ENHANCEMENTS TO THE SAM-GRID INFRASTRUCTURE

by

BIMAL BALAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2005

To my family and friends.

ACKNOWLEDGEMENTS

I express my sincere gratitude to my thesis advisor David Levine for giving me an opportunity to work with him and for all the time and suggestions provided by him for the research. He has instilled in me a passion towards scientific research. I also want to thank Dr. Bob Weems and Dr. Yu for agreeing to serve on my thesis committee. They have given me high motivations for doing research. I am also thankful to Dr. Yu for his excellent mentorship.

I am grateful to Dr. Yu and Fermilab for giving me a wonderful opportunity for doing cutting edge research in Fermilab, through the UTA-FNAL Computer Science MS student exchange program. Also, I express my sincere gratitude to the entire SAM-Grid team in Fermilab. Among them, Gabriele Garzoglio has been really helpful in the successful completion of my work. I am thankful to many of my friends who helped me during this period.

I would like to thank my family for all their support and devout guidance.

November 11, 2005

ABSTRACT

ENHANCEMENTS TO THE SAM-GRID INFRASTRUCTURE

Publication No. _____

Bimal Balan, M.S.

The University of Texas at Arlington, 2005

Supervising Professor: David Levine

SAM-Grid is a grid computing infrastructure for high energy physics (HEP) experiments in Fermilab. It is composed of data handling, job management and information management components. There are several challenges when the number of sites participating in the experiment increases. This thesis presents the enhancements made on the SAM-Grid infrastructure. This includes scalability and performance related enhancements.

The enhancements mainly affect the batch adapter, monitoring and security layers. As a scalability aspect, Monitoring and Information services required changes to make it easier for monitoring large number of jobs. SAM-Grid is integrated with Sun Grid Engine (SGE). The batch adapter layer corresponding to SGE is different from that of other batch systems like PBS. SGE is successfully being used for production in one of the execution sites. Integration of MyProxy with SAM-Grid is one of the performance related enhancements. SAM-Grid system performance will improve because job failures due to proxy expiration are avoided.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

ABSTRACT v

LIST OF FIGURES ix

Chapter

1. Introduction 1

    1.1 Grid Computing 1

        1.1.1 What is Grid Computing? 2

        1.1.2 Virtual Organization 4

        1.1.3 Comparison with other distributed computing technologies 4

    1.2 HEP experiments and Grid Computing 8

    1.3 Goal of the thesis 9

2. Background and Related Work 11

    2.1 LDAP Service 11

    2.2 Globus 12

        2.2.1 GRAM (Globus Resource Allocation Manager) 14

        2.2.2 MDS (Monitoring and Discovery Service) 14

        2.2.3 GridFTP 15

    2.3 Condor and Condor-G 15

    2.4 CORBA (Common Object Request Broker Architecture) 17

3. SAM-Grid Architecture 18

    3.1 Data Management component of SAM-Grid 20

    3.2 Job Management component of SAM-Grid 22

3.3 Information Management component of SAM-Grid 23
4. Sun Grid Engine integration 24
4.1 Grid Fabric interface in SAM-Grid 24
4.1.1 SAM-Grid Job Managers 25
4.1.2 SAM Batch Adapters 28
4.1.3 Batch System Idealizers 30
4.2 Sun Grid Engine (SGE) 33
4.3 Result of Sun Grid Engine integration 36
5. Enhancement of Monitoring System 37
5.1 Overview of the SAM-Grid Monitoring System 37
5.2 Architecture of the Monitoring System 39
5.2.1 Globus MDS 40
5.2.2 Class-Ads from Condor 40
5.2.3 Information Processing Layers of the System 43
5.3 Enhancements in Monitoring 44
5.3.1 Result of Monitoring Enhancements 50
6. Integration of MyProxy with SAM-Grid 52
6.1 Overview of Grid security 52
6.1.1 Proxy 54
6.1.2 Security in SAM-Grid 55
6.2 MyProxy - Credential management system 57
6.2.1 MyProxy Server 57
6.2.2 MyProxy Repository 58
6.3 Integration of MyProxy with SAM-Grid 59
6.3.1 Client site 60
6.3.2 Submission site 61

6.3.3 Execution site 61
6.4 Status of MyProxy Integration 62
7. Conclusions 63
7.1 Current utilization of SAM-Grid 63
REFERENCES 66
BIOGRAPHICAL STATEMENT 68

LIST OF FIGURES

Figure		Page
2.1	Globus Architecture	13
2.2	Condor-G Architecture	16
3.1	SAM-Grid Architecture.	18
3.2	Services in SAM-Grid. Each site represents a job execution site	19
3.3	Job Management	22
4.1	Grid to Fabric interface	25
4.2	Batch Adapter interaction	29
5.1	Resource Class-Ad	41
5.2	Job Class-Ad	42
5.3	SAM-Grid submission site	45
5.4	Grid job Details	46
5.5	Grid job Summary	47
5.6	Progress details of an individual batch system job	48
5.7	Output Files created by a batch job	49
5.8	SAM Web query of an output file	50
6.1	MyProxy with SAM-Grid	60
7.1	Throughput of SAM-Grid	63
7.2	Production in SAM-Grid	64

CHAPTER 1

Introduction

Grid computing is a way of harnessing the resources available from the different clusters. Modern high energy physics experiments need an infrastructure capable of seamlessly integrating computing centers around the world. SAM-Grid infrastructure in Fermilab helps in solving the data challenges involved in the HEP experiments in Fermilab. SAM-Grid integrates standard Grid middleware, such as Globus and Condor.

1.1 Grid Computing

In the last few years, a crucial gap has developed between the advance of networking capability (the bits per second a network can handle) and microprocessor speed (based on the number of transistors per integrated circuit). Networking capability essentially doubles every nine months today, although historically this growth was much slower. And Moore's Law dictates that the number of transistors per integrated circuit still doubles every 18 months. Therein lies the problem. Moore's Law is slow compared with the advancement in network capability. If you accept as a given that core networking technology now accelerates at a much faster rate than advances in microprocessor speeds, then it becomes apparent that in order to take advantage of the advances in networking, a more efficient way of harnessing microprocessor capacity is required. This new point of view changes the historical trade-off between networking and processing costs. Similar arguments apply to bulk storage. Grid computing is the means to address this gap, this change in the traditional trade-offs, by tying together distributed resources to form a single virtual computer [1].

1.1.1 What is Grid Computing?

Grid-computing principles focus on large-scale resource sharing in distributed systems in a flexible, secure, and coordinated fashion. This dynamic coordinated sharing results in innovative applications making use of high-throughput computing for dynamic problem solving. Grid computing uses the resources of many separate computers connected by a network to solve large-scale computation problems. The SETI@home project, launched in 1999, is a widely-known example of a very simple Grid computing project. Most of these projects work by running as a screensaver on users' personal computers, which process small pieces of the overall data while the computer is either completely idle or lightly used.

In 1998, it was stated that a computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [2]. This definition was primarily centered on the computational aspects of Grids. Later iterations broadened this definition with more focus on coordinated resource sharing and problem solving in multi-institutional virtual organizations. Grid computing differentiates itself from other distributed computing technologies through an increased focus on resource sharing, co-ordination, manageability, and high performance. The sharing of resources, ranging from simple file transfers to complex and collaborative problem solving, is accomplished under controlled and well-defined conditions and policies. In this context, the critical problems are resource discovery, authentication, authorization, and access mechanisms.

Grid computing offers a model for solving massive computational problems by making use of the unused resources (CPU cycles and/or disk storage) of large numbers of disparate, often desktop, computers treated as a virtual cluster embedded in a distributed telecommunications infrastructure. Grid computing's focus on the ability to support computation across administrative domains sets it apart from traditional computer clusters

or traditional distributed computing. Grids offer a way to solve grand challenge problems like protein folding, financial modeling, earthquake simulation, climate/weather modeling etc. Grids offer a way of using the information technology resources optimally inside an organization. They also provide a means for offering information technology as a utility bureau for clients, who pay only for what they use, as with electricity or water. Grid computing has the design goal of solving problems too big for any single supercomputer, whilst retaining the flexibility to work on multiple smaller problems. Thus Grid computing provides a multi-user environment. It involves sharing heterogeneous resources (based on different platforms, hardware/software architectures, and computer languages), located in different places belonging to different administrative domains over a network using open standards. In short, it involves virtualizing computing resources.

The following are the characteristics of a Grid [2]:

1) Coordinates resources that are not under centralized control.

2) Uses standard, open, general-purpose protocols and interfaces.

3) Delivers non-trivial qualities of service.

The growing popularity of Grid computing has resulted in various kinds of Grids, common ones being known as Data Grids, Computational Grids, Bio Grids, Cluster Grids and Science Grids. Functionally, one can classify Grids into several types:

1) Computational Grids which focuses primarily on computationally-intensive operations.

2) Data Grids, or the controlled sharing and management of large amounts of distributed data.

3) Equipment Grids which have a primary piece of equipment e.g. a telescope, and where the surrounding Grid is used to control the equipment remotely and to analyze the data produced.

1.1.2 Virtual Organization

A virtual organization (VO) is a dynamic group of individuals, groups, or organizations who define the conditions and rules for sharing resources. The concept of the VO is the key to Grid computing. These are some of the common characteristics that typically exist among participants of a VO:

1. Concerns and requirements exist concerning resource sharing.
2. Resource sharing is conditional, time-bound, and rules-driven.
3. The collection of participating individuals and/or institutions is dynamic.
4. Sharing relationship among participants is peer-to-peer in nature.
5. Resource sharing is based on an open and well-defined set of interactions and access rules.

All VOs share some characteristics and issues, including common concerns and requirements that may vary in size, scope, duration, sociology, and structure. The members of any VO negotiate the sharing of resources based upon the rules and conditions defined by the VO, and the members then share the resources in the VO's constructed resource pool. Assigning users, resources, and organizations from different domains to a VO is one of the key technical challenges in Grid computing. This task includes identification and application of appropriate resource-sharing methods, rules and conditions for member assignment, security delegation, and access control among the participants.

1.1.3 Comparison with other distributed computing technologies

Grid computing has recently enjoyed an increase in popularity as a distributed computing architecture. As Grid computing matures, the application of the technology in additional areas will increase. Grid computing can be differentiated from almost all distributed computing paradigms by this defining characteristic: The essence of Grid computing lies in the efficient and optimal utilization of a wide range of heterogeneous,

loosely coupled resources in an organization tied to sophisticated workload management capabilities or information virtualization [1].

1.1.3.1 Comparison with Cluster computing

Grid computing is often confused with cluster computing. The key difference is that the resources which comprise the Grid are not all within the same administrative domain. Grids consist of heterogeneous resources. Cluster computing is primarily concerned with computational resources; Grid computing integrates storage, networking, and computation resources. Clusters usually contain a single type of processor and operating system; Grids can contain machines from different vendors running various operating systems. Grids are dynamic by their nature. Clusters typically contain a static number of processors and resources; resources come and go on the Grid. Resources are provisioned onto and removed from the Grid on an ongoing basis. Grids are inherently distributed over a local, metropolitan, or wide-area network. Usually, clusters are physically contained in the same complex in a single location; Grids can be (and are) located everywhere. Cluster interconnection technology delivers extremely low network latency, which can cause problems if clusters are not close together. Grids offer increased scalability. Physical proximity and network latency limit the ability of clusters to scale out; due to their dynamic nature, Grids offer the promise of high scalability.

Cluster and Grid computing are completely complementary; many Grids incorporate clusters among the resources they manage. Indeed, a Grid user may be unaware that his workload is in fact being executed on a remote cluster. And while there are differences between Grids and clusters, these differences afford them an important relationship because there would always be a place for clusters; certain problems will always require a tight coupling of processors. However, as networking capability and bandwidth

advances, problems that were previously the exclusive domain of cluster computing could be solvable by Grid computing.

1.1.3.2 Comparison with CORBA

Of all distributed computing environments, CORBA probably shares more surface-level similarities with Grid computing than the others. This is due to the strategic relationship between Grid computing and Web services in the Open Grid Services Architecture (OGSA). Both are based on the concept of service-oriented architecture (SOA).

A key distinction between CORBA and Grid computing is that CORBA assumes object orientation, but Grid computing does not. In CORBA, every entity is an object and it supports mechanisms such as inheritance and polymorphism. In OGSA, there are similarities to some object concepts, but there isn't a presumption of object-oriented implementation in the architecture. The architecture is message oriented; object orientation is an implementation concept. However, the use of a formal definition language (such as WSDL, Web Services Definition Language) in WSRF (Web Services Resource Framework) means that interfaces and interactions are just as precisely defined as in CORBA, sharing one of the major software engineering benefits also exhibited by object-oriented design.

Another distinction is that OGSA Grid computing is built on a Web services foundation. CORBA integrates with and interoperates with Web services. One of the problems with CORBA was that it assumed too much of the "endpoints," which are basically all the machines (clients and servers) participating in a CORBA environment. There are also issues of interoperability between vendors' CORBA implementations, how CORBA nodes are able to interoperate on the Internet, and how endpoints are named. This means that all of the machines in the group had to conform to certain rules and to a certain way of doing things (all assuming the same protocols like IDL, IOR, and IIOP) for CORBA

to work. This is an appropriate approach when building high-reliability, tightly coupled, pre-compiled systems.

Another important distinction between Grid computing and CORBA is that, OGSA Grid computing defines the following three categories of services: Grid core services, Grid data services and Grid program-execution services. CORBA does not pay specific attention to data or program-execution services, because it is based on essentially remote procedure call (RPC). An RPC is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. It is a synchronous operation that requires the requesting program to be suspended until the remote procedure returns results. Many of the services specified and implemented in Grid core services (as well as the WSRF) are similar to foundational services found in CORBA. But data and program-execution services are unique to Grid computing.

1.1.3.3 Comparison with P2P

The hallmark of a P2P system is that it lacks a central point of management; this makes it ideal for providing anonymity and offers some protection from being traced. Grid environments, on the other hand, usually have some form of centralized management and security (for instance, in resource management or workload scheduling). This lack of centralization in P2P environments carries two important consequences:

1) P2P systems are generally far more scalable than Grid computing systems. Even when you strike a balance between control and distribution of responsibilities, Grid computing systems are inherently not as scalable as P2P systems.

2) P2P systems are generally more tolerant of single-point failures than Grid computing systems. Although Grids are much more resilient than tightly coupled distributed

systems, a Grid inevitably includes some key elements that can become single points of failure.

This means that the key to building Grid computing systems is finding a balance between decentralization and manageability. Also, while an important characteristic of Grid computing is that resources are dynamic; in P2P systems the resources are much more dynamic in nature and generally are more fleeting than resources on a Grid. For both P2P and Grid computing systems, utilization of the distributed resources is a primary objective. Given a wealth of computing resources, both of these systems will try to use them as much as possible.

A final distinction between the two systems is standards – the general lack of standards in the P2P world contrasts with the host of standards in the Grid universe. Entities like the Global Grid Forum, refine existing standards and create new ones.

1.2 HEP experiments and Grid Computing

DZero [3] and CDF [4] are high energy physics (HEP) experiments in Fermilab, located at Batavia, IL. They use the SAM-Grid computing infrastructure. The experiments are being conducted on Tevatron, which is currently the highest-energy particle accelerator in the world. These experiments produce data in the order of TB/day. This data is stored on a Mass Storage System based on tape archives called Enstore [5] developed at Fermilab. The Enstore provides a generic interface (an API) so experimenters can use Mass Storage System as easily as if they were native file systems.

The computation for the DZero experiment involves mainly three categories.

1) Monte-Carlo Simulations of the physics events in the DZero Detector.

2) Conversion of raw detector data to a format that is ready for analysis. This stage is also called reconstruction or data processing.

3) Analysis of the processed data.

All the events that have been stored as detector data are occasionally subjected to another round of processing called reprocessing. The entire dataset is reprocessed so that we may obtain high quality of output.

Analyzing the data stored in the Mass Storage System requires a large amount of computational resources. The computational resources of the experiments are distributed across various institutions around the world. Thus there is a need to provide distributed access to the data and also to the resources of the experiments so that the experimenters can study the data irrespective of their geographic location.

HEP experiments resemble the Virtual Organization (VO) concept. In particular the resources of a HEP experiment are owned by different institutions participating in the experiment and there is a need to allow access to these resources to any individual who is a part of the experiment. Thus Grid Computing presents itself as an ideal solution for the computational requirements of HEP experiments such as DZero and CDF. The SAM-Grid project [6] at Fermilab is focused towards providing Grid solution to these two HEP experiments based at Fermilab. SAM-Grid infrastructure enables scientists to access the resources belonging to the experiment as if they are local resources.

1.3 Goal of the thesis

The SAM-Grid infrastructure is composed of three major components: data handling, job management and information management. The data handling layer is interfaced to the information systems for a number of services.

This thesis presents enhancements on the SAM-Grid infrastructure. This includes scalability and performance related enhancements. The enhancements mainly affect the batch adapter layer, the monitoring layer and the security layer. As a scalability aspect, Monitoring and Information services required changes to make it easier for monitoring large number of jobs. SAM-Grid is integrated with Sun Grid Engine (SGE) batch system.

The batch adapter layer corresponding to SGE is different from that of other batch systems like PBS. SGE is successfully being used for production in one of the execution sites. Integration of MyProxy with SAM-Grid is a performance related enhancement. SAM-Grid system performance improves because job failures due to proxy expiration are avoided.

The rest of the thesis is organized as follows: Chapter 2 gives the background for the SAM-Grid project. The architecture of SAM-Grid is described in Chapter 3. Details about the integration with Sun Grid Engine are given in Chapter 4. In Chapter5, changes to the SAM-Grid monitoring and information services are discussed. Chapter 6 handles the integration of MyProxy with SAM-Grid. Chapter 7 gives conclusions and discusses about future work.

CHAPTER 2

Background and Related Work

This chapter discusses the background materials for the SAM-Grid project. The architecture of the SAM-Grid infrastructure is covered in the next chapter. Details about LDAP Service, Globus, Condor, Condor-G and CORBA are given in the subsequent sections.

2.1 LDAP Service

LDAP stands for Lightweight Directory Access Protocol. As the name suggests, it is a lightweight protocol for accessing directory services, specifically X.500-based directory services. LDAP runs over TCP/IP or other connection oriented transfer services [7].

The LDAP information model is based on entries. An entry is a collection of attributes that has a globally-unique Distinguished Name (DN). The DN is used to refer to the entry unambiguously. Each of the entry's attributes has a type and one or more values. The types are typically mnemonic strings, like "cn" for common name, or "mail" for email address. The syntax of values depends on the attribute type. Directory entries are arranged in a hierarchical tree-like structure. Traditionally, this structure reflected the geographic and/or organizational boundaries. In addition, LDAP allows us to control which attributes are required and allowed in an entry through the use of a special attribute called objectClass. The values of the objectClass attribute determine the schema rules the entry must obey.

An entry is referenced by its distinguished name, which is constructed by taking the name of the entry itself (called the Relative Distinguished Name or RDN) and concate-

nating the names of its ancestor entries. Operations are provided for adding and deleting an entry from the directory, changing an existing entry, and changing the name of an entry. Most of the time, though, LDAP is used to search for information in the directory. The LDAP search operation allows some portion of the directory to be searched for entries that match some criteria specified by a search filter. Information can be requested from each entry that matches the criteria.

LDAP directory service is based on a client-server model. One or more LDAP servers contain the data making up the directory information tree (DIT). The client connects to servers and asks it a question. The server responds with an answer and/or with a pointer to where the client can get additional information (typically, another LDAP server). No matter which LDAP server a client connects to, it sees the same view of the directory; a name presented to one LDAP server references the same entry it would at another LDAP server. This is an important feature of a global directory service, like LDAP. LDAP provides a mechanism for a client to authenticate, or prove its identity to a directory server, paving the way for rich access control to protect the information the server contains. LDAP also supports data security (integrity and confidentiality) services.

2.2 Globus

The open source Globus Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy [8]. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. The Globus Toolkit was conceived to remove obstacles that

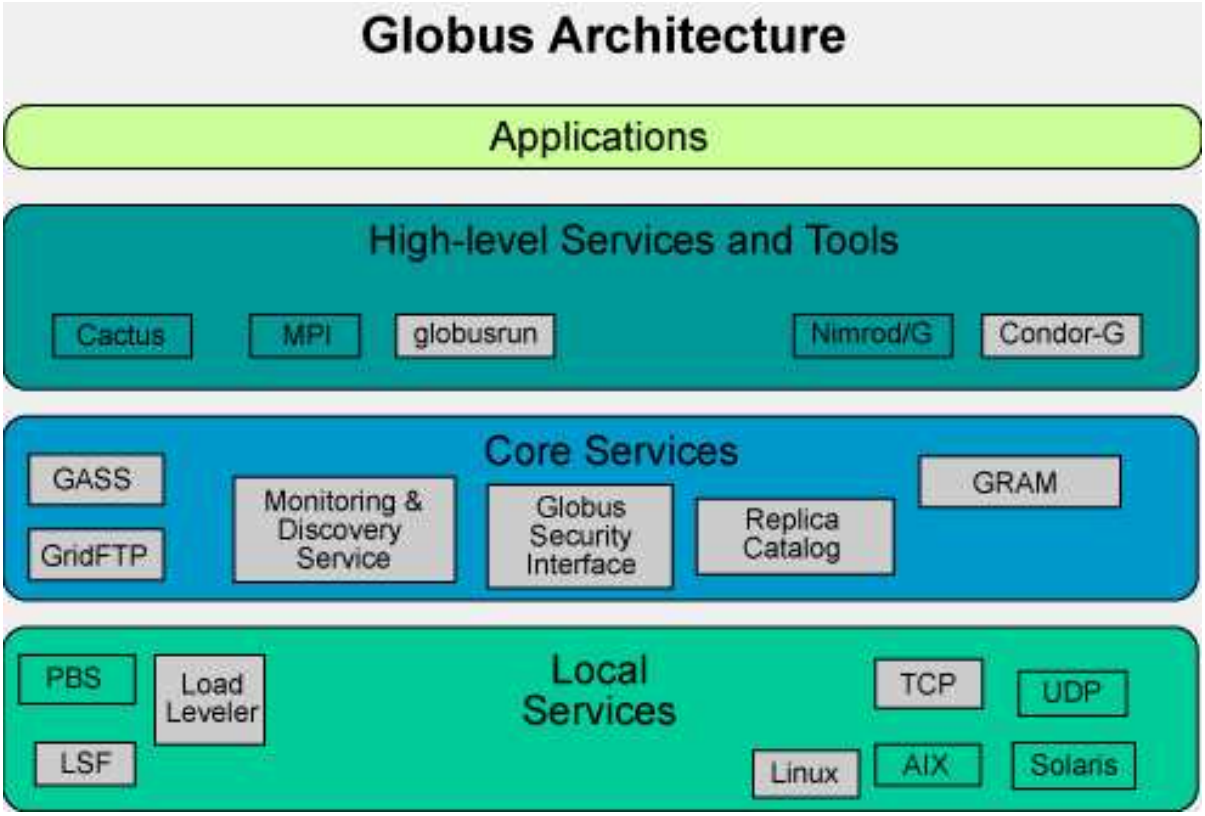

Figure 2.1. Globus Architecture.

prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when.

Globus architecture is shown in figure 2.1 [8]. The toolkit has 3 components known as pillars. These are: Resource Management, Information Services and Data Management. The toolkit uses the GSI (Globus Security Infrastructure) to provide a common security protocol for each of the pillars. GSI is based on public key encryption, X.509 certificates and Secure Socket layer (SSL) protocol.

2.2.1 GRAM (Globus Resource Allocation Manager)

The Globus Resource Allocation Manager provides a standard interface to all the local resource management tools a site uses. The Globus resource management has the high-level global resource management services layered on top of local resource-allocation services. The GRAM service is provided by a combination of the gatekeeper and the jobmanager. The gatekeeper performs the task of authenticating an inbound request using GSI, and mapping the users global ID on the Grid to a local username. The incoming request specifies a specific local service to be launched, the latter usually being a jobmanager. The user needs to compose the request in a Resource Specification Language (RSL) that is handed over to the jobmanager by the gatekeeper. After parsing the RSL, the jobmanager translates it into the local schedulers language. The GRAM also provides the capability to stage in executables or data files, using Global Access to Secondary Storage (GASS).

2.2.2 MDS (Monitoring and Discovery Service)

MDS stands for Monitoring and Discovery Service. MDS in GT2 was called Meta-computing Directory Service. The features provided in GT2 by the Monitoring and Discovery Service (MDS2) are now provided by the GT3 Information Services component, which is now also known as MDS3. When used in conjunction with standard Open Grid Services Infrastructure (OGSI) mechanisms that provide a consistent way of querying any Grid service about its configuration and status information, these services provide all of the capabilities of MDS2 and more, all within an OGSA-compliant environment [8].

The main part of MDS is LDAP server. The information is gathered by information repositories (GRIS - Grid Resource Information Service) and is organized in trees. The composition of information is facilitated by registration service (GIIS - Grid Index In-

formation Service). The information for each node is gathered by launching executables called information providers.

MDS uses slapd as the LDAP directory server. It implements version 3 of Lightweight Directory Access Protocol. It supports certificate-based authentication and data security (integrity and confidentiality) services through the use of TLS (or SSL). slapd is threaded for high performance. A single multi-threaded slapd process handles all incoming requests using a pool of threads. This reduces the amount of system overhead required while providing high performance.

2.2.3 GridFTP

This is a data transfer protocol based on FTP, highly optimized to give secure and reliable performance in a Grid. Among the various features it provides, the important ones are GSI security, partial file transfers, authenticated data channels and third-party (direct server-to-server) transfers. The protocol also allows developers to add plug-ins for customized reliability and fault tolerance features.

2.3 Condor and Condor-G

Condor [9] provides an efficient job scheduling system for distributed computing environments. It aids in harnessing idle CPU cycles of workstations in a transparent manner to the owner of the idle workstation being utilized. Among other job-scheduling functionalities, it implements check-pointing by saving the current state of a remotely executing job, when it is suspended, to restart it from the same stage.

The Condor-G system integrates the two technologies: Globus and Condor. Condor-G architecture is shown in figure 2.2 [10]. Condor-G incorporates features of distributed resource access for multi-domain environments provided by Globus, and the benefits of

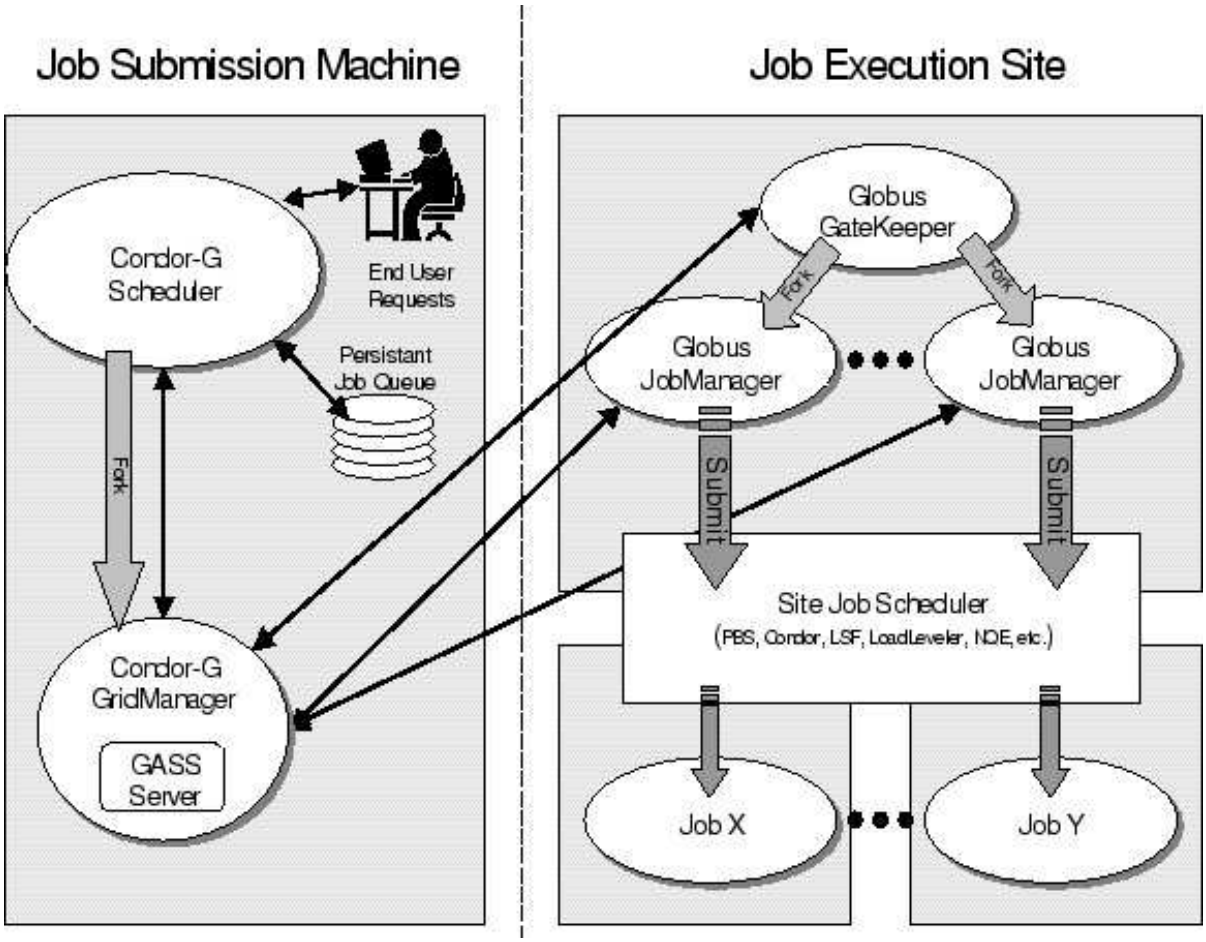

Figure 2.2. Condor-G Architecture.

distributed resource scheduling for a single administrative domain offered by Condor. Condor-G system utilizes various components, chiefly:

1) A Condor-G job scheduler at the submission end

2) A Condor-G Grid manager, that works together with GASS at the submission end

3) A Globus gatekeeper and Globus jobmanager at the execution end.

The authentication and authorization are implemented using GSI mechanisms. Condor-G gives highly reliable performance; it provides precise levels of check-pointing

to resume execution of the jobs from the stage of suspension in a highly transparent and reliable manner.

2.4 CORBA (Common Object Request Broker Architecture)

CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. Distributed object systems are distributed systems in which all entities are modeled as objects. The basic CORBA paradigm is that of a request for services of a distributed object. The services that an object provides are given by its interface. Interfaces are defined in OMG's Interface Definition Language (IDL). Distributed objects are identified by object references, which are typed by IDL interfaces [11].

The Object Request Broker (ORB) is the distributed service that implements the request to the remote object [12]. It locates the remote object on the network, communicates the request to the object, waits for the results and when available communicates those results back to the client. The ORB implements location transparency. Exactly the same request mechanism is used by the client and the CORBA object regardless of where the object is located. The ORB implements programming language independence for the request. The client issuing the request can be written in a different programming language from the implementation of the CORBA object. The ORB does the necessary translation between programming languages. Language bindings are defined for all popular programming languages.

The Object Management Group (OMG) is responsible for defining CORBA. CORBA 2.0 defines a network protocol, called IIOP (Internet Inter-ORB Protocol), that allows clients using a CORBA product from any vendor to communicate with objects using a CORBA product from any other vendor. IIOP works across any TCP/IP implementation.

CHAPTER 3

SAM-Grid Architecture

Details about the architecture of SAM-Grid [13] are discussed in this chapter. SAM-Grid implementation includes technologies like Globus, Condor, Condor-G and CORBA. The SAM-Grid infrastructure is composed of three major components: data handling, job management and information management. Figure 3.1 [6] shows an architectural diagram of the SAM-Grid in which each abstract service is shown with its implementation next to it.

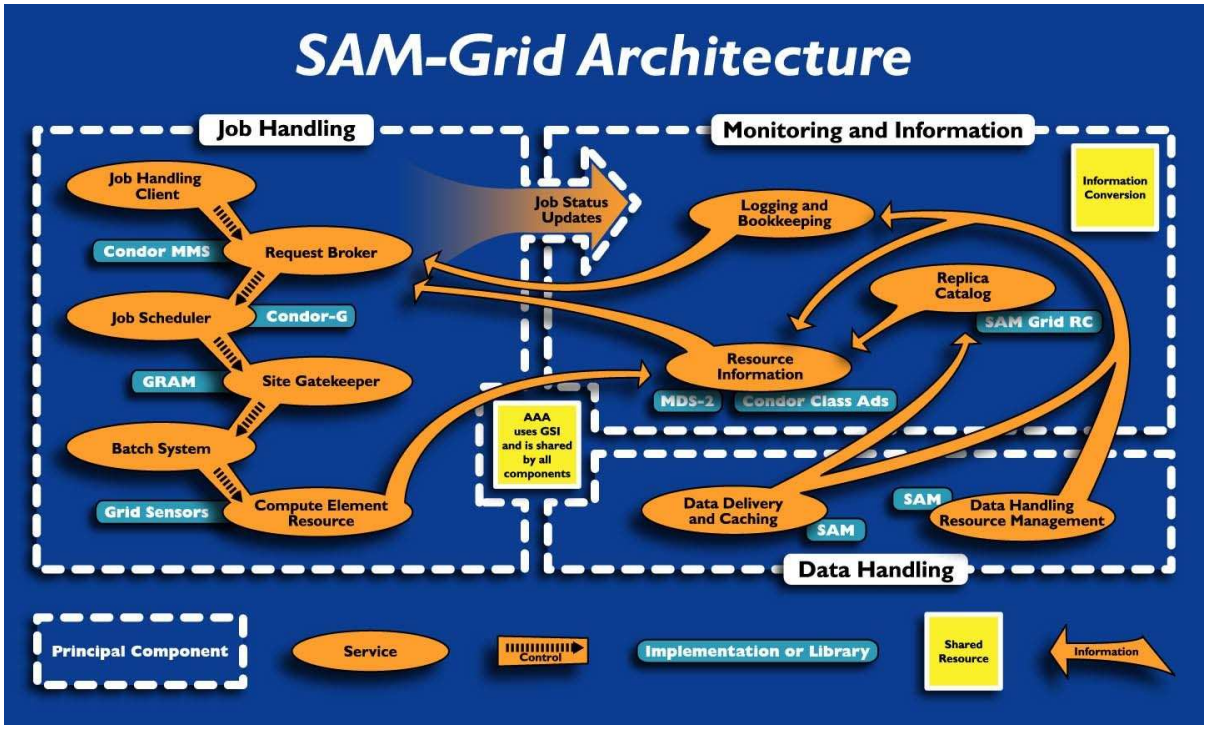

Figure 3.1. SAM-Grid Architecture..

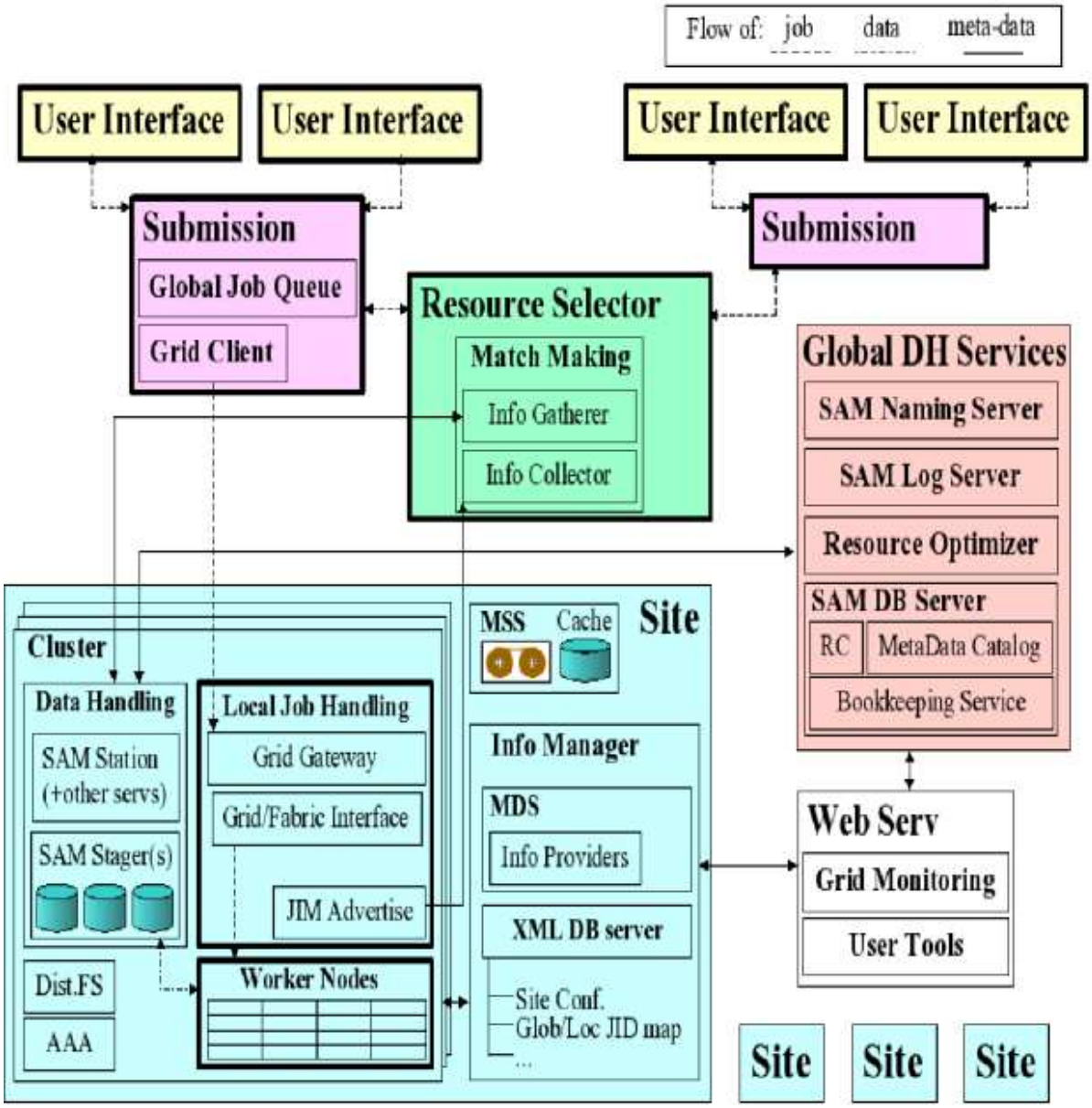

Figure 3.2. Services in SAM-Grid. Each site represents a job execution site.

The data handling system is called SAM (Sequential Access to data via Metadata). Information system uses the interfaces provided by the data handling system. The SAM-Grid resource selection service is based on the information collection mechanisms of the Condor MMS (Match Making Service). Figure 3.2 [16] shows details about various services in the SAM-Grid.

3.1 Data Management component of SAM-Grid

SAM (Sequential Access to data via Metadata) [14] is the data management system used for the experiments in Fermilab. It has the following functions:

1) Store the raw detector data and processed data

2) Maintain a catalog of all data that is present in the system

3) Deliver data to requesting processes

SAM is an acronym for Sequential Access to data via Metadata. The term sequential refers to the layout of physics events stored within files, which are in turn stored sequentially on tapes within a Mass Storage System (MSS). SAM performs the task of transparently delivering files and managing caches of data. It is the sole data management system of the DZero experiment; other major experiments like CDF (Collider Detector at Fermilab) also use this system.

SAM has been designed as a distributed system, using CORBA (Common Object Request Broker Architecture) as the underlying framework. The system relies on compute systems and storage systems distributed over the world. Storage systems have disk storage elements at all locations and robotic tape libraries at select locations. All the storage elements support the basic functionalities of storing/retrieving a file.

Metadata catalogs, replica catalogs, data transformations, and databases of detector calibration and other parameters are implemented using Oracle relational databases. The architecture is organized by physical groupings of compute, storage, network re-

sources termed as Stations. Certain Stations can directly access the tape storage; others utilize routes through the ones that provide caching and forwarding services. The disk storage elements can be managed either by a Station or externally, those managed by Stations together form logical disk caches which are administered for a particular group of physicists.

In SAM, service registration and discovery has been implemented using CORBA Naming Service, with namespace by Station Name. APIs to services in SAM are defined using CORBA IDL (Interface Definition Language) and can have multiple language bindings. UDP (User Datagram Protocol) is used for event logging services and for certain Request Manager control messages. Each disk storage element has a stager associated that serves to transfer or erase a file by using the appropriate protocol for the source and destination storage elements. Rcp, kerberized rcp, bbftp, encp and gridftp are used as the file transfer protocols.

Each Station has a Cache Manager and Job Manager implemented as a Station Master server. The Cache Manager provides caching services and also the policies for each group. The Job Manager provides services to execute a user application either interactively or by using a supported batch system like LSF, FBS, PBS, and schedulers like Condor. Request Managers, which are implemented as Project Master server, take care of the pre-staging of file replicas and the book-keeping about the file consumption. The project master executes for each dataset to be delivered and consumed by a user job. Storage Manager services are provided by a Stations file storage server that lets a user store files in tape and disk storage elements.

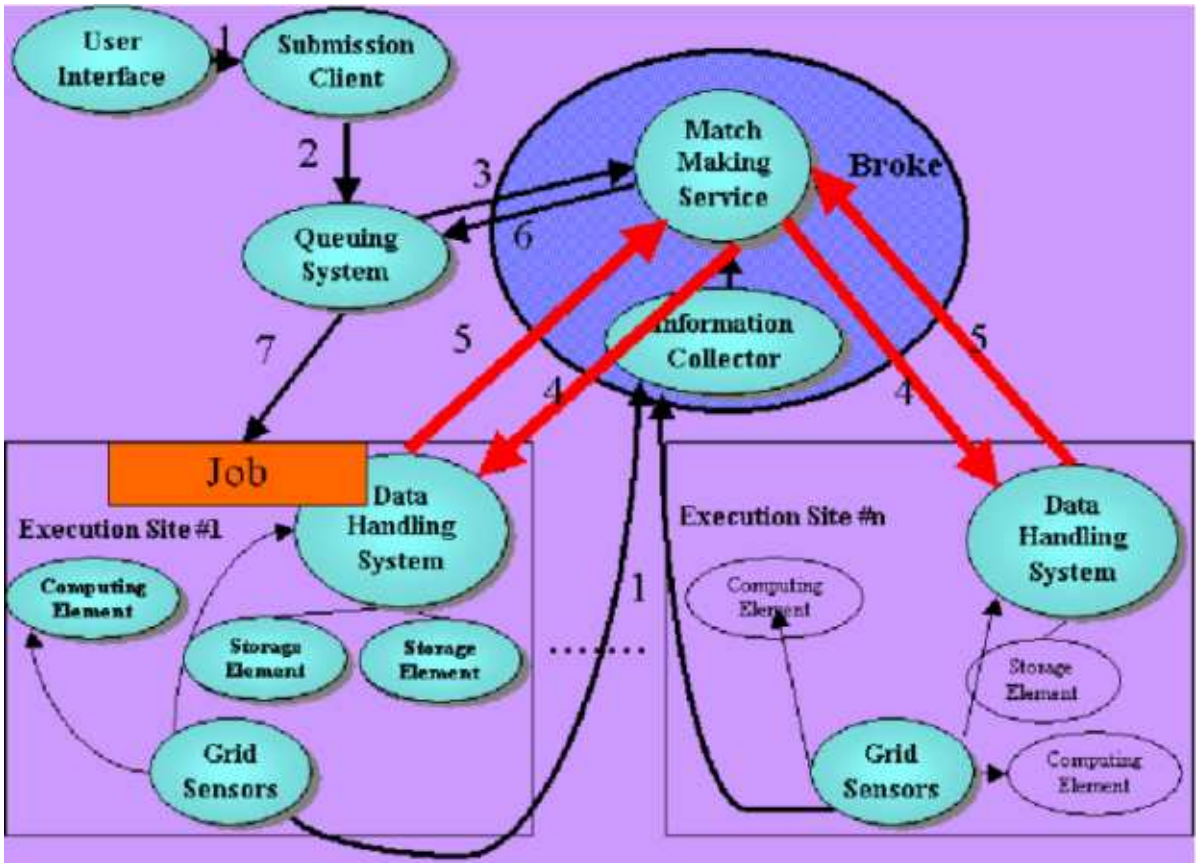

Figure 3.3. Job Management.

3.2 Job Management component of SAM-Grid

The job management component of SAM-Grid handles all life stages of a Grid job, including submission and execution. It is organized in a 3-tier architecture: 1) client site, 2) submission site and 3) execution site

Figure 3.3 [6] shows the architecture of the Job Management infrastructure. A Grid job is submitted from client site using the user interface. Submission site accepts this job and places it in the queuing system. Execution sites advertise themselves to the resource selector. Submission site is responsible for matching the job with the appropriate execution site. Condor MMS (Match Making Service) is used for this purpose. After match making, the job is submitted to the matched execution site.

3.3 Information Management component of SAM-Grid

Information management is composed of configuration, monitoring and logging infrastructures. Information service deals with following types of information:

1) Static or semi-static: These are global parameters of the Grid or the setup of services and resources at a site. Configuration infrastructure handles this.

2) Dynamic: Information associated with the behavior of the entities in the Grid such as resources, services, jobs etc. These are associated with the monitoring infrastructure.

3) Historic: This type of information is handled by the logging infrastructure.

Configuration infrastructure is used for the configuration of the SAM-Grid products and services for a site. It uses Xindice XML database [15]. Apache Xindice is a database designed to store XML data or what is more commonly referred to as a native XML database.

Monitoring infrastructure captures dynamic information. Events relevant to the entities of the Grid are published to the information repositories. We use MDS from Globus toolkit for this purpose. The main part of MDS is LDAP server. The information is gathered by information repositories (GRIS - Grid Resource Information Service) and is organized in trees. The composition of information is facilitated by registration service (GIIS - Grid Index Information Service). The information for each node is gathered by launching executables called information providers. Condor Class-Ads and Xindice XML database are also used by the monitoring infrastructure. More details about the monitoring system are given in chapter5.

Logging infrastructure consists of logging and bookkeeping services. The information that is logged are of three types: data processing history, statuses of services and jobs, and debugging messages [17].

CHAPTER 4

Sun Grid Engine integration

Sun Grid Engine (SGE) [18] is a resource management software for computing clusters. It accepts jobs submitted by users and schedules them for execution on appropriate systems. In this chapter, we discuss about the integration of SAM-Grid with SGE batch system. In the first section, details about Grid fabric interface are given. This interface lies between batch system and the Grid layers outside the execution site.

4.1 Grid Fabric interface in SAM-Grid

The local services and resources at the execution sites are called the Fabric. These services are responsible for coordinating the local resources and executing the jobs that are coming from the Grid. The lack of standard for the basic fabric services has led to the development of ideal fabric services or idealizers. This is an intermediate layer which acts as an interface between fabric and Grid services. This interface adapts the input and the output between the Grid and the fabric in order to comply with the specifics of the fabric. Also, it coordinates the usage of the local resources according to the specifications of the Grid jobs and the local policies. SAM-Grid has a series of job management scripts that use experiment specific interfaces. These scripts are invoked via standard Grid mechanisms, such as the Globus gatekeeper. From these scripts, the invocation of the local batch system commands is done via an intermediate layer which abstracts the basic interactions with the batch system. This layer is called the batch system adapter layer [19, 20].

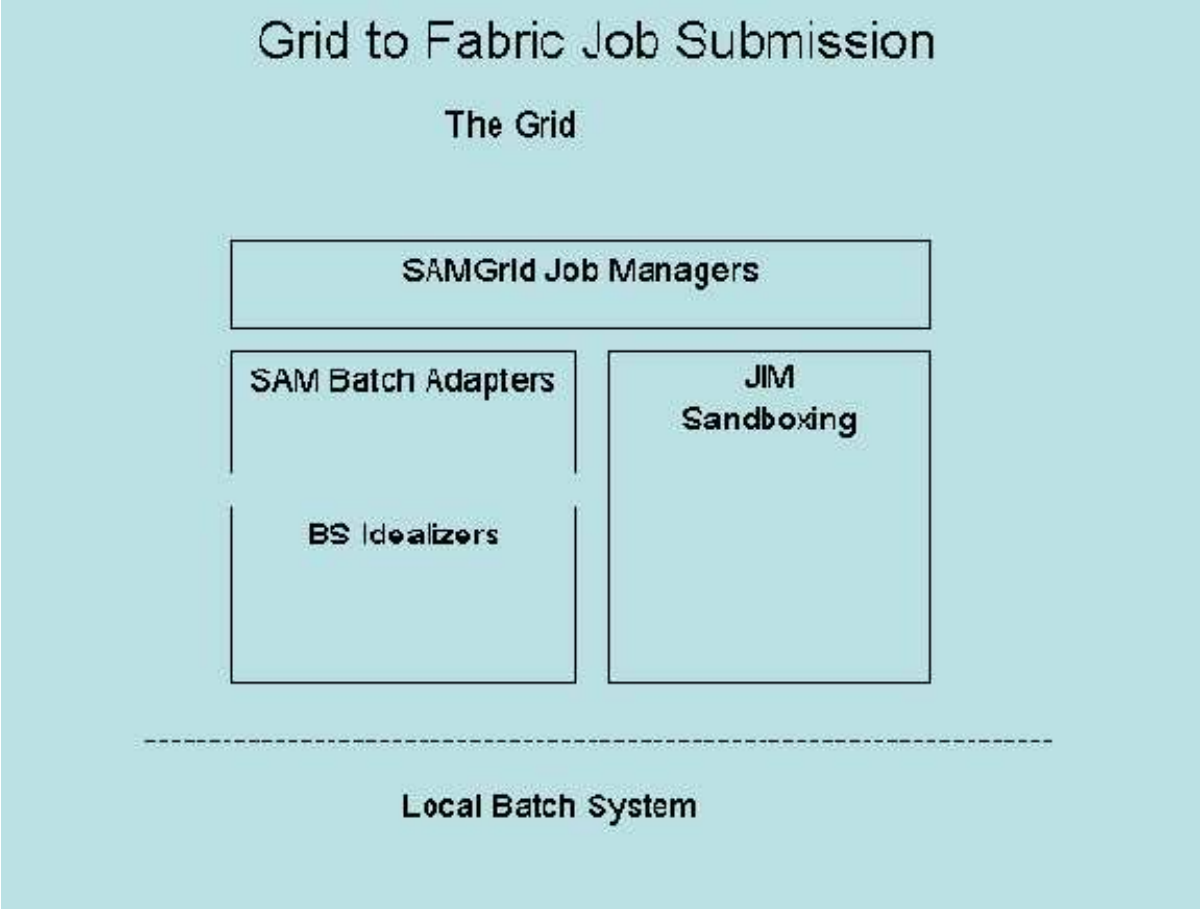

Figure 4.1. Grid to Fabric interface.

Figure 4.1 [17] illustrates the design of the Grid-to-Fabric interface in SAM-Grid architecture. The Grid layer interfaces with the SAM-Grid job managers. SAM Batch Adapters and Batch system idealizers completely abstract the underlying batch system from the job managers. The XML based monitoring service lets the job manager collect more information about the progress of a job.

4.1.1 SAM-Grid Job Managers

The SAM-Grid job managers handles Grid job instantiation at the execution site by mapping a logical Grid job definition to a set of local jobs submitted to the batch system.

The SAM-Grid job managers conform to the standard Globus GRAM protocol and hence can be invoked through the gatekeeper running at the head node of the execution site. Standard Grid tools like Condor-G have a different job manager for each batch system. In SAM-Grid architecture, there is a single job manager for all the batch systems. Instead of the job manager, the lower level components (particularly the batch adapters and the batch idealizers) in the Grid-to-Fabric interface provide the logic to abstract the underlying batch system.

During submission, a Grid job is defined logically as a set of parameters. When a job is submitted to the execution site this set of input parameters is transferred to the SAM-Grid job managers through the gatekeeper. Once invoked the job manager triggers local job submission at the execution site. Firstly, the job manager processes the input parameters and determines the number of jobs that need to be submitted to the local batch system. Next, the job manager initializes a sandbox area for the Grid job using the sandboxing interface. The sandbox area forms the working area for the job manager and all the remaining processing takes place from under this area.

The SAM-Grid job managers depend on the lower layers in the Grid-to-Fabric interface to provide an abstraction of the underlying batch system and its configuration. They use the uniform interface provided by SAM Batch Adapters and the Batch System Idealizers to

1) Submit jobs to any batch system specifying the executable, the input and the directory path under which the output files produced by the local job should be returned by the batch system.

2) Determine the current status of a Grid job. This means given a Grid id, the job managers need to know the list of local jobs and their status, that were created as part of the submission of the Grid job.

3) Kill either a single batch job or all the batch jobs belonging to a Grid job.

In SAM-Grid, each job is categorized into a job type depending on the application being run by the job. Some of the job types are DZero Monte Carlo and DZero Reconstruction. The local job submission results in a set of operations being performed at the head node that depends on the job type of the Grid job. In order to support different job types, the SAM-Grid job managers support the concept of application adapters which are application specific components and perform the pre-submission operations for the application. For example applications such as Reconstruction require a SAM project being started at the head node.

The job manager invokes the job submission process by using SAM batch adapters and Batch System Idealizer. It is important to maintain a mapping between a Grid job and the local jobs so that the status of a Grid job can be tracked. This mapping is created by using the unique Grid id assigned to the Grid job. Essentially during job submission, the job managers provide the id of the Grid job to the batch idealizers which are responsible for creating the mapping. The batch idealizers in turn return to the job manager the id of the local jobs submitted. In order to facilitate more precise monitoring, the job manager creates an entry for each batch job in XML database.

Once the job submission is complete, the job manager is responsible for returning the correct status of the Grid job to the Grid manager running at the submission site. The Grid manager periodically sends a GRAM poll request to the execution site to determine the status of the Grid job. This request is received by the job managers at the execution site. A Grid job is considered to be active as long as there is at least one active or queued local job belonging to the Grid job in the batch system. In order to determine the status of the Grid job, the job manager invokes the batch idealizers supplying them the Grid id. The batch idealizers return to the job manager the list of corresponding local jobs and their batch system status. The job manager analyzes the batch idealizer output to determine the Grid job status. The job manager also updates XML database with the

current status of each local job. If it is determined that the Grid job has finished, the job manager collects the output files and the log files from the sandbox area and transfers them to the submission site using GRAM. It then triggers the cleanup of sandbox area for the Grid job and finally returns the appropriate GRAM status back to the Grid manager. The job manager also provides an interface to terminate a Grid job which when invoked (through GRAM) results in the local jobs at the batch system level being terminated.

4.1.2 SAM Batch Adapters

This layer is configured to interpret the outputs of the batch system commands to enable the extraction of relevant information such as local job id after submission, the status of the job after lookup or the error messages after any command invocation. Figure 4.2 [16] shows the interaction between SAM-Grid jobmanager and batch adapters.

SAM Batch Adapters is a package that provides the job manager interfaces to interact with the batch system. The interfaces themselves are implemented in the Batch System Idealizers. The SAM Batch adapter package is implemented in Python. It has a command line interface and also provides a Python API. The package is fully configurable and does not make any assumptions about the underlying batch system. Thus it can handle any batch system. The configuration of the package is stored in a local python module which can be updated using an administrative interface the package provides. Each command stored in the configuration has a command type associated with it. The command types used are: job submit, job kill and job lookup. Each command has a command string associated with it which may contain any number of predefined string templates. String templates are used for plugging the user input into a command string, which then gives a command that the user can execute to get the desired results. For example, the command string for the job lookup command is ὰqstat %__BATCH_JOB_ID__ὰ The user or the client can read the command string giving its command type and then

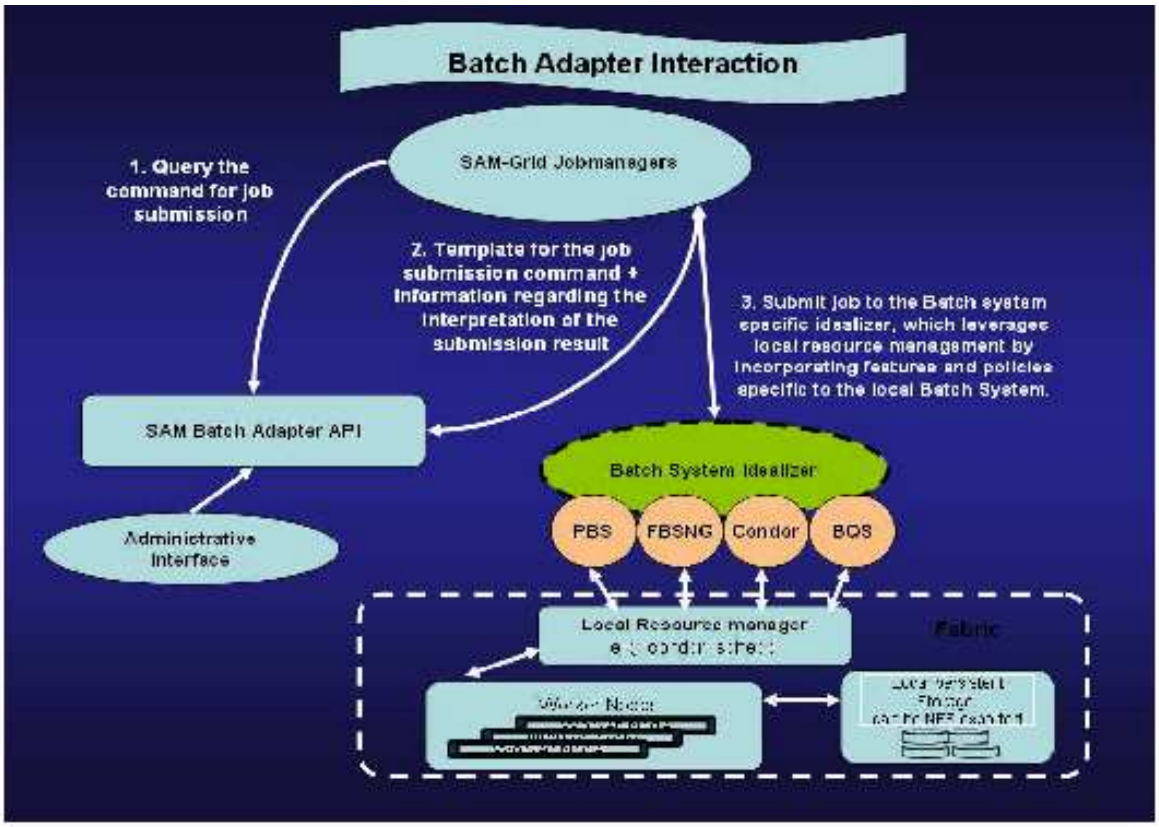

Figure 4.2. Batch Adapter interaction.

replace the template string which in this case is %_BATCH_JOB_ID_ with the id of a local job and then execute the resulting command to perform lookup on a single batch job.

It is important to note that the SAM Batch Adapter itself does not execute the commands to perform Batch System operations. It just provides a functionality to prepare commands for execution. It is the responsibility of the API user to execute commands and interpret their results. In SAM-Grid, the batch system functionalities are provided within Batch System Idealizers. Thus in SAM-Grid, the command strings in SAM Batch Adapter configuration contain the idealizer scripts rather than batch system commands. For example the command string for a look up command looks something

like sam_sge_handler.sh job_lookup. The idealizer script in this case is sam_sge_handler.sh which provides the interface to SGE batch system. Most of the batch adapters in SAM-Grid are written in UNIX shell scripts and a few are written in Python. The batch systems that are currently supported by SAM-Grid i.e. the batch systems for which an idealizer exists are PBS, Condor, FBSNG, BQS and SGE.

4.1.3 Batch System Idealizers

There are mainly 3 types of idealizers in SAM-Grid:

1) Batch adapter: Uniform interface to any underlying local job scheduler.

2) Dynamic product installation: Used to recreate the job environment in the worker nodes of a batch system.

3) Local sandbox management: Responsible for packaging and delivering within the cluster, the software needed to run the job and for gathering the job's output.

Batch System Idealizers together with SAM Batch Adapters provide a complete abstraction of the underlying batch system to the job managers. Batch System Idealizers implement the interfaces required to perform batch system operations such as submitting jobs and checking the status of a job. SAM Batch Adapter is just an interface to invoke the idealizer scripts. Batch System Idealizers as the name implies, idealize the batch systems to make their interactions with the Grid machinery easier by mitigating any imperfections and adding any missing features. The idealizer scripts are totally batch system specific and a new batch system can be added to the Grid infrastructure by writing an idealizer script for the batch system [19].

Different administrators configure the batch system differently because of local constraints. Thus, the users have to submit the jobs using different formats. Submission options are nonstandard and site specific [17]. Some times, special attributes have to be used to adhere to the resource usage agreement. Otherwise, the jobs will not be executed

by the batch system. Also, typical HEP applications are seldom submitted directly to the batch system. They are submitted through experiment specific local interfaces that take care of job preparation. Such preparation include triggering of the data handling systems such as SAM, the use of local job sandboxing mechanisms and the decomposition of job into smaller tasks which are submitted to the batch system.

We can use the interfaces provided by the batch system itself, along with SAM Batch Adapters to provide an abstraction of the batch system. However there are lot of problems that are exposed when the batch system interfaces are used in a Grid scenario. One problem that almost all the batch systems suffer with is the transient failures in executing some commands. While these transient failures may be acceptable to an interactive user, in a Grid scenario they will cause the Grid job to fail. For example if the execution of a batch system polling command fails due to a network glitch or the command times out because of heavy load on the server, the job managers will fail to report the correct status back to the Grid job manager at the submission site. Grid job manager will falsely interpret the job as a failure and proceed with clean up operations. In order to overcome the problems with transient failure in batch system commands, retrials are incorporated with every batch system command in the batch idealizers.

The output produced by a batch system command needs to be parsed by the job managers to interpret the results. However the output produced by commands in different batch systems differs from each other. Also the status of a batch job is represented in different ways in different batch systems. For example some batch system represent the status of a running job simply as running while some batch systems may call it active. The batch idealizers convert the output of the batch system command to a uniform format. They also perform a mapping of the batch system status to a set of status that the job managers understand. The statuses that are currently supported are: active,

failed, suspended, pending, and submitted. As an example if a batch system reports the status of a job as queued the batch idealizers will report its status as pending.

There is a need to create a mapping between a Grid job and the local jobs that were submitted as part of the Grid submission. This mapping is created in a totally batch system specific way. In general, the way this mapping is created varies from one batch system to another. There is a need to abstract the creation of this mapping from the job managers. The Idealizers create a mapping between the Grid job and the local jobs in the batch system. To create this mapping the idealizers accept a unique id associated with the Grid job and associate it with local jobs in a batch system specific way.

In a cluster every worker node has a certain amount of scratch space reserved for a local job that serves as its working area. This ensures mutual isolation between jobs that get scheduled to the same node simultaneously. Not all the batch systems provide support for scratch management at the worker nodes where the actual computation takes place. For example some batch systems like Condor provide a full fledged scratch management support while some other batch systems like PBS do not have any support for scratch management. The batch idealizers provide scratch management support for batch systems that do not already do so. This feature is provided by writing special scratch management scripts that only the batch idealizers know about thus abstracting this limitation from the job managers. The scratch management scripts are staged to the worker nodes using the batch system and form the first stage of execution. The path of the scratch area at worker nodes is read from configuration at the head node. The scratch management scripts create a separate work area or directory for the local job under this path. They then launch the user executable from this unique scratch area. After the job finishes they clean up the directory associated with job and return the appropriate exit status to the batch system.

Another problem that is prevalent in cluster computing is what we term as the Black Hole Effect. In a cluster if even a single node has a configuration problem or hardware problems which results in jobs failing quickly, it reduces the turn around time at that node. This results in the batch system scheduling more and more jobs to the same node, not knowing that they will fail as well. This results in the faulty node acting like a black hole and eating up a lot of jobs from the batch system queue. The batch system idealizers provide a partial solution to the Black Hole Problem by maintaining a neglect list, which contain the names of the nodes known to have such problems. While job submission, they explicitly ask the batch system not to schedule jobs to nodes in the neglect list.

The batch idealizers need not have the same interface because they are invoked through SAM Batch Adapters. However a uniform interface makes configuration of SAM Batch Adapter lot easier and hence it is desired. All the batch idealizers accept three actions: job_submit (To submit a job to the batch system), job_lookup and job_kill. The arguments are supplied to the idealizers using the concept of template substitution.

4.2 Sun Grid Engine (SGE)

The Grid engine system is an advanced resource management tool for heterogeneous distributed computing environments. Workload management is accomplished through managing resources and administering policies. The use of shared resources is controlled to best achieve an enterprise's goals. Sites configure the system to maximize usage and throughput, while the system supports varying levels of timeliness and importance. Job deadlines are instances of timeliness. Job priority and user share are instances of importance [21].

The Grid engine software provides advanced resource management and policy administration for UNIX environments that are composed of multiple shared resources.

The Grid engine system is better than standard load management tools with respect to the following major capabilities:

1) Innovative dynamic scheduling and resource management that allows Grid engine software to enforce site-specific management polices.

2) Dynamic collection of performance data to provide the scheduler with up-to-the-minute job level resource consumption and system load information.

3) Availability of enhanced security by way of Certificate Security Protocol (CSP)-based encryption. Instead of transferring messages in clear text, the messages in this more secure system are encrypted with a secret key.

4) High-level policy administration for the definition and implementation of enterprise goals such as productivity, timeliness, and level-of-service.

The Grid engine software provides users with the means to submit computationally demanding tasks to the Grid for transparent distribution of the associated workload. Users can submit batch jobs, interactive jobs, and parallel jobs to the Grid. SGE system accepts jobs submitted by users and puts them in holding area until they can be executed. It manages the jobs during execution and logs the record of their execution. Four types of hosts are fundamental to the Grid engine system: Master host, Execution hosts, Administration hosts and Submit hosts. The master host is central for the overall cluster activity. It runs the master daemon, sge_qmaster, and the scheduler daemon, sge_schedd. Both daemons control all Sun Grid Engine components, such as queues and jobs, and maintain tables about the status of the components, about user access permissions, and the like. By default, the master host is also an administration host and submit host.

Execution hosts are nodes that have permission to execute Sun Grid Engine jobs. Therefore, they are hosting Sun Grid Engine queues and run the Sun Grid Engine execution daemon, sge_execd. Permission can be given to hosts to carry out any kind of administrative activity for the Sun Grid Engine system. Submit hosts allow for submit-

ting and controlling batch jobs only. A user who is logged into a submit host can submit jobs via qsub, can control the job status via qstat, and can use the Sun Grid Engine user's interface, QMON.

Four daemons provide the functionality of the Sun Grid Engine system. Master Daemon is the center of the clusters management and scheduling activities, maintains tables about hosts, queues, jobs, system load, and user permissions. It receives scheduling decisions from sge_schedd and requests actions from sge_execd on the appropriate execution hosts. The scheduling daemon maintains an up-to-date view of the cluster status with the help of sge_qmaster. It makes the scheduling decision of which jobs are dispatched to which queues. It then forwards these decisions to sge_qmaster, which initiates the required actions. The execution daemon is responsible for the queues on its host and for the execution of jobs in these queues. Periodically, it forwards information such as job status or load on its host to sge_qmaster. Sge_commd is the Communication Daemon. The communication daemon communicates over a well-known TCP port. It is used for all communication among Sun Grid Engine components.

A Sun Grid Engine queue is a container for a class of jobs allowed to execute on a particular host concurrently. A queue determines certain job attributes; for example, whether it may be migrated. Throughout their lifetimes, running jobs are associated with their queue. Association with a queue affects some of the things that can happen to a job. For example, if a queue is suspended, all the jobs associated with that queue are also suspended. In the Sun Grid Engine system, there is no need to submit jobs directly to a queue. You only need to specify the requirement profile of the job (e.g., memory, operating system, available software, etc.) and Sun Grid Engine software will dispatch the job to a suitable queue on a low-loaded host automatically. If a job is submitted to a particular queue, the job will be bound to this queue and to its host, and thus Sun Grid Engine daemons will be unable to select a lower-loaded or better-suited device.

4.3 Result of Sun Grid Engine integration

SGE is different from the other batch systems. It does not do stage-in of input files and stage-out of the outputfiles like PBS [22]. There are also differences in the way the queues are configured. A batch adapter layer is designed to match the special requirements of the SGE system. This layer uses the idealizer to submit jobs to the SGE batch system and provide interface to monitor the job and terminate it.

The batch system idealizer for SGE is implemented as a UNIX shell script called sam_sge_handler.sh. Job submission to SGE requires the creation of a job description file (JDF). sam_sge_handler.sh script creates this JDF at the time of job submission before invoking the batch system command. The mapping of the Grid-id and local jobs is created in the JDF. When performing lookup, the batch system is asked for only those jobs that were submitted with a specific mapping of the Grid job.

Scratch management for SGE is done by the sge_scratch_setup.sh script. This scripts and the user executable are transported to the worker through the batch system. Upon its execution the scratch management script creates a unique directory (based on local job id) for the job and then launches the user executable. When the user executable finishes, the scratch management script then cleans up the job area in the scratch disk. If the job is deleted from the batch system, its scratch area is left dangling. This problem is eliminated by having the scratch management script examine the scratch area and cleaning up any directories belonging to jobs killed. Thus if the scratch directory for a job is left dangling it could be cleaned when the next job is scheduled at that node.

Sun Grid Engine is now integrated with SAM-Grid and is used for production in one of the execution sites. SAM-Grid uses SGE batch adapter and SGE idealizers while interacting with the Grid Engine. Users are able to submit jobs to SGE and make use of the Grid Engine features.

CHAPTER 5

Enhancement of Monitoring System

Monitoring is a very important part of Grid computing. It helps in the detection of faults and failures of the system. Also, it helps in identifying various states of the applications running in the system. SAM-Grid infrastructure now handles large number of jobs submitted to various sites around the world. Users of SAM-Grid need monitoring features which easily show the states of jobs and resources. An efficient monitoring system is required for this purpose. For making monitoring easier for users of the SAM-Grid, monitoring system has been enhanced with new features. Some of these new features help to scale the monitoring system for monitoring large number of jobs. This chapter presents the enhancements done on the monitoring infrastructure of SAM-Grid.

5.1 Overview of the SAM-Grid Monitoring System

SAM-Grid monitoring system [24] helps in monitoring a job from submission through execution. This includes monitoring the submission site, tracking the progress of the submitted job and reaching the unknown execution site from the known submission site. It is possible for the monitoring system to retrieve information from different realms (e.g., Condor, Globus MDS, SAM) and perform integration over all these sets to provide the user with meaningful and comprehensive monitoring data. Only the relevant information is presented to the user, all the back-end mechanisms are transparently abstracted from the front-end layer of the system (the only layer visible to the user) [23].

The monitoring system takes into account the demands of a distributed environment, and is tolerant of failures that may occur in parts of the Grid. The unavailable,

unreachable or under-performing entities (for instance, can be an entire site) does not interfere with the performance of the system. The system provides access to the largest subset of information that can be made available to the user in the event of failure in parts of the Grid.

The system facilitates scalability in increasing number of the entities of the Grid. It is easy to configure and reconfigure it to reflect the new components of the Grid as the need arises. There is a standard way of representing data related to the information about the various entities of the Grid, viz., SAM-stations, SAM-projects, SAM-groups and SAM-users. This standard representation, allows seamless retrieval of information across the entire grid. Dynamic retrieval of information is possible. Ubiquitous presentation through the Internet is another feature of the system. The information is available anytime, anywhere to anyone by means of web browsers using the Internet. The information is generated and collected from the Grid, processed for integration, and finally presented at a users browser.

The architecture of this Monitoring System partitions the components of the Grid according to the functionality exhibited by them as submission sites, execution sites and monitoring sites. Monitoring site collects information about submission sites and execution sites. Submission sites are a subset of the schedulers on the Grid. The user specifies the requirements of the job in the form of a Job Definition Language (JDL) that is parsed and the job is passed on to the broker for match-making. After a successful match according to the users requirements and the best available resource, the job is finally routed to the appropriate execution site.

Information about execution site is retrieved using different Grid middleware than the one used to retrieve information about submission sites. Globus MDS is used to monitor the execution sites, whereas Condor-G is utilized for monitoring of the submission sites. The SAM system is present beneath these Grid middleware realms. The monitor-

ing system performs integration over information retrieved from these different software domains to provide a coherent view of the Grid.

Grid sensor or information provider is a service that is used to access current information about the individual grid-entities, and to notify the monitoring system of the availability of this information. Information providers need to be deployed at each monitoring site. They have the capability to retrieve information from the Grid components at various levels (monitoring site level, execution site level, and job level) and provide it to the backend layers of the monitoring system. Information servers are distributed over the Grid, and are responsible for retrieving monitoring information from the Grid using the Grid sensors. The information servers serve as one of the backend layers to the monitoring system. The system utilizes the slapd servers of OpenLDAP software provided with the Globus MDS, as information servers.

5.2 Architecture of the Monitoring System

A collection of Linux Apache webserver (that has been configured to utilize dynamically loaded modules) and a set of PHP (Hypertext Preprocessor) scripts bundled as a software package serve as an engine to the monitoring system. These PHP scripts process the information retrieved from the backend layers, and render dynamically produced web pages. The integration of information is also largely performed by this engine.

The architecture of the system is of an on-demand nature for most of its components. Hence for those components, only an information request leads to its retrieval from the Grid. This makes the monitoring system incur less cost on the system-resources being utilized for the information generation and processing. However, for monitoring the information from the submission sites, the architecture relies on periodic generation of data by Condors internal mechanisms. The Monitoring System utilizes Globus MDS

and Condor ClassAds for data representation. Condor ClassAds provide information generated by Condor or Condor-G about the entities of the Grid.

5.2.1 Globus MDS

The MDS information model organizes related information into well-defined collections known as entries. MDS contains several entries; each represents an instance of a type of object. Information about an entry is represented by attributes, with name-value pairs. In order to identify an MDS entry uniquely, it needs to have a unique Distinguished Name (DN). All the entries form a hierarchical name space called a Directory Information Tree (DIT). The DIT provides a faster and simpler way to search for a particular entry. The DN for a specific entry can be constructed using the entries on path from the DIT root to the node of the entry. Within the DIT, each entry is associated with a user-defined type, known as Object Class.

The Monitoring System uses the extensions to the MDS/LDAP-defined standard object class definitions. A representation of the DIT designed for the system is available in [23]. All the attributes of an entry are characterized in the object class definition. An inheritance relation can also be made in this definition that extends an existing object class definition. Each definition contains the attributes which must be always present in the information, along with other optional attributes. More complex structures can be defined using attribute names that are themselves Distinguished Names, to represent the graphical nature of entities in a Grid environment.

5.2.2 Class-Ads from Condor

Condor and Condor-G have built-in mechanisms for representing resources, jobs, and schedulers. This information is used for brokering and match-making of jobs sub-

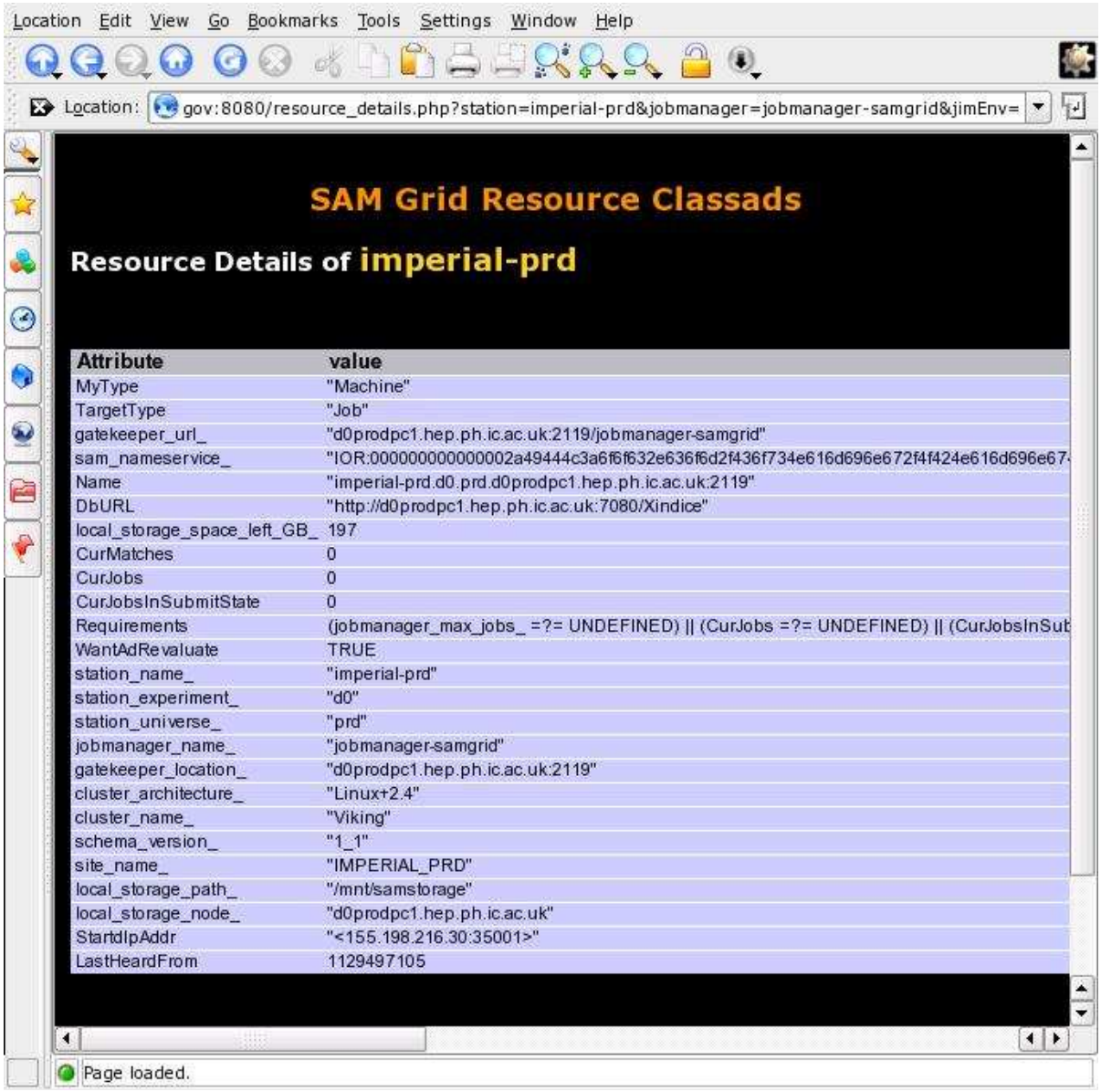

Figure 5.1. Resource Class-Ad.

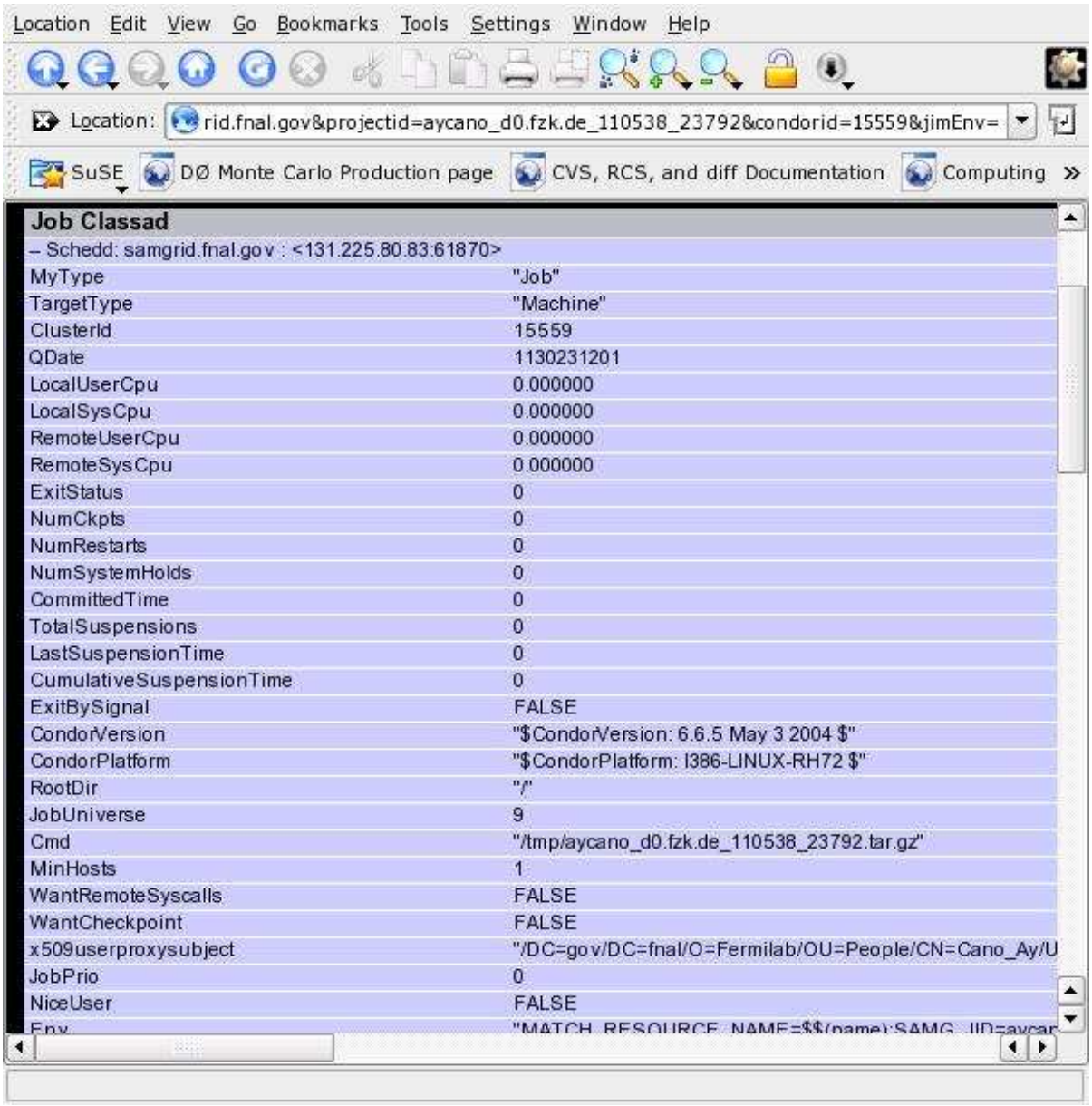

Figure 5.2. Job Class-Ad.

mitted to the Grid. The monitoring system makes use of this information and integrates it with the information received from other sources.

Resource Class-Ad is used for advertising a resource. Figure 5.1 shows the resource Class-Ad of an execution site in SAM-Grid. As long as an Ad is alive, the resource is considered available for brokering, match-making and execution purposes. Similar but more complicated Ads are used for identifying Jobs and Schedulers on a grid. Figure 5.2 shows a Job Class-Ad.

The attributes uniquely identify this entity (resource or job) amongst others of the same category. Class-Ads for jobs are designed with a more complex representation. They are utilized to define the various requirements the job seeks to match with a resource on the Grid, the files involved, various arguments to be passed, and other architecture details.

5.2.3 Information Processing Layers of the System

The Monitoring System comprises of different layers that define the processing of information. Following is a listing of these layers along with a brief description of the configuration features and processing of information that takes place in the monitoring system pertinent to the corresponding layer.

1) Information Generation: At an execution site, the information is configured in XML at the initialization of the monitoring process. This XML formatted information for each grid-component is then processed and configured into shell scripts to be used by Information Services. At a submission site, similar shell scripts are used by Condor and Condor-G Services.

2) Information Transformation: Grid sensors (information providers) utilize the shell scripts to gather information about the state of the site. This information is then delivered to the MDS/LDAP interface that provides a coherent view of the site in con-

formance with the LDIF (LDAP Data Interchange Format) standard. Grid sensors also provide data to Condor interface, that delivers information in the form of Class-Ads.

3) Information Integration & Filtering: All the information available to the Layer 3 is extracted from Layer 1 and Layer 2. Loaded with Apache web services, PHP (Hypertext PreProcessor) scripts perform extensive processing of this data.

4) Information Presentation: After all processing, the information is prepared to be served to a remote user. The user receives a coherent view of the entire grid, and can navigate through the various grid-components using a web interface.

Layers 1 and 2 have two parallel streams of data flow and these are integrated in the third layer. One stream builds upon the globally-distributed Information Servers, whereas the other stream has the globally-distributed Condor Services as their foundation. Layer 3 plays an important role in extracting data from the lower layers. The information is processed, filtered and integrated in this zone. Thereafter, the monitoring data is prepared to be served to users in a dynamically rendered HTML format. The main advantage of choosing HTML at a users end is that, it is widely adopted by most browsers. It also eliminates the need to download plug-ins at the users end.

5.3 Enhancements in Monitoring

The Monitoring System works on logically-partitioned and distributed subsets of the Grid. These subsets comprise of monitoring sites, submission sites and execution sites. Some times, individual components may not be available to the Grid temporarily, due to system or configuration problems; or an entire site may suffer network failure. Monitoring System is robust enough to notice such temporary failures, provide information about which parts of Grid have failed and provide the reasons of failure. Monitoring system has been enhanced with new features to improve the monitoring capabilities. These new features make monitoring easier for users of the SAM-Grid. Some of the features make

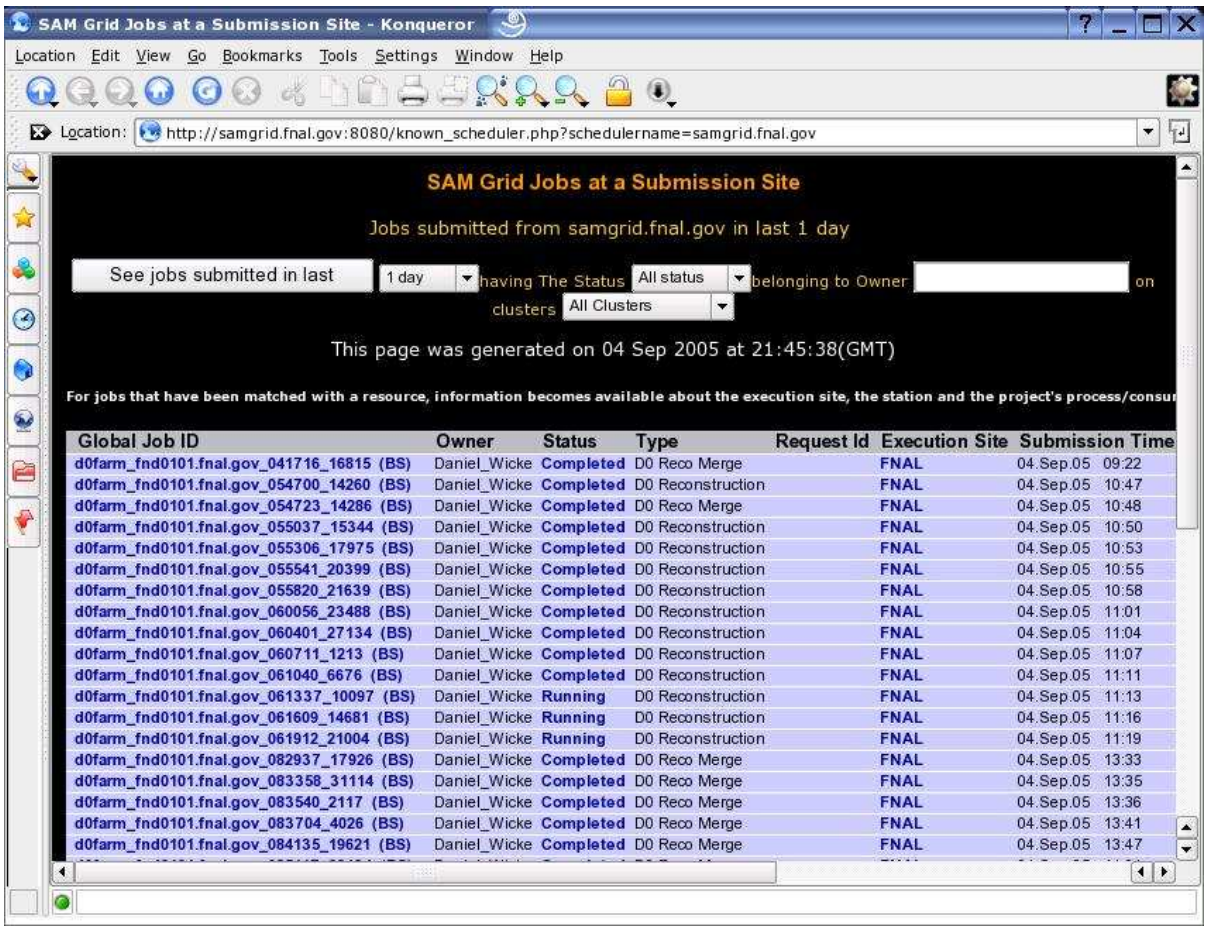

Figure 5.3. SAM-Grid submission site.

it easier for monitoring large number of jobs. Also, monitoring is now possible for many types of jobs including reconstruction.

The implementation of the System directly relies on Linux, Apache/PHP, Shell-Awk scripts; and indirectly relies on C++, Python, XML, Condor, Globus-MDS, and OpenLDAP among other technologies. Areas of monitoring system with enhanced features are described below.

Monitoring of a submission site is the starting point for monitoring Grid jobs. Figure 5.3 shows all the jobs submitted to the submission site on a given date. This

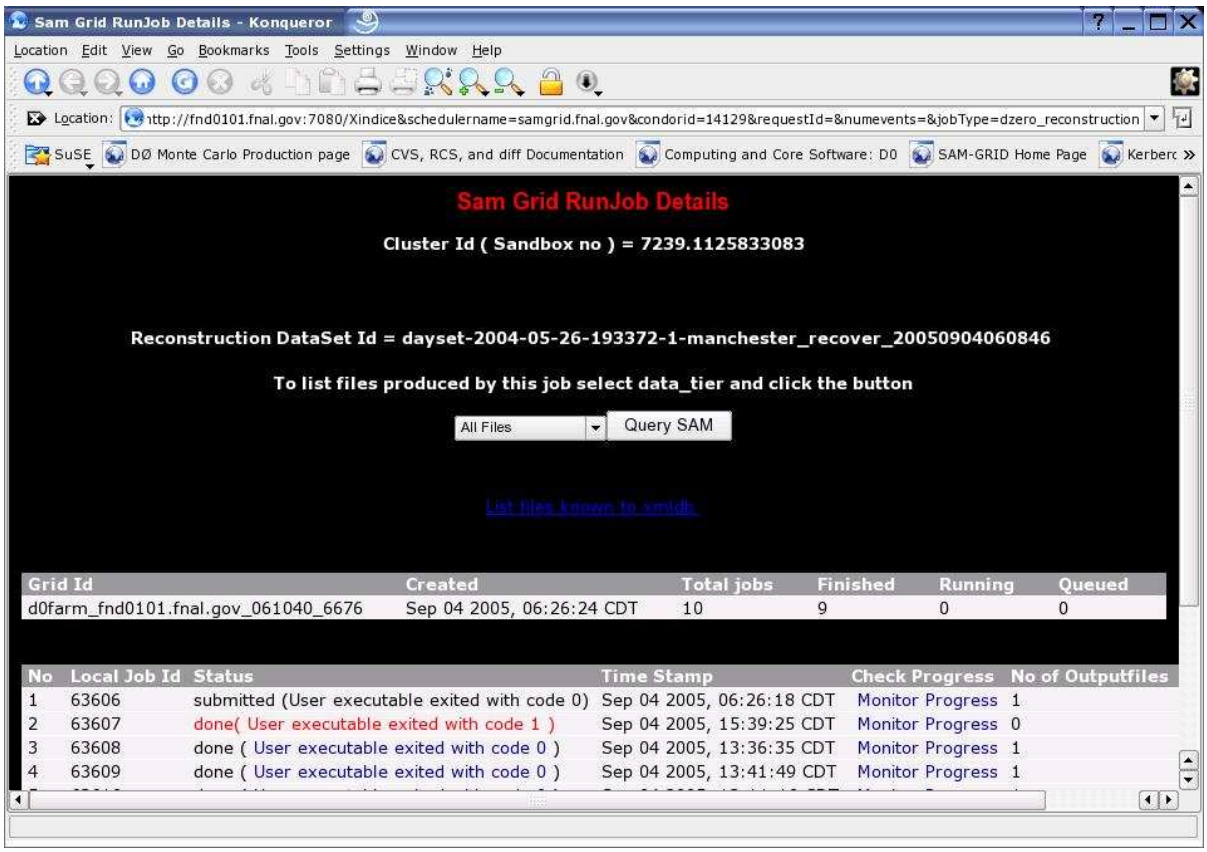

Figure 5.4. Grid job Details.

interface has been enhanced to support monitoring of several new applications. There is a batch system link to monitor the progress of batch jobs in the cluster. The status of a Grid job (submitted, running or completed) is shown in this interface. Other details like application type, owner, execution site and submission time are also shown here.

Monitoring of a job is possible at each stage as it moves through different layers of SAM-Grid. Each job has a global job id associated with it. Once a Grid job reaches the execution site, it might have several batch jobs created corresponding to that. Monitoring system allows us to track the different stages of each batch job.

Next enhanced interface is the one used to view details of a Grid job. Figure 5.4 shows the details of a Grid job submitted to the SAM-Grid system. All the batch jobs

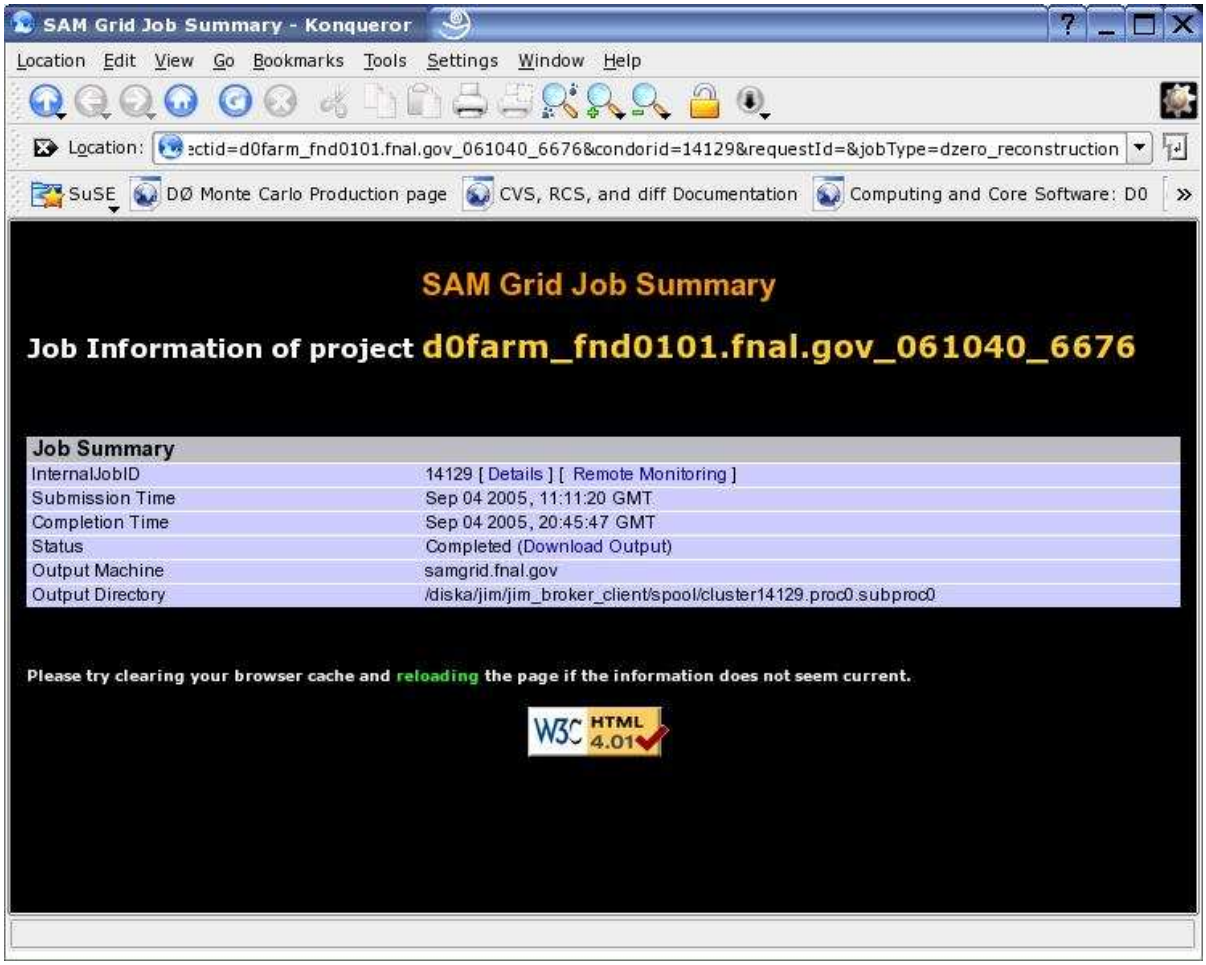

Figure 5.5. Grid job Summary.

created for this job will be shown here. There is a summary of the Grid job, which shows the number of batch jobs which are finished, running and queued. The status of each batch job will be shown with its exit status. The sandbox used by the Grid job is displayed here. This is useful for administrators of the local sites where the job runs. It is also useful while debugging an issue. For reconstruction jobs, the dataset used by the Grid job is shown here. There is a link to monitor the progress of individual batch job. Other details like number of output files created by each batch job are shown here.

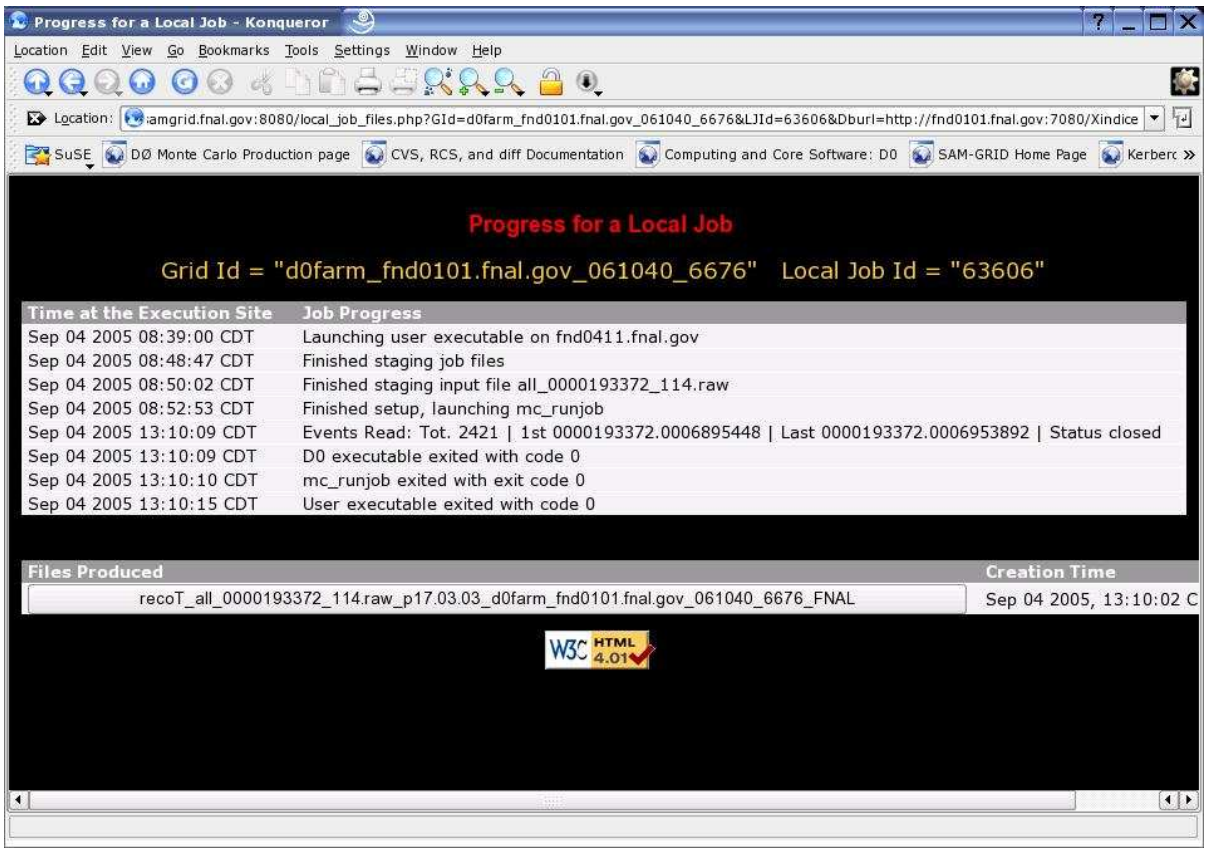

Figure 5.6. Progress details of an individual batch system job.

Figure 5.5 shows the summary of a Grid job. Details like submission time, current status, output machine and output directory are available here. There is a link to monitor the status of individual batch jobs as in figure 5.4. Once the Grid job is completed, an output file can be downloaded from this interface. This file has log files created by individual batch jobs. This is really useful for site administrators and for debugging.

Another enhancement was on the interface which shows the progress details of an individual batch system job created for a Grid job. Figure 5.6 illustrates this. Global id of the Grid job is shown along with batch job id. Job progress is obtained from the XML database. Each event is shown with its time stamp. If the batch job successfully completes, there will be one or more output files created depending on the application

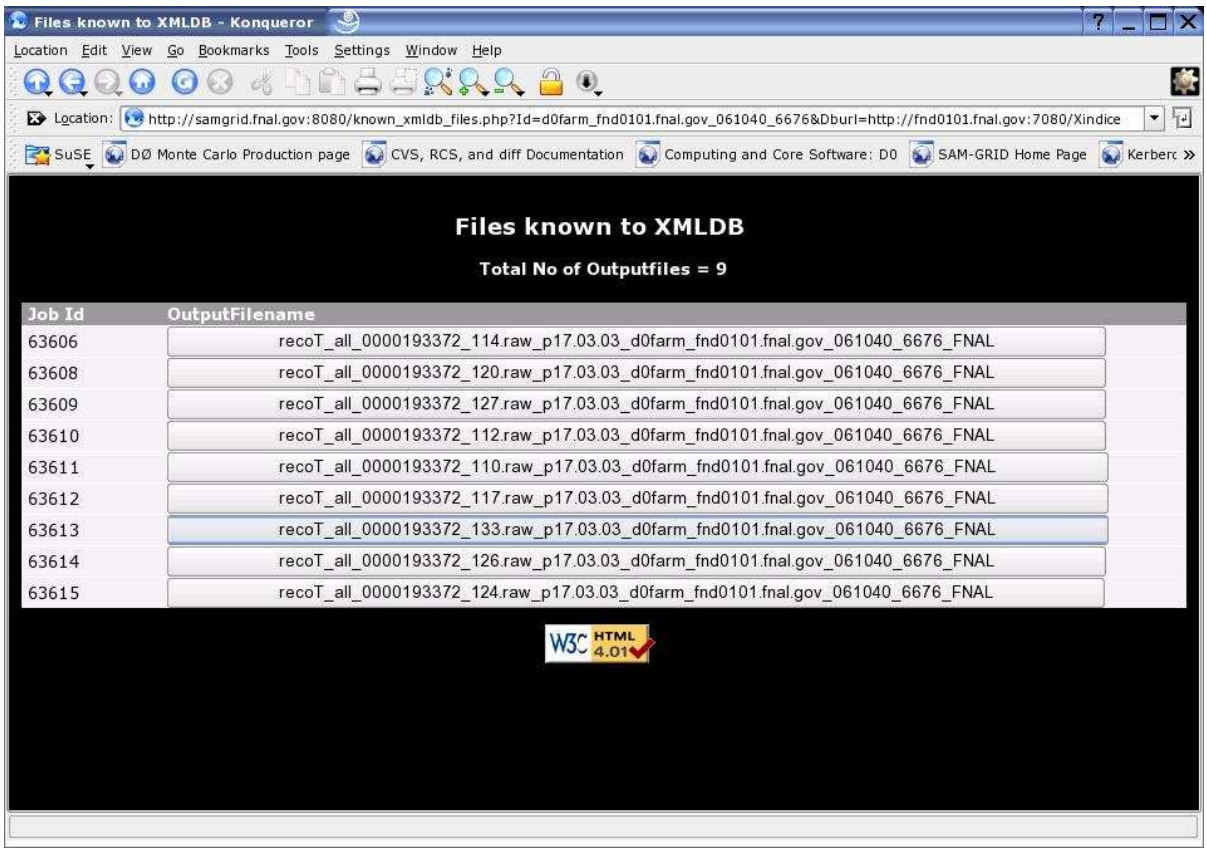

Figure 5.7. Output Files created by a batch job.

type. Each output file produced by the job is shown along with other details like creation time. There is a link to the interface which will show more details about the output file. This interface uses SAM web query with results as shown in figure 5.8.

A new interface is now available for viewing job output files. The figure 5.7 lists all output files created by a Grid job, which are logged in XML database. All the batch jobs which have produced an output are shown here, along with their output files. Thus it is very easy to find out the individual batch job corresponding to an output file and vice versa. The batch jobs are shown in ascending order. From this interface, it is easy to find out the batch jobs which have failed to produce outputs. Here also, there is a link to the interface which will show more details about the output file.

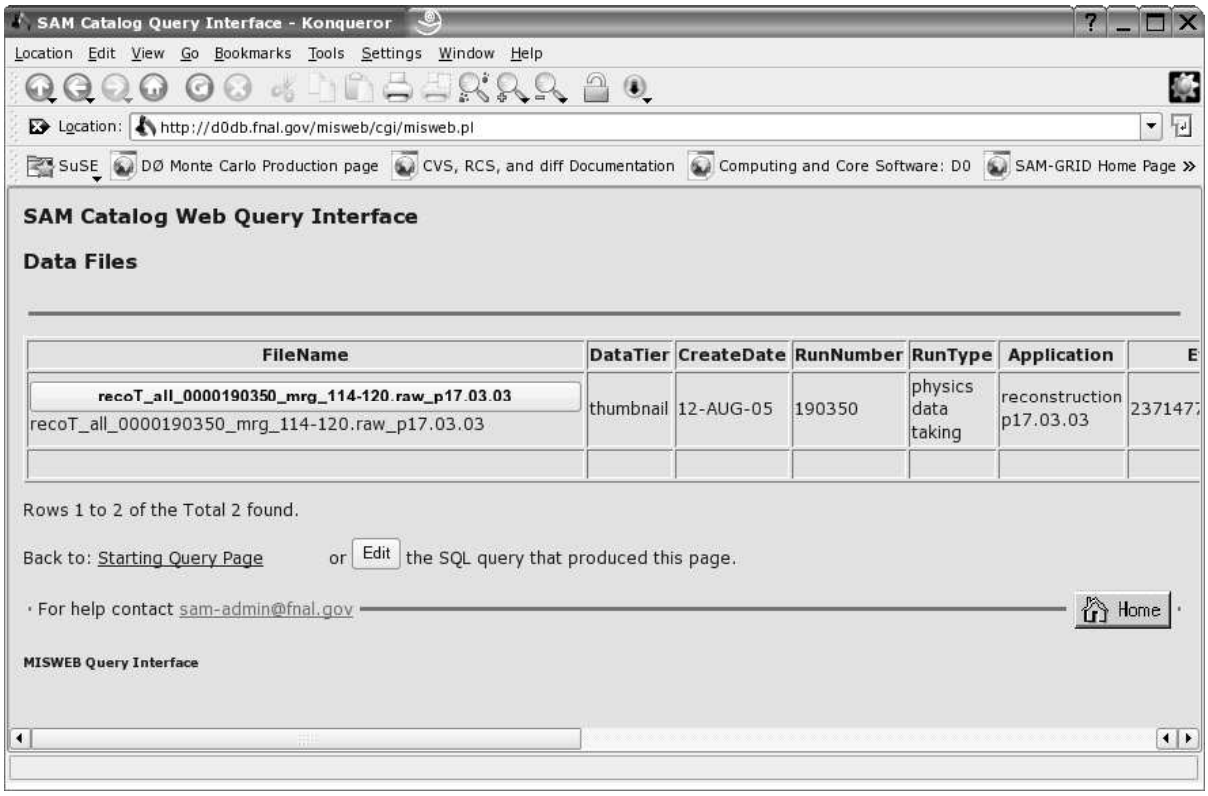

Figure 5.8. SAM Web query of an output file.

The figure 5.8 gives the details of an output file created by the batch job. This is using SAM web query interface. All the output files shown in figures 5.6 and 5.7 are linked to this interface. Using this interface, all the details of an output file can be obtained from SAM. These include creation date, application and number of events.

5.3.1 Result of Monitoring Enhancements

SAM-Grid monitoring system is now able to handle numerous jobs submitted by users. These jobs run on execution sites which are part of SAM-Grid. Because of the enhancements in monitoring, users of SAM-Grid are able to easily track the progress of these jobs. They are also able to easily monitor the batch system jobs created for each Grid job. Using new monitoring features, it is now possible to distinguish between

batch system jobs which produced output files and remaining batch jobs. It is also possible to find out whether output files have been stored in Mass Storage System [5] and find out details of output files. Monitoring system support is now available for running applications like DZero reconstruction. Right now, users of SAM-Grid are making use of the monitoring features extensively.

CHAPTER 6

Integration of MyProxy with SAM-Grid

MyProxy [25] is a credential management system. Integration of MyProxy with SAM-Grid is one of the performance related enhancements to the SAM-Grid infrastructure. SAM-Grid system performance improves because job failures due to proxy expiration are avoided. At this point, there is no statistics available about the percentage of job failures due to proxy expiration. Still, users of SAM-Grid have identified proxy expiration as a reason for some of the job failures. This motivates the integration of MyProxy with SAM-Grid. In the first section, an overview of Grid security is given. Next section discusses security features in SAM-Grid. After this, details about the integration of MyProxy with SAM-Grid are covered. Status of the integration is given in the last section.

6.1 Overview of Grid security

Grid computing has emerged as a common approach to constructing dynamic, inter-domain, distributed computing and data collaborations. The X.509 Public Key Infrastructure is the basis for Grid security. The open source Globus Toolkit [8] middleware has been developed to support Grid environments, and is used in Grid deployments worldwide. The Grid Security Infrastructure (GSI) is the portion of the Globus Toolkit that provides the fundamental security services needed to support Grids. The primary motivations behind GSI are:

1) The need for secure communication (authenticated and perhaps confidential) between elements of a computational Grid.

2) The need to support security across organizational boundaries, thus prohibiting a centrally-managed security system.

3) The need to support "single sign-on" for users of the Grid, including delegation of credentials for computations that involve multiple resources and/or sites.

GSI uses public key cryptography (also known as asymmetric cryptography) as the basis for its functionality. Public key cryptography, unlike other cryptographic systems, relies not on a single key (a password or a secret "code"), but on two keys. These keys are numbers that are mathematically related in such a way that if either key is used to encrypt a message, the other key must be used to decrypt it. By making one of the keys available publicly (a public key) and keeping the other key private (a private key), a person can prove that he or she holds the private key simply by encrypting a message. If the message can be decrypted using the public key, the person must have used the private key to encrypt the message.

A central concept in GSI authentication is the certificate. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service. GSI certificates are encoded in the X.509 certificate format, a standard data format for certificates established by the Internet Engineering Task Force (IETF).

A GSI certificate includes four primary pieces of information:

1) A subject name, which identifies the person or object that the certificate represents.

2) The public key belonging to the subject.

3) The identity of a Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject.

4) The digital signature of the named CA.

A third party (a CA) is used to certify the link between the public key and the subject in the certificate. In order to trust the certificate and its contents, the CA's certificate must be trusted. The link between the CA and its certificate is established via some non-cryptographic means, so that the system is trustworthy.

6.1.1 Proxy

GSI provides delegation capability: an extension of the standard SSL protocol which reduces the number of times the user must enter his passphrase. If a Grid computation requires that several Grid resources be used (each requiring mutual authentication), or if there is a need to have agents (local or remote) requesting services on behalf of a user, the need to re-enter the user's passphrase can be avoided by creating a proxy.

A proxy consists of a new certificate and a private key. The key pair that is used for the proxy, i.e. the public key embedded in the certificate and the private key, may either be regenerated for each proxy or obtained by other means. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA. The certificate also includes a time notation after which the proxy should no longer be accepted by others. Proxies have limited lifetimes.

The proxy's private key must be kept secure, but because the proxy isn't valid for very long, it doesn't have to be kept quite as secure as the owner's private key. It is thus possible to store the proxy's private key in a local storage system without being encrypted, as long as the permissions on the file prevent anyone else from looking at them easily. Once a proxy is created and stored, the user can use the proxy certificate and private key for mutual authentication without entering a password.

When proxies are used, the mutual authentication process differs slightly. The remote party receives not only the proxy's certificate (signed by the owner), but also the

owner's certificate. During mutual authentication, the owner's public key (obtained from his certificate) is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA to the proxy through the owner.

6.1.2 Security in SAM-Grid

The SAM-Grid integrates several standard Grid components, such as the Globus Toolkit and Condor-G: the SAM-Grid therefore inherently uses X509 certificates as the primary way for authentication. SAM-Grid distinguishes between certificates issued to people and to services. GSI allows mutual client-server authentication and authorization. For authentication, certificates are checked against the list of trusted CA where the client/server run. For authorization, the certificate subject is checked against an identity list or map of certificate subject to local UID (grid-mapfile). Clients do not use identity lists i.e. there is no client side authorization policy on the interaction with an authenticated server. Identity lists, used on the server side for authorization, are maintained centrally and can be pulled periodically at the required sites [17].

The client software is used to submit the users job to a submission site, monitor its status, modify its description or cancel it. The client expects the submission site to allocate and maintain the amount of disk space necessary to hold the compressed input sandbox of the job and the delegated users proxy. It also expects the submission site to act on the users behalf when submitting the job to the execution site and when retrieving its status, stdout and stderr. The client expects the submission site to protect the input sandbox against alteration and the delegated user proxy against disclosure. The client contacts the submission site to delegate the job submission. The client needs to trust the CA that signed the certificate of this service.

The submission site is responsible for accepting a job from a client, for keeping a queue of the jobs and for reliably submitting each of them to the execution site selected by the resource selector. The job submission fails if the execution site not trusted by the submission site. The submission site expects the execution site to locally queue the job, execute it, and report its status and output/error streams. It expects the execution site to guard the job and the output/error streams from alteration and the delegated user proxy from disclosure. The submission site needs to trust the CA that signed the service certificate of the resource selector, the execution site gateway (globus gatekeeper) and the CA that signed the user certificate used by the client. It also needs to maintain an authorization list for such users, since they are the primary beneficiaries of its services/resources. The submission site daemon is called condor_schedd.

The execution site is responsible for accepting jobs from the submission site; for advertising itself to the resource selector and for transferring the input files required by the jobs. The servers running at the execution site do the following:

1) Globus Gatekeeper: It is the server that receives the requests for scheduling a job by a submission site on behalf of the user. It needs to trust the CA that signed the users certificate and it needs to keep an authorization list (grid-mapfile) of the users authorized to run at the local resource. The server runs as root and its identity can be the host certificate.

2) gridftpd: We run the Gridftpd daemon to enable external access to the files cached by the local SAM station. The daemon runs under a service certificate, whose subject contains the machine node name. Each gridftpd has access to the list of subjects that are part of the SAM Grid. This mechanism protects against unauthorized use of the local disk space.

3) jim_advertise: It is the service that advertises resources to the Resource Selector. The resource description does not need a high level of protection, since the resource

selector uses it only to recommend an execution site. Administrators can choose to run jim_adverise under a dedicated service identity or the same certificate used by gridftpd.

Resource Selector is responsible to match jobs with resources. It maintains a list of submission and execution sites authorized to register with it, to avoid information flooding from unauthorized services. Resource Selector runs under a service certificate.

6.2 MyProxy - Credential management system

MyProxy is an online credential management system. It is used to store the proxy information at a centralized location. MyProxy server can be installed in a machine which is accessible to all the client programs running on other machines. While executing jobs, we can check the proxy life time and renew their proxy whenever they are about to expire.

6.2.1 MyProxy Server

MyProxy server has to be running on a machine which is highly secured. All the client proxies are stored by the server in a secure location. We should choose a well-protected host to run the server; a host that is secured to the level of a Kerberos KDC, that has limited user access, runs limited services, and is well monitored and maintained in terms of security patches. MyProxy server requires a secure filesystem on which to store credentials. By default, it tries to use /var/myproxy and if that fails, it uses $GLOBUS_LOCATION/var/myproxy [25].

For a typical myproxy-server installation, the host on which the myproxy-server is running must have a host certificate installed. In this case, the myproxy-server will run as root so it can access the host certificate and key. The default configuration does not enable any myproxy-server features to provide the greatest security until we have configured the server. To enable all myproxy-server features, we should provide the details

about accepted_credentials, authorized_retrievers, default_retrievers,authorized_renewers and default_renewers in myproxy-server.config file.

For running the server, we have to make sure that Globus environment is setup in the shell. It accepts connections on TCP port 7512, forking off a separate child to handle each incoming connection. It logs information via the syslog service under the daemon facility. MyProxy Server can be run as a system service. Alternatively, to run the myproxy server out of inetd or xinetd, we need to reactivate the inetd (or xinetd).

6.2.2 MyProxy Repository

Rather than storing the Grid credentials on each machine we use to access the Grid, we can store them in a MyProxy repository and retrieve a proxy credential from the MyProxy repository when needed. To store a credential in the MyProxy repository, myproxy-init command is run on a computer where the users Grid credentials are located. By default, myproxy-init will use credentials in $HOME/.globus/usercert.pem and $HOME/.globus/userkey.pem. To upload credentials from a different location to the Myproxy server, the X509_USER_KEY and X509_USER_CERT environment variables could be set.

MyProxy supports credential renewal, so that long-running tasks don't fail because of an expired credential. An authorized Grid service can renew credentials on the user's behalf, or the user can renew credentials manually as needed. MyProxy server must be configured to allow credential renewal in the authorized_renewers and default_renewers policies in the myproxy-server.config file. To store a renewable credential in the MyProxy repository, we can run the myproxy-init command with the -R or -A option on a computer where Grid credentials are located. For example:

myproxy-init -R 'condorg/samgrid.fnal.gov' -k renewable

This example authorizes the Condor-G service on samgrid.fnal.gov to renew credentials with the -R option, and uses the -k option to specify a name for the credential to distinguish this renewable credential from other credentials we may have in the repository. To renew credentials, we can run the myproxy-get-delegation command with the -a option specifying the filename of the credential we want to renew. For example:

myproxy-get-delegation -a /tmp/x509up_UID -k renewable

If the renewable credential was stored with the myproxy-init -R option, the renewer must have a valid credential matching the -R policy to successfully renew a credential. If, instead, the credential was stored with myproxy-init -A, no additional credential is required.

We can use the globusrun command to update the credentials of submitted Globus GRAM jobs (eg. globusrun -refresh-proxy job-ID). Condor-G version 6.7 also supports renewing credentials via MyProxy.

6.3 Integration of MyProxy with SAM-Grid

In this section, details about the integration of MyProxy with SAM-Grid are given. By using MyProxy, we will be able to avoid the issue of proxy expiring in the middle of the job execution. When the jobs are submitted, there will be a proxy that is associated with them. This proxy has a limited life time. There are two places where the proxy will have to be renewed. One is the submission site and the other is the execution site. Services running in client, submission and execution sites of the SAM-Grid, need changes to use MyProxy.

In figure 6.1, MyProxy service is intergrated with SAM-Grid services. Proxy is stored in the MyProxy repository by the user interface which handles job submission. The jobs are submitted to Condor-G, which queues them before submitting to gate keeper. Condor-G renews the proxy if it expires at this stage. Once the job is submitted

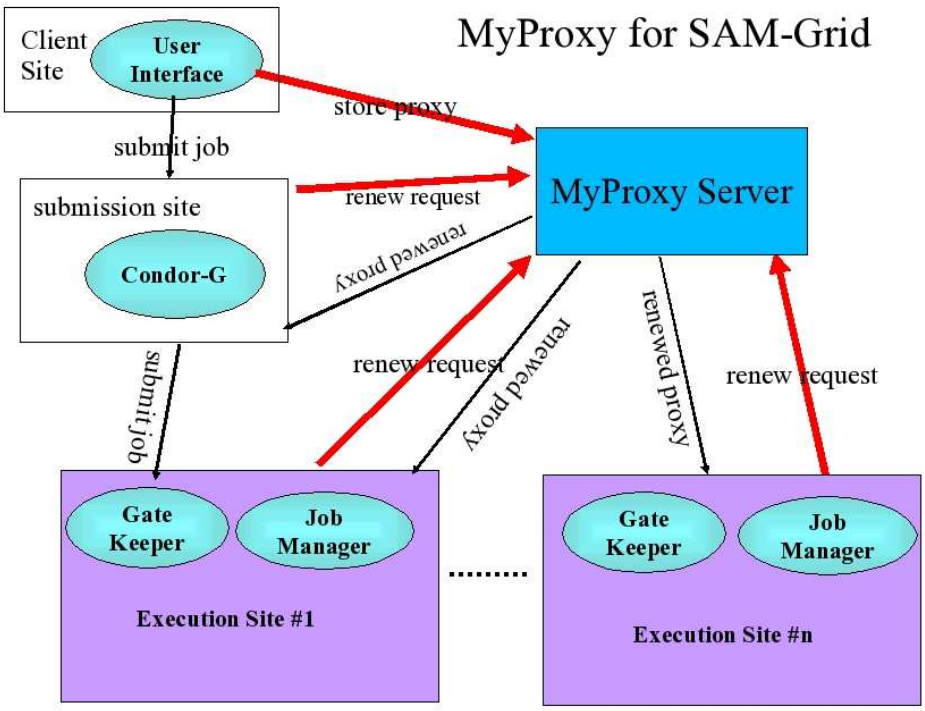

Figure 6.1. MyProxy with SAM-Grid.

to the gatekeeper, the SAM-Grid jobmanager is instantiated. It renews the proxy if it expires before job completion.

6.3.1 Client site

SAM-Grid uses jim_client as the user interface for job submission. While submitting the job, the RSL should have extra parameters related to MyProxy. For example, the RSL could be like this.

MyProxyHost=samgrid.fnal:7512

MyProxyServerDN=/O=DoeGrid/....

MyProxyCredentialName=job1

This requires changes to submission scripts in jim_client. To make it more secure, we can make the credential job specific. The user can store his credential at the beginning of the job submission command from the client site. Using myproxy-init command, the credential can be stored in MyProxy server. It can be deleted from the server once the job completes.

6.3.2 Submission site

Condor-G version 6.7 supports MyProxy. When a job is submitted, the Grid-jobmanager running in the submission site will look at the parameters corresponding to MyProxy. The parameters include MyProxy server and proxy details. When the proxy is about to expire, grid-jobmanager will contact the server and renew it. Schedd would n't need any changes. VDT 1.3 includes Condor/Condor-G v6.7 and MyProxy.

6.3.3 Execution site

In the execution site, job managers can renew the proxy when it is about to expire. This is possible by invoking get_delegation command and connecting to the MyProxy Server. The server name and other parameters should be available in the environment. We can use the SAM-Grid job manager to replace the user proxy when it is about to expire. This will be placed in the sandbox so that the batch jobs will get it, when they transfer the sandbox.

Once the jobs start running in the worker nodes, there is only one way to deal with proxy renewal. Each job will have to renew it by pulling it from the sandbox area in headnode or from the MyProxyServer directly. When the job is waiting in the queue, the proxy might expire. It would be a good idea to make sure that many jobs are not

submitted to the batch system at the same time. If many jobs are submitted, we have to make sure that the proxy of the jobs waiting in the queue is long enough.

Proxy might expire, if the batch job is forced to wait for a long time during the file transfer. In this case, each batch job can renew the proxy by connecting to the MyProxy Server. Transferring renewed proxy from the head-node is another solution, if job-manager will renew the proxy and put it in the head-node.

A job can check the proxy when it is about to use it in the Grid. In sandbox manager script, we are exporting the X509_USR_PROXY. This is used in all the Gridftp transfers later on. Before doing a gridftp, we can execute a script which will check the remaining time for the proxy. If it is not sufficient for the predicted run time of the job, it can be renewed using the myproxy-get-delegation command. This command will use the current proxy, while authenticating to the MyProxy server.

6.4 Status of MyProxy Integration

At this time, SAM-Grid team is evaluating the integration of SAM-Grid with VOMS(Virtual Organization Membership Service). MyProxy integration was tried only in a test environment and would require integration with the production environment for the SAM-Grid. Statistics about the job failures in SAM-Grid due to expired credentials, are to be gathered. The future work might include a tighter integration of VOMS and MyProxy with SAM-Grid. This depends on the requirements of the experiments utilizing the SAM-Grid infrastructure.

CHAPTER 7

Conclusions

SAM-Grid is being used as the Grid computing infrastructure for many of the HEP experiments. This thesis is part of an effort to enhance the capabilities of the SAM-grid infrastructure. SAM-Grid has been scaled to handle large number of jobs submitted to various sites around the world. Monitoring system has been enhanced with new features. These new features are used extensively by the users of SAM-Grid. Sun Grid Engine has been integrated with the SAM-Grid and MyProxy integration has been tried on a test-bed. It is also integrated with components for running experiments like DZero data reprocessing.

7.1 Current utilization of SAM-Grid

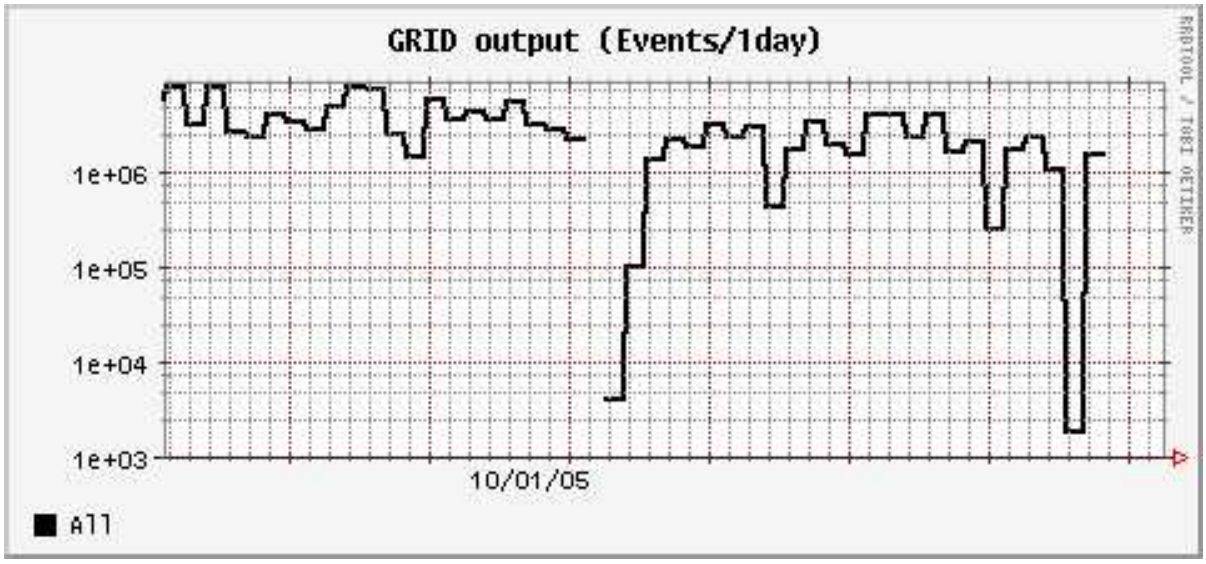

Figure 7.1. Throughput of SAM-Grid.

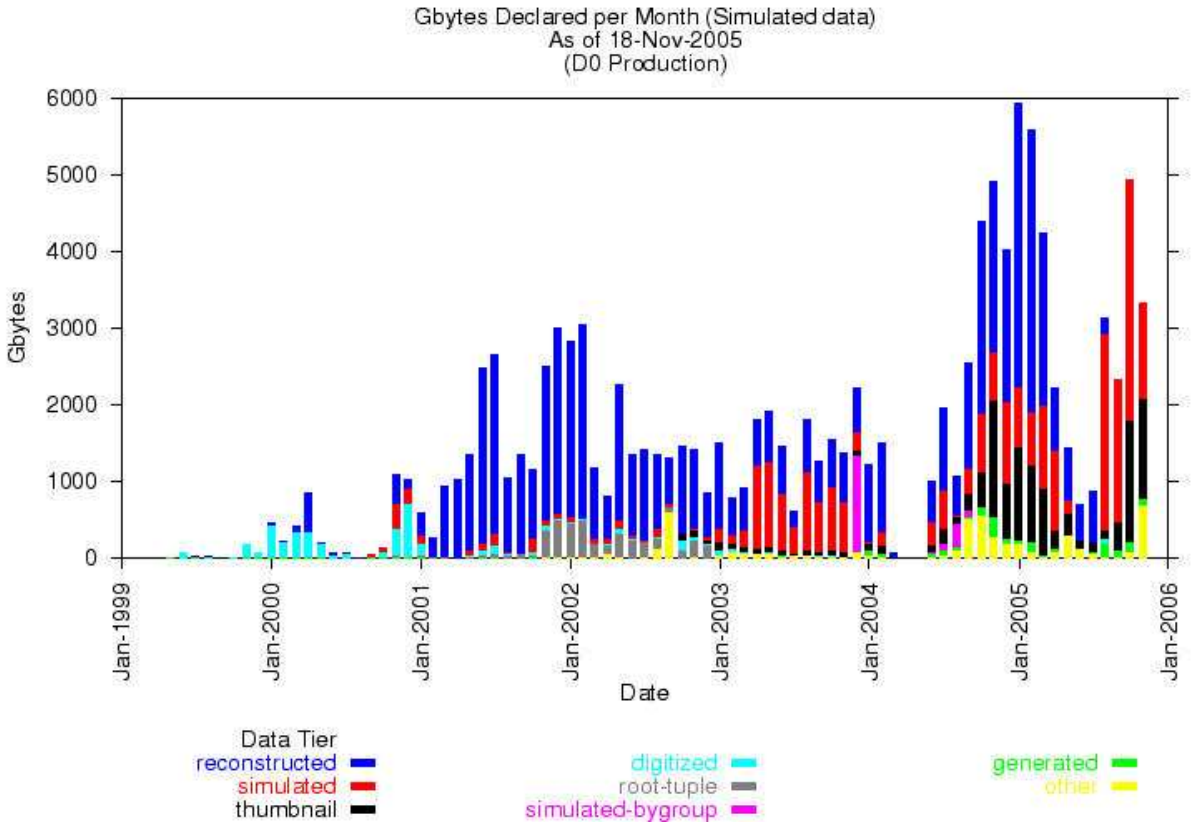

Figure 7.2. Production in SAM-Grid.

Figure 7.1 shows the throughput of SAM-Grid during October 2005. This is the combined output of all applications including DZero reconstruction. There are around 5 million events getting processed every day which corresponds to around 300 Giga Bytes of data.

The SAM-Grid system is being heavily used for running the DZero experiments. Figure 7.2 shows there has been a steady increase in the usage of the system from the year 2001 onwards. Combined total of all simulated data is around 120 Tera Bytes. This includes digitized and reconstructed data. As seen in the figure, from the middle of 2004, there has been much more data produced by the experiments running in SAM-Grid. This

is because SAM-Grid now has more execution sites running jobs and the enhancements done on the infrastructure improved the job and data handling capability of the SAM-Grid. From 2004 onwards, jobs running in SAM-Grid have processed more than 60 Tera Bytes of data.

At this point, Dzero and CDF are the HEP experiments using the SAM-Grid infrastructure. Also, there are new experiments which are being analyzed for using the SAM-grid.

REFERENCES

[1] IBM docs: http://www.ibm.com/developerworks/grid

[2] Foster, I. "What is the Grid?", Grid Today, Vol.1 No. 6, July 22, 2002.

[3] The DZero Experiment: http://www-d0.fnal.gov/

[4] Collider Detector at Fermilab: http://www-cdf.fnal.gov

[5] Fermilab Mass Storage System: http://www-isd.fnal.gov/enstore

[6] SAM-Grid project: http://www-d0.fnal.gov/computing/grid/

[7] LDAP project: http://www.openldap.org/

[8] Globus website: http://www.globus.org

[9] Condor project: http://www.cs.wisc.edu/condor

[10] Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S. "Condor-G: A Computation Management Agent for Multi-Institutional Grids". Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001

[11] CORBA Faqs: http://www.omg.org/gettingstarted/corbafaq.htm

[12] FNORB: http://www.fnorb.org/docs/1.1/Fnorb-Guide/

[13] Baranovski, A., Garzoglio, G., Koutaniemi K., Lueking, L., Patil, S., Pordes, R., Rana, A., Terekhov, I., Veseli, S., Yu, J., Walker, R., White V. "The SAM-GRID project: architecture and plan". Nuclear Instruments and Methods in Physics Research, Section A (Elsevier Science), Proceedings of ACAT'2002.

[14] SAM project: http://www-d0db.fnal.gov/sam

[15] Xindice: http://xml.apache.org/xindice

[16] G. Garzoglio, I. Terekhov, J. Snow, S. Jain, A. Nishandar, "Experience producing simulated events for the DZero experiment on the SAM-Grid", in Proceedings of Computing in High Energy and Nuclear Physics (CHEP04), Interlaken, Switzerland, Sep 2004.

[17] G. Garzoglio, "A globally distributed system for job, data and information handling for high energy physics"; Ph.D. Research Proposal, DePaul University, Chicago; 09/04

[18] Sun Grid Engine: http://www.sun.com/software/gridware/index.xml

[19] S. Jain, "Abstracting the hetereogeneities of computational resources in the SAM-Grid to enable execution of high energy physics applications", Thesis of Master in Computing Science, The University of Texas, Arlington, Dec. 2004

[20] A. Nishandar, "Grid-Fabric Interface For Job Management In Sam-Grid, A Distributed Data Handling And Job Management System For High Energy Physics Experiments", Thesis of Master in Computing Science, The University of Texas, Arlington, Dec. 2004

[21] Sun documents: http://docs.sun.com

[22] Portable Batch System: http://www.openpbs.org/

[23] A.S. Rana, "A globally-distributed grid monitoring system to facilitate HPC at D0/SAM-Grid (Design, development, implementation and deployment of a prototype)", Thesis of Master in Computing Science, The University of Texas, Arlington, Nov. 2002

[24] SAM-Grid monitoring: http://samgrid.fnal.gov:8080/

[25] MyProxy project: http://grid.ncsa.uiuc.edu/myproxy

BIOGRAPHICAL STATEMENT

Bimal Balan received his Bachelor's Degree in Computer Science from REC Calicut (now NIT), India in June 1998. After that he worked in the software industry for 5 years. His experience includes software developer positions in Lucent Technologies and Computer Science department of Indian Institute of Science. Bimal started his graduate studies at the University of Texas at Arlington in the fall of 2003. He received his Master of Science in Computer Science in Dec 2005. Through UTA-FNAL Computer Science MS student exchange program, he worked in Fermi National Accelerator Laboratory in Batavia, IL as a software developer. His research interests include computer architecture, distributed systems and networking.