EGI-InSPIRE

SERVICES FOR HIGH ENERGY PHYSICS

EU MILESTONE: MS610

Document identifier:	EGI-doc-540-V2.docx
Date:	**12/08/2011**
Activity:	SA3
Lead Partner:	EGI.eu
Document Status:	FINAL
Dissemination Level:	PUBLIC
Document Link:	https://documents.egi.eu/document/540

Abstract

The computing systems of the LHC experiments at CERN are probably the most complex Grid-integrated applications currently in production. This milestone describes the critical services on which these computing systems are based and how they interact with each other. This description represents the current state of the art in the high energy physics community. This document revises and replaces MS603 (EGI-doc-147-V5.doc).

I. COPYRIGHT NOTICE

II. DELIVERY SLIP

	Name	Partner/Activity	Date
From	Andrea Sciabà	CERN	14/7/2011
Reviewed by	Moderator: John Gordon Reviewers: Antun Balaz	STFC IPB	10/8/2011
Approved by	AMB & PMB		11/8/2011

III. DOCUMENT LOG

Issue	Date	Comment	Author/Partner
1	14 July 2011	First draft	CERN IT-ES group
2	10 Aug 2011	Second draft	CERN IT-ES group
3	11 Aug 2011	Final	CERN IT-ES group

IV. APPLICATION AREA

This document is a formal deliverable for the European Commission, applicable to all members of the EGI-InSPIRE project, beneficiaries and Joint Research Unit members, as well as its collaborating projects.

V. DOCUMENT AMENDMENT PROCEDURE

Amendments, comments and suggestions should be sent to the authors. The procedures documented in the EGI-InSPIRE "Document Management Procedure" will be followed: https://wiki.egi.eu/wiki/Procedures

VI. TERMINOLOGY

A complete project glossary is provided at the following page: http://www.egi.eu/about/glossary/.

VII. PROJECT SUMMARY

To support science and innovation, a lasting operational model for e-Science is needed − both for coordinating the infrastructure and for delivering integrated services that cross national borders.

The EGI-InSPIRE project will support the transition from a project-based system to a sustainable pan-European e-Infrastructure, by supporting 'grids' of high-performance computing (HPC) and high-throughput computing (HTC) resources. EGI-InSPIRE will also be ideally placed to integrate new Distributed Computing Infrastructures (DCIs) such as clouds, supercomputing networks and desktop grids, to benefit user communities within the European Research Area.

EGI-InSPIRE will collect user requirements and provide support for the current and potential new user communities, for example within the ESFRI projects. Additional support will also be given to the current heavy users of the infrastructure, such as high energy physics, computational chemistry and life sciences, as they move their critical services and tools from a centralised support model to one driven by their own individual communities.

The objectives of the project are:

1. The continued operation and expansion of today's production infrastructure by transitioning to a governance model and operational infrastructure that can be increasingly sustained outside of specific project funding.
2. The continued support of researchers within Europe and their international collaborators that are using the current production infrastructure.
3. The support for current heavy users of the infrastructure in earth science, astronomy and astrophysics, fusion, computational chemistry and materials science technology, life sciences and high energy physics as they move to sustainable support models for their own communities.
4. Interfaces that expand access to new user communities including new potential heavy users of the infrastructure from the ESFRI projects.
5. Mechanisms to integrate existing infrastructure providers in Europe and around the world into the production infrastructure, so as to provide transparent access to all authorised users.
6. Establish processes and procedures to allow the integration of new DCI technologies (e.g. clouds, volunteer desktop grids) and heterogeneous resources (e.g. HTC and HPC) into a seamless production infrastructure as they mature and demonstrate value to the EGI community.

The EGI community is a federation of independent national and community resource providers, whose resources support specific research communities and international collaborators both within Europe and worldwide. EGI.eu, coordinator of EGI-InSPIRE, brings together partner institutions established within the community to provide a set of essential human and technical services that enable secure integrated access to distributed resources on behalf of the community.

The production infrastructure supports Virtual Research Communities (VRCs) − structured international user communities − that are grouped into specific research domains. VRCs are formally represented within EGI at both a technical and strategic level.

VIII. EXECUTIVE SUMMARY

The physics experiments using the Large Hadron Collider (LHC) facility located at CERN are running the vast majority of their offline distributed computing activities on the infrastructure provided by the Worldwide LHC Computing Grid (WLCG). The LHC collider has been operational since late 2009, but the computing systems were already fully commissioned in 2008.

In this document, first we classify the computing services that serve as basic components of the computing systems. These services can be classified in four main categories:

1. Experiment services;
2. Middleware services;
3. Fabric services;
4. Infrastructure services.

The most important among these services are explicitly listed.

Then, we describe for each of the LHC experiments the high-level services, which provide most – if not all – of the functionality available to the end users. For each of them we summarize the main features (scope, dependencies, interfaces) and give some insight of their architecture and their implementation.

Finally we conclude with a description of the main high-level middleware services provided by external projects (EMI, OSG, Condor, NorduGrid, etc.) and explain their function and their role in the LHC computing systems.

Table of Contents

1 INTRODUCTION 7

2 OVERVIEW 8

3 EXPERIMENT COMPUTING SYSTEMS AND SERVICES 10

3.1 ALICE 10

3.1.1 AliEN 11

3.2 ATLAS 12

3.2.1 PanDA 13

3.2.2 Distributed Data Management 13

3.2.3 PanDA Dynamic Data Placement (PD2P) 14

3.3 CMS 14

3.3.1 CRAB 15

3.3.2 CRAB Analysis Server 16

3.3.3 ProdAgent 16

3.3.4 WMAgent 16

3.3.5 PhEDEx 17

3.3.6 DBS 17

3.3.7 Data Popularity 18

3.4 LHCb 18

3.4.1 DIRAC 19

4 MIDDLEWARE SERVICES 21

4.1 Data Management 21

4.1.1 LCG File Catalogue 21

4.1.2 File Transfer Service 21

4.2 Workload Management 21

4.2.1 Ganga 22

4.2.2 Condor-G 22

4.2.3 gLite Workload Management System 22

4.2.4 GlideinWMS 23

4.3 Persistency 23

4.3.1 CORAL 23

4.3.2 POOL 23

4.3.3 COOL 24

4.3.4 Frontier/Squid and CORAL server/proxy 24

4.4 Monitoring 24

4.4.1 Experiment Dashboard 24

4.4.2 SAM/Nagios 25

4.4.3 HammerCloud 25

5 SERVICES SUPPORTED BY SA3 26

5.1 Common Solutions with EGI-InSPIRE Involvement 26

5.2 Services and Operations 26

6 CONCLUSION 28

7 APPENDIX 29
8 REFERENCES 30

1 INTRODUCTION

The goal of this document is to provide a concise yet complete description of the distributed systems that the High Energy Physics experiments at CERN use for their offline computing, which includes, in very general terms, data processing, data analysis and event simulation. In particular the services used by these systems are briefly described together with their relative dependencies. This document can therefore serve as a reference for the current status of the LHC computing and its basic components.

This document focuses on describing the main experiment services used for their distributed computing activities and the high-level middleware services. It does not cover:

1) experiment computing services not related to distributed activities (for example, prompt reconstruction at the CERN Tier-0)
2) low-level middleware services (computing and storage elements, VOMS and MyProxy servers, etc.), short of mentioning them where appropriate.

2 OVERVIEW

The computing systems of the LHC experiments can be seen as some of the most complex distributed data processing systems, both from an architectural point of view and for their scale. In fact, each of them is able to process several Petabytes of data each year and serve thousands of users; finally they are truly distributed on a worldwide level and are integrated with several Grid infrastructures and middleware stacks.

Although each of them was developed independently, they inevitably address and implement similar use cases and functionality and rely on some underlying services, in particular those belonging to the middleware layer. Therefore, all the computing systems can be represented by the schema in Figure 1, showing a layered service stack, having at its lowest levels the basic, non-Grid aware computing services typically provided by a computer centre.

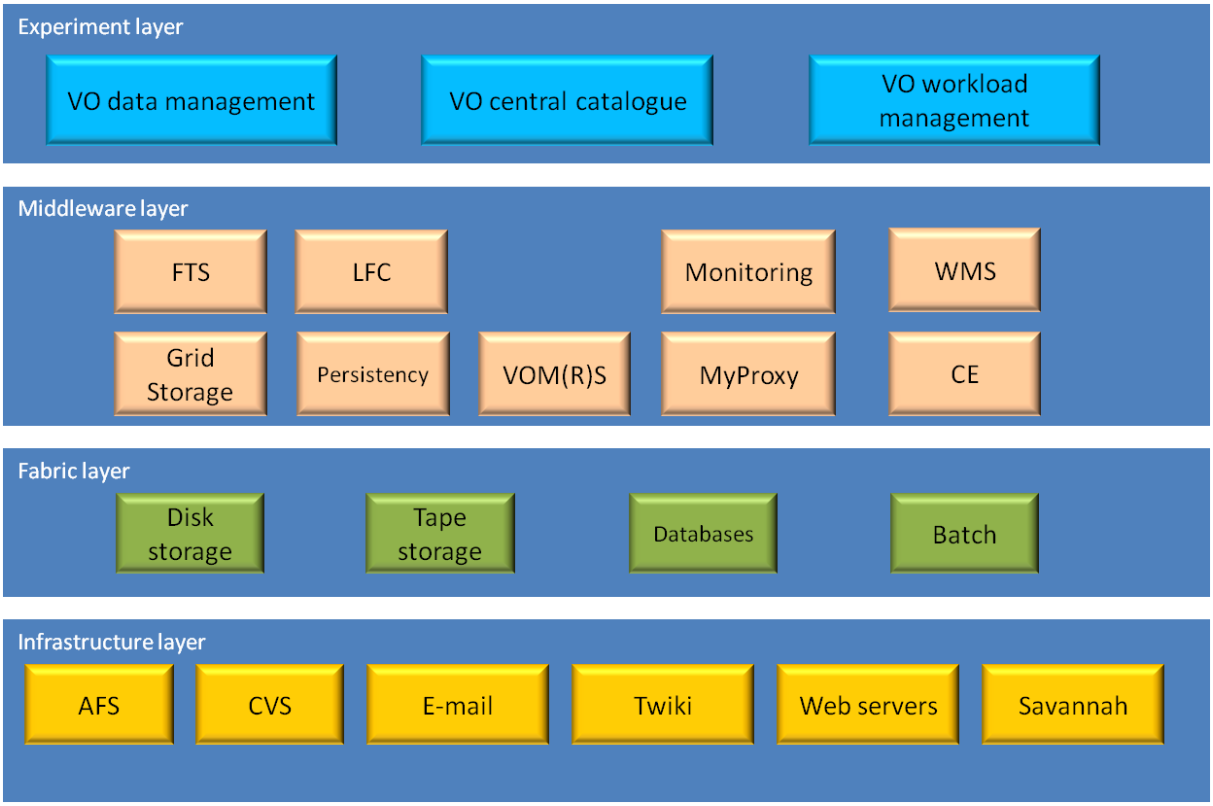

Figure 1. Global service architecture.

We therefore classify services in the following categories:

1. **Experiment services**: These are services developed, maintained and operated by the collaborations themselves; providing functionality very specific to the experiment applications. They are, by experiment:
   a. ALICE: AliEN
   b. ATLAS: PanDA, DDM, PanDA PD2P
   c. CMS: CRAB, Analysis Server, Production Agent, PhEDEx, DBS, Data Popularity

d. LHCb: DIRAC

2. **Middleware services**: These are generic services at the Grid middleware layer, providing high-level but application-independent functionality, used by one or more experiments (but typically also by a wide spectrum of VOs outside HEP). They are developed by a variety of Grid projects and software providers and are typically operated by the WLCG in the LHC context. They include:
   a. Data management services: LFC, FTS
   b. Workload management services: *Ganga*, *Condor-G*, *gLite WMS*, *glideinWMS*
   c. Persistency services: *CORAL*, *POOL*, *COOL*, *Frontier*
   d. Monitoring services: HammerCloud, Experiment Dashboard, Nagios
   e. Security services: VOMS, VOMRS, MyProxy
   f. Computing elements: LCG CE, CREAM CE, OSG CE, ARC CE
   g. Storage elements: CASTOR, dCache, DPM, *XrootD*, StoRM, BeSTMan
3. **Fabric services:** These are fabric-related services operated by the sites and include:
   a. Batch systems: LSF, PBS, Torque/Maui, Condor, etc.
   b. Tape systems: CASTOR, TSM, Enstore, HPSS, etc.
   c. Disk servers or distributed file systems: GPFS, Lustre, AFS, NFS, etc.
   d. Database services: Oracle, Oracle Streams, MySQL, PostgreSQL, etc.
4. **Infrastructure services:** These are utility services which are not part of the experiment computing systems but are anyway important for their operations. Examples include: documentation services, web services, bug tracking systems, etc. and are not further mentioned here.

The services described in detail in this document are those written in italics in the previous list.

3 EXPERIMENT COMPUTING SYSTEMS AND SERVICES

All the computing systems of the LHC experiments are based on a variety of services developed inside the collaborations. Typically these services provide functionalities very specific to the experiment and strongly coupled to the computing and data model; however, in some cases two or more experiments have developed similar services which could possibly have been implemented as a common generic service. There are some possible reasons for this:

a) the experiment needed a service on a timescale incompatible with the one of the Grid projects;
b) the development cycle needed to be much faster than it was possible within a Grid project;
c) the experiment requirements were not fully compatible with those of the user communities served by the Grid project.

Nevertheless, there are a number of areas where common solutions have been developed. These include those the pre-date EGI-InSPIRE: the (W)LCG Persistency Framework – POOL, COOL, CORAL and CORAL server, Ganga and the Experiment Dashboards being the main examples. These are all described in detail below (see sections 4.2.1, 4.2.14.3 and 4.4.1). In addition, there are common solutions that have been made possible through EGI-InSPIRE. Whilst these are also described below (see sections 4.4.3 and 3.3.7), we include a short summary in section 5.1 to highlight those areas to which EGI-InSPIRE has contributed.

In this section the main experiment-specific services are briefly described. These services are typically (but not always) used by a single experiment, although, given the similarities of the use cases of each VO, the functionalities they provide are similar.

Rather than giving a fully comprehensive description of all services, we choose to concentrate on those providing the workload management and the data management and transfer functionalities, which are undoubtedly the most important in a distributed environment.

In the next sections, we will first summarize the experiment-specific services and describe their main dependencies, and then provide a more detailed description.

3.1 ALICE

In the case of ALICE, computing system is fully integrated and based on a single framework with a limited number of external dependencies.

ALICE framework

Problem area	Service	Depends on	Interfaces
Workload management	AliEN	CE	Web, CLI
Data management	AliEN	SE, xrootd	Web, CLI
Data Catalogue	AliEN	Database	Web, CLI, API
Security	AliEN	MyProxy, VOMS	API
Monitoring	MonALISA		Web, API

3.1.1 AliEN

AliEN [R1] is the set of middleware tools and services developed and used by ALICE and other collaborations for data production and analysis in the Grid. The ALICE computing services are summarized in Figure 2 and their most important components are:

a) the file catalogue
b) a data management system based on xrootd and the File Transfer Daemon
c) a workload management system
d) authorization services
e) a monitoring system based on MonALISA

The **file catalogue** service provides the mapping from the logical file name (LFN) to one or more physical file names (PFN), with an interface that resembles a UNIX file system. It is used to record all data, including software packages used for data production and analysis. Furthermore, it supports file collections as user-defined lists of entries and arbitrary metadata information. The file catalogue is built on top of a relational database and accessed via several interface layers (the AliEN DB interface, a generic Perl DB interface and a specific DB driver). Each branch in the catalogue directory tree can, in principle, be supported by different RDBMS engines running on different hosts.

The AliEN data management allows to remotely accessing any file by automatically resolving a LFN into the "best" PFN given the client location. Direct access is handled via the xrootd protocol, while scheduled transfers are run via the **File Transfer Daemon** (FTD). All storage systems used by AliEN are required to support the xrootd protocol (as is the case for those used in WLCG).

The workload management system is based on the so-called *pull* approach. A central **Task Queue** contains all the submitted jobs, while on each site a **Computing Element** (CE) service advertises its capabilities. A **Job Broker** tells eligible CEs to start **Job Agents** (an implementation of the "pilot job" concept) and then sends the jobs to them, by taking into account job requirements such as the input files needed, the CPU time, the operating system architecture, the amount of needed disk space and the user and group quotas. At the end of each job, the Job Agent takes care to register the job output files in the file catalogue.

Security is provided by the **Authorization Service** and the **Database Proxy Service**. The authentication service is implemented using SASL and uses GSSAPI via a Perl module based on Globus GSI, which allows it to use various authentication mechanisms (token, RSA key, X.509 certificates) as well as Grid proxy certificates. Once authenticated, the user is given a database token which allows him to connect and identify himself to the database engine using the Database Proxy service.

All the AliEN monitoring is based on the MonALISA framework [R2], which is used to collect and aggregate all relevant information about jobs, resources and services. The MonALISA information repository can also be used to take automatic actions depending on the information received.

AliEN is coded in Perl, mainly because of the wide availability of reusable Open Source modules, which provide a complete security support, a full featured SOAP platform and easy Web integration.

The user interacts with the Web Services via SOAP messages and the Web Services constantly exchange messages between themselves acting as a web of collaborating services.

Finally, AliEN is interfaced to several Grid middleware implementations, namely all those used in WLCG: gLite, VDT and ARC.

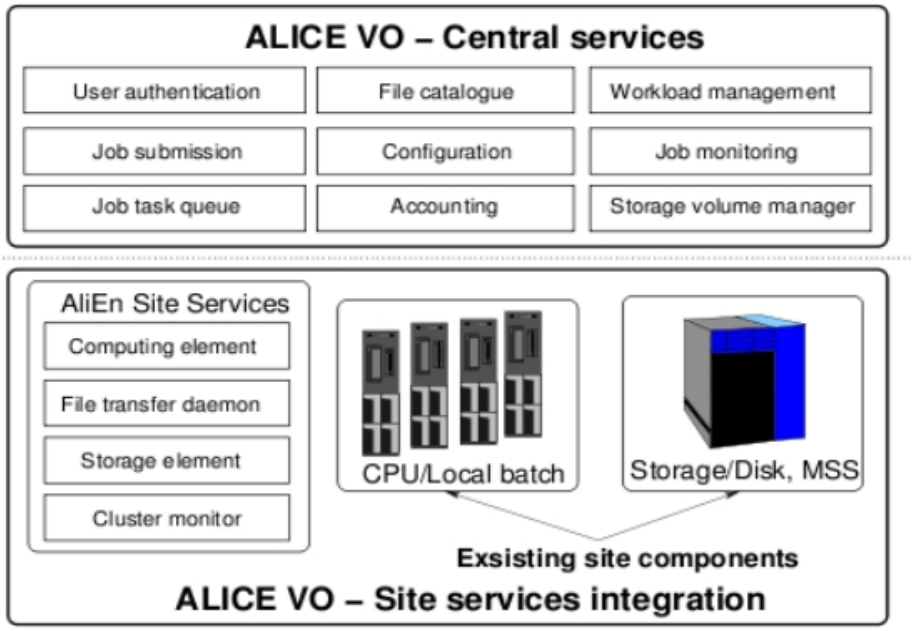

Figure 2. ALICE computing services.

3.2 ATLAS

The ATLAS computing system is built on two high-level services: PanDA, the workflow management system used for both production and analysis jobs, and DDM, the data management system. PanDA is used also as a "back-end" for Ganga, a generic job management framework that will be described later; Ganga is also used to directly send jobs to the gLite WMS as back-end, although this method is gradually being phased out. In this section we describe their architecture and functionality and their relationship with the underlying Grid services.

ATLAS framework.

Problem area	Service	Depends on	Interfaces
Workload management	Ganga	DDM, PanDA, CE, gLite WMS, VOMS, MyProxy	API
	PanDA	DDM, CE, Condor-G, VOMS, MyProxy	Web, CLI, API
Data management	DDM	FTS, SE, VOMS	Web, CLI, API
Data Catalogue	DDM	LFC, Oracle	CLI, API
Monitoring	Dashboard		Web, API
	Panda monitor		Web
	DDM monitoring		Web

3.2.1 PanDA

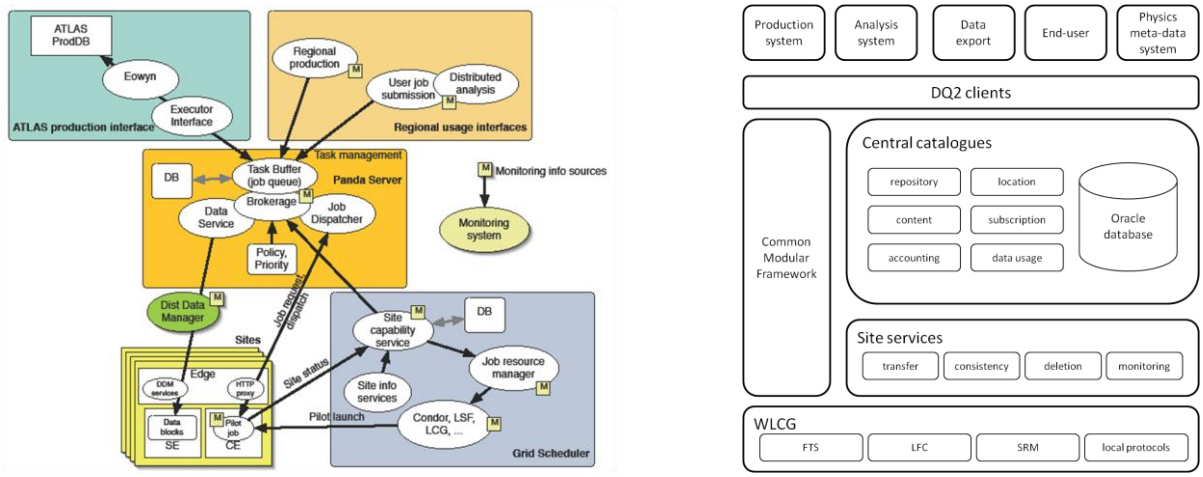

Figure 3. (a) PanDA architecture; (b) DDM architecture.

The **Production and Distributed Analysis** (PanDA) system (Figure 3a) is a Grid workload management system developed by the ATLAS collaboration [R3]. As in the case of AliEN, the system is built around the concept of "pilot jobs": Grid jobs (workloads) are submitted to a **Task Buffer** and generic pilots, already running on a worker node, retrieve these jobs and execute them. The PanDA server implements fairshare policies and priorities and assigns work via a **brokerage** module, while pilots contact the job dispatcher to request a job to run; this mechanism allows reducing job latency and increases efficiency and throughput.

Pilots are submitted to Grid CEs from multiple pilot factories developed around Condor-G. The PanDA server and clients are implemented in Python to allow trivial portability across operating systems and architectures. The server maintains its state in an Oracle database.

PanDA is tightly coupled with the ATLAS Distributed Data Management system described in the next section: this integration enables PanDA to replicate datasets to sites before jobs are submitted.

PanDA end-user clients (**pathena** and **Ganga**) are used by physicists to package and send user jobs to the PanDA server, while production jobs are submitted via a dedicated interface. Finally, PanDA provides a web-based monitoring tool that is used by users and operators to track the progress of the grid jobs. Since late 2006 PanDA has processed over 269 million jobs; in particular, during 2010 PanDA processed 110 million jobs and in the first half of 2011 it processed over 80 million jobs.

3.2.2 Distributed Data Management

The ATLAS **Distributed Data Management** (DDM) project (Figure 3b) is responsible for the replication, access and bookkeeping of ATLAS data across the participating WLCG sites [R4]. It also enforces data management policies defined in the ATLAS Computing Model and provides a central link between the WLCG and ATLAS analysis components.

To ensure the DDM scalability and fault tolerance, the core of the system has been designed as a set of independent clients and services. One of the main components of the system are the **Central Catalogues**, which hold the information about which datasets exist in the system (repository), their composition (content), where they are located (location), which replication requests have been submitted (subscription), how often the datasets are accessed (data usage); finally an *accounting* catalogue contains information such as the amount of data existing at each site and metadata.

The DDM **Site Services** are the software agents that take care of the transfer requests, of the deletion of datasets, of finding and fixing consistency issues and of recording monitoring information.

At the lowest level DDM is interfaced to the WLCG data management and storage services: FTS to run file transfer jobs, LFC to implement the local dataset catalogue and SRM to remotely access and write files to storage.

The DDM interface to external components is implemented by the **DQ2 Clients** that allow users, production and analysis systems to interact with DDM. ATLAS DDM is currently managing 64 PB of data and has achieved aggregated transfer rates of over 10 GB/s.

3.2.3 PanDA Dynamic Data Placement (PD2P)

The early distribution of ATLAS data was defined in the ATLAS Computing Model in an overly generous way in order to guarantee the persistency of the data, but also to facilitate the analysis of the data and its access to the users. However, by examining the data usage statistics, it was noticed that a large fraction of the replicated data was never accessed and thus the usage of network and storage resources was done in a suboptimal fashion: uninteresting data was being preplaced in the same way as interesting data.

The evolution of the early pre-placement model is known as the PanDA Dynamic Data Placement. This new algorithm defines a minimal fraction of pre-placement to Tier1s according to their agreed share in order to guarantee the persistency of the data. The workload management system dynamically triggers further replication of datasets that have been recently accessed. The destination sites are chosen by the probability to run promptly a job on the new replica.

3.3 CMS

The CMS offline computing system includes a large number of services, but for the purpose of this document we will focus on those which are closer to the Grid infrastructure: the workload and data management systems and the data catalogue. Other services, such as the Tier-0 production and monitoring system, are not covered as they are inherently non-distributed. In the table below the WMAgent is mentioned, the WMAgent is a replacement for ProdAgent, thus it is used by the same community.

CMS Key Grid infrastructure services framework.

Problem area	Service	Depends on	Interfaces
Workload management	CRAB	DBS, Analysis server	CLI
	Analysis server	DBS, CE, gLite WMS, Condor-G, Condor glideins, MySQL	API
	ProdAgent	DBS, CE, gLite WMS, Condor-G, Condor glideins, MySQL	CLI
	WMAgent	DBS, CE, gLite WMS, Condor-G, Condor glideins, MySQL, CouchDB	Web, CLI, API
Data management	PhEDEx	DBS, FTS, SE	Web

Data Catalogue	DBS	Oracle, MySQL, SQLite	Web, CLI, API
Monitoring	PhEDEx		Web, API
	Dashboard	Oracle	Web, API
	Data Popularity	Oracle	Web, API

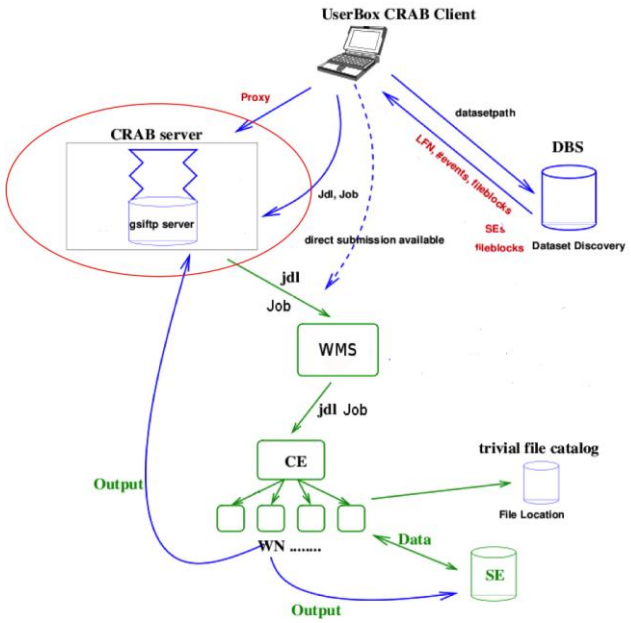

Figure 4. A workflow diagram of the CMS user analysis system.

3.3.1 CRAB

The CMS Remote Analysis Builder (CRAB) is a tool to allow users to run analysis jobs over distributed datasets and collect the results by hiding as far as possible the complexity of the underlying system [R5]; it can be used to execute jobs both on CERN local resources (as the CMS Analysis Facility at CERN) and remote (the WLCG sites).

As shown in Figure 4 the interaction with the Grid can be either direct, leaving to the user tasks such as job submission, status check and output retrieval, or via a **CRAB Analysis Server** (see next section). The direct interaction requires a very small set of operations by the user, but full workflow automation can be reached only using the server, leaving to the user just the task of preparing a configuration file, while the server will notify the user when the analysis is completed.

The user specifies the dataset to be analysed and CRAB queries the **Database Bookkeeping System** (DBS), which is the CMS data catalogue, to resolve a list of sites hosting the dataset; then the analysis task can be split in several jobs which are submitted to eligible sites. Finally, when the jobs are finished, their output is retrieved to the user local host and the produced data is remotely copied to an appropriate site and published in the DBS.

CRAB is implemented in Python as a batch-like command-line application. In order to interact with the Grid middleware and with the CMS analysis software (CMSSW), CRAB must be installed on a Grid user interface where CMSSW is also available. CRAB uses an SQLite database for logging purposes. CRAB transparently interacts with all middleware flavours in WLCG (gLite, VDT and ARC) and can use ad back-end the gLite WMS, Condor-G, glideinWMS and several batch systems.

During 2011 the CRAB user community exceeded one thousand with a submission rate greater than 200,000 jobs per day and it is stable around 1200 unique users per month.

3.3.2 CRAB Analysis Server

The purpose of the Analysis Server is to fully automate the workflow management, leaving to the user just the preparation of the configuration file and notifying him of the output availability [R5]. It also allows implementing complex workflows, such as the possibility to automatically schedule analysis jobs on new data as soon as it appears in the DBS.

The server architecture is completely modular and shared with the old CMS production core system (ProdAgent, see next section).Thanks to its design model, the Analysis Server is comprised of a set of independent components implemented as daemons and communicating asynchronously through a shared messaging service supporting the publish/subscribe paradigm.

Most of the server components are multi-threaded to allow a multi-user scalable system and to avoid bottlenecks in the most intensive and slower operations such as job (re)submission, job status tracking and output handling. The status of the server is defined in a MySQL database.

A crucial element of the Analysis Server architecture is an external Storage Element where user input and output data are stored.

A completely new implementation of the Analysis Server is near completion. It represents one of the specializations of WMCore, the common data and workload management framework developed by CMS in the recent years with the aim to improve the sustainability of the overall system.

3.3.3 ProdAgent

The ProdAgent is the system used to manage all production activities (simulation, reconstruction and skimming) and was designed aiming at automation, scalability, absence of single points of failure and support for different Grid middleware. In addition, ProdAgent is integrated with the CMS event data model, data management system and data processing framework (CMSSW).

The ProdAgent interacts with the data management system to discover data to be processed or to register produced data. Input data are read directly from the local storage system using the appropriate local I/O access protocol and output data are staged out into the local storage system. The CMS data transfer and placement system, PhEDEx, takes care of harvesting production files and transferring them to the appropriate sites.

ProdAgent is implemented as a set of loosely coupled Python daemons that communicate through a MySQL database. As for the Analysis Server, components use an asynchronous publish/subscribe model for communication and their states are defined in the database. Scaling is achieved by running in parallel several ProdAgent instances, where every instance makes use of a local ProdAgent MySQL database for operation and monitoring of the components as well as a local DBS instance for data bookkeeping. Produced data are published into the data transfer system database and into the global DBS instance to make them available for transfer and to the collaboration for analysis.

The ProdAgent is being gradually phased out and replaced by the WMAgent.

3.3.4 WMAgent

The WMAgent represents another example of a specialization of the WMCore framework.

In term of objectives and roles the WMAgent project does not have any main difference with respect to the ProdAgent, but the implementation is completely different.

In the WMAgent the core framework and the adopted technologies were reviewed with the main purpose to solve some known issues and to improve the overall performance of the system. Although the distributed nature of the system and the modular approach has been preserved, the various

components are not communicating any more via a MySQL-based message service but they use a well-defined Job State Machine, which prevents messages from being lost.

Another important change which improves the overall performance is the adoption of CouchDB, a NoSQL database that it is fully integrated with an HTTP framework and also exposes a RESTful based interface.

Another new feature of the system indeed is the usage of the RESTful-based interface to expose APIs and to enable the communication between the various pieces of the distributed framework.

The WMAgent is assessed to be of pre-production quality and the integration phase is close to finish in the next months.

3.3.5 PhEDEx

The **Physics Experiment Data Export** (PhEDEx) is a software project started by the CMS experiment in 2004 to reliably manage by a simple mechanism large-scale data transfers and data placement policies across the Grid [R5]. It was derived from a prototype, developed for the DC04 CMS data challenge, which consisted in a number of agents using a central database (the TMDB, see below) as a central blackboard. It is notable how some of the PhEDEx architectural choices were later adopted in the development of the gLite File Transfer System.

In PhEDEx, data transfers are requested by specifying only the destination storage area, while the source is selected using an algorithm which calculates the path of least cost, determined from the recent history of the corresponding link. This allows to automatically balance the load and to be fault-tolerant in case a link becomes unavailable.

PhEDEx is based on a high-availability Oracle database cluster hosted at CERN (**Transfer Management Data Base**, or TMDB) acting as a "blackboard" for the global system state, including the data location and the current tasks.

Furthermore, PhEDEx is composed by software daemon processes or agents implemented in Perl, which contact the central database to retrieve their work queue.

A set of service agents run centrally at CERN, while each site runs the agents that directly interact with the local storage to execute file transfers to the site (usually by submitting a job to FTS), for file deletion and to run on-demand data consistency checks.

Finally, PhEDEx provides two interfaces for data operations management (transfer request and approval), and for transfer and activity monitoring: a web site implemented in HTML and JavaScript as an interactive interface, and a Web Data Service using Apache, for the upload into and retrieval from TMDB of data in machine-readable formats such as XML and JSON.

Since the beginning of the 2011 LHC physics run in March, CMS has been steadily transferring data with PhEDEx at an average global speed between all sites above 2 GB/s with peaks exceeding 3.5 GB/s, with up to 120,000 file transfers per day and 37 PB of replicas distributed over all the sites.

3.3.6 DBS

The **Dataset Bookkeeping Service** (DBS) describes all the CMS event data by cataloguing CMS-specific data definitions like run number, the algorithms and configurations used to process it and the composition of each dataset in terms of files [R5]. It can be used for data discovery via either an API or a Web interface. There are different DBS differing by scope: the global scope DBS is a single instance describing all official CMS data, while local scope DBS instances can be used as temporary data catalogues, by production teams or individuals.

The DBS is a multi-tier Web application and supports Oracle, MySQL and SQLite as database back-ends. The global scope DBS is hosted in the CERN Oracle RAC cluster and local scope DBS instances are installed both at CERN and at remote sites.

The DBS does not contain the physical file names: it knows only the logical file names and the sites where the dataset is replicated. Translation from the logical to the physical file name is performed locally at sites according to a set of simple string manipulation rules, called the **Trivial File Catalogue**.

3.3.7 Data Popularity

The CMS Data Popularity project started at the beginning of 2011 inspired by the ATLAS Popularity System. The main role of this service is to allow the experiment to measure data access patterns, in particular by recording how often files are accessed, where, by which users, how much CPU time was spent processing them, etc. Understanding these patterns is a crucial step ahead towards the automation of data cleaning and dynamic data placement.

The project aims at providing a generic framework which can be adopted by the LHC experiments. While the source of information can be experiment-specific, the core database and the presentation layer can implement a generic design. For this reason the architecture is designed to be modular as possible, where specific plugins can be added.

The current implementation is composed by three main components named as:

- Oracle popularity database
- Populator daemons
- Presentation framework

The **populator daemon**, written in Python, runs once a day and fills the **popularity database** by extracting the information generated from analysis jobs and temporarily stored in the Dashboard database and converting it in data-related information.

The **presentation layer**, which includes both the web application and an API, has been implemented using the Django framework and is written in Python. This layer represents a key point in the overall implementation since is at this level that data from the popularity database and other services are combined, and this approach allows to keep the database schema as simplest as possible.

Finally the web interface provides three types of presentations: a JSON API, built-in graphics and specialized interfaces for various external applications to guarantee versatilities of the tool.

At the time of writing the described system is assessed to be of pre-production quality and first results are now ready. After a complete validation and integration phase the plan is to move the system in to production.

3.4 LHCb

As in the case of ALICE, in LHCb the computing system is based on a single, all-encompassing framework, but with stronger dependencies on the middleware layer, as shown below:

Problem area	Service	Depends on	Interfaces
Workload management	Ganga	DIRAC	API
	DIRAC	gLite WMS, CE, MySQL	Web, CLI, API
Data management	DIRAC	LFC, FTS, SE, MySQL	Web, CLI, API
Data Catalogue	DIRAC	Oracle	Web, GUI, CLI, API
Security	DISET	VOMS, MyProxy	API
Monitoring and accounting	DIRAC		Web, API

3.4.1 DIRAC

DIRAC (**Distributed Infrastructure with Remote Agent Control**) (Figure 5) is a distributed computing framework designed to meet all the requirements for the LHCb data processing [R6]. It covers all tasks, including raw data distribution from the detector to the Grid storage, and data processing from reconstruction to analysis. DIRAC provides a secure framework (DISET) for building service-oriented distributed systems and a complete pilot job framework for workload management. Finally it includes several subsystems to manage various operations like data production and distribution.

DIRAC is implemented as a network of lightweight agents written in Python and follows the Service Oriented Architecture paradigm. It is interfaced to various types of batch systems and Grid interfaces, including GRAM-based and ARC computing elements, the CREAM CE, PBS/Torque, LSF, Sun Grid Engine, Condor, BQS and Microsoft Compute Cluster. For LHCb the DIRAC services maintain their status in a MySQL database and are deployed at CERN and Tier-1 sites. The DIRAC user interface includes command line tools, a Python API and a Web portal providing users with a secure access to the system. In LHCb end users interact with DIRAC via Ganga to submit jobs.

In contrast to other experiment frameworks, the DIRAC development is not completely specific to LHCb and is used by other communities, like the Belle II collaboration.

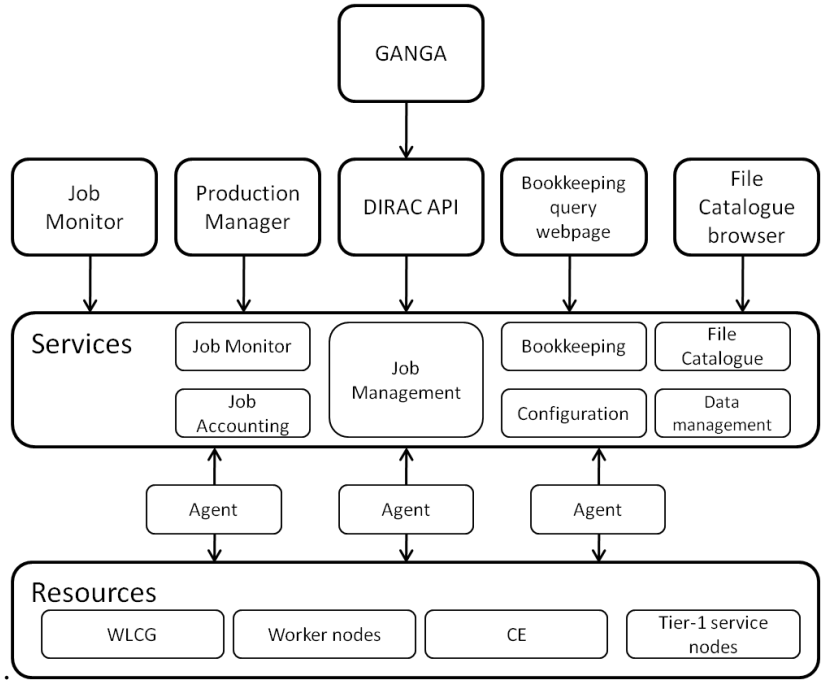

Figure 5. The DIRAC architecture.

DIRAC Framework. The DIRAC secure client/service framework (**DISET**) provides authentication and authorization to determine the user's rights to access the service functions. Moreover, DIRAC provides a set of tools for managing extended proxy certificates with VOMS attributes to encode group membership information. Finally DISET offers a security logging service where all the operations in the secure distributed environment are traced.

Workload Management. The DIRAC Workload Management System is again based on the pilot job concept. Jobs are submitted to the central **Task Queue** where they wait until they are picked up by

pilot job agents running close to the computing resources. The pilot job agents are submitted by **Pilot Director** components specialized for each type of computing resources and perform sanity checks on the local environment before pulling jobs. This approach allows to increase the overall reliability and to apply global policies on how to share common resources via a central prioritization system.

Configuration service. This component hosts all the static configuration data and makes it available to all the clients

Data Management System. The Data Management System takes care of all data operations: data transfer, data access from the worker nodes, data integrity checks and organized recall from tape. An automatic data distribution system is used to create replication requests for incoming data and these are executed using the FTS or specialized transfer agents. Another subservice, the Data Integrity Checking System, takes care of finding and fixing inconsistencies between storage systems and file catalogues.

File catalogue. In LHCb the file catalogue service used for data is provided by LFC (see next section), with a single master write accessible instance at CERN and multiple read-only mirrors at Tier-1 sites. Another simpler file catalogue is used in the context of the production management system.

Request Management System. The Request Management System is a specialized database with a service interface, which allows collecting and serving requests for various operations (data management operations, framework operations and WMS operations). The RMS itself is organized as a distributed redundant set of services. An instance of the RMS system is running at each site.

Production Management System. The Production Management System allows a large number of production jobs to be defined and managed by automating job submission and job management via a set of convenient user tools and interfaces.

Bookkeeping. The Bookkeeping Database Service collects and serves the provenance information about all the data files produced by LHCb. It contains all the necessary metadata to allow users selection of their preferred data sets and to allow the production system to select the input data to be processed.

Monitoring and accounting. This DIRAC component allows collecting, querying and displaying monitoring and accounting information such as job-related information and history and data transfer information.

4 MIDDLEWARE SERVICES

In this section we describe the high-level Grid services developed by Grid projects to be VO-independent. In the context of the LHC computing, they are deployed at different sites and are operated as part of the WLCG infrastructure. They can be classified in four main areas of functionality: data management, workload management, data persistency and monitoring. A more detailed description of the middleware used by WLCG is available elsewhere [R7].

4.1 Data Management

The purpose of the middleware data management services is to address in a VO-agnostic way two main needs: data discovery and data transfer. To do this, the traditional Grid paradigm of a central File Catalogue, mapping logical file names to their actual replicas, is being adopted, while data transfer is optimized to transfer very large numbers of files in a reliable way. At this level, data is considered to be just a set of individual files, while the concept of dataset is implemented only in the experiment service layer.

4.1.1 LCG File Catalogue

The LCG File Catalogue (LFC) is the evolution of the EDG Catalogues, providing more features and fixing the performance and scalability problems seen on the previous file catalogues. Some of the features it provides are the use of a hierarchical namespace, namespace operations, built-in security, use of Access Control Lists, checksums and the possibility to use bulk methods to avoid long round-trip times. It is implemented as a client/server model, where the server is multi-threaded and well decoupled from the database backend. The available client interfaces span from a command line interface to C, Python and Perl APIs. LFC instances are used in over 60 sites by several Virtual Organisations, including ATLAS and LHCb. In the case of ATLAS, as of today, 118 million files are registered between 11 LFC instances.

4.1.2 File Transfer Service

The File Transfer Service (FTS) is a data movement service used by experiment frameworks to manage high volume data streams. It provides site resources usage balancing and prevents network and storage overload. Additionally it features service monitoring and statistics gathering.

FTS is based on the concept of **channel**, which defines a unidirectional management queue for transfer jobs, where the endpoints might as well be single sites, site groups or all possible destinations. The FTS channel defines the transfer protocol (direct GridFTP or via srmCopy), transfer limits and parameters, as well as VO shares and transfer priorities.

The server is deployed as decoupled components communicating through a common database. The **Web service** is responsible to receive the transfer jobs submitted by the user. These transfer jobs are picked up by the **VO Agents**, which queue them on an appropriate channel between the source and the destination. Finally the **Channel Agent** is responsible to select the transfers according to shares and priorities and start them.

The agents can be split across multiple nodes to ensure load balancing. FTS is security-aware using X.509 credentials and logs all the operations carried out. In WLCG, FTS is used by ATLAS, CMS and LHCb.

4.2 Workload Management

The middleware provides several types of workload management. For all of them the basic goal is to provide a convenient way for a user to submit and manage generic batch jobs on Grid resources. Each

system differs from the others in terms of the level of automation provided in the job management and in the user interface. All systems described here are now able to interact with all flavours of Computing Element used in the WLCG.

4.2.1 Ganga

Ganga [R8] is an end-user tool used to manage jobs running on arbitrary execution back-ends, be they local to the user, on a nearby computing batch system, or on the Grid.

Ganga jobs are composed of a few configurable plugins including an **Application** plugin, **Input** and **Output Datasets** plugins, a **Splitter** which divides the task into smaller work units, and a **Backend** execution service.

Ganga is implemented in Python and is a community-driven project: the core of Ganga provides basic job management functionality such as job persistency and functions to interact with jobs (submit, retry, copy, kill), while end-user communities develop plug-ins which map their applications to the execution back-ends.

Ganga is used extensively by heavy user communities in HEP (ATLAS and LHCb) and by many other VOs including Life Sciences and Earth Sciences. Built-in back-ends provide the ability to run jobs on gLite Computing Elements either directly or via the gLite Workload Management System. During 2010 Ganga has been used by more than a thousand users at more than hundred sites.

4.2.2 Condor-G

Condor-G is a Grid workload management system developed by the Condor project and based on the Globus Toolkit and the Condor technologies. Condor-G allows submitting, managing and executing jobs on distributed resources, while Globus is used to communicate with the remote resources via the GRAM protocol for job submission.

Condor-G was and is still used for job submission by ATLAS CMS, and it is used internally by other workload management systems, like the gLite WMS and glideinWMS (see next sections).

4.2.3 gLite Workload Management System

The gLite WMS was developed as a high-level job submission service for EGEE implementing the "push" paradigm. It takes care of the distribution and management of tasks to remote Grid resources and supports several types of tasks, including single batch jobs, collections, MPI jobs and workflows with arbitrary dependencies.

The gLite WMS includes several components: **WMProxy**, a Web Service interface to access the WMS functionality; the **Workload manager**, which is the core component of the system; the **Resource Broker**, which finds the best suitable resource for a job; the **Information Supermarket**, which contains a local cache of the information needed for the matchmaking; the **Task Queue**, holding the submission requests, **CondorC**, which performs the actual job management operations, the **Log Monitor**, which is responsible to react to specific events in the job history, the **Proxy Renewal Service** and finally the **Logging and Bookkeeping**, which records all information about submitted jobs.

The gLite WMS has been (and still is) extensively used by the LHC experiments and by several other VOs, in particular on the EGEE and SEE-GRID infrastructures, and it is capable to submit to different types of Computing Elements (LCG CE, OSG CE, CREAM CE, ARC CE), which is why it is the only WMS directly supported by the WLCG project. However, in the recent years part of the workload was migrated to pilot job-based systems; the only experiment running a significant fraction of its jobs via the gLite WMS is CMS, but also in this case the plan for the future is to rely exclusively on pilot jobs.

4.2.4 GlideinWMS

GlideinWMS [R9] is a general purpose pilot-job based Workload Management System that works on top of Condor by creating and using a virtual private pool over Grid resources.

User jobs are handled by the glideinWMS **Condor pool**, whose execution daemons (called **condor_startd**) run on remote Grid worker nodes; jobs are matched to the existing execution daemons and sent to a matched worker node.

The condor_startd daemons are started by dedicated pilot jobs, called glideins; glideins are submitted to remote sites by **glidein factories** using Condor-G. A component called **VO frontend** uses a submission logic by which the number of glideins dynamically adapts to the number of centrally queued jobs.

Among the LHC experiments only CMS is using GlideinWMS, for part of its production and analysis activities. Outside CERN, it is used by the CDF and the MINOS experiments at Fermilab.

4.3 Persistency

The Persistency Framework consists of three software packages (**POOL**, **CORAL** and **COOL**) which address the requirements of the heavy user communities in HEP for storing and accessing several different types of scientific data produced by the LHC experiments. The software is developed by the CERN IT department in collaboration with the three LHC experiments that are using it to store their data (ATLAS, CMS and LHCb). In contrast to the middleware services that make up the infrastructure for the submission and monitoring of data processing jobs and for the management and distribution of the input and output data sets of these jobs, the Persistency Framework software is used inside these jobs to allow the data processing algorithms to read or write data in the appropriate user-defined format.

The Persistency Framework software is used by the LHC experiments for both of their main categories of scientific data: **event data** (which contain information about the response of the detectors to the passage of the particles generated in the collisions of the two LHC beams) and **conditions data** (which record the state of the detector at the time the event data were collected). The functionalities provided by the software are however not specific to the HEP experiments and some of the components may be reused by other communities to store their own, different, types of data.

All three software packages provide a set of libraries and APIs which are used directly by the user code of the HEP experiments. The APIs are often defined via abstract interfaces which decouple the user code specific from the details for a given storage technology. The APIs and implementation code are all written in C++, but Python bindings are also available for most components.

4.3.1 CORAL

The CORAL package [R10] is an abstraction layer with an SQL-free API for accessing data stored using relational database technologies. It is used by ATLAS, CMS and LHCb to store their conditions and other types of data, both directly by experiment-specific applications and via COOL or POOL. CORAL supports data persistency for several back-ends and deployment models, including local access to SQLite files, direct client connections to Oracle and MySQL servers, as well as client connections to database servers through intermediate caching/multiplexing tiers using the Frontier and CORAL server technologies (described in a later section).

4.3.2 POOL

The POOL package is a hybrid technology store for C++ objects, using a mixture of streaming and relational technologies to implement both object persistency and object metadata catalogues and collections [R11]. POOL provides generic components that can be used by the experiments to store

event data (ATLAS and LHCb, using object streaming to ROOT files), event 'tag' collections (ATLAS, using Oracle collections) and conditions data (CMS, using object streaming to Oracle).

4.3.3 COOL

The COOL package provides specific software components and tools for the handling of the conditions data of the HEP experiments (ATLAS, LHCb) [R12]. The main properties of conditions data are that they vary with time and that they can exist in several versions. COOL defines a data model where conditions data are assigned an interval of validity and provides an API for the retrieval of the conditions data valid at a given point in time, with particular emphasis on the optimization of the relevant database queries.

4.3.4 Frontier/Squid and CORAL server/proxy

The master repositories of the conditions data (and many other types of relational data) of ATLAS, CMS and LHCb are hosted on Oracle database servers at CERN. These data, that are typically written once and seldom updated, generally need to be read back simultaneously by several CORAL clients at several geographical locations in the Grid (either from the Oracle master repository at CERN or from one of the Tier-1 sites where an Oracle Streams replica exists). To reduce the load on the Oracle servers and serve all relevant clients in the fastest and most efficient way, CORAL supports two technologies that allow the deployment of intermediate multiplexing and caching tiers between the database servers and the client applications.

In the **Frontier/Squid** technology [R13], based on the HTML protocol, a CORAL plugin in the client application encodes SQL queries into HTTP requests that are decoded back into SQL queries by a Frontier server deployed close to the Oracle server, to which it is connected using JDBC. The results of the queries, encoded as HTML pages, may be cached in a tree of Squid caches deployed between the client and the Frontier server. This read-only technology is used for conditions data access on the Grid by both CMS (since several years) and ATLAS (since the end of 2009). It is also used in the CMS online system for the configuration of the High Level Trigger farm.

Similarly, in the **CORAL server/proxy** technology, SQL requests and replies are exchanged using a custom binary protocol between the client applications and a CORAL server deployed close to the Oracle server. The CORAL server is itself a CORAL application that may connect to Oracle servers via OCI, but also to any other supported backend (e.g. MySQL servers). The results of queries may be cached in a tree of CORAL server proxies deployed between the client and the CORAL server. At present, this technology is read-only and is used in the ATLAS online system to configure the High Level Trigger farm. Work is ongoing to extend it by implementing read-write functionalities with secure authentication based on SSL and Grid certificates.

4.4 Monitoring

In the context of the LHC experiment computing systems, monitoring is particularly relevant in two areas: application monitoring and site monitoring. Application monitoring is crucial to understand the status of the experiment activities and their time evolution, while site monitoring is important to measure the health of the Grid infrastructure.

4.4.1 Experiment Dashboard

In order to monitor the computing activities of the LHC experiments, several monitoring systems were developed, most of which are coupled with the Data Management and Workload Management System of specific experiments. They were developed independently by each experiment along with their data management and workload management systems. And as a consequence they work in the scope of a single experiment. Experiment Dashboard is an attempt to provide a common solution which can be

shared by several experiments. In addition, the Experiment Dashboard was developed as a generic monitoring framework for the LHC experiments.

The Experiment Dashboard system [R14] covers the complete range of the LHC computing activities, that is job processing, data transfer and site monitoring, across the whole WLCG infrastructure (EGI, OSG and NorduGrid).

It is extensively used by the LHC experiments; for CMS alone, more than 5,000 unique visitors use it per month and approximately 100,000 pages are accessed daily. These numbers are steadily growing.

The structure of the Experiment Dashboard monitoring system consists of several information collectors, the data repositories (implemented in an ORACLE database) and the user interfaces. Its framework is implemented in Python. All the output data produced by the Experiment Dashboard can be retrieved in HTML, XML, CSV, JSON or image formats. This flexibility allows the system to be used not only by the users but also by other external, third party, applications. A set of command line tools is also available.

4.4.2 SAM/Nagios

The Service Availability Monitoring (SAM) framework provides a site-independent, centralised and uniform monitoring for all Grid services. SAM can also offer greater flexibility for individual VO's with the option of tailoring the monitoring to align with their specific services. Within EGI, SAM is used in the validation of sites and services and to calculate the site availability and reliability.

SAM is based on Nagios, a well-known open-source monitoring system, and provides a set of probes which are submitted at regular intervals, and a database that stores test results. In effect, SAM provides monitoring of grid services from a user's perspective. At the present time, SAM is being used by more than 330 certified and production sites in 48 countries to monitor more than 3400 grid services.

Due to the evolution of EGI from EGEE to a multitude of national grid initiatives, the resources and responsibility for grid monitoring had to move away from a centrally administered monitoring function. This is why the original SAM architecture was realigned to encompass the organisational charges that took place as we moved towards EGI by using the Nagios open-source framework for monitoring network hosts and services.

4.4.3 HammerCloud

HammerCloud [R15] is a distributed analysis stress testing system built around Ganga. Inspired by an older and less advanced service, the CMS Job Robot, it was motivated by a requirement from the ATLAS collaboration for site- and central-managers to easily test a set of Grid sites with an arbitrarily large number of real analysis jobs. These tests are useful during site commissioning to validate and tune site configurations, and also during normal site operations to periodically benchmark the site performance.

HammerCloud generates a test report including metrics such as the event processing rate, the mean CPU utilization, and timings related to various stages of the user analysis jobs. The report is presented in a web interface which makes it simple to compare sites and observe trends over time. The system has been used by the ATLAS experiment to run more than 200,000 CPU-days of test analyses. HammerCloud is implemented as a Django web application, with state maintained in a MySQL database and job management built around Ganga in Python. Jobs can be submitted to Grid sites using the gLite User Interface commands and to all ATLAS sites using PanDA.

Although HammerCloud was born as an ATLAS-only service, recently CMS and LHCb adopted it and it is actively used to run site functional tests.

5 SERVICES SUPPORTED BY SA3

The EGI InSPIRE WP6-SA3 activity supports or contributes development effort to almost all of the experiment services and to some of the middleware services. The following table summarizes the services supported by SA3 and their corresponding subtasks.

Summary of SA3 supported services.

Service	Subtask
Analysis tools and support	
Ganga	TSA3.2.2
CRAB	TSA3.3
CRAB analysis server	TSA3.3
Data management	
ATLAS DDM	TSA3.3
PhEDEx	TSA3.3
Data Popularity	TSA3.3
Persistency and conditions	
CORAL	TSA3.3
CORAL server	TSA3.3
POOL	TSA3.3
COOL	TSA3.3
Frontier/Squid	TSA3.3
Monitoring	
Experiment Dashboard	TSA3.2.1
HammerCloud	TSA3.3

5.1 Common Solutions with EGI-InSPIRE Involvement

As is shown in the table above, EGI-InSPIRE effort contributes to a number of common solutions, including Ganga, the LCG Persistency Framework and the Experiment Dashboard that pre-date the EGI-InSPIRE project, as well as HammerCloud and the Data Popularity framework – both of which have been adopted (and where necessary adapted) via EGI-InSPIRE funded resources. The goal is to increase the level of commonality both within the HEP community whilst ensuring that the solutions are applicable also to others with the additional goal of reduced long-term support as part of the overall strategy for addressing sustainability.

5.2 Services and Operations

In order to provide the above services to the HEP community, a number of additional tools and services are required. These are defined in the Memorandum of Understanding (draft) between EGI.eu

and WLCG and are expected to be reviewed approximately twice per year. The current list is given below:

- (EGI.eu) Quality verification and Staged Rollout of software provided by the EGI Technology Providers, which is made available for deployment on EGI.
- (EGI.eu) EGI Community Repository for software contributed and supported by the WLCG community.
- (EGI.eu) The EGI Help desk (GGUS): provided by EGI.eu and its partners to WLCG VRC and other user communities.
- (EGI.eu) First, second and third-level support (this with the involvement of the Technology Providers) to users and site administrators about EGI-supported software and operations support.
- (EGI.eu) Support Units: EGI.eu will maintain and develop the EGI Helpdesk to ensure the support units and workflow needed to support the WLCG VRC are implemented in a timely manner.
- (EGI.eu) Core middleware services: EGI.eu in collaboration with its NGI providers will provide a highly-available core middleware services according to the WLCG demand (e.g. top-level information discovery services, workload management services, etc.) to support their communities.
- (EGI.eu) Monitoring: EGI.eu provides in collaboration with its NGIs the distributed monitoring infrastructure needed to check the status of the deployed services (central MyEGI portal, the central databases and the messaging infrastructure).
- (EGI.eu) Configuration Database: EGI.eu will provide a configuration database (GOCDB) that will provide information on the sites and services accessible to the WLCG VRC.
- (EGI.eu) Accounting: EGI.eu will provide an accounting database and portal that will allow the WLCG VRC to review its usage of EGI resources, together with the messaging infrastructure needed to centrally collect usage records.
- (WLCG) Dashboard: WLCG VRC will provide to EGI.eu and its community the Dashboard functionality so that EGI's user communities can integrate their own domain specific probes into the Dashboard infrastructure.
- (WLCG) Availability Computation: WLCG will provide and maintain the system, including consideration of functionality enhancements specific to EGI, needed to calculate availability and reliability statistics for the (OPS VO) and to customise profiles according to the EGI needs.
- (WLCG) GSTAT: WLCG will provide and maintain the system, including consideration of functionality enhancements specific to EGI.

6 CONCLUSION

We have seen how the HEP experiments rely on a very large number of services to implement their computing systems. As expected, given the complexity of the experiment activities, both the core functionality and the user interfaces are in the domain of the experiment services (with the exception of Ganga). The boundary between the experiment and the middleware domains is not always clear, with some functionality being sometimes offered by the middleware, sometimes by the experiment, and this boundary moved over time (as an example, the diminishing role of the gLite WMS in favour of a pilot job approach). The middleware services, on the other hand, play an essential role in the operation of the WLCG infrastructure according to common and agreed policies in the provision of the computing, storage and network resources.

7 APPENDIX

Several of the experiment services have a web interface. As a reference, we give some web links that may be of interest.

Service	URL
AliEN home page	http://alien2.cern.ch/
MonALISA	http://monalisa.caltech.edu/
PanDA monitor	http://panda.cern.ch/
DDM monitoring	http://bourricot.cern.ch/dq2/
PhEDEx	http://cmsweb.cern.ch/phedex
DBS Data Discovery	https://cmsweb.cern.ch/dbs_discovery/
LHCb DIRAC portal	http://lhcbweb.pic.es/DIRAC/
Ganga home page	http://cern.ch/ganga
Persistency home page	https://twiki.cern.ch/twiki/bin/view/Persistency
Frontier home page	http://frontier.cern.ch/
Experiment Dashboard	http://dashboard.cern.ch/
HammerCloud	http://hammercloud.cern.ch/

8 REFERENCES

R 1	S. Bagnasco *et al*, *AliEN: ALICE environment on the GRID*, J. Phys.: Conf. Ser. **119** (2008) 062012
R 2	I. Legrand *et al*, *MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications*, Comp. Phys. Comm. **180**, **12**, (2009), 2472
R 3	T. Maeno, *PanDA: distributed production and distributed analysis system for ATLAS*, J. Phys.: Conf. Ser. **119** (2008) 060236
R 4	M. Branco *et al*, *Managing ATLAS data on a petabyte-scale with DQ2*, J. Phys.: Conf. Ser. **119** (2008) 062017
R 5	A. Fanfani *et al*, *Distributed analysis in CMS*, J. Grid Comp. **8** (2010) 159
R 6	A. Tsaregorodtsev *et al*, *DIRAC: a community solution*, J. Phys.: Conf. Ser. **119** (2008) 062048
R 7	S. Burke *et al*, *gLite User Guide*, https://edms.cern.ch/document/722398/
R 8	J. T. Mościcki *et al*, *Ganga: A tool for computational-task management and easy access to Grid resources*, Comp. Phys. Comm. **180, 11** (2009) 2303
R 9	I. Sfiligoi, *GlideinWMS – A generic pilot-based Workload Management System*, J. Phys.: Conf. Ser. **119** (2008) 062044
R 10	I. Papadopoulos *et al*, *CORAL, a software system for vendor-neutral access to relational databases*, proceedings of CHEP06
R 11	G. Govi, *POOL Developments for Object Persistency into Relational Databases*, proceedings of CHEP06
R 12	A. Valassi, *COOL Development and Deployment - Status and Plans*, proceedings of CHEP06
R 13	L. Lueking *et al*, *CMS Conditions Data Access using FroNTier*, J. Phys.: Conf. Ser. **119** (2008) 072007
R 14	J. Andreeva *et al*, *Experiment Dashboard for Monitoring Computing Activities of the LHC Virtual Organizations*, J. Grid Comp. **8** (2010) 323
R 15	D. C. Van Der Ster *et al*, *Functional and large-scale testing of the ATLAS distributed analysis facilities with Ganga*, J. Phys.: Conf. Ser. **119** (2008) 072021