the disk-only characteristic of this storage pool, hence avoiding any kind of tape staging latency, which is a typical bottleneck encountered for large storage systems. In addition, CAF sub-groups and users are sharing 9 TB AFS space for POSIX-like storage needed for analysis. Finally, a set of 5 worker nodes (40 cores, 50 slots) are reserved for interactive access to CAF batch and storage resources, and two 8-core computers are hosting the CRABServers for jobs submission.

The CAF user management is handled via the "LSF WEB"[76] application provided by CERN/IT: the tool is used for user registration and to set the fair share among CAF subgroups. It is also automatically interfaced to the main CAF resources (CASTOR, AFS, CRABServer, Email lists) for granting access permission to CAF users. This tool turned out to be very convenient for user management, since it represents the single entry point to any CAF resource.

The main CAF data management and workflow management components include the job submission tool CRAB, the DBS and PhEDEx. These are standard CMS tools, with only minor adaptations compared to those used for distributed analysis on the Grid.

The CAF batch queues rely on a fair share system, in which ∼30 CAF subgroups are allocated resources averaged over time. The total number of jobs submitted, during a six months period after the beginning of LHC collisions data taking of 2010, was $\sim$ 2x10^6, while the cumulative number of active users has reached nearly 300 during the same period, see Figure 4.8. The main CPU management characteristics are a split into 3 batch queues (with maximum CPU time of 1 hour, 1 day and 1 week, respectively) and a limit of maximum 100 running jobs/user for the two shorter queues, to avoid batch congestions by a single user, while this limit is reduced to 10 for the longer queue.

4.5.2 Description of the first supported workflow

The first workflow that is being supported by the Automation system is the beam spot determination. The beam spot is the luminous region produced by the collisions of the LHC proton beams. Knowing the position of this is important for accurate physics measurements, and it needs to be measured precisely in an

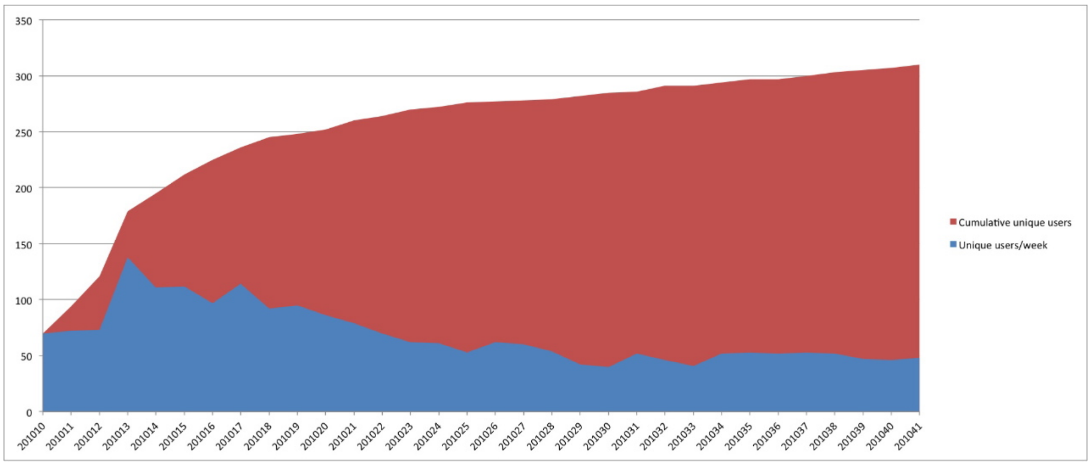
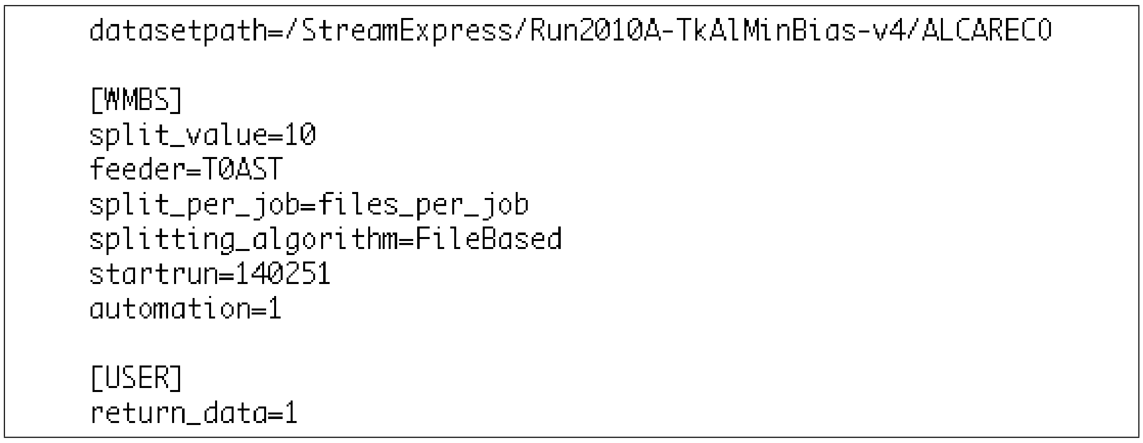


Figure 4.8: Number of distinct and cumulative CAF users per week, during the first 6 months LHC collisions data taking of 2010.

automated fashion for a correct offline data reconstruction. The new section in CRAB configuration file, named [WMBS], is used to specify all configurable parameters needed to drive the automatic process (see Figure 4.9).



```
datasetpath=/StreamExpress/Run2010A-TkAlMinBias-v4/ALCARECO

[WMBS]
split_value=10
feeder=T0AST
split_per_job=files_per_job
splitting_algorithm=FileBased
startrun=140251
automation=1

[USER]
return_data=1
```

Figure 4.9: WMBS section added in CRAB configuration file.

For the beam spot workfow, the data source is the Tier-0. The T0AST Feeder is developed in order to regularly poll the Tier-0 Data Aggregation System (DAS) [93] interface for a given workflow. T0AST Feeder injects new files once they appear in the Tier-0 into the WMBS instance.

Measuring the beam spot using CRABServer required the development of four

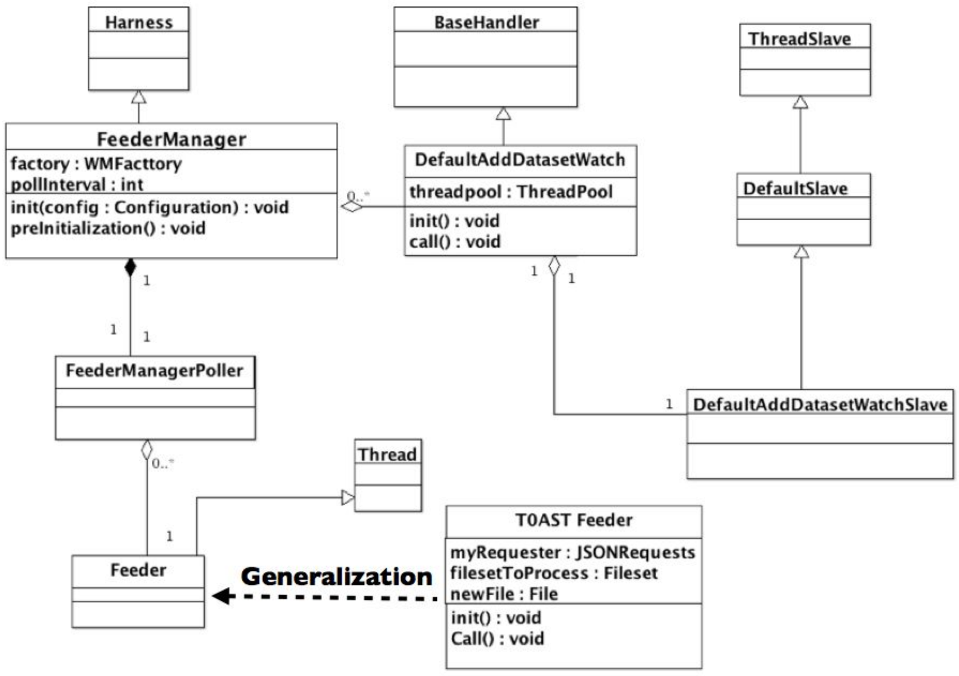


Figure 4.10: General Class diagram of the FeederManager

Feeders:

- T0ASTChain allows to submit jobs over outputs of another workflow as soon as they appear in the CAF storage system. This Feeder allows users to iterate automatically between their workfows;
- T0ASTRun allows to submit jobs over runs as soon as they are considered as finished by the Tier-0 system;
- T0AST allows to submit jobs over files as soon as they appear in the CAF storage system;
- DBSFeeder allows to submit jobs over files as soon as they appear in DBS. It has been developed to automate also the other kinds of alignment and cal-

ibration workflows (Strip Tracker calibration workflows...), and the typical analysis workflow. It is the Feeder used for Grid jobs.

All WMBS Feeders inherit from the super-class Feeder. The Feeder based architecture allows to add a new Feeder whenever needed by just writing a class which inherits from Feeder parent class and using correctly the WMBS library. Figure 4.10 shows the general design of the FeederManager module.

Chapter 5

Design and development of the asynchronous stage-out tool: AsyncStageOut

To address the issues described in 3.2.2, it is decided to adopt an asynchronous strategy for the remote stage-out. This required the design and development of a machinery able to stage-out the outputs locally, in the storage of the site where the code is executing, followed by a subsequent outputs harvesting step where the users outputs are copied to a remote storage using FTS.

In this chapter, the asynchronous stage-out system is presented, introducing briefly tools and terminologies. The design and implementation of the tool are detailed. The integration of asynchronous stage-out tool with current CMS distributed analysis tool, CRAB, is also described. Finally, it is reported the results of the first asynchronous stage-out service tests.

5.1 File Transfer Service

To address the issues caused by the remote stage-out, it is decided to rely on FTS. This service, developed as part of the gLite middleware, is the underlying data movement service of Phedex for transferring files between sites. It was designed to:

- balance resource usage at sites used by multiple experiment Virtual Organizations (VOs),
- prevent network or storage overload,
- enforce prioritization in transfer jobs.

FTS exposes a Web service interface to submit asynchronous bulk requests for parallel file transfers (transfer jobs). Transfer requests are stored in an Oracle database and assigned to unidirectional queues, called "channels". A channel is not tied to a physical network path but is defined between endpoints, where each endpoint might be a site or a group of sites. The configuration of each channel defines the resource restrictions such as the maximum number of transfers for each of the VOs using the channel.

The FTS performs the transfers using either third-party GridFTP or Storage Resource Manager (SRM)[80]. These third-party transfers enable FTS to drive transfers in multiple parallel streams, thus being able to scale up to the limits of the underlying network and reach an aggregated transfer rate of 1.5GB/sec. Protocol and protocol parameters, such as timeouts or the maximum number of simultaneous GridFTP streams, are set in the channel configuration. The resulting throughput on a site-to-site channel is the transfer rate for a GridFTP stream, times the number of GridFTP streams per files times the number of files in active transfer on the channel. The data flow in an FTS transfer is shown in Figure 5.1

The FTS servers are deployed at the Tier-0 at CERN and at each of the Tier-1 sites. The CERN FTS server manages transfers on dedicated channels between the Tier-0 and the Tier-1s. The FTS server at each Tier-1 centre is typically configured to manage:

- Imports on dedicated channels from other Tier-1s
- Exports and imports on dedicated channels to and from the associated Tier-2s
- Exports and imports on shared channels to and from the non-associated Tier-2s

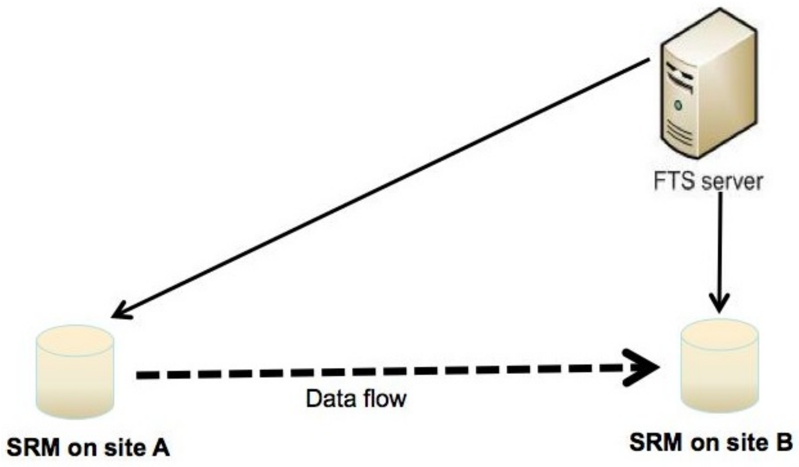


Figure 5.1: Data flow in an FTS transfer.

- Third-party transfers to the associated Tier-2s from other Tier-2s on shared channels.

The latter kind of transfers will be used by the asynchronous stage-out transfers.

PhEDEx is described in 2.5. The backend module of the download agent in PhEDEx is the core of the data-movement engines. FTS is used as agent backend in the most WLCG sites. It is responsible for splitting the transfers into chunks and submits them as transfer jobs to FTS, tracking the status of those jobs, and updates the PhEDEx database with results. It was used heavily to transfer large amounts of data, order of Peta-Bytes, between WLCG sites and has shown satisfactory performance in term of scalability and stability.
PhEDEx is customized to transfer the official CMS data and its current design cannot handle the users outputs for free since:

- Phedex can transfer only files from a given LFN path in a site to the same path in the destination site, while users outputs must be transferred from an LFN path in the standard temporary users area, namely /store/temp/user, in the site where the jobs were executing to the permanent area of users in /store/user/ of the destination site;

- To optimize data transfer, the unit of data that PhEDEx transfers is a block, while analysis jobs produce simply outputs files. It is clear that transferring datasets by files instead of blocks will affect the scalability of the tool since its central database, TMDB, should be updated at each file transfer instead of each block transfer;

- Phedex is optimized to transfer files that the average size is reasonably large, 1-2 GB, while the outputs of analysis jobs vary in size;

- Using Phedex to transfer the end-users analysis outputs means that every removal or transfer of the outputs should be performed by means of Phedex, otherwise the TMDB will be corrupted. Consequently, physicists have to learn this tool, which makes the analysis task more complex for them.

For all these reasons, it is decided to rely only on FTS and not on the entire PhEDEx tool.

5.2 Implementation strategy and used technologies

The analysis workflow including the asynchronous stage-out steps can be summarized as follows:

- User submits its analysis workflow from an UI.

- Based on the configuration and the location of the data that he needs, the jobs are scheduled by the WMS to run in matched Tier-2s,

- Once the execution of the analysis code in the WN is done, the output is copied in the local SE of the site (Local stage-out) in /store/temp/user area. If the local copy of the output succeeds, a request is automatically submitted to FTS to copy the output to the remote SE (Remote stage-out) in /store/user area.

- The transfer request is then tracked and resubmitted if the submission fails.

The asynchronous stage-out workflow is shown schematically in Figure 5.2. The use of FTS allows to balance site resources usage, prevent network and storage overload, and manage transfer resubmission. The latter will avoid to waste resources by resubmitting the whole analysis workflow as it is the case for the synchronous remote stage-out approach and reduces also the delay in the execution of the analysis workflows since only the transfer of the output will be resubmitted. To reduce the delays in accessing the outputs, a first secure copy of the output, in the local SE of the site where the job run, is provided to the user.

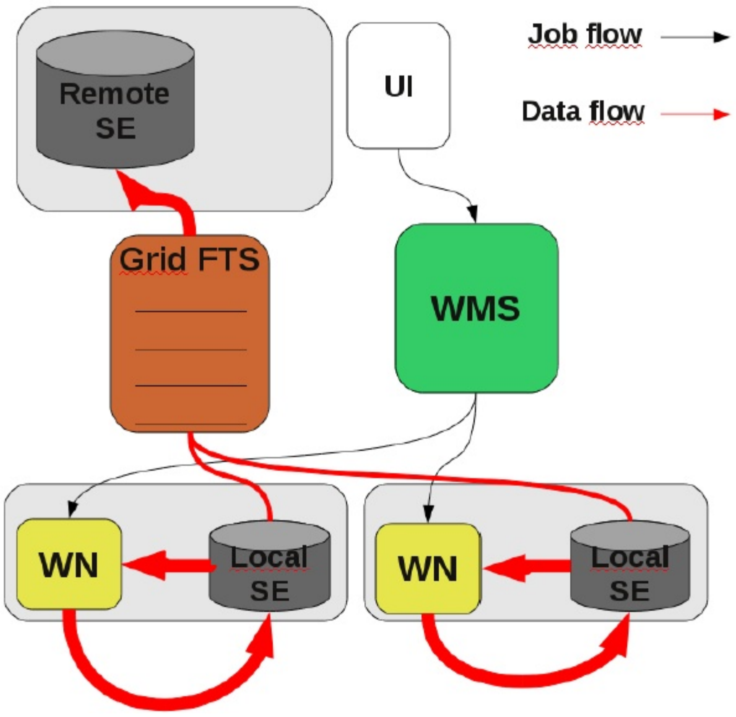


Figure 5.2: Asynchronous stage-out approach.

The NoSql database, CouchDB, is extensively used in Data Management / Workload Management (DMWM) project and the APIs for using this technology are already available in WMCore. This technology is used to persist the details of user outputs to transfer using the AsyncStageOut tool. The next section describes CouchDB and the benefit in using it.

5.2.1 CouchDB technology

CouchDB is a NoSQL database management system developed by the open source Apache Software Foundation [84]. The term Couch stands for "Cluster Of Unreliable Commodity Hardware" emphasizing the aim of CouchDB to provide a highly scalable and reliable system by means of commodity hardware. CouchDB was originally written in $C++$. In April 2008, the project has been ported to the Erlang OTP platform [85]. Apache CouchDB is a document-oriented database, accessible via a RESTful[86] API. It uses a system of creation and management of indices for accessing and querying the database through views. It also provides a multi-master and bi-directional replication mechanism with decentralized conflict management. The following is a high-level introduction of key model and benefits of CouchDB:

- **Document database**: CouchDB does not use the relational model, where data are represented by tables and relations, but a document-oriented approach. Data document are stored in schema-less documents, each one with its own implicit structure not tied to a rigid database design. In such systems, the document is the main unit of data and consists of a finite number of fields and attachments. Even the metadata managed by the DataBase Management System (DBMS) are represented as documents. The fields can contain values of various types (text, numbers, boolean, lists, objects, etc) with no limit to the size of the text or number of elements. The documents are written in JavaScript Object Notation (JSON)[87], a standard created for the exchange of data structures in human-readable format. It is based on JavaScript[88] programming language. It is a text format completely language independent and easy for machines to parse and generate.

- **RESTful HTTP API**: CouchDB considers all stored objects as resources and provides a set of RESTful APIs to access and modify the documents in the database. These APIs are accessible via HTTP GET and POST requests and return objects formatted in JSON, making communication and execution of operations with the database simple regardless the language of the applications that use it.

- **ACID Properties**: CouchDB storage system verifies all the Atomicity, Consistency, Isolation, Durable (ACID) proprieties. Operations involving the documents are serialized, except for binary attachments that are written concurrently but committed serially. The database is never locked out when reading documents, furthermore, documents readers never have to wait on writers or other readers, even when using the same document. CouchDB uses a model of Multi-Version Concurrency Control (MVCC) and therefore each client is able to access a snapshot of the database for the duration of the read operation. When a document is modified, the changes are not made on the document itself but a new version of the document is created. Consequently, a full document history is automatically maintained.

  The documents are indexed in a B-tree [90] data structure by their unique name (DocID), and a Sequence ID for a quicker access. The Sequence IDs are used to keep track of the operations performed over the documents. The indices of b-trees are updated simultaneously with the creation, modification, and deletion of documents.

- **Distributed updates and replication**: The distributed database system of CouchDB allows users and servers to access and modify shared data while disconnected and then replicate the changes bi-directionally. Both ordinary and "design" documents can be replicated providing the possibility to get a copy of the entire applications in a different servers. This feature allow to eliminate quickly the problems caused by bottlenecks in a distributed systems or, for example, allows to use the applications in offline or in a local servers and updating later the shared database.

- **Views**: To extract relevant documents from the database, CouchDB integrates a view model. Views are the means by which they are reported and aggregated the actual document contents of a database. They are executed on-demand and defined, using JavaScript functions, in a special document called "design" document. The views functions compute, by means of "map / reduce"[89] approach, the CouchDB documents to determine which data is to be made available. The view is only a dynamic representation of the actual document contents of a database and do not affect in any case the document content.

Generate a view of a database that contains a large amount of documents is a task that requires considerable time and resources and it cannot be then calculated from scratch at each time. To ensure rapid interrogation of CouchDB documents, the view engine maintains indexes of their views in B-trees. It incrementally updates the view since the last read access rather regenerating it from scratch at each time when requested. The CouchDB's core is largely optimized for an efficient creation of views and their indexes.

The numerous benefits of this new technology, in particular:

- Fast and agile "schema" updates and changes;
- Easy replication, synchronization, and load balancing;
- Extremely quick development of applications;
- Very efficient querying of large data;
- Every manipulation of data stored in CouchDB can be done via HTTP, hence it is possible the communication with a plenty of Client options.

and the experience acquired by DMWM developers in using CouchDB as back-end for various applications makes it a choice coming before any Relational DataBase Management Systems (RDBMS) when prototyping a new application in DMWM project.

5.2.2 AsyncStageOut design description

The asynchronous stage-out tool is a CRAB 3 module, implementing a modular architecture, based on the common DMWM library, WMCore. Figure 5.3 shows the sequence of the interactions between the main modules and libraries in CRAB 3.

To be flexible as possible, the AsyncStageOut tool provides a set of configurable parameters such as expiration_days, max_transfers_retries, max_files_per_transfer and database_source. Its core machinery is as follows:

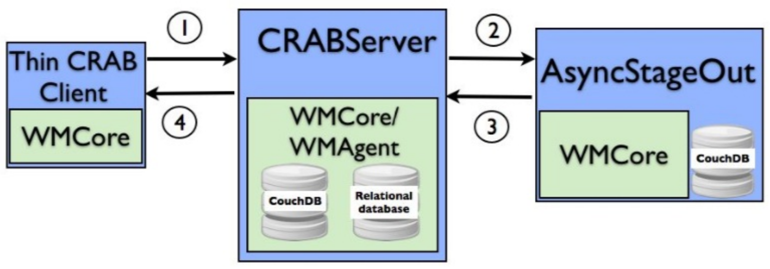


Figure 5.3: General architecture of CRAB 3.

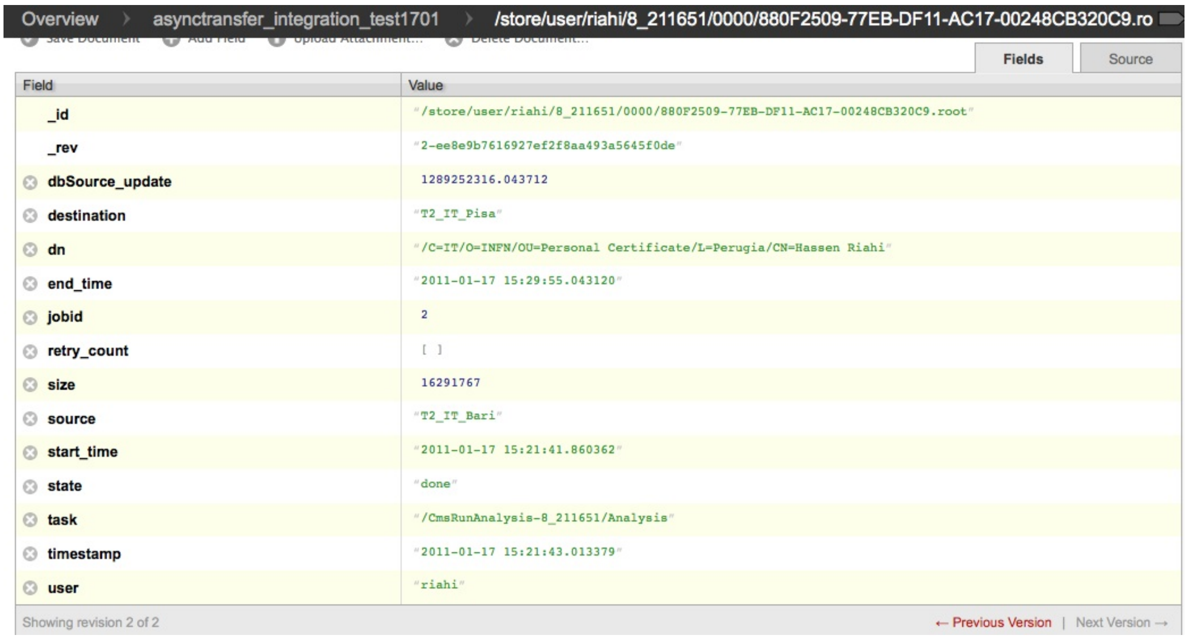
1. AsyncStageOut interacts with database_source in CRABServer's Couch instance to get the details of the output to transfer. Each document in this database represents a job FrameWorkJobReport (FWJR) and should be organized in steps describing the status of the analysis job execution in the WN, namely logArchive, cmsRun and stageOut.

   The analysis job output is copied locally in the storage of the site where the code is executing. Once this step is achieved, the analysis job is considered as done and their information, such as status, output location and output path are updated in its FWJR.

2. The LFNDuplicator module polls regularly the database_source to get the outputs details of jobs done and store them as transfer documents (the format of a transfer document is shown in Figure 5.4) in a proper database in Couch, named files_database. Once added, the transfer document is in a "new" state.

3. The TransferDaemon module polls regularly the files_database to get the details of files to transfer. At each polling cycle, it instantiates, for each user, a TransferWorker object which, marks the transfer documents as acquired, and creates and submits then FTS jobs for transferring the user outputs to the final destination site and:

   - If a transfer fails, the transfer document is marked as failed in files_database. The TransferWorker marks the transfer as expired if the transfer still fails after max_transfers_retries times.


Overview › asynctransfer_integration_test1701 › /store/user/riahi/8_211651/0000/880F2509-77EB-DF11-AC17-00248CB320C9.ro

Field	Value
_id	"/store/user/riahi/8_211651/0000/880F2509-77EB-DF11-AC17-00248CB320C9.root"
_rev	"2-ee8e9b7616927ef2f8aa493a5645f0de"
dbSource_update	1289252316.043712
destination	"T2_IT_Pisa"
dn	"/C=IT/O=INFN/OU=Personal Certificate/L=Perugia/CN=Hassen Riahi"
end_time	"2011-01-17 15:29:55.043120"
jobid	2
retry_count	[ ]
size	16291767
source	"T2_IT_Bari"
start_time	"2011-01-17 15:21:41.860362"
state	"done"
task	"/CmsRunAnalysis-8_211651/Analysis"
timestamp	"2011-01-17 15:21:43.013379"
user	"riahi"

Figure 5.4: Transfer document format in files_database.

- it marks the transfer as done if it succeeds.

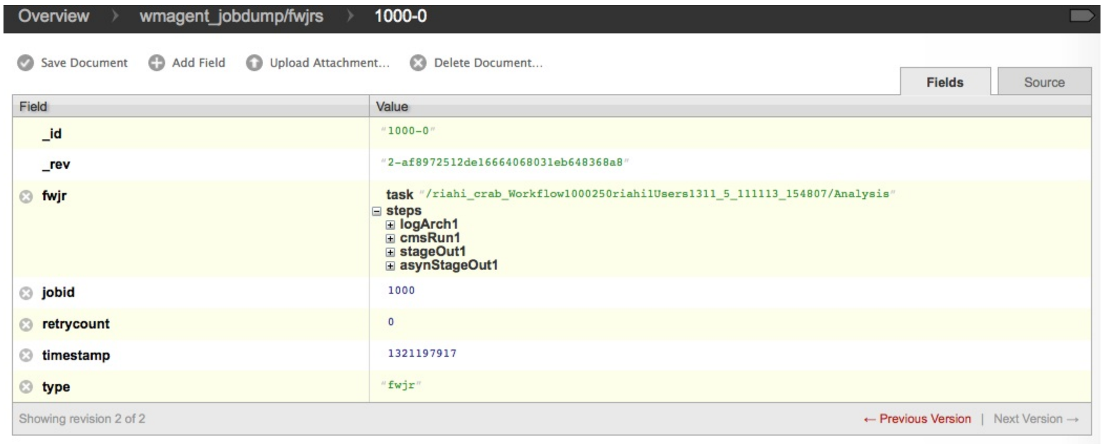
4. Updates the database_source by adding an asyncStageOut step to the FWJR document. Figure 5.5 shows a FWJR updated with an asyncStageOut step.

The main interactions of the asynchronous stage-out components are shown in Figure 5.6, while the state machine of files documents in files_database is shown in Figure 5.7.

To allow the management of many transfer jobs in parallel, the tool implements a parallel processing approach, by means of multiprocessing library of Python, to take advantage of multiple cores and processors in a machine.

Expired and done transfer documents will remain in files_database for expiration_days before being removed from there by the StatDaemon which tasked to merge the transfer documents by FTServer used in a single document and insert it in the stat_database.

In order to be able to interact with any DMWM tools requiring the remote stage-out service, this tool is developed following a plugin-based architecture to



Field	Value
_id	"1000-0"
_rev	"2-af8972512de16664068031eb648368a8"
fwjr	task "/riahi_crab_Workflow1000250riahilUsers1311_5_111113_154807/Analysis" ⊟ steps ⊞ logArch1 ⊞ cmsRun1 ⊞ stageOut1 ⊞ asynStageOut1
jobid	1000
retrycount	0
timestamp	1321197917
type	"fwjr"

Figure 5.5: FWJR document updated with an asyncStageOut step.

allow to add a new source whenever needed by just writing a class which inherits from the Source parent class. The first plugin developed is the Job State Machine (JSM) to allow the interaction with WMAgent framework. The general class diagram of the asynchronous stage-out is shown in Figure 5.8.

However, the asynchronous stage-out system requires much more care, in particular regards the use of disk storage and time required by the operation of the remote stage-out. In fact, in such system, each output must be copied first in the local SE and transferred later to the remote site, which requires more storage capacity than the current synchronous remote stage-out strategy. In addition, the operation of the local copy and the subsequent transfer with FTS, introduces latency between the completion of the job and the time at which the asynchronous stage-out tool begins effectively to transfer the output. For this reason, the asynchronous stage-out system has been integrated into a version of CRAB 2, and used to perform preliminary tests of scalability and latency.

5.3 Integration of asynchronous stage-out and CRAB

CRAB 3 will make use of an asynchronous stage-out server to interact with FTS in order to manage the transfers of analysis jobs outputs. To test the asynchronous

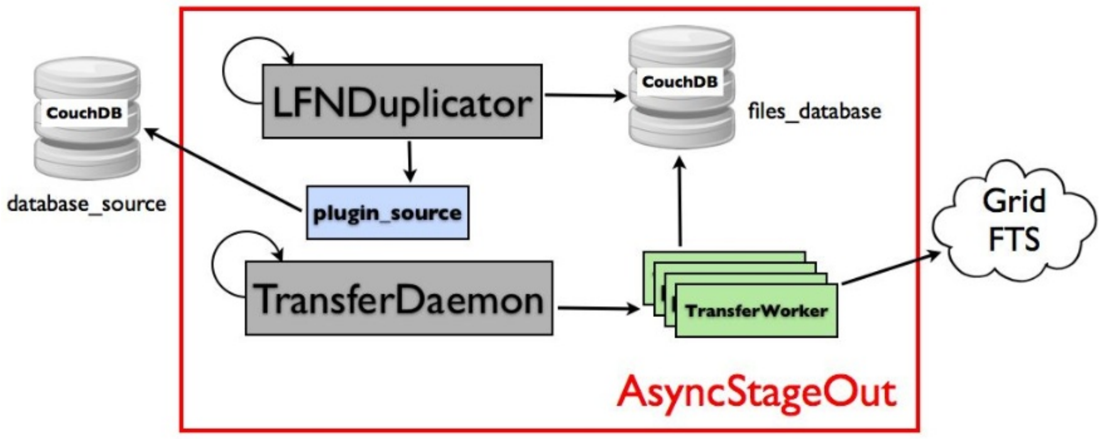


Figure 5.6: Asynchronous stage-out core machinery.

stage-out during the development of CRAB 3, it was decided to integrate the asynchronous stage-out in a development version of CRAB 2. To enable the use of the asynchronous stage-out in CRAB, a new component, named CrabJobAccountant, has been implemented in CRABServer to allow the communication with the asynchronous stage-out server. Furthermore, the following parameters were added in CRAB configuration file:

- default_local_stage_out: This parameter forces CRAB job wrapper to stage-out the output in the local site rather the remote one specified by storage_element parameter, which will be ignored.
- destination: It is used to specify the name of the SE to where the output must be transferred by the asynchronous stage-out server. This parameter is needed since, as mentionned previously, the storage_element parameter is ignored.

A diagram of the asynchronous stage-out and CRAB integration is shown in Figure 5.9. The workflow can be summarized as follows:

1. CRAB submits the jobs to the Grid.

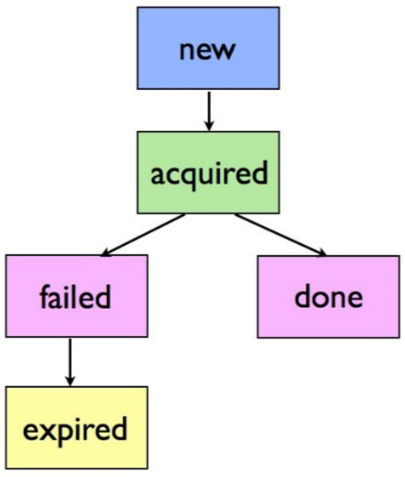


Figure 5.7: Asynchronous state machine.

2. CRAB checks periodically the status of not completed jobs.

3. Once the execution of a job on the WN has finished successfully, its output is copied by CRAB job wrapper to the local SE of the site.

4. At this point, the CrabJobAccountant component adds a transfer document for this output in the asynchronous stage-out database.

5. The asynchronous machinery, described previously, start the transfer of the output.

The next section describes the first functional tests and related results.

5.4 Functional tests of the asynchronous stage-out tool

5.4.1 First round of tests

The first test was carried out using jobs for the hadronic decays of the top analysis. The functional test foresees the execution of a large number of jobs producing

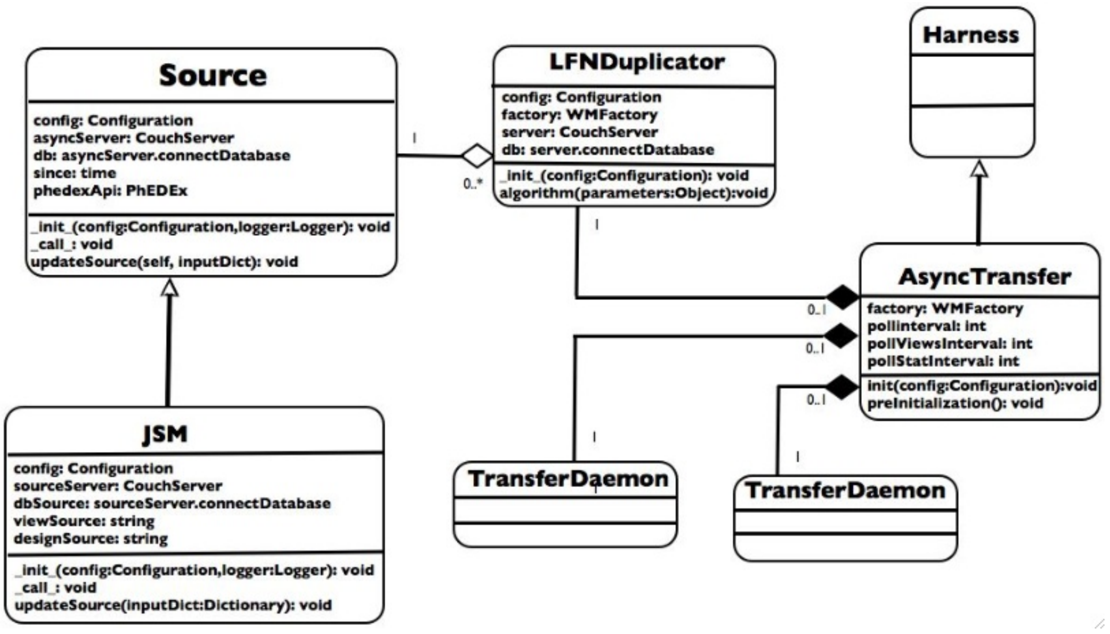

Source
config: Configuration
asyncServer: CouchServer
db: asyncServer.connectDatabase
since: time
phedexApi: PhEDEx
init(config:Configuration,logger:Logger): void
call: void
updateSource(self, inputDict): void
LFNDuplicator
config: Configuration
factory: WMFactory
server: CouchServer
db: server.connectDatabase
init(config:Configuration): void
algorithm(parameters:Object):void
Harness
AsyncTransfer
factory: WMFactory
pollInterval: int
pollViewsInterval: int
pollStatInterval: int
init(config:Configuration):void
preInitialization(): void
JSM
config: Configuration
sourceServer: CouchServer
dbSource: sourceServer.connectDatabase
viewSource: string
designSource: string
init(config:Configuration,logger:Logger): void
call: void
updateSource(inputDict:Dictionary): void
TransferDaemon
TransferDaemon
1
0..*
1
0..1
0..1
0..1
1
1

Figure 5.8: General class diagram of the asynchronous stage-out.

order of gigabytes of output size. It was decided to take advantage of this test and the Grid resources that requires to achieve a "real" analysis.

The analysis foresees the processing of five datasets. A new task was created over each dataset with different number of jobs and different average of outputs sizes.

The number of jobs successfully completed was not as expected and this due to wrong TFC file at some sites and to a bug in the customized version of CRAB used. In particular, the TFC at some sites was translating correctly the LFNs to PFNs but was not able to perform the reverse operation, required by the asynchronous stage-out machinery.

This test has promptly highlighted the broken TFCs files in CMS sites, which, of course, should be fixed. To do so, an action plan was discussed and decided together with CMS integration team.

Regarding CRAB, the customized version, used for these tests, was crashing when retrieving the outputs and, consequently, it prevents the recovery of the FWJR files which are required in a later step to complete the analysis. This bug

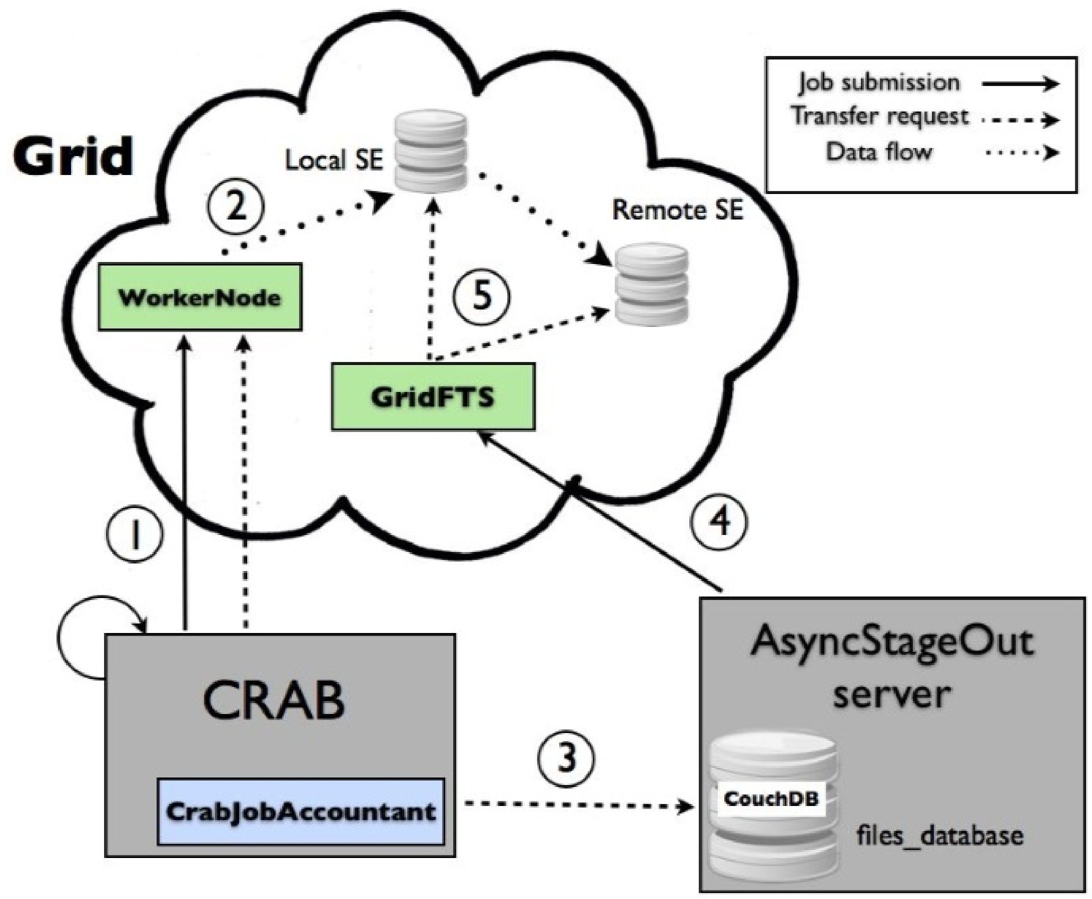


Figure 5.9: Asynchronous stage-out and CRAB integration.

is fixed for the next round of tests.

Table 5.1 is listing the names of the datasets used, the number of jobs submitted and the number of successful jobs. As shown by this table, several jobs have completed successfully, and, in particular, their outputs were transferred to the final destination. Using the outputs of these jobs, a new attempt was made to go ahead with the next steps needed to achieve the analysis, but more issues were met. In particular, the FWJR files report the close SE (the close SE is the SE of the site where the job has run) as location of the outputs even when they were transferred successfully to the final destination. This makes the publication in DBS not possible, since it is based on the information in the FWJR. Since such issue was complicated to fix in the customized version of CRAB, it was decided

Dataset	Total Jobs	Succeeded jobs
MultiJet	580	451
Jet	364	250
JetMET	725	496
JetMETTau	186	186
MC	158	158

Table 5.1: Summary of the total number of jobs created and succeeded over each dataset required for our analysis.

to submit test jobs rather than "real" analysis ones for the next round of tests.

In this test, $\sim$ 2.000 jobs were submitted to analyze the hadronic decays of the top, of which, 1.541 were completed successfully. It was also submitted some other jobs for preliminary checks, which brought the total number of succeeded jobs to 1.692.

These jobs have allowed to identify the problem of translation from LFN to PFN in some sites, which were manually excluded from the possible destinations for the remaining jobs of this first round of tests. Most of the failures reported were caused by sites configurations. The datasets used for the analysis of hadronic top were located in a few sites and, after removing manually the ones that have problems, each dataset becomes available in only one site. Among 445 jobs available for transfer, 444 have succeeded, and only one results as failed for a bug in CrabJobComponent, promptly fixed.
The fraction of successful and failed transfers and the reasons of the failures are shown in Figure 5.10.

The fact that the tasks submitted are split into jobs producing outputs differing in size, has allowed to estimate the system latency by varying the output size (Figures 5.12, 5.13 and 5.11).
Basing on the legend below each plot, the colors of points represent the time when the asynchronous stage-out server has received the request of transfer.
As shown by these plots, the latency is highly dependent on when the transfer was requested, and does not depend on the size of the output to transfer. This phenomenon is particularly evident in case of Monte Carlo simulations jobs (Figure 5.11). Most of the output files turned out to have dimensions near to 790 MB, 1.1

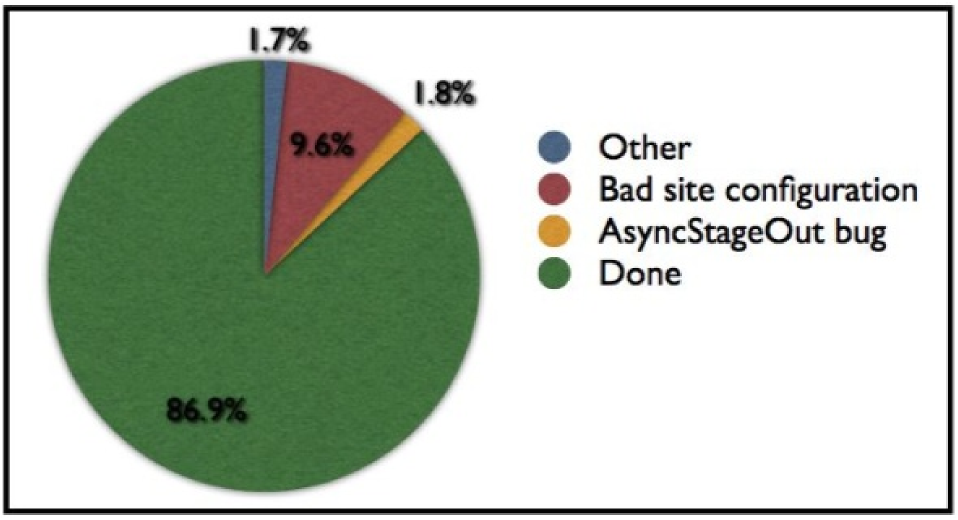


Figure 5.10: The fraction of failed and succeeded transfers, and the reasons of the failures.

GB and 1.3 GB. The dependence on when the transfers are recorded by the asynchronous stage-out server may be due to the latency in requesting the transfer to FTS by the server or a latency of the FTS server used itself, which, in different times, can be a subject to different loads and different length of transfer queues to manage. Figures 5.12 and 5.13, representing respectively output transfers of JetMETTau jobs and MultiJet jobs, seem to confirm this analysis. The fact that these datasets correspond to real data makes the outputs size more distributed. JetMETTau jobs are completed in a very short time interval, as it can be seen from the uniform color of the points, and were transferred in times comparable to the variation of the output size. The MultiJet task, however, consisted of heavier jobs, which ended in different time intervals. The associated Figure shows an accumulation near 2 GB of the output size, and a scattering along the y-axis due to the different times of registration of the transfer requests. Typically, transfer jobs recorded later were ending faster. This result shows that the transfers depend on the time when they are requested, emphasizing the benefit of the asynchronous stage-out system, which allows to retry again failed transfers.

Despite the issues met, these tests have shown a large fraction of succeeded transfers, which reflects the smoothness of the tool. In addition, they were inter-

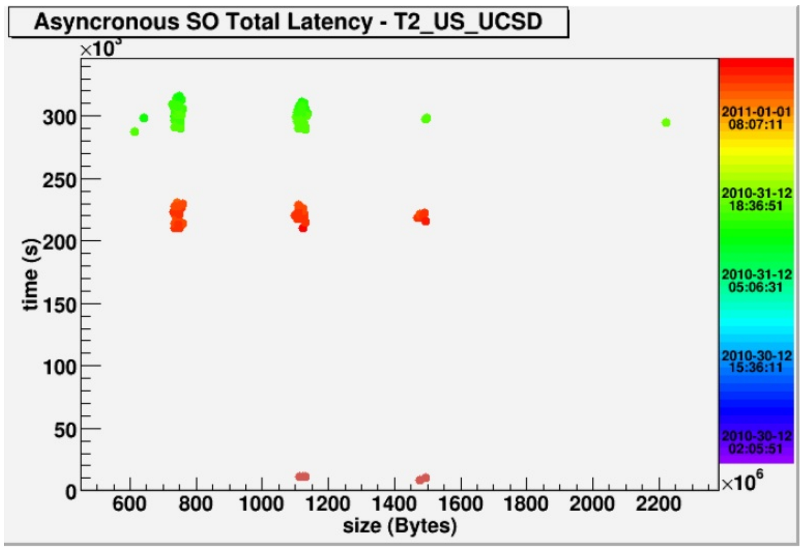


Figure 5.11: Latency in transferring the outputs of Monte Carlo jobs to the SE of T2_US_UCSD

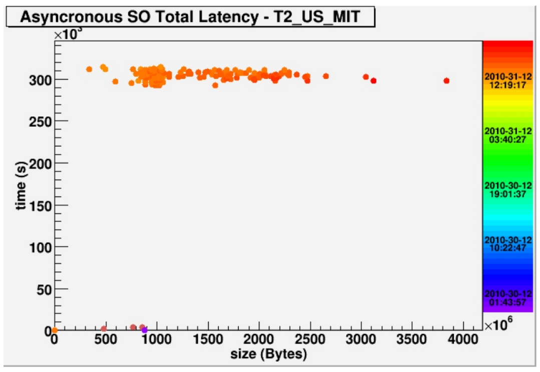


Figure 5.12: Latency in transferring the outputs of JetMETTau jobs to the SE of T2_US_MIT

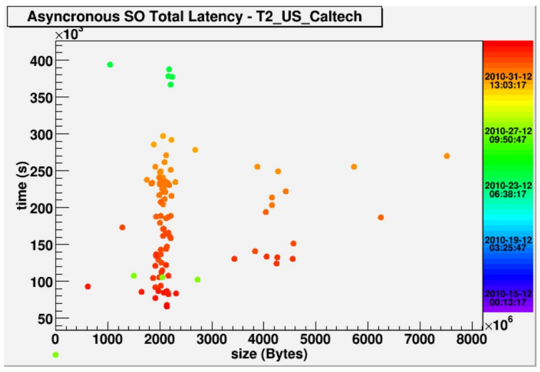


Figure 5.13: Latency in transferring the outputs of MultiJet jobs to the SE of T2_US_Caltech

esting as they provided an estimation of the latency since the output is available in the close SE to when it is copied and becomes available in the target SE.

5.4.2 Second round of tests

For this round of tests, it was chosen to use the same dataset used to test the performances of CMS sites, named JobRobot dataset, to benefit from its high availability in various computing sites that support the experiment. Such dataset allows to create a large number of jobs producing equal outputs sizes. However, it was necessary to "blacklist", in CRAB configuration file, the sites that do not support the translation from PFN to LFN described previously, limiting the number of appropriate sites to 20.

The task created for this test consisted of 400 jobs, producing each ~ 1.3 GB file. This test has promptly revealed the incompatibility of some of these sites with the asynchronous stage-out system, since it did not support yet the local stage-out. The task was submitted several times to reach a total of 1686 transfer jobs.

Figure 5.14 shows the fraction of successful and failed transfers, while Figure 5.15 shows the failures rate by site.

Out of a total of 1686 transfers, approximately 98.3% of transfers succeeded. The transfers failure in this test were caused by a wrong FTS server used or a temporary issue of sites involved in the transfer.

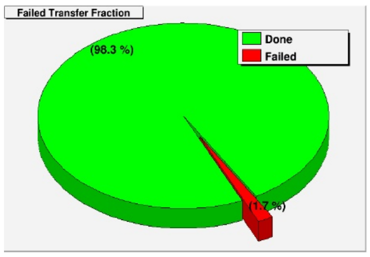


Figure 5.14: Fraction of failed and succeeded transfers.

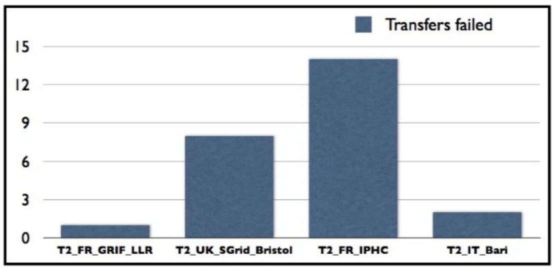


Figure 5.15: Transfers failed by site.

The configuration of the asynchronous stage-out allows to retry failed transfers more than one time. Given that, the asynchronous system can be considered to be more reliable than synchronous one since it uses a robust system of data transfer allowing to resubmit the request of the copy in case of failure.

The histogram in Figure 5.16 shows the number of transfer attempts by site. This histogram shows the results for only succeeded jobs. Note that if the system were synchronous, the only succeeded jobs could in the first column, while the other would collapse and their jobs would have to be resubmitted. From Figure 5.16, the number of attempts seems to be greater for some sites, for example, T2_UK_Bristol, which also appears to be one of the sites with the highest number of failures (Fig. 5.15). This might indicates some incompatibility of the site with the asynchronous stage-out system, since the latter is still experimental.

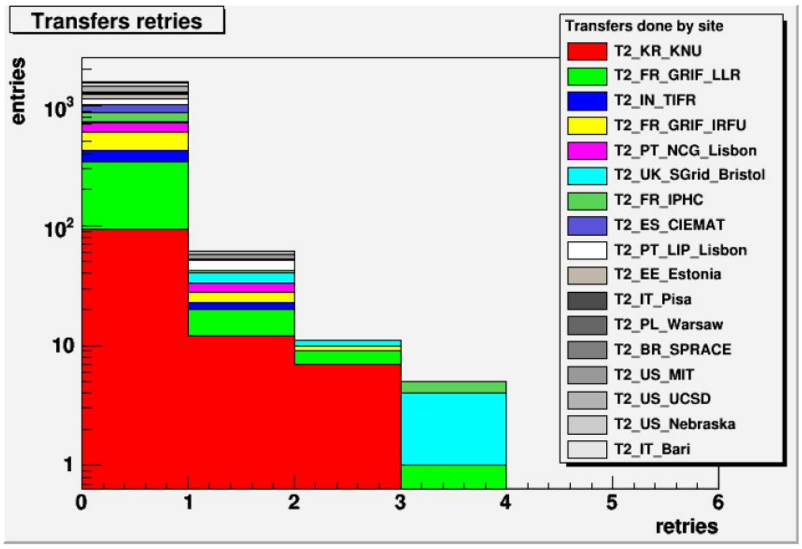


Figure 5.16: Transfer retries by site.

The total latency introduced by the asynchronous stage-out system is more distributed in the second round of tests, due to different network performances available to the various sites and the different times when the transfers of outputs were requested. Furthermore, the output sizes of each job were very similar, nearly 1.3 GB. Table 5.2 shows the average time required to transfer an output from a site source to a site destination. The performance was varying between sites.

The load supported by the asynchronous stage-out server during these tests is shown in Figure 5.17. The histogram shows the trend of the total number of transfers per hour, and the number of transfers notified as completed to the asynchronous stage-out server for each hour. The maximum number of simultaneous transfers was reaching almost 600 jobs.

Within the conditions described above, the system did not show problems of scalability. The maximum number of transfers notified as completed per hour was approximately 350 jobs. Comparing with Figure 5.18, where it is shown the

Source site	Latency (min)	File
T2_US_UCSD	158	14
T2_EE_Estonia	312	86
T2_FR_GRIF_LLR	610	254
T2_FR_GRIF_IRFU	381	183
T2_IN_TIFR	390	92
T2_US_Nebraska	6	109
T2_PT_LIP_Lisbon	540	142
T2_ES_CIEMAT	318	137
T2_KR_KNU	400	113
T2_FR_IPHC	425	149
T2_PT_NCG_Lisbon	352	126

Table 5.2: Transfer latency by source site of $\sim$ 1.3 GB file.

number of analysis jobs completed per hour in the first half of December 2010, a period in where there has been intense analysis, the number of jobs completed is over about two orders of magnitude than the results obtained in our test and shown by the green histogram in Figure 5.17. However it is not realistic to make a test with numbers comparable to those of real analysis, because the consumption of resources would be very high.

The scalability performance, that the system has shown, allows to wait to evaluate the true scalability of the system on ”real analysis” once the first prototype of CRAB 3 is in production, reaching the number of transfers required.

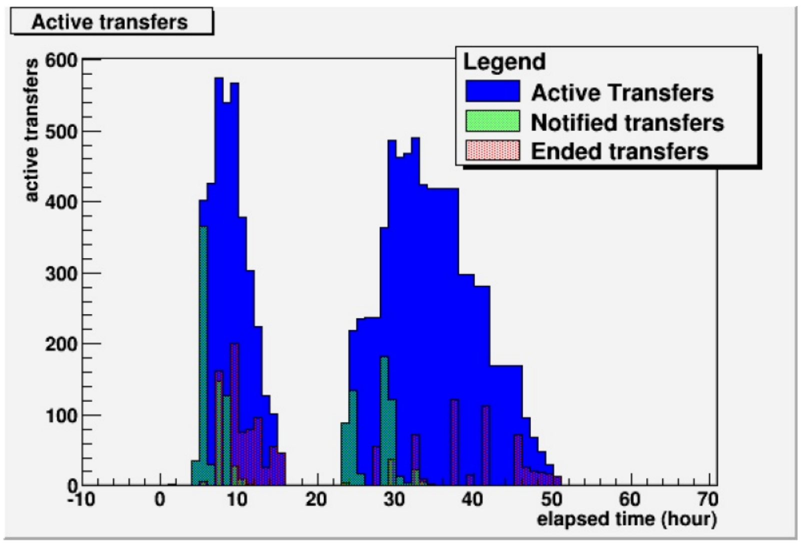


Figure 5.17: Load supported by the asynchronous stage-out server to manage 1349 transfers in 60 hours. In blue, the number of active transfers at each hours. In green, the number of transfers notified to the asynchronous stage-out at each hour, while the red is representing the number of transfers completed at each hour.

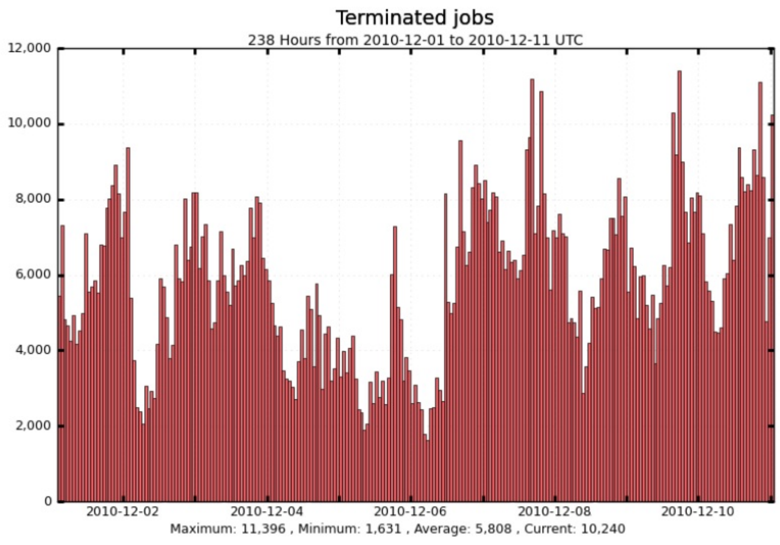


Figure 5.18: Number of analysis jobs terminated per hour during the first half of December 2010 (the period when these tests are done).

Chapter 6

Results

The Automation system and the AsyncStagOut tool previously described in chapter 4 and chapter 5 are designed to optimize the end-user analysis workflows by reducing, as much as possible, the latency of their executions.

In particular, the AsyncStagOut tool improves the Grid resources usage by reducing the rate of failures of analysis jobs, and consequently the resources that they waste.

In the first part of this chapter, the Automation tool performance over the case of study, alignment and calibration workflows, is reported, while the performance of asynchronous stage-out tool is reported in the second part.

6.1 Automation tool performance over alignment and calibration workflows

The support of alignment and calibration workflows by the Automation tool represents a high priority task for CMS collaboration. In order to reduce the timescales to reach this goal, the T0Emulator daemon has been developed. It allows to run the Automation machinery over the alignment and calibration workflows during the down periods of the Tier-0. This daemon reproduces continuously significant data for these workflows, which allows the development against the Automation system during the LHC downtimes.

By the LHC startup of February 2010, a dedicated production instance of

CRABServer at CAF running the Automation machinery has been setup to serve the alignment and calibration workflows. The latency and the usage of the system in production are described in the next sections.

6.1.1 Latency over the first supported workflow

During the first period of the Automation tool production, a particular attention was given to the beam spot determination workflow since it requires very short latency.

Practically, to execute the beam spot determination workflow, an expert configures and submits its workflow to CRABServer, once CMS announce that there will be collision runs in the next few days. Then, it is CRABServer that takes care to submit automatically the jobs of the workflow once data of interests becomes available and starting from the run specified in the configuration of the workflow. These jobs will execute over collision and non collision data starting from the run specified. "datasetpath" parameter is used in CRAB configuration to indicate data of interests while "start_run" parameter is for specifying the run to start from. The beam spot can be determined from the outputs of jobs executed over collisions data. So, the expert starts, just after the collisions, to look at their storage area at CAF to check whether their outputs exist.

Given that the alignment and calibration workflows are running at CAF and thus were limited by the Andrew File System (AFS) token lifetime, the immediate feedback was asking to extend the lifetime of the workflows to be as long as possible. The ability to initialise a renewable token for 3 days was added in CRAB Client and the automatic user token renew was included in CRABServer. The renew period was extended in a second step to one month, maximum allowed at CERN.

Figure 6.1 shows the fitted X positions of the beam line produced using the automation system as a function of luminosity sections for the run 132440.

Monitoring closely the calibration of the beam spot workflow, for one month starting from 15 June 2010, when the maximum user token validity is already extended to 3 days and the workflow becomes stable, the machinery takes less than 2 hours between the start of collision runs processing in Tier-0 and the

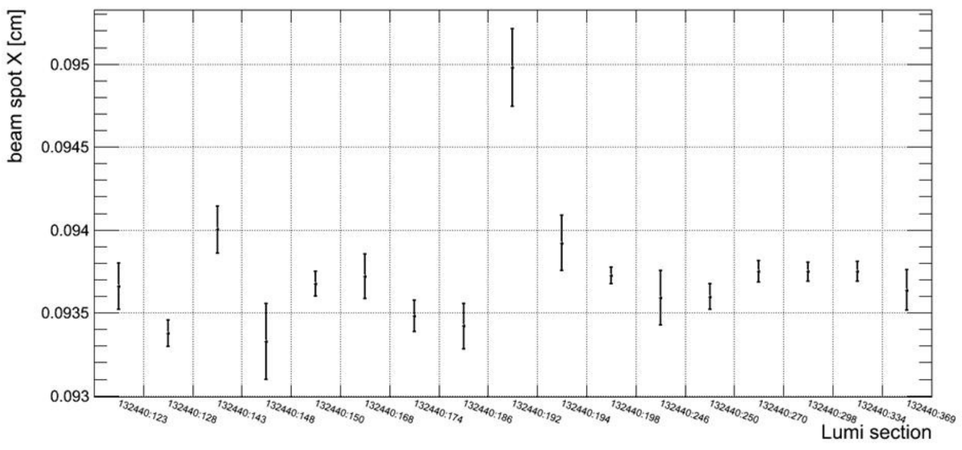


Figure 6.1: Fitted X positions of the beam line as a function of luminosity sections.

beginning of the automatic jobs submission in CRABServer over the data that is produced by these runs. Figure 6.2 shows this time interval, during this month, from run 138560 to 140317. Such latency allows alignment and calibration experts to update the conditions database with beam spot constants on the time scale required by CMS.

Figure 6.2 shows also the long latency seen between the start of non collision runs processing in the Tier-0 and the start of the automatic jobs submission in CRABServer over data produced by these runs. This latency is caused by the fact that experts do not know in advance the exact numbers of the collision runs and, consequently, they set the ”start_run” parameter to a run, in general a non collision one, already processed by the Tier-0. Such configuration allows to experts, in particular, to validate their workflows and check whether they produce correctly the outputs.

Data from non collision runs are not used for the beam spot calculation. In the future the feeder will be improved to filter out data taken during non collision runs.

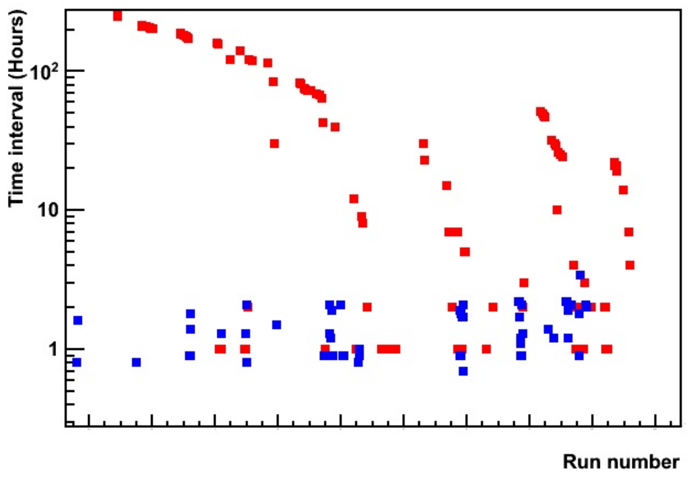


Figure 6.2: Time interval between Tier-0 processing and jobs submission in CRABServer from run 138560 to 140317. Non collision runs are represented by red squares, while collision runs by blue squares.

6.1.2 Automation tool Usage

The Automation tool was heavily used by the alignment and calibration experts once available in production. Figure 6.3 shows the number of distinct users during the first six months, while Figure 6.4 shows the weekly number of jobs during the same period.
These figures show that the numbers of jobs and users are high during the first period, from March to June, compared to the last 2 months. During the first 4 months, experts were stressing the system to define the configuration of their specific workflows. Once the workflows become more stable, by June, the numbers of users and jobs are decreased considerably to be limited to experts that submit well understood workflows for specific alignment and calibration activities.
The fall of the number of jobs during July is caused by a major upgrade of CRABServer which took 3 days.

This experience has allowed to alignment and calibration experts to understand their workflows. It will make easy the later migration of these workflows into the more structured and controlled Tier-0 system.
The satisfactory performance of the Automation tool shown, has made useful to

follow up and evolve the system to support, not only alignment and calibration workflows at CAF, but also the workflows running in the Grid as well as the typical analysis workflows.

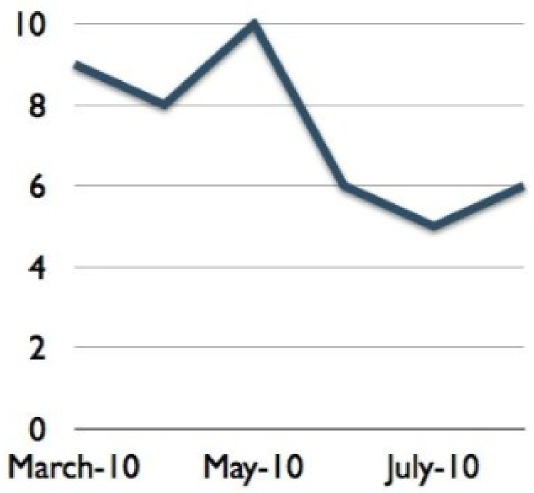


Figure 6.3: Number of distinct users during the first six month of the Automation tool production.

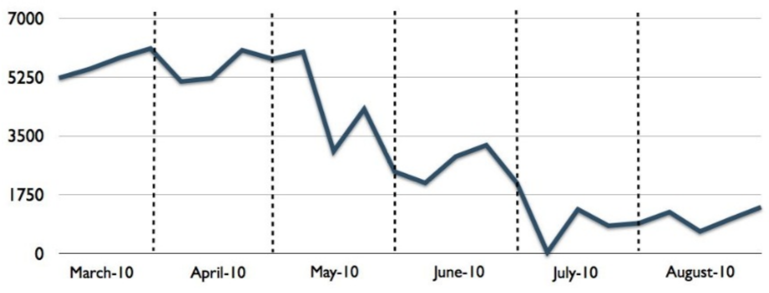


Figure 6.4: The weekly number of jobs during the first six month of the Automation tool production.

6.2 Asynchronous stage-out tool performance

While the functional tests of the Asynchronous stage-out tool are described in 5.4, these tests aim to stress the basic functionalities of CRAB 3 at some scale. They come in the context of the decision of CMS to provide CRAB 3 to beta users before the technical stop of the LHC, which will happen by the end of 2012. These tests are coordinated by the integration team of CMS.

6.2.1 Description of the setup

To motivate physicists to contribute on these tests as beta users, CMS has decided to open for them the large Tier-1 facilities for analysis workflow executions. Such strategy allows to these physicists to execute their analysis before anyone else by using the highly available Tier-1 facilities and accessing data stored there. The collaboration has provided these users the special role, "t1access", to be able to submit their jobs and store their outputs using Tier-1s resources. The interactions of the analysis workflow with CRAB 3 components can be summarized as follows:

- User deploys CRAB Client to configure and submit their workflows to CRABServer. The interaction between Client and Server in CRAB3 is done through RESTfull[91] Web services.
- CRABServer takes care then to:
  - split the workflow to jobs using the algorithm selected by the user in the configuration file,
  - submit and track jobs,
  - update the job FWJR database in CouchDB.
- The AsyncStageOut polls periodically the job FWJR database to execute its machinery described in the last chapter.

These tests are performed using following CRAB 3 tags:

- Client : 3.0.2

- Server : 3.0.2
- AsyncStageOut : 0.0.1
- WMCore : 0.8.5

Workflows are submitted by several users and executed over the JobRobot dataset, described in 5.4.2

6.2.2 Validation of basic functionalities

The aim of this first test is to check the basic functionalities of the AsyncStageOut and define a working CRAB configuration to be able to perform later the scale tests.

This test has shown a large number of failures caused by the AsyncStageOut component restarts. It was noticed in particular that by the shutdown of the AsyncStageOut server, the on-going transfers were interrupted and the files states were not updated. So, by the next server startup, the files, which have been transferred before the shutdown, will fail forever when retrying to transfer them, since they already exist in the target SE.

A few transfers were failed for temporary target SEs problems. Tickets, describing the problems met, were opened, using the CMS computing infrastructure ticketing system, against involved sites.

Monitoring closely the AsyncStageOut and CRABServer, it was noticed that they are are not using the user role neither when submitting the job nor during the stage-out step. Since the jobs submission and the stage-out of outputs to Tier-2 do not require in general special roles, this bug was not noticed before.

In addition, from the log files, it was seen that the AsyncStageOut does not give the right path to the remove command for the pre-clean file transfer. This bug was fixed and committed promptly.

The scale tests aim to understand the scalability of the system with respect to increasing the number of files to transfer at time and the number of users. During these tests, the jobs execution site and the remote SE will be fixed to avoid any kind of leverage that can be introduced by such parameters. The FTS server to use will be also fixed consequently since it is the one located at the Tier-1 of the remote SE site.

To choose a stable remote storage for the next tests, a workflow of 50 jobs is submitted over the JobRobot dataset. The T2_IT_Bari, known as the most reliable Italian Tier-2 site, is specified as final SE of the outputs. Jobs are then scheduled to run in T2_BE_IIHE site in Belgium and T2_ES_CIEMAT in Spain. The failure and success rates by site are shown in Figure 6.5.

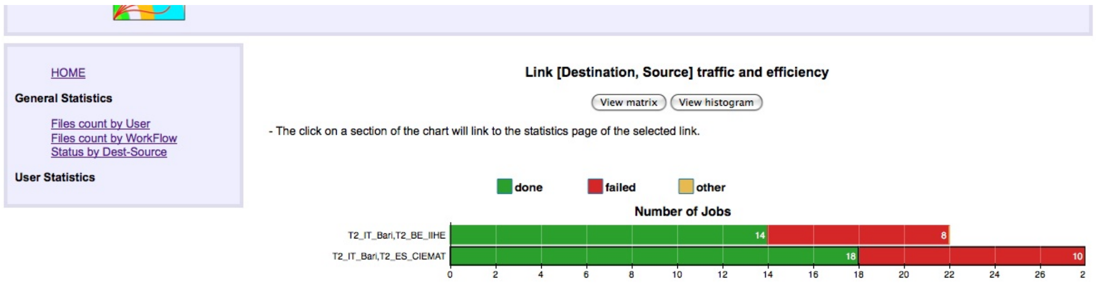


Figure 6.5: Success and failure rates by site.

Figure 6.6 shows the error message reported by the ftslog files of transfers.

```
Source: srm://srm.ciemat.es:8443/srm/managerv2?SFN=/pnfs/ciemat.es
/data/cms/scratch/user/riahi/RelValProdTTbar/MyReskimForTwo/v1/
0000/D473C235-4CF0-E011-A946-002481DE47D0.root
Destination: srm://storm-se-01.ba.infn.it:8444/srm/managerv2?SFN=//
cms/store/user/riahi/RelValProdTTbar/MyReskimForTwo/v1/0000/
D473C235-4CF0-E011-A946-002481DE47D0.root
State: Failed
Retries: 1
Reason: TRANSFER error during TRANSFER phase:
[GRIDFTP_ERROR] globus_ftp_client: the server responded with
an error
500 500-Command failed. : callback failed.
500-globus_xio: System error in write: No space left on device
500-globus_xio: A system call failed: No space left on device
500 End.
```

Figure 6.6: Example of FTS error messages..

Such message can indicate a temporary problem of the T2_IT_Bari SE or the user space in the SE is really full. Cleaning the user space using srm commands has resolved partially the problem. A new error has appeared occasionally in the log file. This error is shown in Figure 6.7. It indicates that the user credential is seen as not valid by the T2_IT_Bari SE.

```
Source: srm://srm.ciemat.es:8443/srm/managerv2?SFN=/pnfs/ciemat.es
/data/cms/scratch/user/riahi/RelValProdTTbar/MyReskimForTwo/v1
/0000/20369431-B4F2-E011-B2B7-002481CFE25E.root
Destination: srm://storm-se-01.ba.infn.it:8444/srm/managerv2?SFN=/
/cms/store/user/riahi/RelValProdTTbar/MyReskimForTwo/v1/0000/
20369431-B4F2-E011-B2B7-002481CFE25E.root
State: Failed
Retries: 1
Reason: TRANSFER error during TRANSFER phase: [SECU-
RITY_ERROR] globus_ftp_client: the server responded with an error
530 530-globus_xio: Authentication Error
530-OpenSSL Error: s3_srvr.c:2010: in library: SSL routines, function
SSL3_GET_CLIENT_CERTIFICATE: no certificate returned
530-globus_gsi_callback_module: Could not verify credential
530-globus_gsi_callback_module: The certificate is not yet valid:
Cert with subject: /C=IT/O=INFN/OU=Personal Certifi-
cate/L=Perugia/CN=Hassen Riahi/CN=proxy is not yet valid-
check clock skew between hosts.
530 End.
```

Figure 6.7: Credential error as reported in the ftslog.

Since this problem does not appear with all jobs of the user, it was likely that Bari SE has some temporary problems. This behavior is shown by the Figure 6.8 where for the last workflow submitted by the user, split into 100 jobs, some transfers have failed and other ones succeeded.

At this point, it was decided to resubmit a few test jobs and to use another storage as destination SE. Using T2_IT_Pisa SE, all jobs are completed successfully as shown in Figure 6.9. So, for the next tests, T2_IT_Pisa SE will be used as final remote storage. The AsyncStageOut will interact with the FTS server of the T1_IT_CNAF, located in Bologna, since it is the Tier-1 of T2_IT_Pisa.

Next jobs will run in T2_IT_Legnaro since it was seen that jobs, scheduled to

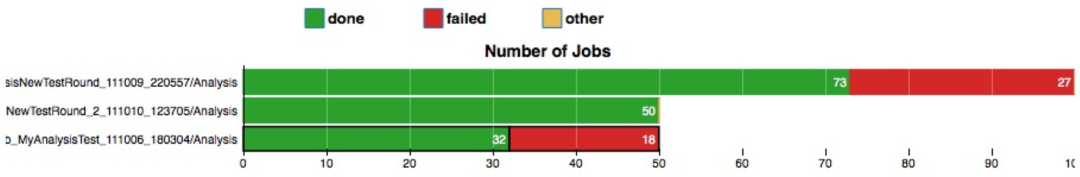


Figure 6.8: Transfers status by workflow.

be executed in this site, do not spend too much time in the batch queue before being effectively executed.

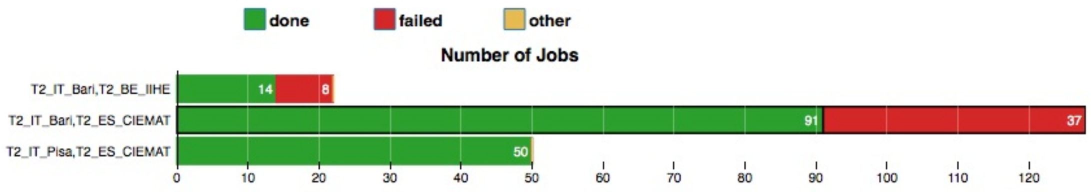


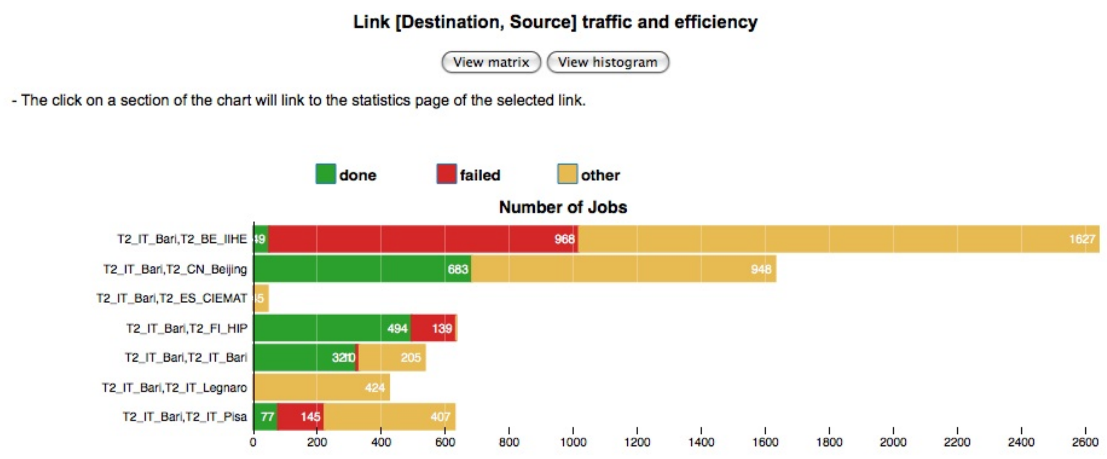
Figure 6.9: Transfers status by site.

Given the issues and bugs met, this test was interrupted. A snapshot of the failure and success transfer rates is shown in Figure 6.10. The bugs will be fixed for the next rounds of tests.

6.2.3 AsyncStageOut scalability

6.2.3.1 Scalability with respect to the number of files

For the first bunch of jobs, the max_files_per_transfer is set to 1000 and the number of users is set to 1, which means that the AsyncStageOut will try to transfer, at each time, all outputs of jobs done for this user.
The first workflow in this first bunch submits 50 jobs, the second one 100 jobs and the third workflow 250 jobs. The latency in transferring the outputs of these jobs are shown in the left of Figure 6.11.


Link [Destination, Source] traffic and efficiency

- The click on a section of the chart will link to the statistics page of the selected link.

Figure 6.10: States of transfers in the basic functionalities test.

In a second step, the same numbers and configurations of jobs for a single user were submitted to CRABServer but this time with 10 as max_files_per_transfer. The latency in transferring the outputs are shown in the right of Figure 6.11.

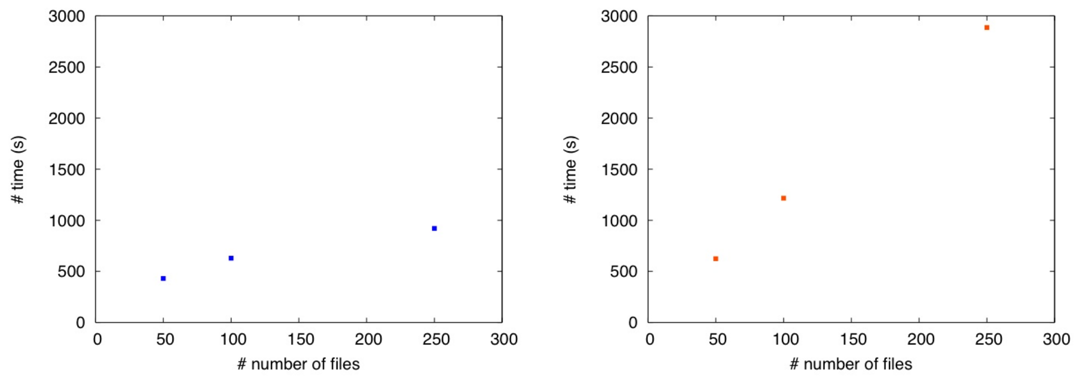


Figure 6.11: The latency in transferring the outputs when max_files_per_transfer = 1000 is shown in the left, while the latency in transferring the outputs when max_files_per_transfer = 10 is in the right.

The plot in the right of Figure 6.11 shows that the AsyncStageOut took more than 500 seconds to transfer 500 files, more than 1000 seconds to transfer 100 files, and ~ 3000 seconds to transfer 250 files. Within this configuration, the latency of transfers is growing almost proportionally with the number of files, which is not the case when the max_files_per_transfer is set to 1000. The plot in the left of Figure 6.11 shows that setting max_files_per_transfer to a high number, 1000 in this case, the latency of transfers grows slowly compared to when it is configured to transfer a low number, such as 10 files.

These tests have allowed to understand the scalability of system regarding the number of files to transfer at time. They have shown that the implementation of FTS allows the AsyncStageOut, configured to transfer an high number of files at time, to scale better than when it is configured to transfer a few files at time.

6.2.3.2 Scalability with respect to the number of users

Within the third bunch of jobs, the max_files_per_transfer is set to 1000, since the previous tests has shown that the AsyncStageOut scales better with this configuration.
The first workflow submits 50 jobs for a single user. Next, 2 workflows created by 2 different users and each workflow creates and submits 50 jobs. Finally, 5 workflows created by 5 different users and each workflow creates and submits 50 jobs. The latencies in transferring the outputs of these jobs are shown in Figure 6.12.

Even if, by AsyncStageOut design, the request transfer of users are submitted to FTS server in parallel, Figure 6.12 does not reflect this and the FTS server seems to manage them sequentially.
This behavior raises an immediate drawback when setting max_files_per_transfer to a large number: such configuration may result in occupying the FTS server for long time when trying to transfer a large number of files and consequently it may introduce latencies to the transfers requested by other activities, such as CMS data production.

It can be also noticed from Figure 6.12 that the latencies to transfer 50 files at time are almost the same. It took between 220 and 250 seconds, which is not the case of the plot in the left of Figure 6.11, where the transfer of 50 files at time took $\sim$ 500 seconds.
This indicates that FTS servers can be subjects to different loads at different times and consequently they provide different performances.

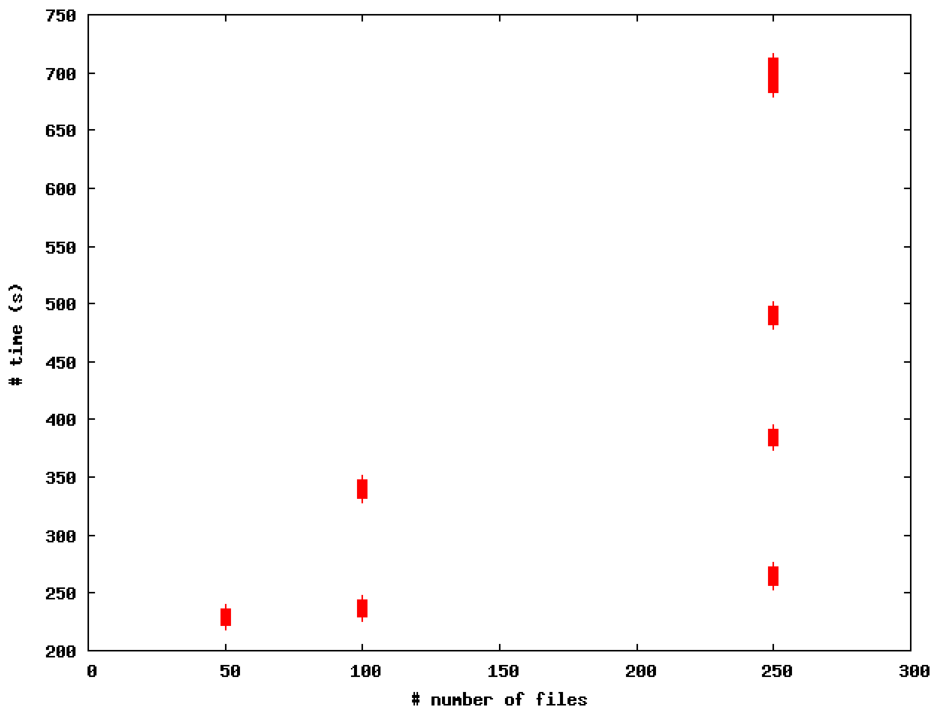


Figure 6.12: Latencies in transferring the outputs by varying the number of files and users.

These scalability tests have allowed to understand how the AsyncStageOut parameters can be tuned to make the tool scaling better in term of users and number of files to transfer. Moreover, they have allowed to understand the behavior of the FTS service. The AsyncStageOut, as component of CRAB 3, has shown satisfactory scalability, which is limited by the the FTS service, namely:

- the implementation of the service itself,
- the configuration of the FTS channel used for the transfers,
- the load on the FTS server at a given time.

6.2.4 Gains compared to the synchronous stage-out approach

The total number of jobs submitted in this test is 23.909 jobs, where 8.414 have succeeded to execute their codes and staged-out the outputs locally while 14.495 have failed. Figure 6.13 shows the success and failure rates by user of the asynchronous transfers of outputs. The failures of jobs in these tests were caused mainly by:

- Various bugs in CRAB 3.
- Missed sanity check in the Client of CRAB making possible the submission of misconfigured workflows to CRABServer, which will fail then to submit them to the Grid.
- Bugs in the configurations of some sites making them incompatible with CRAB 3 workflow.
- Temporary infrastructural problems of sites.

CRAB 3 has been fixed and the integration team has contacted the concerned sites and given them a timescale to fix their configurations.

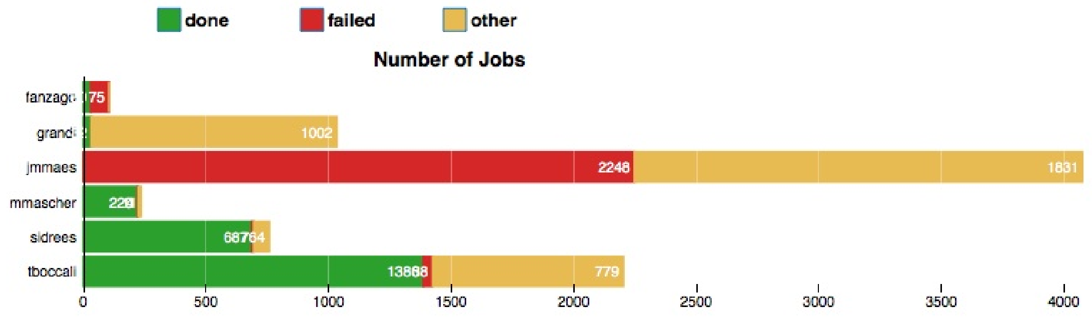


Figure 6.13: Success and failure rates by users.

Based on Figure 3.3, if these jobs were submitted using CRAB 2 and, so, the remote stage-out step is performed synchronously, an approximation of the number of failed jobs for the stage-out from a total of 8.414 would be $\sim$ 223 and the number of succeeded jobs would be $\sim$ 5.545. So, $\sim$ 223 CPU slots would be effectively wasted. Moreover, as in particular shown by the plot (b) of Figure 3.4, 31% of failed jobs for the remote stage-out spend between 3500 - 4000 seconds to perform the stage-out step, making the physicists waiting for that time before the resubmission of their jobs. So in this test, $\sim$ 69 jobs would spend between 3500 - 4000 seconds for the remote stage-out, which means that between $\sim$ 5 - $\sim$ 6 days of delay would be introduced by the synchronous stage-out approach to perform physics analysis.

Regarding succeeded jobs, the asynchronous stage-out approach makes available a first secure copy to users in /store/temp area in the storage of the site where the job was executing. So, physicists can already access their outputs from there. The distribution of the number of jobs done over time intervals spent for the remote stage-out in CRAB 2 is shown in Figure 3.2.2, while the distribution of the number of jobs done during these tests over time intervals spent for the local stage-out is shown in Figure 6.14. If the jobs of this test were submitted using CRAB 2, $\sim$ 95 % of jobs done, which is $\sim$ 5.268 jobs, would spend between 0 - 500 seconds to perform the stage-out, while they required only 0 - 50 seconds to perform the local stage-out in CRAB 3. So, in this test, the asynchronous stage-out approach has allowed to users the access to their outputs before 0 - 450 seconds for $\sim$ 5.268 jobs compared to the synchronous stage-out. Consequently, it reduced the latencies in performing physics analysis by 0 - $\sim$ 27 days.

This approximation of the gain in term of CPUs slots and times seems to be promising. This makes useful to follow up and wait for CRAB 3 production to measure the exact gain in term of time and resources at a real scale.

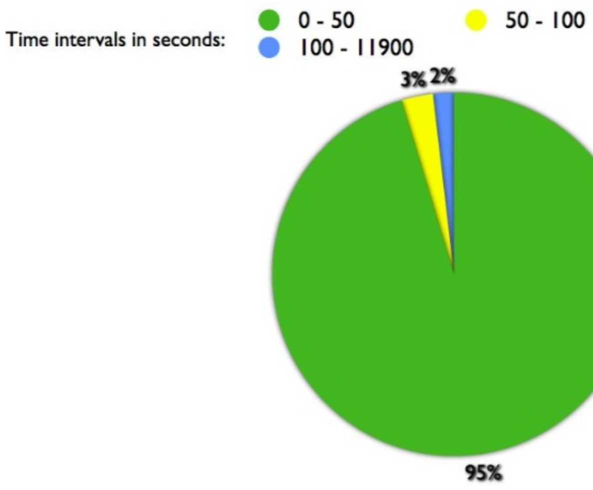


Figure 6.14: The distribution of the number of jobs done during these tests over time intervals spent for the local stage-out.

Conclusions

The subject of the present work is to optimize the design of the distributed analysis workflow in CMS experiment. The presented solutions rely on the design and implementation of tools, Automation and AsyncStageOut, to reduce the latencies in executing the workflow and to optimize its efficiency in using the distributed resources of CMS infrastructure.

An accurate study of distributed systems and in particular the distributed data analysis infrastructure in CMS computing model has been done. Such study has made possible the analysis of the end-user analysis workflow performance.

The Automation system has been used with success to automate the end-user analysis workflows and has shown satisfactory performance when tuned over the highest priority analysis workflow in CMS.

The execution of the beam spot determination workflows represent stringent latency requirements, making its support a critical task for CMS. Even if the Automation system was heavily tested over emulated data, during the technical stop of the LHC in 2009, further optimizations and real time bugs fixes were needed, during the first period of the Automation system deployment.

The new stage-out approach implemented by the AsyncStageOut system has allowed in particular:

- the gain of CPUs times consumed by the analysis job stucking in the remote stage-out step,
- to reduce the delays in executing the analysis workflows by removing the fraction of failures caused previously by the synchronous stage-out step,
- the gain of resources wasted previously by the failures of jobs to perform the remote stage-out synchronously,

- to organize the transfers to Tier-2s using dedicated service to avoid the overload of their networks and storage systems,
- to reduce the latency of the availability of analysis jobs outputs by storing them first in /store/temp/user area of the sites where the jobs were executing.

The AsyncStageOut has been heavily tested by the integration team of CMS and has shown satisfactory performance.

The AsyncStageOut relies on CouchDB as database. Implementing advanced feature using such recent technology has shown to be a not trivial task.

Further developments of this thesis will deal with scale tests and tuning of the Automation tool in Grid environment. Regarding the AsyncStageOut, the monitoring of the system in production will allow to understand better the system behavior, scalability and whether, in particular, is introducing issues to Phedex transfers of CMS production system by overloading FTS servers, and to design and implement then required optimizations. The generic approach adopted to implement the new system, WMCore/WMAgent, will allows to easily spread it once stable to other disciplines.

Appdx A

Event model

CMS uses a number of event data formats with varying degrees of detail, size, and refinement. Starting from the RAW data produced from the online system, the successive levels of processing refine this data, apply calibrations and create higher level physics objects.

Each bit of data in an event must be written in a supported data format. A data format is essentially a C++ class.

The most important data tiers from a physicist's point of view are Raw Data (RAW), Reconstructed Data (RECO) and Analysis Object Data (AOD).

RAW

The first level (L1) of the CMS trigger system, composed of custom hardware processors, runs synchronously with the LHC bunch crossing frequency of 40MHz. It is designed to reduce the incoming data rate to a maximum of 100 kHz, by processing fast trigger information coming from the calorimeters and the muon chambers, and selecting events with interesting signatures. This is further reduced by a factor of 1000 using a high-level trigger (HLT), a software filtering system running HLT selection algorithms on a large processor farm. Output events are written to a temporary disk buffer before being transferred to the computing center (Tier 0) at CERN for offline processing.

Detector data after the High Level Trigger selections are called RAW data.

The RAW data event size is approximately 1 MB. It is processed at CERN, creating new higher level physics objects, RECO.

RECO

RECO is the name of the data-tier which contains objects created by the event reconstruction program. It is derived from RAW data and provides access to reconstructed physics objects for physics analysis in a convenient format.

Event reconstruction is expensive in terms of CPU. It is performed by a central production team, rather than individual users, in order to make eective use of resources and to provide samples with known provenance and in accordance with CMS priorities.

The RECO provides compact information for analysis to avoid the necessity to access to RAW data for most analysis. The algorithms, provided by the CMS event reconstruction software, build physics objects (e.g., muons, electrons, jets) from the RAW data recorded by the detector. All events collected by the CMS trigger system are reconstructed by the CMS prompt reconstruction system soon after being collected. Subsequent processing are carried out in order to incorporate improvements to software algorithms and updated detector calibrations. In addition, a subset of the event reconstruction algorithms contribute in the HLT system.

AOD

AOD are derived from the RECO information to provide data for physics analysis in a convenient and compact format. AOD data are useable directly by physics analyses. The AOD contains enough information about the event to support all the typical usage patterns of a physics analysis. Thus, it contains a copy of all the high-level physics objects (such as muons, electrons, taus, etc.), plus a summary of the RECO information sucient to support typical analysis actions. The AOD size is about 5 times smaller than the next larger (RECO) data format.

Bibliography

[1] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed System: Concepts and Design.* Addison-Wesley, 2005.

[2] Andrew S. Tannenbaum. *Distributed Operating Systems.* Prentice Hall, 1994.

[3] Sanjay Ghemawat, Howard Gobio, and Shun-Tak Leung. *The Google File System.* 19th ACM Symposium on Operating Systems Principles, 2003.

[4] Max K. Goff. *Network Distributed Computing.* Fitscapes and Fallacies Prentice Hall, 2003.

[5] Sun Microsystems. *Lustre networking: High-performance features and flexible support for a wide array of networks.* Sun Microsystems white paper, Version 1.0, 2008.

[6] Grayce M. Booth. *Distributed information systems.* Proceedings of AFIPS National Computer Conference, 789-794, 1976.

[7] Ian F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. *Wireless Sensor Networks: A Survey.* Computer Networks Elsevier Journal, 38:393-422, 2002.

[8] Leslie Lamport. *Time, clocks, and the ordering of events in a distributed system.* Communications of the ACM, 21: 558-565, 1978.

[9] R. Buyya. *High Performance Cluster Computing.* Prentice Hall, 1999.

[10] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann Publishers, 1999.

[11] R. Subramanian and B. Goodman. *Peer-to-Peer Computing: Evolution of a Disruptive Technology*. Idea Group Inc., 2005.

[12] Nadiminti, Dias de Assuno, and Buyya. *Distributed Systems and Recent Innovations: Challenges and Benefits*. InfoNet Magazine, 16(3):1-5, 2006.

[13] Steve Chapin, Mark Clement, and Quinn Snell. *A Grid Resource Management Architecture*.Grid Forum Scheduling Working Group, 1999.

[14] A. S. Grimshaw and W. A. Wulf. *The legion vision of a worldwide virtual computer*. Communications of the ACM, 40(1):39-45, 1997.

[15] I. Foster. *Globus Toolkit Version 4: Software for Service-Oriented Systems*. Journal of Computer Science and Technology, 21:513-520, 2006.

[16] R. Buyya, S. Chapin, and D. Dinucci. *Architectural models for resource management in the grid*. Proceedings of 2nd International Workshop on Grid Computing (Grid '01), 1971:18-35, 2001.

[17] I. Foster and C. Kesselman. *The Globus project: A progress report*. Heterogeneous Computing Workshop, 1998.

[18] K. Hickman and T. Elgamal. *The SSL protocol*. Netscape Communications Corp., Version 3.0, 1995.

[19] J. W. Ross and G. Westerman. *Preparing for utility computing: The role of IT architecture and relationship management*. IBM Systems Journal, 43(1):5-19, 2004.

[20] http://www.nist.gov/index.html

[21] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. Communications of the ACM, 51(1): 107-113, 2008.

[22] John Allspaw and Jesse Robbins. *Web Operations: Keeping the Data On Time*. O'Reilly Media, Inc., 2010.

[23] Ashlee Vance. *Hadoop, a Free Software Program, Finds Uses Beyond Search*. New York Times, 2009.

[24] http://www.openldap.org/

[25] http://lhc.web.cern.ch/lhc/

[26] CMS Collaboration. *LHC Computing Grid (LCG) Technical Design Report*. CERN-LHCC-2005-024, 2005.

[27] http://public.web.cern.ch/public/

[28] http://www.egi.eu/

[29] http://www.opensciencegrid.org/

[30] http://www.euindiagrid.eu/

[31] http://www.euindiagrid.eu/

[32] http://www.eu-emi.eu/

[33] http://www.nordugrid.org/arc/

[34] http://glite.cern.ch/

[35] http://www.unicore.eu/

[36] http://www.globus.org/

[37] http://www.omii.ac.uk/

[38] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. *The Globus Striped GridFTP Framework and Server*. Proceedings SC '05 Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 54-54, 2005.

[39] http://www.dcache.org/

[40] X. Zhang and J. Schopf. *Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2*. Proceedings SC '04 Proceedings of the 2004 ACM/IEEE conference on Supercomputing, 843-849, 2004.

[41] A. W. Cooke, A. J. G. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall and A. Djaoui, *et al. The Relational Grid Monitoring Architecture: Mediating Information about the Grid.* Journal of Grid Computing, 2(4):323-339, 2004.

[42] Karl Czajkowski, Carl Kesselman, Steven Fitzgerald, Ian Foster. *Grid Information Services for Distributed Resource Sharing.* Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), 181-184, 2001.

[43] https://twiki.cern.ch/twiki/bin/view/EGEE/BDII

[44] P Andreetto *et al. The gLite workload management system.* Journal of Physics Conference Series **119** 062007 doi: 10.1088/1742-6596/119/6/062007, 2008.

[45] R.-D. Heuer. *Electron-Positron colliders.* Proceedings of the XX International Symposium on Lepton and Photon Interactions at High Energies, 2001.

[46] http://public.web.cern.ch/public/

[47] Samuel Reich E. Erratum. *Tevatron faces final curtain.* Nature, 469(7330):280, 2011.

[48] http://www.fnal.gov/

[49] W. N. Cottingham and D. A. Greenwood. *An introduction to the standard model of particle physics, Second Edition.* Cambridge University Press, 1998.

[50] ALEPH, DELPHI, L3 and OPAL Collaborations, The LEP Working Group for Higgs Boson Searches. *Search for Neutral Higgs Bosons at LEP'.* Eur.Phys.J.C47:547-587, hep-ex/0602042, 2006.

[51] http://lhc.web.cern.ch/lhc/

[52] http://atlas.web.cern.ch/Atlas/Collaboration/

[53] The CMS Collaboration, Adolphi R, *et al. The CMS experiment at the CERN LHC.* JINST 3 S08004, 2008.

[54] http://lhcb.web.cern.ch/lhcb/

[55] http://aliweb.cern.ch/

[56] http://www.cern.ch/MONARC/

[57] O Buchmller, *et al. The CMS CERN Analysis Facility (CAF).* Journal of Physics Conference Series **219** 052022 doi: 10.1088/1742-6596/219/5/052022, 2010.

[58] Christopher Z. Mooney. *Monte Carlo simulation.* Sage Publications, Inc., 1997.

[59] K Gopalakrishnan. *Oracle Database 11g Real Application Clusters Handbook, Second Edition.* McGraw-Hill Companies, Inc., 2012.

[60] Afaq A, *et al. The CMS dataset bookkeeping service.* Journal of Physics Conference Series **119** 072001, 2007.

[61] http://www.mysql.com/

[62] http://www.sqlite.org/

[63] J. Rehn, *et al. PhEDEx high-throughput data transfer management system.* Proceedings of Computing High Energy Physics, 2006.

[64] http://www.json.org/

[65] Wakfield S, *et al. Large Scale Job Management and Experience in Recent Data Challenges within the LHC CMS experiment.* Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research, 2008.

[66] Spiga D, *et al. The CMS Remote Analysis Builder (CRAB).* 14th Int. Conf. on High Performance Computing (Goa, India), 4873:580-586, 2007.

[67] S. Metson, *et al. SiteDB: Marshalling the people and resources available to CMS.* Proceedings of Computing in High Energy and Nuclear Physics (CHEP), 2009.

[68] LSF web site http://www.platform.com/Products/platform-lsf

[69] Brett Bode, David M. Halstead, Ricky Kendall, and Zhou Lei, *et al. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters.* Proceedings of 4th Annual Linux Showcase and Conference, 27-27, 2009.

[70] G. Codispoti, *et al. Use of the gLite-WMS in CMS for production and analysis.* Journal of Physics Conference Series **219** 062007 doi:10.1088/1742-6596/219/6/062007, 2009.

[71] http://www.w3.org/TR/soap/

[72] Julia Andreeva, *et al. Experiment Dashboard for Monitoring Computing Activities of the LHC Virtual Organizations.* Journal of Grid Computing, 8(2):323-339, 2010.

[73] J. Spalding, Collaboration for two CMS working groups: the Machine Interface Group and Trigger Simulation Group. *Background Issues for CMS.* LHC Workshop on Experimental Conditions and Beam-Induced Detectors Backgrounds, CERN, 60-64, 2008.

[74] LSF web site http://www.platform.com/Products/platform-lsf

[75] Presti L, *et al. CASTOR: A Distributed Storage Resource Facility for High Performance Data Processing at CERN.* Proceedings of the 24th IEEE Conf. on Mass Storage Systems and Technologies (IEEE Computer Society), 275-280, 2007.

[76] https://j2eeps.cern.ch/service-lsfweb/home/index

[77] Erik T. Ray. *Learning XML.* O'Reilly Media, Inc., 2003.

[78] Craig Larman. *UML 2 et les Design Patterns.* Pearson Education, 2005.

[79] Frohner A, *et al. Data management in EGEE.* Journal of Physics Conference Series **219** 062012, doi: 10.1088/1742-6596/219/6/062012, 2009.

[80] Abadie L, *et al. Storage Resource Managers: Recent International Experience on Requirements and Multiple Co-Operating Implementations.* Proceedings of 24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007), 47-59, 2007.

[81] Allcock B, *et al. Data Management and Transfer in High-Performance Computational Grid Environments.* Parallel Computing Journal, 28:749-771, 2001.

[82] Jon Erickson. *HACKING the art of exploitation, 2nd Edition.* No Starch Press, 2008.

[83] http://python.org/

[84] http://www.apache.org/

[85] http://www.erlang.org/

[86] Mary Shaw and Paul Clements. *Toward boxology: preliminary classification of architectural styles.* Proceeding in ISAW '96 Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops, 50-54, 1996.

[87] http://www.json.org/

[88] David Flanagan. *JavaScript: The Definitive Guide.* O'Reilly Media, 2001.

[89] Frédéric MAGOULéS,Yann LE BIANNIC. *Executing multiple group-by query in a MapReduce approach.* Proceedings in Communication Systems, Networks and Applications (ICCSNA), 38-41, 2010.

[90] Knuth Donald. *The Art of Computer Programming: Sorting and Searching.* Addison-Wesley, 1973.

[91] Subbu Allamaraju. *RESTful Web Services Cookbook.* O'Reilly Media / Yahoo Press, 2010.

[92] Edsger W. Dijkstra. *A note on two problems in connexion with graphs.* Numerische Mathematik, 1:269-271, 1959.

[93] Kuznetsov V, *et al. The CMS Data Aggregation System.* Procedia Computer Science, 1:1529-1537, 2010.