

GRID-FABRIC INTERFACE FOR JOB MANAGEMENT IN SAM-GRID,
A DISTRIBUTED DATA HANDLING AND JOB MANAGEMENT
SYSTEM FOR HIGH ENERGY PHYSICS EXPERIMENTS

The members of the Committee approve the master's
thesis of Aditya P. Nishandar

Supervising Professor Name
David Levine

Jaehoon Yu

Leonidas Fegaras

Copyright © by Aditya P. Nishandar 2004

All Rights Reserved

GRID-FABRIC INTERFACE FOR JOB MANAGEMENT IN SAM-GRID,
A DISTRIBUTED DATA HANDLING AND JOB MANAGEMENT
SYSTEM FOR HIGH ENERGY PHYSICS EXPERIMENTS

by

ADITYA P. NISHANDAR

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 AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2004

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervising professor David Levine for his constant encouragement and valuable guidance throughout my research. I am also grateful to the SAM-Grid project's technical leader, Igor Terekhov and Gabriele Garzoglio of the Computing Division, Fermilab.

I would also like to thank Jaehoon Yu, Department of Physics, UT Arlington, for his excellent mentorship and suggestions while I was conducting this work. Most importantly, the credit for the achievement of this work goes to the SAM-Grid team, specifically Gabriele Garzoglio, Igor Terekhov and Andrew Baranovski, their numerous original ideas form part of this work. I am grateful to Sankalp Jain and Drew Meyer for their great help during the entire work. Above all, I would like to thank Wyatt Merritt for her highly efficient management of the project.

My heartfelt thanks go to my Family and to my friends. This work could not have been accomplished without their support.

November 17, 2004

ABSTRACT

GRID TO FABRIC INTERFACE FOR JOB MANAGEMENT IN SAM-GRID, A DISTRIBUTED DATA HANDLING AND JOB MANAGEMENT SYSTEM FOR HIGH ENERGY PHYSICS EXPERIMENTS

Publication No. _____

Aditya P. Nishandar, M.S.

The University of Texas at Arlington, 2004

Supervising Professor: David Levine

Modern science and engineering are increasingly done in a collaborative fashion. These collaborations are multi-institutional, multi-disciplinary and geographically distributed environments. A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities. An effective computational Grid assumes the existence of a correctly operating large scale Grid fabric. The Grid fabric is the collection of physical and logical resources such as computing and storage facilities, file systems and high performance networks to which shared access is mediated by Grid protocols. A number of middleware implementations have been proposed and implemented that interface the user applications executing on the Grid infrastructure to the fabric, but majority of them assume an overly simplistic view and fail to adapt to shortcomings in the fabric. This thesis presents the challenges and application imposed complexities of interfacing the

standard Grid middleware to the underlying fabric in SAM-Grid, a software infrastructure that addresses the globally distributed computing needs of the Run II experiments at Fermilab.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iv
ABSTRACT	v
LIST OF ILLUSTRATIONS.....	x
LIST OF TABLES	xi
Chapter	
1. INTRODUCTION.....	1
1.1 Grids and high energy physics (HEP) experiments.....	3
1.2 Need for Grid middleware – Fabric interface	4
1.3 Summary of work and thesis outline	6
2. BACKGROUND.....	8
2.1 A brief introduction to Grid Computing	8
2.2 Evolution of Grid Computing.....	9
2.3 Definitions of Grid Computing	12
2.4 Grids and traditional distributed systems.....	15
2.5 High energy physics and Grid computing.....	18
2.6 Grid Computing for Run-II experiments using SAM-Grid.....	19
3. A BRIEF REVIEW OF THE SAM-GRID ARCHITECHTURE.....	21
3.1 Introduction to SAM-Grid.....	21
3.2 Job Management.....	22

3.3 Monitoring and Information Management.....	25
3.4 Data Handling System	30
4. PROBLEM STATEMENT.....	34
5. GRID TO FABRIC INTERFACE IN SAM-GRID.....	35
5.1 Job Managers.....	39
5.1.1 GRAM Job Managers.....	39
5.1.1.1 Limitations with GRAM Job managers.....	41
5.1.2 SAM-Grid Job Managers.....	45
5.1.2.1 Flow of control for SAM-Grid Job managers.....	46
5.2 Sandboxing Mechanism.....	47
5.2.1 Introduction to sandboxing	47
5.2.2 Need for sandboxing in Grid environments.....	48
5.2.3 Sandboxing in SAM-Grid.....	51
5.3 Uniform interface to batch systems - Batch Adapters	54
5.3.1 Need for Batch Adapters	54
5.3.2 Batch Adapter – Jobmanager interaction.....	55
5.4 Batch System Idealizers	57
5.4.1 Need for Batch System Idealizers	57
5.4.1.1 Scratch management on worker nodes.....	57
5.4.1.2 Limit on local job names	58
5.4.1.3 Job lookup failures	58
5.4.1.4 Black hole effect	59

5.4.2 SAM-Grid Batch System Idealizers	60
6. EXPERIMENTAL RESULTS	61
7. CONCLUSIONS.....	68
7.1 Conclusions	68
7.2 Future Research	70
REFERENCES	71
BIOGRAPHICAL INFORMATION	78

LIST OF ILLUSTRATIONS

Figure	Page
2.1 Moore's law	10
2.2 Major milestones in networking and computing technologies	12
2.3 A layered architecture for computational Grids and related technologies	17
3.1 SAM-Grid architecture.....	22
3.2 SAM-Grid: Service perspective.....	23
3.3 Job control flow in SAM-Grid.....	24
3.4 Static <i><attribute, value></i> pair for an execution site	29
3.5 Dynamic attributes of an execution site	29
3.6 Software layers for data handling and data management.....	33
4.1 Gray area between core Grid middleware and fabric.....	34
5.1 Globus job management on the gateway node	40
5.2 SAM-Grid job management	46
5.3 Batch Adapter – Sam-Grid job manager interaction.....	56
6.1 Efficiency of Monte Carlo production using SAM-Grid	61
6.2 Monte Carlo production using SAM-Grid	62
6.3 Reduction in efficiency and throughput due to “Black Holes”	64
6.4 Number of physics events produced per execution site from March	66
6.5 Number of Global (Grid) jobs created in a month	67

LIST OF TABLES

Table	Page
2.1 Comparison of conventional distributed environments and Grids	16
2.2 Comparison of HEP applications on SAM-Grid	20
2.3 Data generated and consumed by HEP applications on SAM-Grid	20
5.1 SAM-Grid sandboxing python API's	53

CHAPTER I

INTRODUCTION

The history of mankind shows that the way we live our lives has been largely driven by great innovations. Humans have always found novel ways of meeting requirements. Whether the advent of the wheel or the personal computer, the way these revolutionizing tools have been used has a profound impact on the human society. The thirst for knowledge has led us to a better understanding of the environment we live in. Understanding the fundamental nature of matter is the main goal of the high energy physics community. Particle physicists try to understand the nature of matter that surrounds us at the smallest scales possible. By studying the particles, physicists learn about the elementary building blocks and fundamental forces that determine the nature of matter and the ultimate structure and evolution of the universe. The answers to some of the most profound problems have been discovered by conducting high energy physics experiments. These experiments generate huge amount of data in which the answer to many fundamental questions might lie. To analyze this unprecedented amount of data, huge amount of computational and storage resources are required. Over the past decade, the field of *Grid Computing* has enabled distributed computing infrastructures called Grids capable of satisfying the unprecedented requirements of scientific and engineering collaborations by providing coordinated, shared access to geographically distributed resources. The heterogeneity of the resources, coupled with the lack of

administrative control of the user over the resources make enabling Grids challenging. These resources constitute the *Grid Fabric* [2]. “Fabric” refers to a “layer” of Grid components (underneath applications, tools, and middleware). The Fabric encompasses local resource managers, operating systems, queuing systems, device drivers, libraries and networked resources such as compute and storage resources, data sources. Grid applications use Grid tools and middleware components to interact with the fabric. Middleware is a term used to describe a logical layer of software that facilitates distributed applications to access and utilize the resources in a way that is transparent to the user and the application. Middleware provides consistency, security and privacy. The work presented in this thesis addresses the challenges and application imposed complexities of interfacing the fabric to the Grid middleware in context of SAM-Grid [3], a software infrastructure that addresses the globally distributed computing needs of the Run II high energy physics experiments at Fermi National Accelerator Laboratory [78].

It should be noted that the solutions proposed in this thesis have resulted due to the hard work and efforts of the SAM-Grid team. The author *does not* claim individual credit for the proposed solutions and believes that the SAM-Grid team at Fermilab along with the collaborators from various participating institutions should be credited for the success of the SAM-Grid infrastructure. In particular Igor Terekhov, Gabriele Gargozlio and Andrew Baranowski along with the past students from the Computer Science and Engineering department at The University of Texas at Arlington stationed at Fermilab have developed the initial prototypes for the Grid-Fabric interface for job

management. Drew Meyer from The University of Texas at Arlington helped with the initial implementation of *merging* jobs. The author was involved in design and implementation of the Grid-Fabric interface in a production environment. This involved among many other things, the implementation of SAM-Grid job managers having application specific plugins, the sandboxing mechanism (described in Chapter 5) and the deployment of the Grid-Fabric interface at different execution sites around the world. The major content of this thesis is derived from publications by the SAM-Grid team at numerous international conferences [6] [5] [3]. The reader is highly encouraged to visit the reference section for a comprehensive list of publications.

1.1 Grids and high energy physics (HEP) experiments

The Grid has been described as a distributed computing infrastructure for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [2]. A virtual organization (VO) is a set of individuals and/or institutions with some common purpose or interest and that need to share their resources to further their objectives [2]. High energy physics (HEP) experiment collaborations have been large and international for many years. The World Wide Web [9], initially developed at CERN [52] has provided an efficient means for geographically separated collaborators (as well as the rest of the world) to communicate among each other for over a decade. These collaborations are typically geographically distributed, dynamic in nature and multi-institutional involving large number of collaborators (500 physicists for DØ [46] and 1000 each for CMS [48] and ATLAS [49]). Physicists are typically part of multiple collaborations and hence members of multiple virtual organizations. The data collected

from experiments are becoming orders of magnitude more voluminous. Petabytes are expected [50] per year. Institutions collaborating on these experiments typically want the computing resources that they buy to remain under their administrative control. Participating institutions requiring strict (Kerberos [53] at Fermilab) and varied authentication methods for accessing their computing resources. These characteristics make computational Grids [2] an ideal infrastructure for satisfying the computing requirements of the HEP community. In the rest of the thesis the abbreviation HEP refers to High Energy and Nuclear Physics.

1.2 Need for Grid middleware – Fabric interface

Middleware is the software that abstracts the distributed resources for the applications and users. More specifically, the term middleware refers to an evolving layer of services that resides between the network and more traditional applications for managing security, access and information exchange to

- 1) Let scientists, engineers, and educators transparently use and share distributed resources, such as computers, data, networks, and instruments,
- 2) Develop effective collaboration and communications tools such as Grid technologies, desktop video, and other advanced services to expedite research and education, and
- 3) Develop a working architecture and approach that can be extended to the larger set of Internet and network users [54].

The Grid Fabric layer provides the resources to which shared access is mediated by Grid protocols [2]. A resource need not be a physical entity like a compute node. It can

be a logical entity like a distributed file system, or a Condor [7] pool. Most of the institutions of the HEP collaborations have local resources at their disposal. Traditionally these resources have been used by the HEP community in an isolated manner and under local administrative policies. The existing fabric is configured to satisfy the needs of local users and safeguard the resources by having a myriad of security mechanisms to control access to the local resources. Interfacing existing fabric to the Grid middleware is challenging, especially when the fabric is deficient in providing services that are required for the efficient and complete operation of the Grid. Middleware implementations like the Globus¹ Toolkit [55] try to compensate for the deficiencies in the fabric services by incorporating these services in the middleware. A *service* is an entity that provides some capability to its clients by exchanging messages. A service is defined by identifying some sequence of messages that cause a service to perform some operation. By encapsulating operations in terms of message exchanges, service orientation isolates the users from details of location of the service and how the service is implemented. The Grid Resource Allocation manager (GRAM) [10] is the component in the Globus[®] Toolkit responsible for managing Grid resources. GRAM is a basic library service that provides capabilities to do remote-submission job start up. The user application utilizes the Grid fabric via the GRAM interface. The current GRAM implementation has very limited functionality, making it difficult to deploy complex data and compute intensive applications. With respect to Grid computing in general, "application" refers to a "layer" of Grid components (above infrastructure and

¹ [®] Globus Toolkit is a registered trademark held by the University of Chicago.

resources). An application is a name used to identify a set of software that will execute computational jobs, manage data (access, store, read...) and has many attributes. Any application when invoked (executed) includes information that allows tracing back to the individual who is responsible for the execution.

1.3 Summary of work and thesis outline

The work presented in this thesis highlights the need for a Grid-Fabric interface and discusses an implementation in the context of SAM-Grid, an infrastructure currently deployed for satisfying the computational needs of the Run-II (DØ and CDF [47]) experiments at Fermilab. The main focus of this thesis is on the job management aspect of the interface Grid-Fabric interface. In the rest of the thesis, the term SAM-Grids refers to the SAM-Grid infrastructure as described in Chapter 3. The Grid-Fabric interface refers to the layer of software (as described in Chapter 2) between the core Grid middleware and the fabric layer. The rest of the thesis is organized as follows.

Chapter 2 provides a brief background on Grid computing. It briefly describes the evolution of Grid computing and various interpretations of the Grid concept. Differences between conventional distributed computing environments and the Grid computing environment have been highlighted and an effort has been made to distinguish between various types of Grids, difference between computational Grids and peer to peer Grids. It concludes with a brief description of the DØ and CDF [47] experiments at Fermilab and the role of SAM-Grid in satisfying the computing and storage needs of these two HEP experiments.

Chapter 3 gives the reader a brief overview of the SAM-Grid architecture. The three main components (job handling, data handling and information and management) are described. Chapter 4 states the problem and the goal of this thesis. Chapter 5 proposes a solution within the SAM-Grid framework. Chapter 6 discusses some results, particularly in the increased efficiency of SAM-Grid due to the Grid-Fabric interface and the occurrence of the “Black Hole Effect”. This thesis concludes with Chapter 7 highlighting the contributions of this thesis.

This work was partially supported by the Department of Computer Science and Engineering, Department of Physics at the University of Texas Arlington, TX, USA; and Fermi National Accelerator Laboratory, Batavia, IL, USA, FNAL-PO-546763 and conducted with the United States Department of Energy SciDAC program, the Particle Physics Data Grid (PPDG), and the Grid for UK Particle Physics (GridPP).

CHAPTER II

BACKGROUND

2.1 A brief introduction to Grid computing

In today's daily life, electric power has pervaded almost every aspect of modern human life. It is so ubiquitous that we hardly notice it or ponder about its source and method of distribution. In fact its presence is felt only by its absence. When designing and developing a new electric device, the device just has to adhere to the required voltage and current specifications, and not be concerned about the origin of electric power. The device essentially can use a defined interface (electric socket) and tap into the electric Grid.

With the increase in network speeds and proliferation of the personal computer, computer scientists started exploring the idea of a computational Grid. The arrival of the personal computers in 1980s, sophistication and advances in chip manufacturing technologies, increase in high speed networks, emergence of new paradigms for distributed computing and the ever increasing need to communicate and share information made designing computational Grids the next logical step in the evolution of distributed computing. Emerging computational Grids have enabled geographically distributed collaboration of scientists to collaborate on a previously unimagined scale. Harnessing idle compute cycle when people in different time zones are not using them, peer to peer networks for data sharing, distributed resource sharing between different

high energy physics experiments are some of the applications domains for different types of Grids.

In this chapter a brief history of Grid computing is presented. Differences between Grid environments and other related distributed environments are highlighted; different views on what Grids really mean, followed by a brief discussion of Grid computing in the context of the Run-II experiments ($D\bar{O}$ and CDF) at Fermilab is presented. This chapter is not meant to be an exhaustive description of Grid technologies or the history of Grid computing. Rather the attempt is to provide the reader with a brief description of this emerging field. The reader is highly encouraged to visit the references for a more thorough understanding of different concepts. The author also believes that Grid computing is in an evolutionary stage. As most of the technologies in the field of computer science, contemporary Grid technologies may be soon outdated. The emphasis of this chapter is not on specific technologies or tools, but on the underlying concepts.

2.2 Evolution of Grid computing

Grids are large scale shared distributed systems. Distributed computational systems consist of diverse end systems. These end systems typically consist of computing elements, computer systems like CPU and memory, storage elements like disks, NAS, tapes etc, and connected via high speed networks [1]. During the last three and a half decades the number of transistors per chip has increased by six orders of magnitude as shown in figure 2.1. Gordon Moore [58] made a famous observation in 1965 regarding the number of transistors that can be fabricated on a chip.

$$\text{Transistor} = 20 \times 2^{(\text{yr}-1965)/1.5}$$

This popular observation by is termed as Moore’s law.

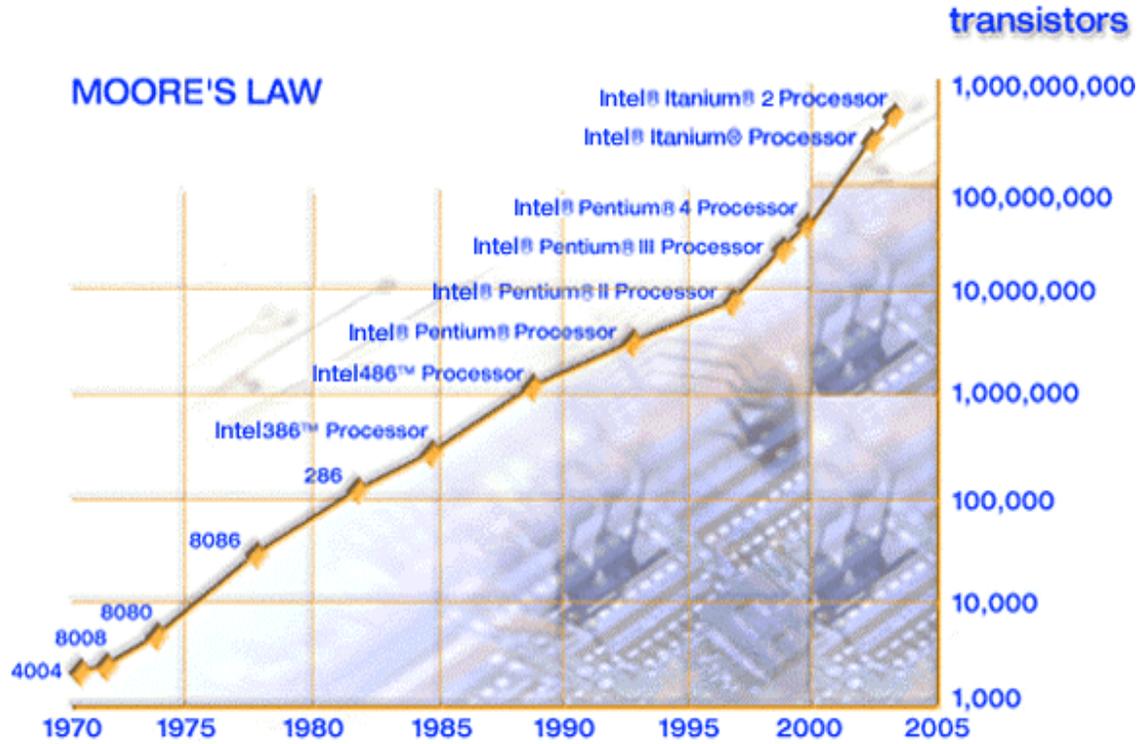


Figure 2.1 Moore’s law (Source [58])

Moore observed an exponential growth in the number of transistors per integrated circuit and predicted that this trend would continue. According to the formula, at present the capacity (number of transistors per chip) of a single chip doubles every 18 months. This is highlighted from the fact that a personal computer’s processing power has gone from 100,000 IPS in 1975 to 1000 MIPS in 2000 [1]. In terms of quantity, approximately 100 million personal computers are produced per year. This has resulted in geometric decrease in system cost performance [1]. Large quantitative change will push computational capability, resulting in large qualitative

change. The storage technologies have shown significant acceleration in terms of capacity. [1] States that between 1970 and 1988 there was a steady increase of 29% in the areal (bits per unit area) density of storage devices, and in recent years the increase is almost 60%. A personal computer in 2001 is as fast as a supercomputer of 1990 [29]. Personal computers now ship with up to 100 gigabytes (GB) of storage, as much as an entire 1990 supercomputer center [29].

Xerox PARC [59] estimates that a \$100 computer in 2022 will have the capability of 500,000 MIPS (million instructions/sec), 1 terabyte of RAM, and 2 terabytes of disk. Probably the most important aspect from distributed point of view is the exponential increase in the communication performance. According to a recent article in ZD-Net [11] [79], Gartner [66] analysts recently released a list of 10 predictions for enterprise businesses. The predictions cover technology, economics, and social boundaries that will morph during the next eight years. One of the predictions is that bandwidth is becoming more cost-effective than computing. Network capacity will increase faster than computing, memory, and storage capacity to produce a significant shift in the relative cost of remote vs. local computing. Cheap and plentiful bandwidth will catalyze a move toward more centralized networks services, using Grid computing models and thin clients. This is based on the fact that optical technologies are growing at a more rapid rate than processors (silicon) or memory. Figure 1.2 below, derived from [1] highlights these trends in networking and computing technologies.

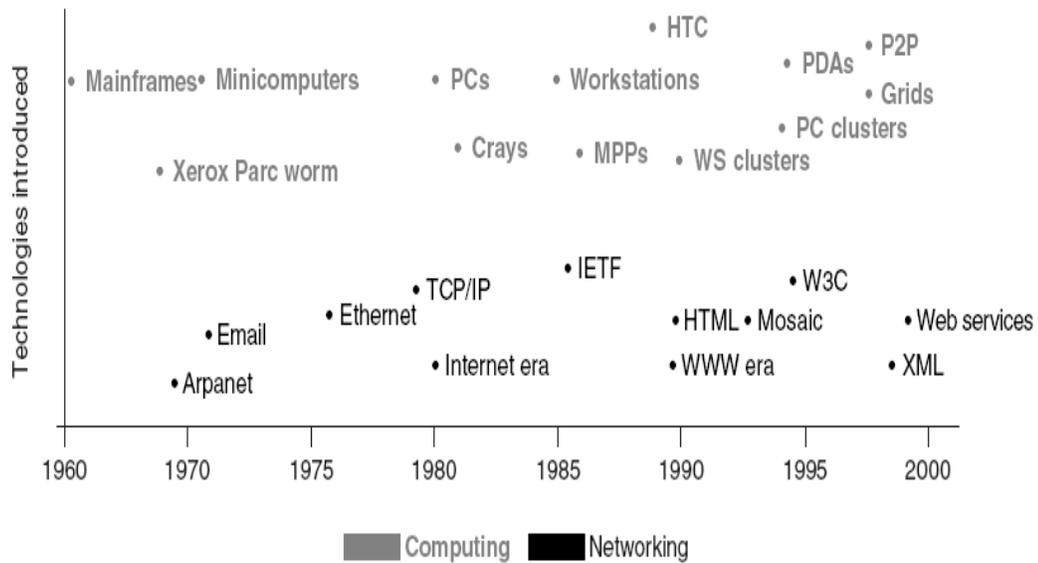


Figure 2.2 Major milestones in networking and computing technologies (Source: [1]).

One important contribution that computational Grids can make is that they can accelerate the deployment of high bandwidth communication infrastructure, because doing so enhances the value of computational elements [1]. Thus the advances in computational, storage and communication technologies coupled with the increasing need of modern scientific, engineering and business applications have given the rise to the concept of computational Grids.

2.3 Definitions of Grid Computing

Grid technologies and architectures have proliferated significantly in the past couple of years. The number of academic Grids has increased approximately six times in past couple of years, and the trend is ongoing. This section describes the essential characteristics of Grids, and highlights the differences between traditional distributed

computing environments and the more recent peer to peer computing. According to Buyya [12], the four main aspects that characterize a Grid are

- 1) *Multiple administrative domains and autonomy.* Grid resources are geographically distributed across multiple administrative domains and owned by different organizations. The autonomy of resource owners needs to be honored along with their local resource management and usage policies.
- 2) *Heterogeneity.* A Grid involves a multiplicity of resources that are heterogeneous in nature and will encompass a vast range of technologies.
- 3) *Scalability.* A Grid might grow from a few integrated resources to millions. This raises the problem of potential performance degradation as the size of Grids increases. Consequently, applications that require a large number of geographically located resources must be designed to be latency and bandwidth tolerant.
- 4) *Dynamicity or adaptability.* In a Grid, resource failure is the rule rather than the exception. In fact, with so many resources in a Grid, the probability of some resource failing is high. Resource managers or applications must tailor their behavior dynamically and use the available resources and services efficiently and effectively.

Resource sharing is the core concept of Grid computing, but there is less agreement on issues such as, unique characteristics of Grids, the functionalities that Grid support, programming model suitable for Grid computing or even abstractions to model and represent Grids [13]. Many definitions of Grids have emerged as different prototypes

were developed and deployed. Grids may be viewed from the utility perspective, where the consumers obtain resources from utility providers or from a resource sharing perspective as mentioned by [2]. Most of the scientific Grids, including SAM-Grid adhere to the resource sharing perspective and utilize the Globus® Toolkit developed at Argonne National Laboratory [57] and ISI [56] as their core middleware. Another significant view regarding the Grid is Grid as a high-performance distributed environment. Some popular definitions for Grids are

- 1) “A flexible, secure, coordinated resource sharing among dynamic collection of individuals, institutions and resources” [2]
- 2) “A single seamless computational environment in which cycles, communication, data are shared and in which the workstation across the continent is no less than the one down the hall” [41]
- 3) “Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements.” [80]
- 4) “A wide area environment that transparently consists of personal computers, workstations, graphic rendering engines, supercomputers and non traditional devices: e.g. TVs, toasters, etc.” [39]
- 5) “Grid computing involves the distribution of computing resources among geographically separated sites (creating a "Grid" of resources), all of which are

configured with specialized software for routing jobs, authenticating users, monitoring resources, and so on [50].

The natural question that arises is “What makes Grids different from traditional distributed environments?” Would sharing the printer on the network, networked file systems and utilizing idle cycles in a network of workstation constitute Grid computing? Next section describes the operational characteristics of a Grid infrastructure and highlights the aspects that make Grids unique from traditional distributed environments. The literature resource for this and the next section is derived from [13].

2.4 Grids and traditional distributed systems

The definitions described in the previous section focus on how a Grid system can be constructed, i.e. the components, layers, protocols and interfaces to be provided by the Grid system are described. Buyya and Chetty [33] describe computational Grids as an extension of the scalable computing concept: Internet-based networks of geographically distributed computing resources that scientist can share, select from, and aggregate to solve large scale problems. The fundamental differences between traditional distributed systems like PVM, MPICH etc. and Grid systems lie in the semantics as shown in Table 1.1. [80] Highlight the differences between clusters and Grids from a resource management perspective. The key distinction between clusters and Grids is mainly in the way resources are managed. In case of clusters, the resource allocation is performed by a centralized resource manager and all nodes cooperatively work together as a single unified resource. In case of Grids, each node (a Grid node can

be thought as an execution site) has its own resource manager and don't aim for providing a single system view. [81] Provide a comparison between Grid computing and peer to peer computing.

Table 2.1 Comparison of conventional distributed environments and Grids
(Source: [13])

Conventional distributed environments	Grids
A virtual pool of computational nodes.	A virtual pool of resources.
A user has access (credential) to all the nodes in the pool.	A user has access to the pool but typically does not have access to individual nodes.
Access to a node means access to all resources on the node.	Access to a resource may be restricted.
The user is aware of the capabilities and features of the nodes.	The user has little or no knowledge about each resource.
Nodes belong to a single trust domain.	Resources span multiple trust domains.

The Grids focus on the user [14]. This is perhaps the most important, and yet the most subtle, difference. Previous systems were developed for and by the resource owner in order to maximize utilization and throughput. In Grid computing, the specific machines that are used to execute an application are chosen from the user's point of

view, maximizing the performance of that application, regardless of the effect on the system as a whole. It is these differences that make the Grid a more usable system than its predecessors. Figure 2.3 illustrates a layered Grid architecture.

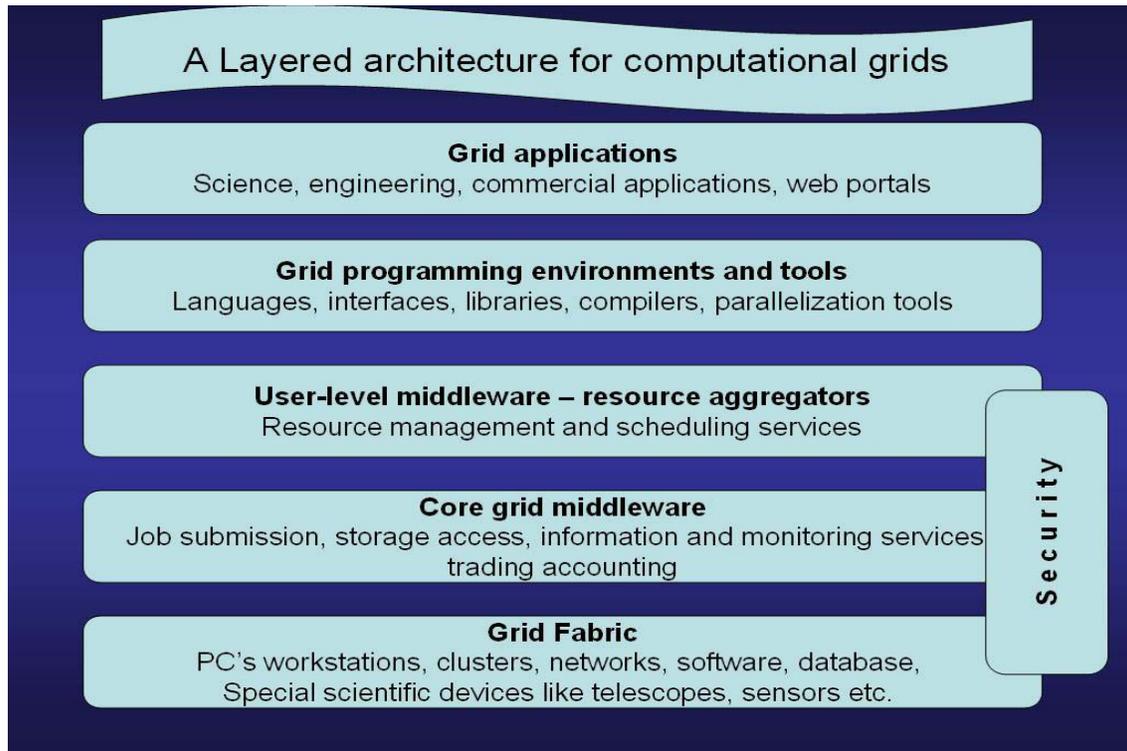


Figure 2.3 A layered architecture for computational Grids and related technology ([34]).

Throughout this thesis when Grids are mentioned, it implies a virtual organization specific distributed infrastructure that distributes compute resources among geographically separated sites, all of which are configured with specialized software for routing jobs, authenticating users and monitoring resources.

2.5 High energy physics and Grid computing

Particle physics laboratories and universities around the world are working to create a global data Grid, a new technology with the potential to put the power of the world's computing capacity at the fingertips of everyone with a computer. Grid technology pools computing power over the Internet, linking and managing global computing resources for solving the fantastically challenging computer problems of particle physics research. It allows physicists to access worldwide distributed computing resources from their desktops as if they were local [73]. Professor Steve Lloyd, Chair of the UK Particle Physics Grid, mentioned that, “Individual scientists using the Grid won't need to know where the data is held or which machines are running their programs. So, whereas a PC on the web provides information or access to services, such as banking or shopping, a PC on the Grid offers its computing power and storage” [43].

The study of the fundamental particles at the highest energies depends on huge devices called particle accelerators. Fermilab's four-mile Tevatron, the world's highest-energy particle accelerator to date, can reach an energy level of 0.980 trillion electron volts (TeV) for each of its particle beams: clockwise-circulating protons and anticlockwise-circulating antiprotons [74]. While the particle accelerators are located in a few selected locations around the world, the collaborations of physicists analyzing the data from them are geographically distributed. High energy physics (HEP) experiments have grown to be more challenging and more expensive over every decade. The challenges have not only grown in terms of investigation of rare particle interactions,

but also in terms of the data generated by the accelerators. The physics phenomena studied are statistical in nature and huge amount of data (petabytes) is needed to derive significant conclusions. The amount of data collected from the detectors is approximately 100 terabytes per year. This coupled with simulated and reconstructed data amounts to 400 terabytes per year.

2.6 Grid computing for Run-II experiments using SAM-Grid

The DØ experiment was proposed for the Fermilab proton-antiproton Tevatron Collider in 1983 and approved in 1984. After 8 years of design, testing, and construction of its hardware and software components, the experiment recorded its first antiproton-proton interaction on May 12, 1992. The data-taking period referred to as "Run 1" lasted through the beginning of 1996. Collisions were studied mainly at an energy of 1800 GeV in the center of mass (the world's highest energy), with a brief run taken at 630 GeV. Currently, the DØ Collaboration consists of more than 500 scientists and engineers from 60 institutions in 15 countries. The Run-II commenced in 2001 with an improved particle accelerator with greater energy and more data [46].

SAM-Grid is a distributed data handling and job management system, designed for experiments with large (petabyte-sized) datasets and widely distributed production and analysis facilities. It mainly caters to the needs of Run-II experiments. SAM-Grid [3, 4, 5] was started at the Fermi National Accelerator Laboratory in January 2002 and it is one of the first Grids deployed for the HEP community. The project is conducted as a collaborative effort between physicists and computer scientists and it is financed by the Particle Physics Data Grid (PPDG) [75], in the US, and GridPP [76], in the UK. The

goal of the SAM-Grid is to enable fully distributed computing for the DØ and CDF experiments, integrating standard Grid tools with in-house software, when the standards do not provide an appropriate solution. The components now in production provide a versatile set of services for data transfer, data storage, and process bookkeeping on distributed systems. The job management components are built around standard Grid middleware from Condor and Globus®. SAM-Grid is used by DØ and CDF, and is being tested for use by MINOS [60].

Table 2.2 Comparison of HEP applications on SAM-Grid
(Source: [83])

Activity	Description	No. of Users	CPU-IO	Time per job
Reconstruction	Data filtering	Small (~5)	CPU + IO	10 Hours
Monte Carlo	Data simulation	Small (~10)	CPU	10 Hours
Analysis	Data mining	Large (~100)	CPU + IO	0.5 – 5 Hours

Table 2.3 Data generated and consumed by HEP applications on SAM-Grid
(Source: [83])

Activity	Data input Per job	Data output Per job	Input per year	Output per year
Reconstruction	1-5 GB	1-5 GB	100 TB	100 TB
Monte Carlo	N/A	10 GB	N/A	1-5 TB
Analysis	100 GB	1-5 GB	varies	varies

CHAPTER III

A BRIEF REVIEW OF THE SAM-GRID ARCHITECTURE

3.1 Introduction to SAM-Grid

A Grid has been described as a distributed computing infrastructure for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [2]. A virtual organization (VO) is a set of individuals and/or institutions with some common goal. Resource sharing takes place according to a set of predefined rules. These resources are typically computers, data, software, expertise, sensors, and instruments. A Grid environment can be considered as a problem-solving environment. A problem solving environment (PSE) has been defined as a computer system that provides all the computational facilities necessary to solve a target class of problems [15]. SAM-Grid provides a problem-solving environment for two of the largest High Energy Physics (HEP) virtual organizations, namely the DØ collaboration and the CDF collaboration.

The SAM-Grid architecture is composed of three major components: the data handling component, the job handling component and the monitoring and information component. This logical division is mostly natural as it closely follows the organization of the standard middleware like Globus® and best capitalizes on the software already developed at Fermilab for the experiments. The most notable of this in-house software is the Sequential Access via Metadata (SAM) [84], the data handling system of the

experiments. Figure 3.1 shows an architectural diagram of SAM-Grid. Figure 3.2 shows the SAM-Grid services.

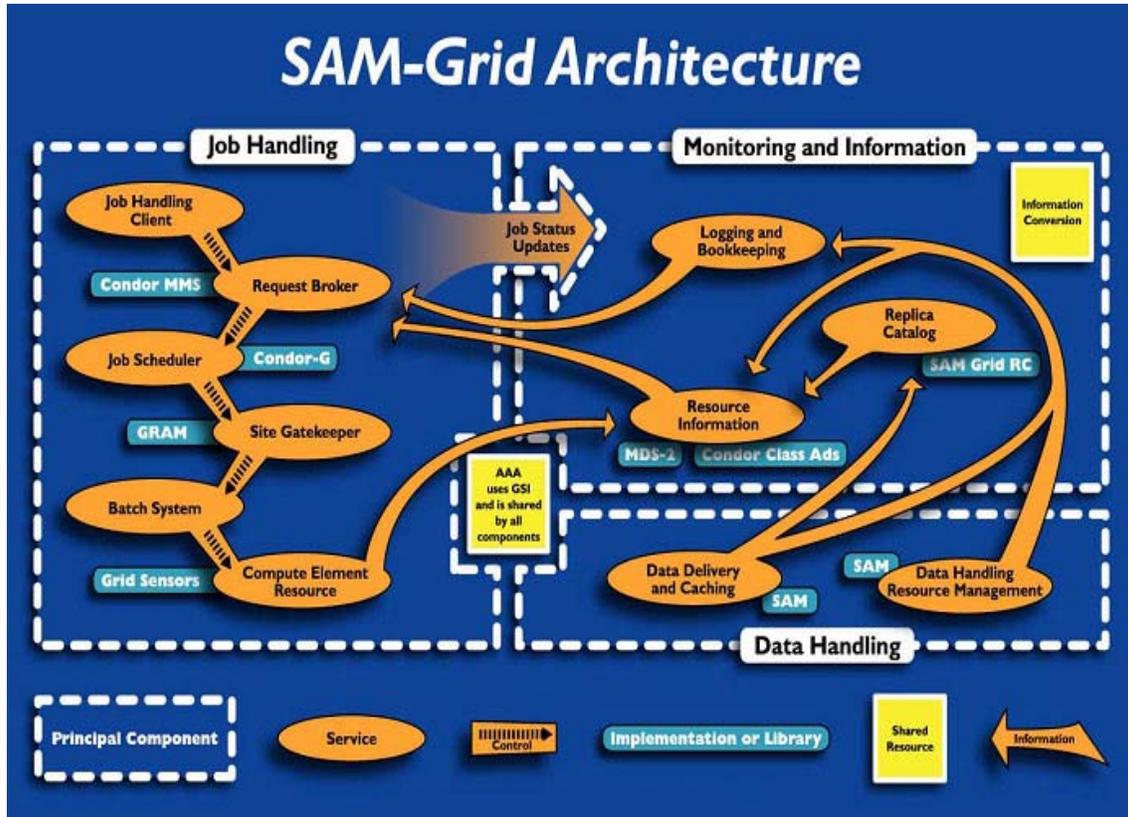


Figure 3.1: The SAM-Grid architecture (Source [4])

3.2 Job Management

A vital component of any computational Grid is its job management system. The Sam-Grid Job management system provides a very lightweight interface (Job Client in Figure 3.1) to the users. This interface is designed to support a multitude of high energy physics applications, like Monte-Carlo simulation, data reconstruction and

data analysis. The interface is extensible and handles application specific job interdependencies.

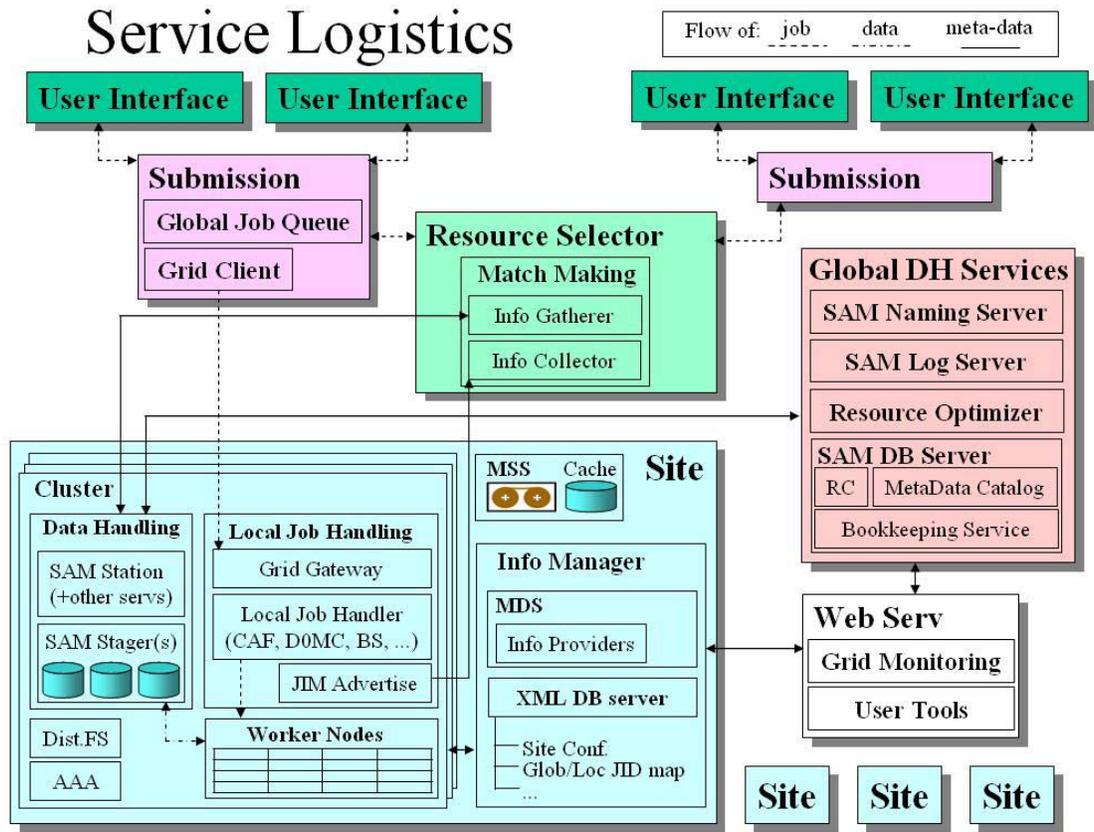


Figure 3.2 SAM-Grid: Service perspective (Source [6])

Scheduling of jobs (Job Scheduler) to resources and negotiating access to them with the help of an automatic resource selection service (Request Broker) is handled by the job management infrastructure as shown in figure 3.3. The Broker analyzes the job requirements and selects the best resource available for the job, according to a dynamically configurable algorithm. The scheduling and the brokering capabilities are provided by the Condor-G [8] middleware. The Condor-G system leverages software

from Globus® and Condor to allow users to harness multi-domain resources as if they all belong to one personal domain.

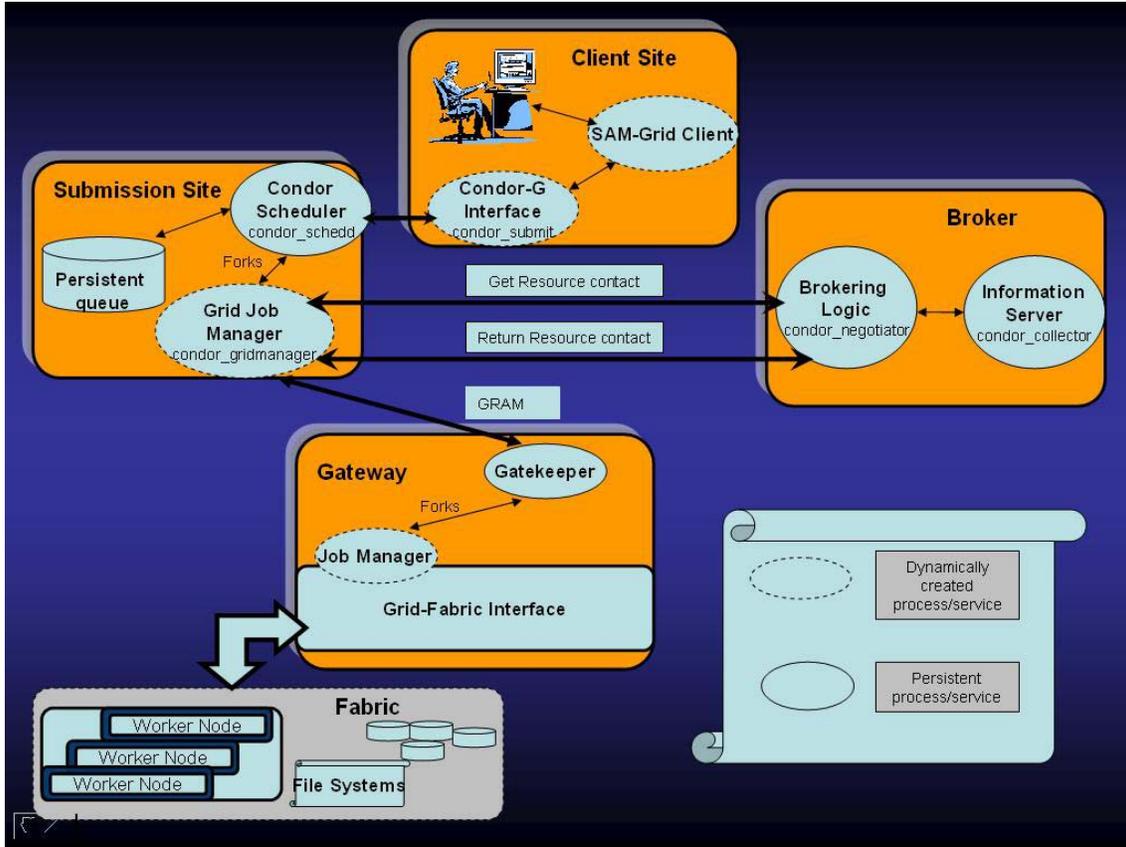


Figure 3.3 Job control flow in SAM-Grid.

Thanks to the efforts of the Particle Physics Data Grid (PPDG) collaboration, the Sam-Grid team along with the Condor team enhanced the Condor-G matchmaking [4] framework, thus enabling to implement high level functionality on top of the Condor-G framework. The Condor-G framework uses Globus® protocols like GRAM, GASS [17] RSL [72] and the security framework GSI [16] to negotiate access and manage jobs submitted to the Grid resources. GSI (Grid Security Infrastructure) is the

Grid middleware component, Public Key- based security component, provided by the Globus® Toolkit. GASS simplifies the porting and running of applications that use file I/O to the Globus® environment. The Globus® Resource Specification Language (RSL) provides a common interchange language to describe resources.

The Grid to Fabric Interface which is the main focus of the work presented in this thesis is the layer between the core Grid middleware and the Grid fabric. The fabric typically consists of a batch queuing systems, computing resources and storage resources. The batch system has the responsibility of managing the jobs when they are submitted to the execution site via SAM-Grid's Grid to Fabric interface. The job execution time and execution host parameters are controlled by local policies enforced by the batch system.

3.3 Monitoring and Information Management

Monitoring and information management is an important aspect of any distributed system. In order for the fabric resources to interface to the Grid middleware, the resources should at a minimum provide an interface for the discovery of the resources. Monitoring of resources helps in planning and adaptation of the application behavior. It provides the user a view of various states that the application is going through, and helps in detection of faults and failures. Collecting and managing a variety of information in a Grid environment presents significant challenges as outlined below. Some desirable properties for monitoring and information system in a Grid environment are mentioned by [18] as follows.

- 1) *Platform independence*: A Grid typically contains nodes running on different operating systems, so platform independence is a must for any monitoring and information system.
- 2) *Scalability with respect to number of producers and consumers*: Because a Grid is a large scale system, scalability is one of the biggest problems to be solved. A system has to be scalable to any number of information producers without response time being affected too much. The system also has to be scalable to the number of consumers (requests) it receives.
- 3) *Ability to survive node / link crashes*: In a Grid system a node crash should be considered the rule and not the exception, the same holds for network links. Because of this, a good system should be designed such that a disappearance of a node, either by a crash or crash of the network connecting the node, should have no impact on the overall system, no matter which node crashed.
- 4) *No restrictions on the data type of information*: Good information/monitoring system should not put any restrictions on the data type of the information being used in the system. Doing so enables users of the system to create their own data types, giving a much richer and more extensible system.
- 5) *Security*: Security is of course also a very important property. The information being accessed through an information/monitoring system (IM) is not something that most people would like to share with just anyone. Proper authentication and authorization should therefore be present.

- 6) *Minimal resource consumption*: A system should always try to minimize resource consumption because an information management system is a secondary system and resources should be saved for the actual computational work being performed.
- 7) *Lightweight user interface*: A lightweight and easy to use user interface should be the rule for any good system.

The SAM-Grid monitoring and information management component satisfies the above properties by leveraging from existing technologies like Globus® MDS [19], Xindice [70] XML database and the Condor advertisement framework. The Monitoring and Discovery System (MDS) is the information services component of the Globus® Toolkit and provides information about the available resources on the Grid and their status. The Monitoring and Directory Service is the Globus® Toolkit implementation of a Grid Information Service (GIS). The MDS has the ability to function as a white pages directory for retrieving information associated with a particular name (distinguished name). Examples for such lookups are the number of CPUs and the operating system associated with a particular machine. Apache Xindice is a database designed to store XML data. It is a native XML database. The monitoring and information is categorized as

- 1) *Static information*: This type of information is typically constant for a sufficiently long period of time. Figure 3.4 below shows the static attributes of an execution site.

- 2) *Dynamic information*: Information associated with entities that change their state frequently over a period of time. Some dynamic entities in SAM-Grid are, the data handling component (SAM projects) conveying information regarding the data staged, amount of data necessary to start job execution and jobs in the batch queue. Figure 3.5 captures the dynamic attribute “Projects” [71]. “Projects or “Analysis Projects” is a SAM specific term explained in the next section. It refers to the activity of retrieving and processing data (files) via the SAM data handling system.
- 3) *Historic information*: Message logs and information regarding the history of jobs executed on a compute cluster as well as appropriate synthesis of the static and dynamic information constitutes the historical information. This kind of information is typically used for bookkeeping purposes and for reproducibility of scientific results. Debugging messages are logged by each service on the Grid to a centralized log server. These message logs are of interest to developers only and in case some error condition occurs. Messages are using the UDP network protocol.

Two different models are used to publish and extract the aforementioned information types. The static information is used by the broker to match the job requirements with the site attributes. The static information is published to the information repositories using a push model. The push model records the changes independent from the external interest in the information at a particular time instant. External entities may comprise of users accessing the static information.

Attribute	value
MyType	"Machine"
TargetType	"Job"
Requirements	TRUE
gatekeeper_url_	"sam.farm.particle.cz:2119/jobmanager-samgrid"
sam_nameservice_	"IOR: 000000000000002a49444c3a6f6f632e636f6d2f436f734e616d696e672f41"
Name	"d0_fzu_prague.d0.prd.sam.farm.particle.cz:2119/jobmanager-samgrid"
DbURL	"http://sam.farm.particle.cz:7080/Xindice"
d0software_D0RunII_p14_05_01	"Installed"
d0software_cardfiles_v00_04_06	"Installed"
station_name_	"d0_fzu_prague"
station_experiment_	"d0"
station_universe_	"prd"
jobmanager_name_	"jobmanager-samgrid"
gatekeeper_location_	"sam.farm.particle.cz:2119"
cluster_architecture_	"Linux+2.4"
cluster_name_	"Prague-Grid"
schema_version_	"1_1"
site_name_	"FZU_GRID"
local_storage_path_	"/raik_3_d0/jim_tmbs"
local_storage_node_	"sam.farm.particle.cz"
StartdIpAddr	"<147.231.19.52:56649>"
LastHeardFrom	1099041656

Figure 3.4 Static <attribute, value> pair of an execution site [87].

Monitoring at the WISCONSIN Site

Please click on a station's name to get its Server-Version and Start-time.
For stations that are grid-enabled, the Cluster Details can be viewed through the available link.

Station Name	Universe	Grid-enabled	Projects	Disks	Groups	Experiment
d0ppdg-wisconsin-2	prd	● Yes	● 0	● 1	● 1	d0
d0ppdg-wisconsin-2	prd	● Yes	●	●	●	cdf

Legend

For a listing of jobs at this site click [here](#)

Figure 3.5 Dynamic attributes of an execution site [87].

For the push model monitoring, the SAM-Grid uses a XML database infrastructure (Xindice) that is used by the configuration framework. As for the configuration infrastructure, a library has been developed to facilitate the insertion and the update of information to the database. This event-driven monitoring infrastructure is currently used to log the statuses associated with the user jobs [20]. The push model is adopted by the logging infrastructure (historic information), albeit in an unreliable (UDP packets) manner.

Retrieval of dynamic information is related to the time of request of such information by an entity (such as a user, broker or a load balancer). The dynamic information is gathered according to a pull model. The technology used by the SAM-Grid for its pull model monitoring is the Globus® MDS, complemented by the information collector of the resource selection service provided by the Condor-G framework.

Monitoring via the web is enabled by a set of PHP [61] scripts that present a consistent hierarchical extensible view of the whole system. The web interface is developed by collaborating with NorduGrid [62].

3.4 Data Handling System

Data intensive problems require large scale data management methods to transfer the data needed for solving the problem to the machine assigned to solve it. Data intensive applications such as high energy physics and bioinformatics require both computational and data management solutions to be present in the Grid infrastructure. The data handling component comprises of the SAM system. The SAM project started

in 1997 to handle the data handling needs of the DØ experiment. 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) [63]. SAM performs the task of transparently delivering files and managing caches of data. It is the sole data management system of the DØ experiment; other major experiments like CDF have also started using this system.

The information that follows in this section has been derived from [21] and [22]; more details are available in [21], [22] and [84] about SAM. SAM has been designed as a distributed system, using CORBA (Common Object Request Broker Architecture) [65] as the underlying framework. The system relies on compute systems, database servers and storage systems distributed over the world robotic tape libraries like Enstore [63] are present 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 Oracle² relational databases. Figure 3.6 illustrates the various software layers for data handling and management.

The SAM architecture is organized by physical groupings of compute, storage, network resources 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

² Oracle is the registered trademark of Oracle Corporation.

a particular *group* of physicists. *Stations* own the resource partitions, yet they do not have exclusive control over them. For instance, for the same set of compute resources, there can be two different *Stations* sharing the compute elements but with distinct and disjoint disk storage elements. This aids in running *production* and *development* *Stations* sharing the compute elements and tape storage systems but with discrete sets of files, catalogs, and disks. 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 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 provide 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. 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.

The metadata catalog provided by SAM allows users and applications acting on behalf of the user to search for data according to physics parameters. Instead of having to know a file by its name, the physicists are able to retrieve and store data according to

the description of the physics processes involved in production the desired data. In a typical use case for data retrieval, the user or the application acting on behalf of the user explores the data to be retrieved by describing pertinent conditions. They thus create a dataset comprising of files. An *Analysis project* [71] refers to the activity of retrieving and processing files. Every time a user processes a dataset, the SAM system creates and starts an analysis project. All the data processed by an application is recorded in the SAM system and is essential for reproducibility.

SAM has been in successful use for handling the Monte Carlo data on the order of terabytes produced off-site from Fermilab. Over 650 TB of data is stored for the DØ experiment in the mass storage system.

Software Layers for Data Management and Data Analysis

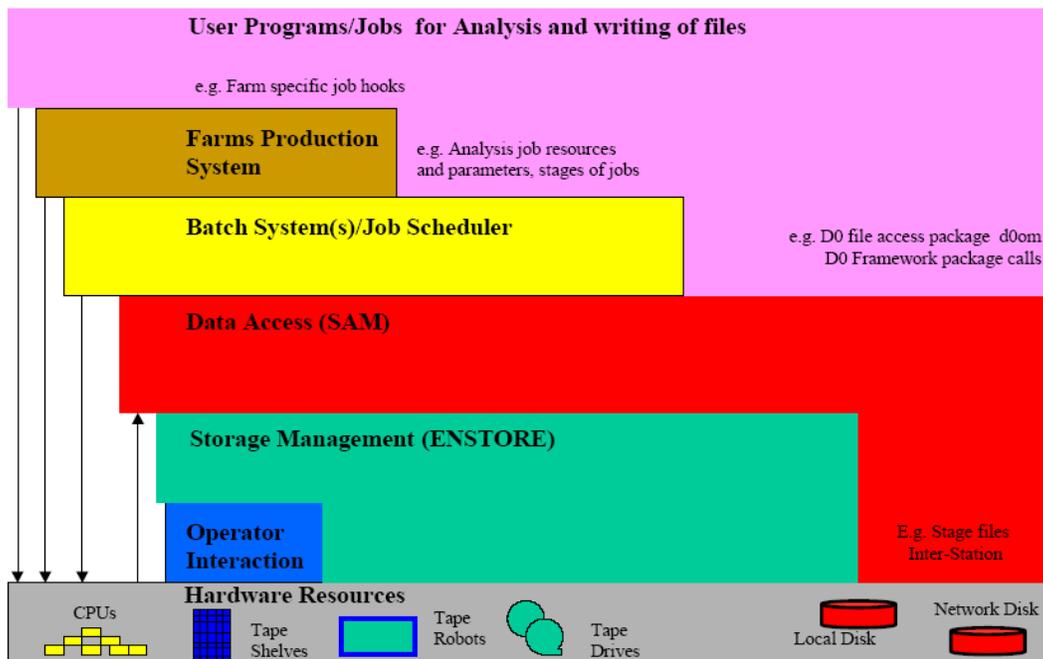


Figure 3.6 Software layers for data handling and data management (Source [63])

CHAPTER IV

PROBLEM STATEMENT

The goal of this thesis is to design, develop and implement an interface between the core Grid middleware technologies and the fabric. The application and fabric imposed complexity for data and compute intensive applications like HEP applications provided the impetus for defining such an interface. Success was measured as a ratio of the number of physics events requested to the number of physics events produced.

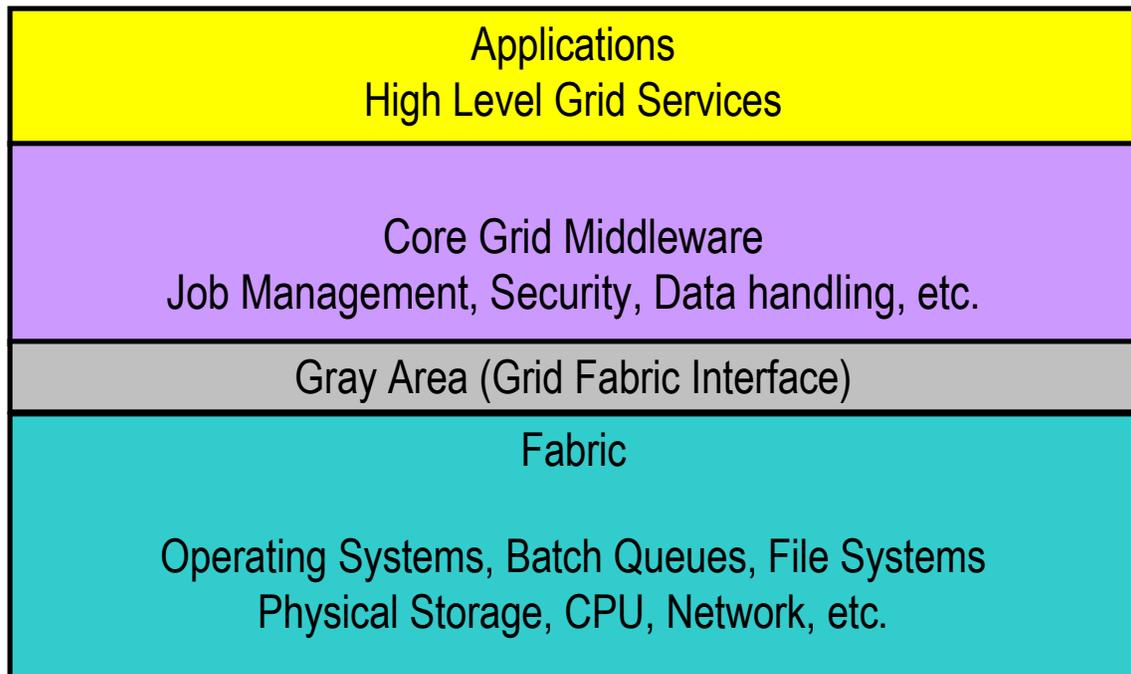


Figure 4.1 Gray area between core Grid middleware and fabric

CHAPTER V

GRID TO FABRIC INTERFACE IN SAM-GRID

The Grid Fabric layer provides the resources to which shared access is mediated by Grid protocols [2]. A resource need not be a physical entity like a compute node. It can be a logical entity like a distributed file system, or a Condor pool. In data intensive Grids and computational Grids like SAMGrid, the fabric typically consists of, but is not limited to logical entities like a compute cluster, distributed compute nodes (e.g. Condor pool) and a distributed file system. It is important to point here that a resource is identified by its interface, for example a compute cluster may use NFS storage access protocol to stage in the executables, input et cetera at the compute nodes of a cluster. From the Grid perspective, the internal protocols and implementation are not important. The most important aspect which separate Grids from a cluster based environment is the lack of control over the local policies and resources. This lack of control over the fabric and the policies that control it makes the development of Grid technologies and their deployment challenging. Richer functionality offered by the fabric in terms of coordinated access, enables sophisticated sharing operations [1]. The resources owned by the collaborating members of a virtual organization form the fabric layer of the Grid architecture.

SAM-Grid resources are mainly contributed by the members of the two high energy physics experiments, DØ and CDF. The Condor pool at the University of

Wisconsin is a major resource, thanks to the PPDG collaboration. The experience of the SAM-Grid team suggest that the existing fabric has a myriad of configurations tailored for operating in an isolated environment, which assume among many other things, user accounts, complete or no access to compute and storage resources and single administrative domain. At a minimum, the resources should implement *enquiry* mechanism, by which their state, structure and capabilities are discovered [1]. For computational resources like clusters, a mechanism for starting jobs on the worker nodes, enquiry of these jobs and their termination if necessary and capability of inquiring the state of the local resource management system like batch systems.

Grid Middleware like Globus® Toolkit provide the basic *bag of services* for implementing Grid infrastructure for virtual organizations. The Globus® Grid Toolkit is a set of low-level tools, protocols and services that has become a de facto standard for basic Grid computing infrastructure. The deployment of Complex data intensive applications like high energy physics application when deployed on a Grid infrastructure, give rise to interesting set of problems. This chapter which is the crux of the work presented in this thesis concentrates on the problems arising in deploying high energy physics applications on the Grid infrastructure in a production environment. In particular we highlight the challenges in interfacing the fabric to the Grid middleware and the solutions proposed and implemented in the context of the SAM-Grid to overcome those challenges.

The three tier architecture of the SAMGrid uses Globus® and Condor-G as its middleware for job management. Condor-G is deployed at the submission site and the

GRAM gatekeeper service provided by the Globus® Toolkit is deployed on the head node (gateway node) of the execution sites. A gatekeeper is a process used to take incoming job requests and check the security to make sure they are allowed to use the site resources. The gatekeeper process is responsible for starting up the job-manager process after successful authentication.

The Globus® Resource Allocation and Management (GRAM) service provides for the management and remote execution of jobs defined using a standard Resource Specification Language (RSL). Currently, the GRAM has very limited functionality, making it difficult to deploy complex data intensive applications. The GRAM runs a jobmanager service that is responsible for job execution and monitoring, and accepts jobs defined using a standard Resource Specification Language (RSL).

High energy physics community has been traditional using resources at individual institutions like universities and government labs for satisfying their computational and storage needs. Applications like Monte Carlo simulations have been traditionally executed in a very controlled environment and make many assumptions regarding resources on which they execute. Assumptions like locally available data, availability of a shared file system, application code installed on every node of a cluster or in a shared area accessible by the worker nodes et cetera are commonplace. The present Grid services like the Globus® jobmanager service lack the support for application specific environment. Extensions to the RSL for dealing with application specific environments have been proposed by [23]. These extensions are right steps towards the final goal, but still lack the stability, required in a production environment.

Data staging is an important aspect of executing data intensive applications on a Grid infrastructure, the current data staging capabilities at present lack the support for caching, and tracking the data usage, necessary for reproducibility and statistical purposes. High energy physics applications operate on huge data of the order of hundreds of gigabytes. Data staging facilities integrated with GRAM are useful for retrieval of files of the order of few hundred megabytes. These files are typically log files and output and error streams of the job execution. The output and error streams of the job execution might not convey sufficient information to the user, and valuable information might be lost as well as the loss in computing time. Retrieval of application specific logs, which indicate to the user the behavior of the application, is limited by the local resource management system.

The Globus® Toolkit jobmanager service, instantiated by the GRAM service at the gateway provides interface to only standard scheduling systems like Condor, PBS and LSF. Separate plug-ins have to be written for scheduling systems like Batch Queuing System (BQS) [64] used by the DØ collaborators in Lyon-France at the *Centre de Calcul l'IN2P3* (CC-IN2P3), Farm Batch System Next Generation (FBSNG) [24] developed at Fermilab and McFarm [85] developed at The University of Texas at Arlington. The standard plug-ins implemented for the Globus® Toolkit fail to implement site specific features out of the box. Tweaking the plug-ins to incorporate site specific features, like avoiding preemption, utilizing access to high performance local storage and adapting to specific fault conditions requires manual intervention of

site administrators who might not be familiar with the Grid protocols or the implementation language for the plug-ins.

GRAM Jobmanagers do not provide facility for retrieval of application specific logs. Temporary disruptions or slow communication between the local resource management software like batch scheduler and the Jobmanager might lead to the false conclusion that the application has failed or worse completed successfully, thus leading to a resource leak.

5.1 Job Managers

5.1.1 GRAM Job Managers

The Globus® Resource Allocation Manager (GRAM) is the lowest level of Globus® resource management architecture. GRAM provides an API for submitting, monitoring, and terminating Grid jobs. When a job is submitted by the Grid scheduler, the request is sent to the gatekeeper of the remote computer. The gatekeeper handles the request and creates a job manager for the job. The job manager starts and monitors the remote program, communicating state changes back to the submission site, which is the GRAM client. When the remote application terminates, normally or by failing, the job manager terminates as well.

The job manager is typically started by the Gatekeeper service. It interfaces with a local scheduler (e.g. condor_schedd for the Condor batch system) to start jobs based on a job request RSL string. One job manager is created by the gatekeeper to fulfill every request submitted to the gatekeeper. It starts the job on the local system, and handles all further communication with the client.

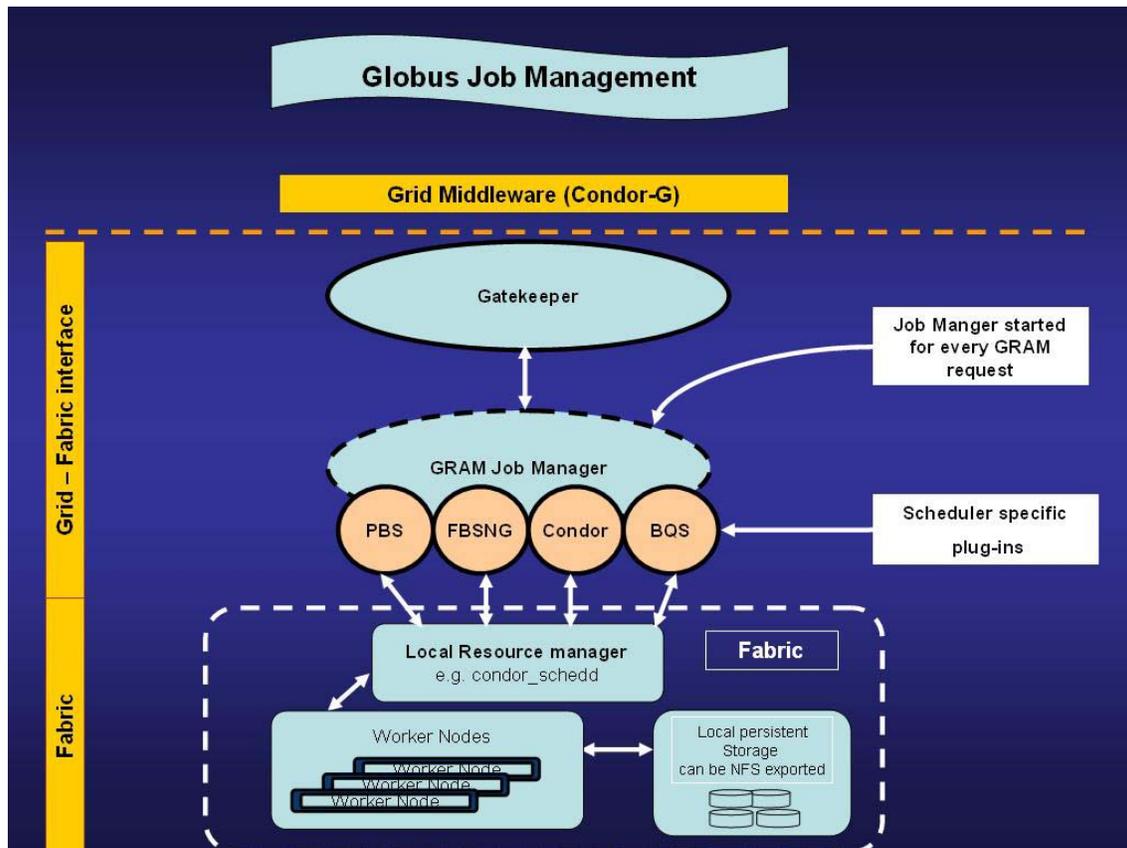


Figure 5.1 Globus® job management on the gateway node

It is made up of two components:

- 1) *Common Component* - translates messages received from the gatekeeper and client into an internal API that is implemented by the machine specific component. It also translates callback requests from the machine specific components through the internal API into messages to the application manager.
- 2) *Machine-Specific Component* - implements the internal API in the local environment. This includes calls to the local system, messages to the resource monitor, and inquiries to the MDS.

5.1.1.1 Limitations with GRAM Job Managers

The current Globus® GRAM implementation suffers from several problems and limitations with the current Globus® GRAM implementation. A major limitation is that the standard RSL fields understood by the GRAM lack the functionality of staging files of the order of gigabytes, this is because GRAM relies on the Globus® Global Access to Secondary Storage (GASS) protocol, which is a fast and secure protocol not meant for transferring files greater than a few hundred megabytes. During a job submission, a GASS client is started on the execution node to handle the data transfers. It uses the RSL parameters to determine what files and what location they need to be transferred to and from. The GASS client communicates with a GASS server that is started by Condor-G on the submission site. GASS is used for transferring files like small executables, and redirections of standard input and standard output to and from the remote execution host. The main issue we are attempting to address is the lack of support by the GRAM for arbitrary execution environments. Specifying a job that requires a special execution environment requires mangling the semantics of a standard RSL request. The content of this section is derived from [23] and [6].

5.1.1.1.1 Lack of Flexibility

One of the goals of Grid computing is to support submission of jobs to a Grid of multiple compute clusters, either in a local area (a “campus Grid”) or a wide area (e.g. a national or global Grid). These clusters will all be running their own batch systems, which may not all be the same. The GRAM provides a very useful (and necessary) mechanism for translating a GRAM job specification in RSL into a job submission

script for a variety of batch systems, using the scheduler-specific plugin architecture as shown in figure 4.1. This is straightforward since the RSL job specification is restricted to very few fields, basically just those that would be used by the default fork jobmanager. Unfortunately this means that information that could be useful for the batch scheduler on a cluster compute resource is not available. For example, as part of a special agreement, the University of Wisconsin at Madison runs some of the DØ jobs on their condor cluster without pre-emption. The intention to take advantage of this local policy must be expressed at the time of local job submission. The submission command is specific and cannot be expressed using the standard job-managers. Another example is the special option used at the IN2P3 computing centre in Lyon, France, to inform the scheduler that a job plans to access data via HPSS [81], the local mass storage system. In case of HPSS downtime, the batch system can schedule those jobs specially, avoiding crashes due to denial of access to the data. This option is also site specific and cannot be part of the standard job managers. In general, the job-managers do not provide a way to customize the interface to the local batch system. The main issue here is that the GRAM assumes that an appropriate computational resource has already been identified by a resource broker or scheduler based on some resource requirements for the application (e.g. architecture, operating system, memory, software availability et cetera), so the job is submitted to a particular node in the Grid that satisfies these requirements. Hence the standard RSL job submission request only provides job execution details, not resource requirements. This is different to a typical job specification for a batch system, where the user specifies both the job and its

resource requirements in one submission script, and the batch system provides the resource broker and scheduler as well as a job manager. Thus, problems arise when jobs are submitted to the GRAM using standard RSL that has no resource requirement specifications. The GRAM passes this information on to the local scheduler using the scheduler-specific plugin, but this does not include resource information. Even if information on resource requirements is added to the RSL job submission script, it is thrown away by the GRAM and not passed to the batch system. This is not a problem if the cluster has homogeneous nodes, but for a heterogeneous cluster resource requirement information is useful (and in many cases, necessary) for the batch scheduler to be able to effectively schedule the job.

5.1.1.1.2 Inefficiency and lack of scalability

Another potential problem with the GRAM is inefficiency. Submitting and executing jobs on a remote system will inevitably introduce some inefficiency. Monte Carlo simulations constitute a major percentage of Grid applications. These simulations typically operate on independent data for the same executable. For every Grid job submitted to the gateway we have three ports and a *GRAM Job Manager* process, on the average commodity machine this limits the number of Grid jobs to a few hundreds. Most batch systems support parameter sweep applications by providing a mechanism for queuing multiple jobs with multiple input parameters and output data. However the GRAM supports only individual jobs. The inefficiency in this method mainly comes from two places:

- 1) Files need to be transferred before and after each job execution. The same files used in each execution, such as the executable, cannot be shared by multiple jobs, and must be staged in repeatedly.
- 2) Mutual authentication must be performed for each submission and file staging.

The SAMGrid aggregates multiple local jobs to form a single Grid job. This aggregation is not part of the standard *GRAM Job Managers*. GRAM uses a simple, standard format for specifying jobs to the GRAM is necessary for interoperability on the Grid. However it comes at the cost of potential loss of flexibility and efficiency. Ideally the GRAM would also offer a mechanism for handling more advanced job information when the client application and the local scheduler support it.

5.1.1.1.3 Robustness

The GRAM Jobmanagers cannot react to temporary problems when interacting with the local scheduler. If the batch server is managing hundreds and thousands of jobs, it responds slow and is misconstrued by the GRAM Jobmanagers as a failure.

5.1.1.1.4 Comprehensiveness

The Globus® job-managers interface to the local batch systems only. There are a series of other fabric services that in general need notification when a job enters a site. The data handling system could start pre-staging the input data, while the job is idle in the scheduler queue. The monitoring system can observe the status of the job in the queue, while it is not running; this cannot be achieved if the job is responsible for sending monitoring information. Database accesses common to all the batch processes can be aggregated, thus reducing dramatically network traffic.

5.1.2 SAM-Grid Job Managers

The SAMGrid Job Managers try to overcome the deficiencies of the GRAM job managers, by utilizing the set of services developed as part of the Grid-Fabric interface. As shown in the figure 5.2, an application specific environment is provided by the application specific plugin in the SAMGrid job manager. A single GRAM job manager is started for multiple local jobs; according to local or virtual organization policies (i.e. the split factor is configurable). This aggregation is also convenient for the Grid users, who can manage their Grid jobs as single entities, irrespectively of their local multiplicity. Large scale data staging for local jobs is done by the sophisticated data handling service provided by SAM. The interactions with the local batch system, or rather its “idealizer”, are mediated via a layer of abstraction, called the “batch adapter”. The batch adapter is a virtual service that is set up at installation time to reflect the specifics of the configuration of the local batch system. Using this extra layer of indirection we could customize the Grid-fabric interface to the batch system of every collaborating site, reflecting the peculiarities of the local policies and hardware/software configuration. The idealizers are batch system specific plugins that use the batch system in an efficient way and provide basic fault detection and tolerance to the higher job management layers. To provide interoperability the SAMGrid job manager is driven by GRAM job manager as shown in figure 5.2.

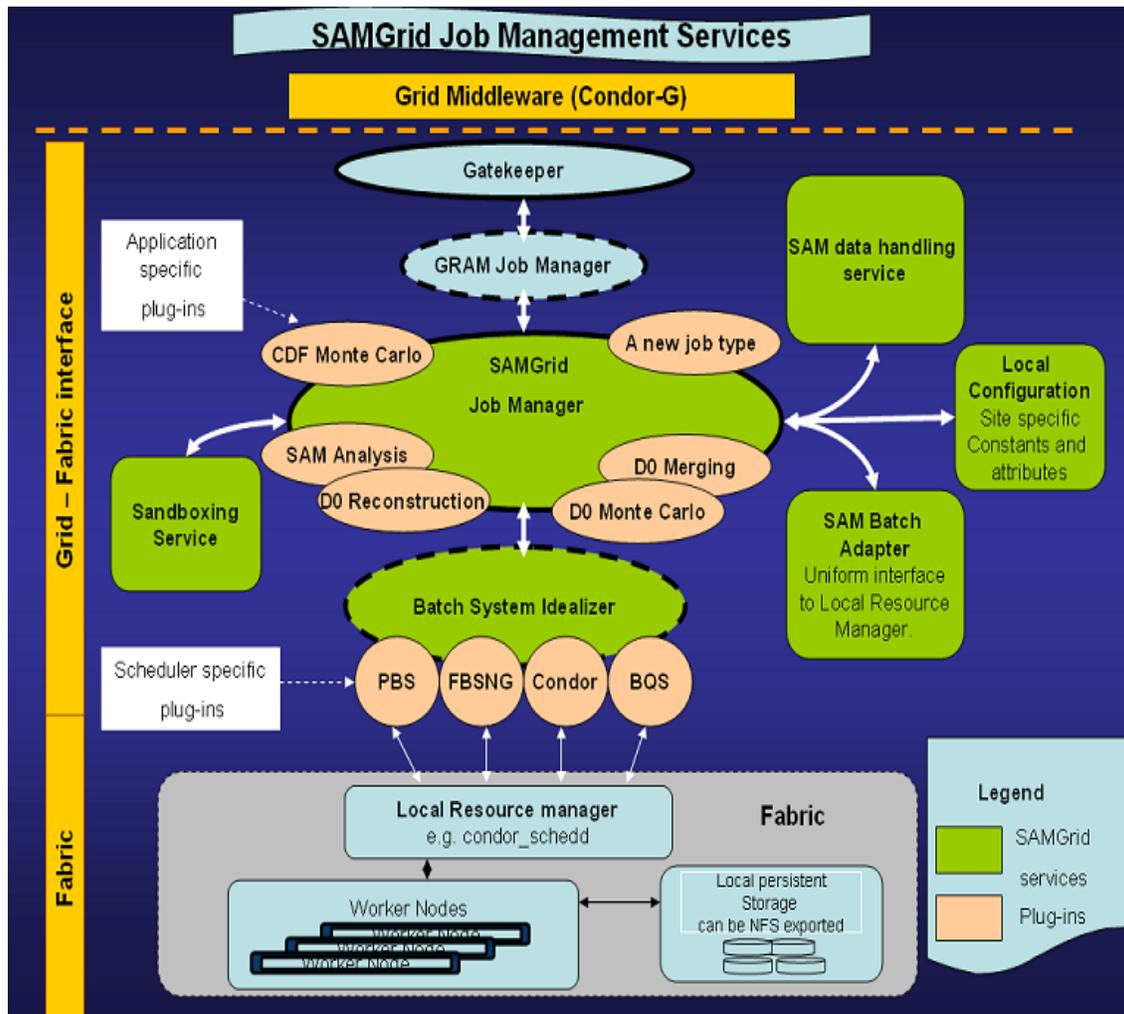


Figure 5.2 SAM-Grid job management.

The overall flow of control is driven by the Globus® job managers, the aim here is to maximize the use of Globus® toolkit and minimize the development of application-specific or VO-specific services.

5.1.2.1 Flow of control for SAM-Grid Job Managers

When the initial GRAM request is received by the gatekeeper service running in the gateway node, it spawns the *GRAM Job Manager* process which interfaces with the

SAMGrid job manager for the purposes of actually managing the jobs. The GRAM Job Manager relies on the SAMGrid job manager for submitting jobs, polling for existing jobs and terminating jobs. The SAMGrid Jobmanagers can be also thought of providing a job management service. SAMGrid job managers in turn return certain information like job submission success, whether the job is active or not and if the termination of the job was successful in the case of job termination to the GRAM Jobmanager. This information has to be returned in a certain specific time frame. Since the batch system response time is uncontrollable, it is technically difficult to ensure the timeliness of completion. As an implementation detail, the timing constraints are avoided by launching the processes in the background which look for the requested information and carefully avoiding duplicate processes.

5.2 Sandboxing Mechanism

5.2.1 Introduction to sandboxing

Sandboxing is a popular technique for creating confined execution environments, which could be used for running un-trusted programs. A sandbox *limits*, or reduces, the level of access its applications have - it is a *container* [77]. The term sandboxes are used in numerous contexts. Sandbox environments range from those that look like a complete operating environment to applications within, to those that provide a minimum level of isolation (to a few system calls). The term *Sandboxing* within this thesis and the SAM-Grid terminology in general is used in a complementary sense to the popular notion associated with security.

Sandboxing in SAM-Grid provides a container service. SAM-GRID provides a container service which is responsible for bringing the user code in and carrying the log files and output out of the sandbox area. The physical implementation of creating a sandbox translates to creating a directory as in UNIX terminology. The sandboxing term is widely used in terms of a secure environment in which code that is not trusted (for example mobile agents) is executed. System call by the foreign code are trapped and inspected for maliciousness. SAM-Grid sandboxing does not trap system calls, though such functionality could be added on later.

5.2.2 Need for sandboxing in Grid environments

In a Grid environment the compute nodes are typically shared among many virtual organizations. The need for a sandboxing abstraction of the execution environment and related services like intra-cluster data transfers stem from the following characteristics of prevalent configurations at execution sites.

- 1) In a typical cluster based environment of many execution sites, standard software is installed on the worker nodes by installing it in an exported shared file system like NFS. Standard software here implies application specific software, special libraries and numerous other packages that help execute the application, also referred to as “infrastructure” packages such as interpreters, compilers, archiving tools et cetera.
- 2) Cluster based computing typically has a durable local storage area (home area), where the jobs and agents (such as batch systems) acting on behalf thereof can

safely deposit small files. These files include both those needed to bootstrap the job (input) and any logs produced (output).

- 3) Intra-cluster file transfer mechanisms are different for different batch systems. Most systems relied on a shared file system; some used batch systems with built-in file transfer mechanisms, etc. Development of a uniform job submission interface, which could be used by standard Grid middleware, was severely complicated by this heterogeneity of job file transfer mechanisms.

The first characteristic is in keeping with the model of submitting jobs to locally owned resources, and the as the applications become more Grid aware, the application developers are lifting the requirement by developing tools to envelop their applications and provide appropriate run-time environment. Dynamic installation of application specific software environment on the worker nodes is a very important aspect for deployment point of multidisciplinary Grid applications on the worker nodes of an execution site. Dynamic installation on a worker node consists of user code and other run-time environment augmented with additional infrastructure packages like interpreters, data handling client software et cetera. In SAMGrid, this type of dynamic deployment is accomplished by the packaging API provided by the sandboxing mechanism (also known as the sandboxing service).

The second characteristic is almost always implied. The *home* area concept has a long history and is part of the broader concept of an account, whereby computer access is controlled statically. Grid computing strives to provide a fuller and much more dynamic resource control by virtue of sophisticated authorization frameworks. Thus, it

has become increasingly necessary to be able to move job files in and out of the execution node bypassing home area. The sandboxing mechanism in SAM-Grid obviates the home area concept. Incidentally, both of the above characteristics involve usage of a shared file system such as NFS [86]. Shared file systems have suffered from the issues of performance bottleneck (because of the centralized topology and UDP-based communication) and low security (NFS authentication is IP based). The sandboxing mechanism provides complete independence of the shared file system.

Experiences in deploying the SAM-Grid infrastructure indicate the well-known issue of dramatic variation of the computing environment of the execution sites. Development of a uniform job submission interface, which could be used by standard Grid middleware, is severely complicated by this heterogeneity, especially when the mechanisms of intra-cluster file transfer are considered. Most systems rely on a shared file system; some use batch systems with built-in file transfer mechanisms. To be independent of the local intra-cluster file transfer mechanism, a data transport service in the form of a GridFTP [44] [26] server is instantiated, if not already available. In SAM-Grid a job execution site has a GridFTP server running on the head node. GridFTP is the Grid version of the *File Transport Protocol* for moving large datasets between storage elements within a Grid. The head node generally refers to a node in a cluster through which jobs are submitted. Most of the execution sites in SAM-Grid have the head node running a gatekeeper and have the monitoring software installed. A head node is networked to a group of worker nodes and has the batch system server or scheduler running on it. If a job execution site does not have a GridFTP server running

on its head node, then a dynamic GridFTP server is instantiated by the sandboxing service for intra-cluster sandbox transfers. This ensures a uniform, secure (X509 based), robust and scalable (multiple transfers) file transfer mechanism to transfer data between the nodes in the cluster, in particular the head node and one or many worker nodes.

5.2.3 Sandboxing in SAM-Grid

During job submission, a user typically supplies or specifies what program to execute on the Grid, for example the name of the executable and optionally the input files and the place to deposit the output of the execution. When the user specifies some inputs such as application name, product dependencies, optional input dataset and an archive containing the software and configuration files needed to run the application, the archive should be transported to the execution site. This archive is sometimes referred to as the *input sandbox*. Depending on the information collected by the resource broker the job is scheduled on an execution site. When the execution or the run is complete, the output if any is collected and deposited in either a user specified place or some default location depending on the data handling system. The output and the history of the execution, i.e. the logs could be deposited in separate locations. The output of the job, such as the error and output streams, relevant log files and output files is called the *output sandbox*. It should be noted the SAM-Grid handles the large input and output data files and potentially the product releases via the SAM data handling service, hence dramatically reducing the size of the output and input sandboxes, since these files do not need to be part of them. As highlighted in section 5.2.2, application specific software is typically installed on the worker nodes via shared file system

leading to the possibility of version mismatch if the brokering is not sophisticated, in terms of the deciding if a version of software is compatible with the application to execute, for example if an user program uses python 2.2 [27] libraries and the execution site has python 2.1, either the resources at that site are rendered useless because of version mismatch or if the broker does not know the versions of the software and schedules the job on the execution site in question, the user program is bound to crash. Either case is unacceptable. Hence the need to dynamic deployment of software environment on the worker node, this includes instantiation of services on the worker node, setup of the environment required for the user program and handling and transfer of the input and output sandboxes.

The Grid Security Infrastructure relies on the machine clock to determine the validity of the security tokens (X.509 proxies). If the system clock is running slow then the security mechanism fails because the security token (X.509 proxy is not valid). In addition of requesting the site administrators to run ntpd [82][82], the sandboxing service shield's the application from slow clocks at the worker nodes by introducing artificial delays during the bootstrapping process at the worker node. The sandbox manager is responsible for deploying the sandbox on the worker node, it checks if the system clock of the worker node is within the specified range with respect to the timestamp on the security token, if not an artificial delay is introduced.

SAM-Grid sandboxing service offers a suite of Python API's and command line interface. Table 5.1 provides the list of API's along with description. As an implementation detail, the API is accessed via an object of the python class named

Sandbox. An object of this class is created by the SAM-Grid jobmanager during the job submission and during the output collection, when all the local jobs finish.

Table 5.1 SAM-Grid sandboxing python API's

Python API	Description
<p>sandbox_create(char *<i>name</i>, [char *<i>file</i>])</p> <p><i>name</i> = Name of the sandbox area <i>file</i> = Optionally the name of the input sandbox.</p>	<p>Implementation currently is limited to creation of a directory in a predefined location. This directory location is called the sandbox area or the work directory. If an input sandbox is supplied, a soft link is created in the sandbox area.</p>
<p>sandbox_add(char *<i>workDir</i>, PyObject *<i>files</i>)</p> <p><i>workDir</i> = Absolute path to the sandbox area on the file system. <i>files</i> = A python list object containing files to be added in the sandbox area.</p>	<p>Add files specified as a python list object into the sandbox area. Currently this is implemented by creating soft links pointing to relevant that are to be added.</p>
<p>sandbox_package (char *<i>name</i>, char *<i>command</i>, [PyObject *<i>environ</i>], [char *<i>output</i>], [char* <i>template</i>])</p> <p><i>name</i> = Name of the sandbox <i>command</i> = command to be executed by the bootstrapping code. <i>environ</i> = Python dictionary object used to specify arbitrary environment for the user program. <i>output</i> = Name of the self extracting archive <i>template</i> = Used to create the bootstrapping code.</p>	<p>Starts the dynamic GridFTP server if the site is not using one for intra-cluster transfers. Creates a self extracting archive containing the software environment required for the user program to successfully execute on the worker node, containing the bootstrapping code and GridFTP client software.</p>

5.3 Uniform interface to batch systems - Batch Adapters

5.3.1 Need for Batch Adapters

Interfacing fabric components like the batch system to the Jobmanager layer is one of the most important aspects of the job management of any successful Grid middleware. The Globus® toolkits GRAM Jobmanagers provide interface to various popular batch systems like PBS, Condor and LSF. The GRAM Jobmanagers also provides the capability of plugins for any arbitrary batch system. As mentioned in section 5.1.1.1.1 the GRAM Jobmanagers are inflexible because the GRAM RSL does not allow any arbitrary parameters, which might result in the efficient use of the Compute Nodes by the batch system.

The SAMGrid overcomes this inflexibility by providing an API for interfacing arbitrary batch systems via the *Batch Adapter* layer. The *SAM Batch Adapter* layer is a python API which serves as an interface between the SAM-Grid Jobmanager and batch systems used for submitting user jobs. The API is fully configurable and does not make any assumptions about underlying batch systems. It has an administrative interface that can be used for adapter configuration. Once it has been configured, it will contain knowledge about all batch systems available at the execution site.

The execution site can have any number of batch systems available for submitting and running user jobs. For each of those batch systems there should be an adapter configured. As an implementation detail, the batch adapter configuration is kept in a local python module which gets updated every time a valid administrative command is executed. Adapter configuration consists of batch commands and queues

available to users, as well as of the default batch system limits. Batch commands are described by their type (e.g., *job submission command*) and command string which may contain any number of predefined string templates (e.g. *qstat %__BATCH_JOB_ID__*). They can be associated with any number of possible outcomes characterized by the command exit status, as well as by its output string which also may contain templates (e.g. *JobId=%__BATCH_JOB_ID__ Status=%__BATCH_JOB_STATUS__*). It is important to note here that the Batch Adapter API does not execute batch commands. It simply provides functionality for preparing commands before their execution, as well as for analyzing their outcome. It is responsibility of the API user, which is the SAM-Grid Jobmanager, to execute commands and interpret their results. There are several types of batch queues that can be configured for a given adapter. The Batch Adapter API does not make any assumptions about client usage of those queues, so that different clients may use the same type of queue for different purposes. Adding new queue types is straightforward, which makes the API flexible and extensible. The batch queues can have different limits configured, and those limits override the default adapter limits.

5.3.2 Batch Adapter – Jobmanager interaction

The SAM-Grid Jobmanager utilizes the Batch Adapter API to get the information regarding job submission, job lookup, and job termination and also the information about interpreting the result of each of the aforementioned commands. The Jobmanager reads the template, replaces the appropriate fields and executes the action. The Batch Adapter API has adapters for different batch systems. To make the Grid-Fabric interface portable on any execution site, the Batch Adapters are configured with

a “Grid” adapter. This adapter serves as an extra layer of indirection and return to the SAM-Grid Jobmanager the specific idealizer to invoke as shown in the figure 5.3. During the job submission by the SAM-Grid Jobmanager, the Jobmanager invoke the Batch Adapter API to query for the command to submit to the batch system by specifying the “Grid” adapter. The Grid adapter returns the job submission command to the Jobmanager layer as well as the template for interpreting the outcome of the submission and information regarding return status is also provided. The job submission command for the Grid adapter is not a batch system command, but rather the information (path to) regarding the batch idealizer to invoke. This interaction is transparent to the Jobmanager layer.

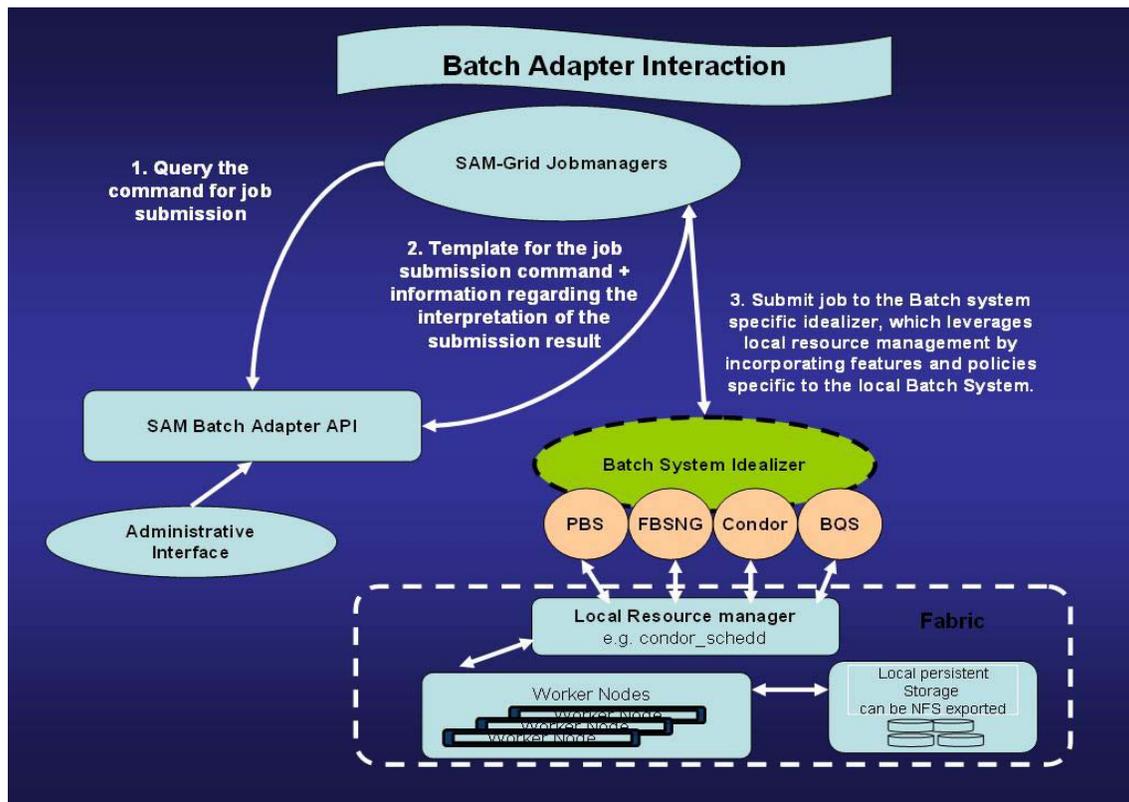


Figure 5.3 Batch Adapter – SAM-Grid Jobmanager interaction

The batch idealizers as discussed in the next section are responsible for incorporating fault tolerance during the interaction with the batch scheduler and provide for incorporating local policies regarding resource limits, like cpu limits, queues to submit et cetera. For example during the job submission if the local batch system has special flags to enable access to a high capacity durable storage for storing the output of the job, or if the local batch system has an efficient implementation of the API for looking up submitted jobs, the idealizer layer takes into consideration such local optimizations.

5.4 Batch System Idealizers

5.4.1 Need for Batch System Idealizers

Batch scheduling systems are typically the local resource manager for compute nodes at an execution site. In a typical configuration, jobs are submitted to the compute nodes via the interface provided by the batch system. For example to submit, poll and remove jobs to the PBS batch system the command line interface is qsub, qstat and qdel respectively. In an interactive job submission session, the user typically specifies a job queue (for PBS and its clones) or the list resources on which the job should execute via the classAd mechanism for the Condor batch system.

5.4.1.1 Scratch management on worker nodes

Modern batch systems typically have some kind of scratch management mechanism on the worker nodes providing a temporary area on the worker node where the job files are staged via some intra cluster transfer mechanism, execution is triggered by a batch system daemon running on the worker node and the temporary area cleaned

(removed) when the job execution is done and necessary output transferred. If the batch system does not provide scratch management, it results in the disk on the worker node getting full, eventually leading to an unusable worker node if the temporary areas created for individual jobs are not cleaned manually.

5.4.1.2 Limit on local job names

Batch systems like OpenPBS and BQS have limit on the characters that are used to name the job. For example PBS has a 10 character job name limit; BQS has a 20 character job name limit. It is hence essential to map a Grid job name greater than the local batch system limit to a job name with characters within the local batch system limit.

5.4.1.3 Job lookup failures

The SAM-Grid was initially interfaced to three different batch systems: PBS, BQS, and Condor. After submitting jobs on the order of hundreds, the SAM-Grid periodically polls their status. In our experience, all of these batch systems, especially when under stress, have failed to report the status of the local jobs, either because the polling request (*condor_q* for Condor and *qstat* for PBS) timed out (as observed for PBS, or because the batch system temporarily couldn't find the job in the queue for BQS). It should be noted that this transient condition would not disrupt the activity of an interactive user. To the contrary, it causes the Grid to consider the job terminated, thus creating a resource leak.

5.4.1.4 Black hole effect

Physics simulations using Monte Carlo methods constitute a major percentage of Grid applications. It's their inherent parallelism that lends them to execute on computational Grids. Parallelism is a way to accelerate the convergence of a Monte Carlo computation. If N processors each execute an independent copy of a Monte Carlo computation, the accumulated result will have variance N times smaller than a single copy. Grid jobs in production Grids like SAM-Grid manifest themselves into multiple local jobs at an execution site. Most of the execution sites have a mix of dedicated and non-dedicated computational resources, managed by a network batch queuing system such as Condor, PBS and BQS, operating in distributed, networked environments. When a Grid job is instantiated at the execution site it usually outnumbers the compute elements (nodes), resulting in local jobs getting queued in the batch system. The batch system schedules these local jobs on compute resources as they become available. The application relies on some basic services provided by the compute nodes, if any of these required services malfunction, the application fails and is evicted by the batch system. The batch system and the monitoring software typically do not deal with application specific failures. If other jobs of the same application are scheduled on the faulty node, it is bound to fail. From the application point of view the compute resource in question is as good as unavailable, but the batch system might continue to schedule jobs on the same compute node causing spurious high turnaround, and eventually yielding very low throughput for the Grid job. This effect is called the Black Hole effect, and the compute nodes which malfunction with respect to the application are called black holes. The

effect is not limited to only Monte Carlo jobs, but can manifest itself for any type of application.

5.4.2 SAM-GRID Batch System Idealizers

The aforementioned problems are solved in the Grid-Fabric interface layer by providing a level of abstraction on top of the batch systems, with the purpose of increasing the reliability of the interaction with them. This layer is the “idealizer”, as it idealizes the behavior of the underlying batch system. The *Batch System Idealizers* provide features like aggregating polling requests in order to reduce the stress to the batch system, providing a scratch management service by bootstrapping the executable, providing a mapping between the Grid job names (Global job ID) and local batch system constraints regarding the limit on number of characters to be used for the job name. The idealizers shield the Grid application from black hole effects by statistically determining the faulty nodes. Retrials are incorporated with exponential back off for job lookups, thus providing a buffer to deal with transient failures. The Batch System Idealizer layer gives full consideration to local batch system configurations and takes advantage of some of the custom built services. For example the *qselect* command for the BQS batch system is an optimized version of the *qstat* command to poll jobs in the batch system. On the Condor cluster at the University of Wisconsin, a job submitted to a category of special nodes had high priority and no preemption.

CHAPTER VI

EXPERIMENTAL RESULTS

The SAM-Grid infrastructure has been actively used for producing simulated events using Monte Carlo methods for the DØ collaboration since March 2004. Efficiency of the Grid infrastructure for producing Monte Carlo events is measured as the ratio of number of simulated physics events produced by the SAM-Grid to the number of events requested.

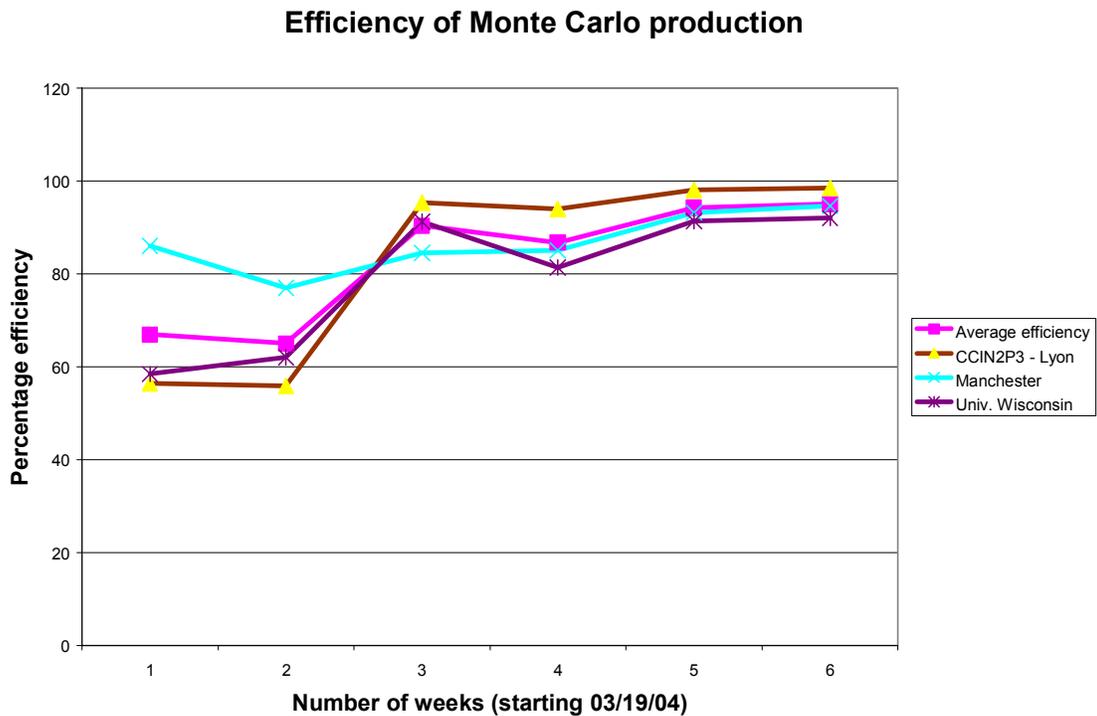


Figure 6.1 Efficiency of Monte Carlo production using SAM-Grid

$$\text{Efficiency} = \frac{\text{Number of events produced}}{\text{Number of events requested}} \times 100$$

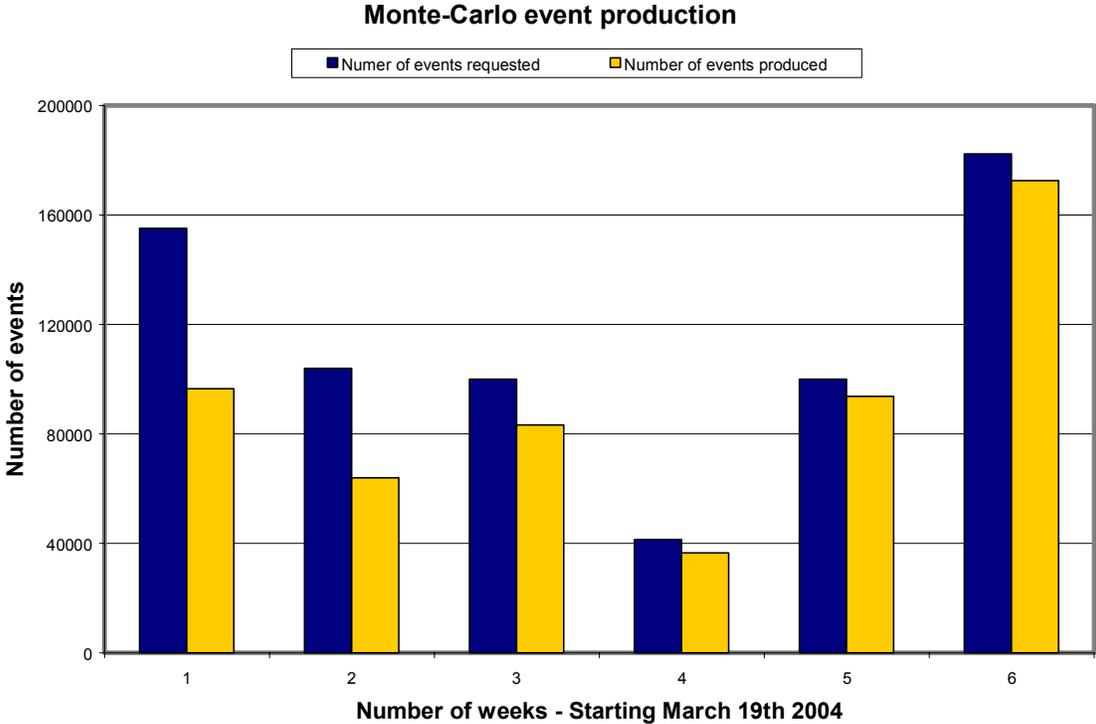


Figure 6.2 Monte Carlo production using SAM-Grid

As shown in figure 6.1, the average efficiency (indicated by the pink line) of SAM-Grid was improved from 67% to 95% over a period of six weeks. As seen from the figure the SAM-Grid efficiency increased from 65% to 90% after the second week. The main reasons for this increase were the incorporation of retries with exponential back-off at different layers (for example SAM-Grid job managers and batch idealizers)

of the Grid-Fabric interface; this improved the performance of the system when the central database server was heavily loaded. A lightweight mechanism for transferring data from other SAM *Stations* was developed. This mechanism improved on the process of fetching input data by running SAM projects on the execution site and then selecting the required files from all the retrieved files. Instead of fetching all the files in a dataset and then selecting from them, only file identities were retrieved and the necessary files were retrieved using these file id's from preferred replica locations. Aggregating requests to the database and working closely with site administrators were some of the other contributing factors for the improvement of efficiency.

Figure 6.1 also shows the efficiency of three execution sites individually. The brown line shows the efficiency for CCIN2P3 execution site in Lyon, France. The efficiency for this site was better than the efficiency of the other two sites shown in the graph for most of the weeks. The primary reason being that the cluster was operated and supported 365 days a year. Configuration related issues like migration of worker nodes from Red Hat Linux 7.3 to Red Hat Linux 9 and pre-emption on the University of Wisconsin's Condor cluster were the main causes of inefficiency due to incompatibilities between the glibc libraries. On the Manchester cluster (indicated by sky blue color) the main causes of inefficiency were issues relating to system clock skew on the worker nodes and the frequent deactivation of the local data handling interface (SAM disks) due to manual interference by the local users.

Figure 6.2 illustrates the Monte Carlo event production corresponding to the period during which the efficiency was measured. It is important to note that the

number of events requested fell during the 4th week, but the efficiency increased as illustrated in figure 6.1. A special mention of a particular category of failures that was observed in two of the execution sites (University of Wisconsin and Manchester). As discussed in section 5.4.1.4, even if a single node in the cluster is ill configured and makes the jobs scheduled on it crash, the batch system keeps sending idle jobs to it, leading to the whole queue of jobs crashing. In the SAM-Grid framework, A single Grid request is mapped into multiple local jobs. Typically the number of local jobs exceeds the compute resources, resulting in local jobs being queued in the batch system.

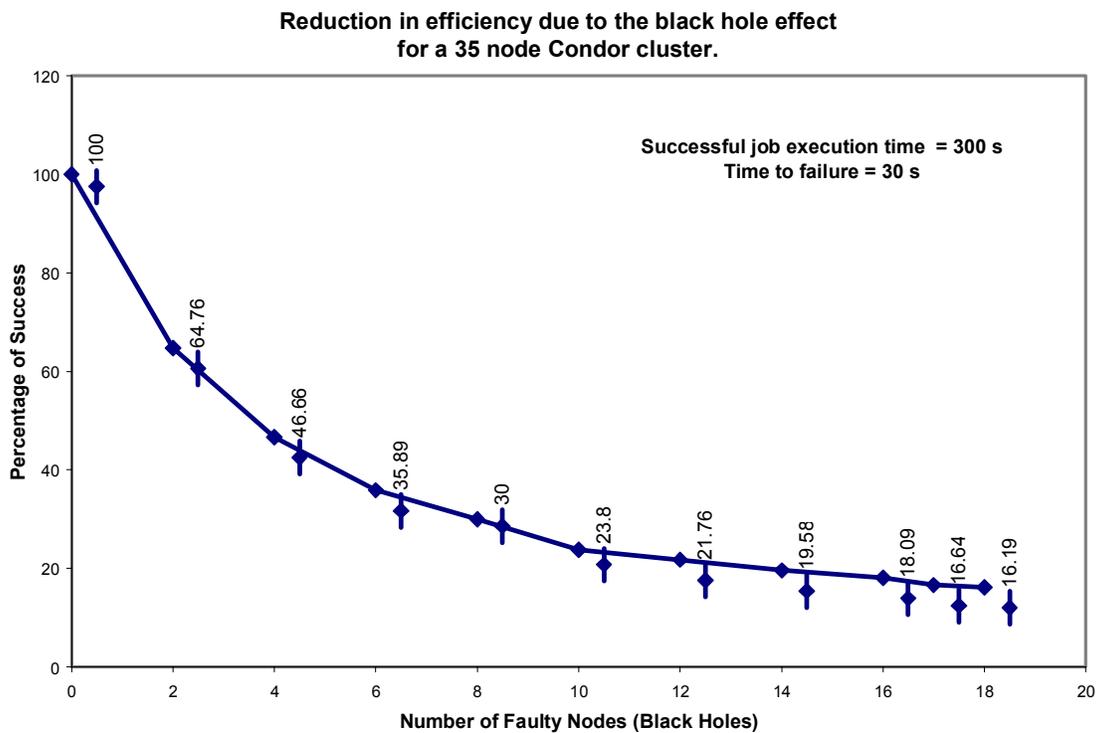


Figure 6.3 Reduction in efficiency and throughput due to “Black Holes”

The detrimental effect that these ill configured nodes (Black Holes) have on the efficiency of a Grid job due to the failure of local jobs is illustrated in Figure 6.3.

At the University of Wisconsin's Condor cluster, one of the worker nodes had configuration problems (older version of the GNU tar utility). This dual processor worker node failed every job scheduled on it and contributed to two "Black Holes", thus reducing the maximum efficiency of the Grid submission. This problem was fixed by statistically determining faulty nodes and eliminating them for the application via the Batch System idealizers. Lessons learned in deploying SAM-Grid for DØ Monte Carlo production and the resolutions of problems are discussed in detail in [6]. The *Black Hole effect* is discussed in detail in [41].

SAM-Grid infrastructure currently has around 13 execution sites. Figure 6.4 shows the number of events produced from March 2004 for 7 execution sites. Around 2 million physics events were produced during this period from these execution sites, which is approximately equivalent to ~19 years in wall clock time of a single 1GHz processor (300 seconds per physics event).

Figure 6.5 illustrates the number of global (also known as Grid) jobs submitted to SAM-Grid. The figure shows the Grid jobs submitted to three execution sites, the Condor cluster at University of Wisconsin- Madison (Red line), consisting of single and dual processor machines running Linux (Red Hat 9). Approximately 200 jobs can start simultaneously on the cluster. The blue line indicates the Grid jobs submitted to the CCIN2P3 in Lyon, France running Linux (Red Hat 7.2). The batch queuing system is BQS and the cluster can execute 130 jobs concurrently in the queue to which the Grid

jobs can be submitted. The green line indicates the amount of Grid jobs submitted to the 30 (Dual processor) machines PBS cluster at the Manchester University, UK.

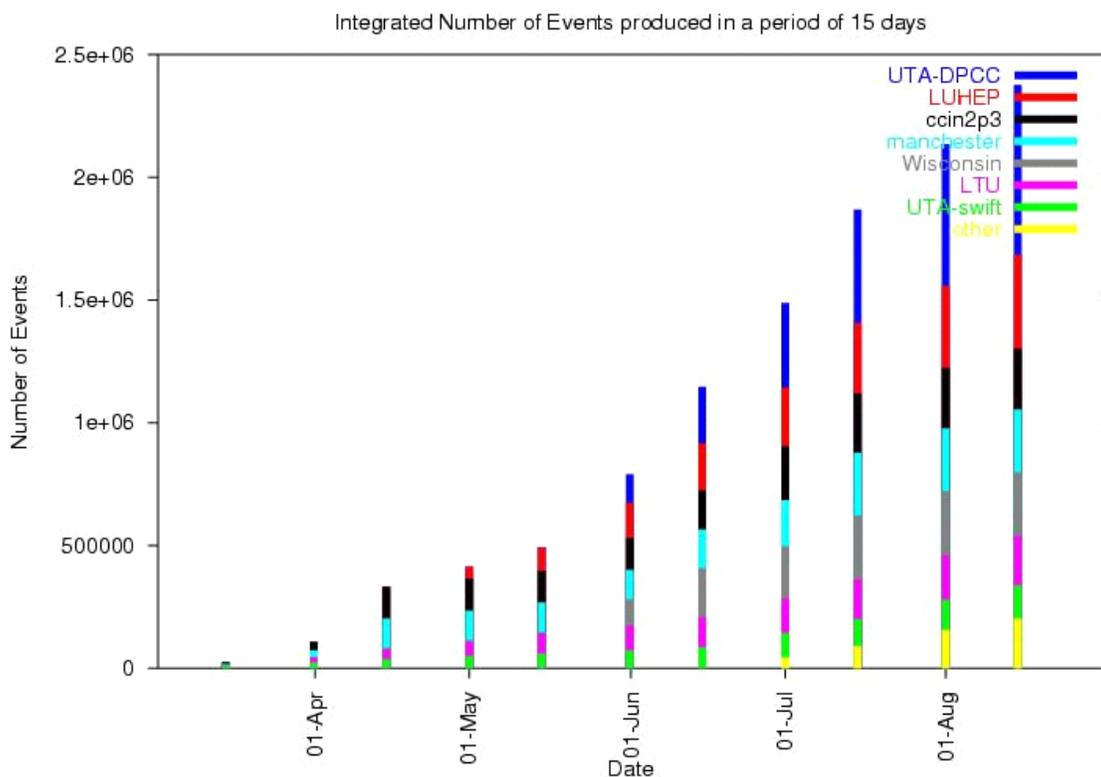


Figure 6.4 Number of physics events per execution site from March

A typical production Grid job, translates into multiple cluster jobs depending on the number of events requested and the site configuration. For example a 50,000 event Grid job is translated into 200 local jobs queued in the batch system if the site configuration parameter in 250 events per local job.

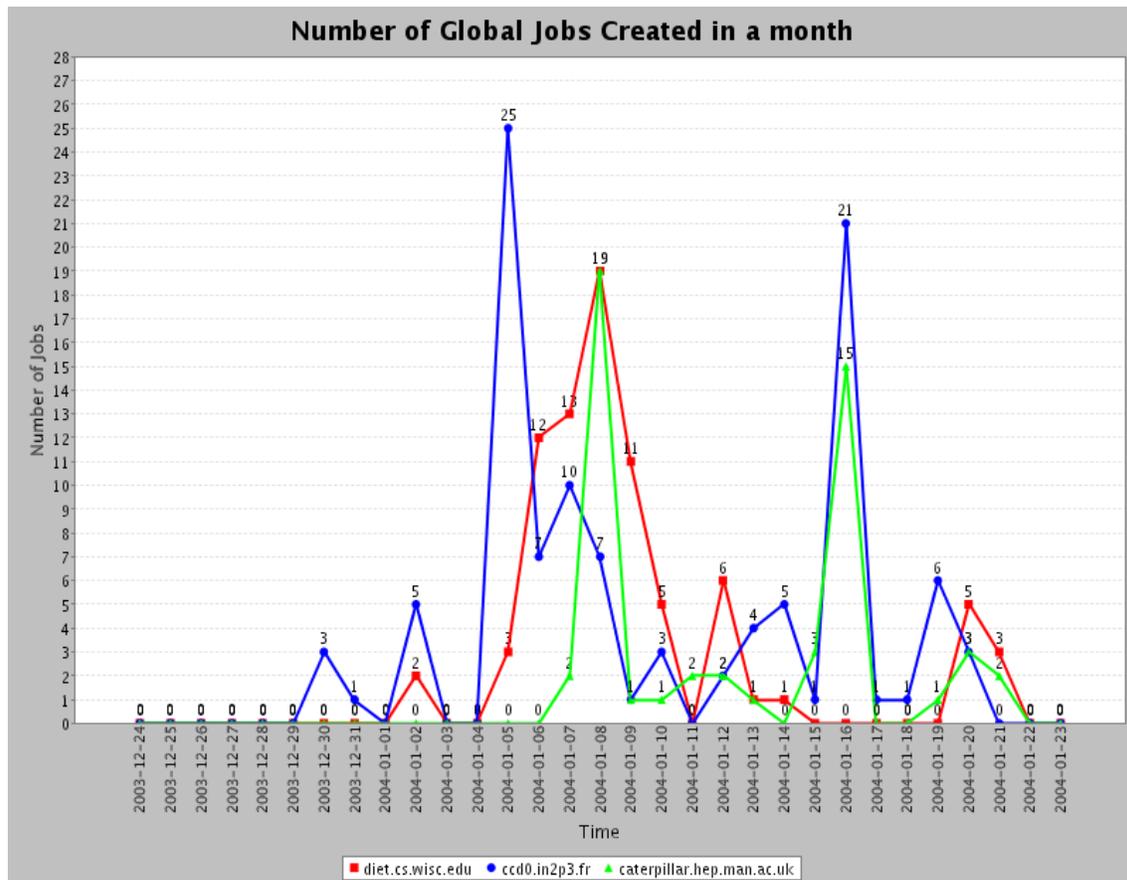


Figure 6.5 Number of Global (Grid) jobs created in a month.

CHAPTER VI

CONCLUSIONS

7.1 Conclusions

Grid computing is an active area of research and development. The number of academic Grids has jumped six-fold to 500 through the year 2004 [82]. The “customer base” of scientists, who are often also application writers, drive Grid developers to produce tangible solutions. History indicates that technologies have always been driven by need of applications which have a large user base. In 1998 Tim Berners-Lee’s need to communicate his own work and the work of other physicists at CERN led him to develop HTML, HTTP, and a simple browser, which form the crucial elements of today’s World Wide Web [9].

SAM-Grid has been developed to satisfy the needs to Run-II experiments at Fermilab. The DØ collaboration has been extensively using the SAM-Grid infrastructure for its computing needs, and the CDF collaboration has started using the Grid framework for Monte Carlo production. The CMS [48] experiment has expressed interest [45] in some of the services that the Grid-Fabric interface offers. A solution to the problem of interfacing the fabric to the Grid middleware as described in Chapter 4 has been presented in Chapter 5 by

- 1) Implementing extensible application specific Jobmanager, thus providing due consideration to the application imposed complexities and leveraging the use of the SAM data handling system.
- 2) Providing a sandboxing service for dynamic installation of execution environment at the worker nodes.
- 3) Interfacing the Grid fabric with existing Grid middleware like Globus® and Condor.
- 4) Handling the inefficiencies of the local resource managers, such as batch systems.
- 5) Providing a virtual service, Batch Adapters, to interface different batch systems to the Grid middleware via simple API's.

The efficiency measurements presented in the last chapter, as well as the SAM-Grid team's experience in deploying Grid technologies suggest that the major cause of inefficiencies in a practical Grid of heterogeneous resources are the issues related to the configuration of the fabric. The greatest asset that SAM-Grid has is the SAM data handling system. Most of the contemporary Grid's lack a sophisticated data handling system.

Fabric elements like computing clusters and the services provided by the Grid-Fabric interface are prone to misconfigurations. Knowledge transfer between Grid software developers and site administrators is crucial. There is a more human aspect to a working Grid than the software aspect. If the site administrators migrate to a new version of the operating system or create directories in disk areas used by the data handling system, this typically causes application failing. Sophisticated site

configuration tools and more Grid ware fabric elements can make the Grid function in a better way from the user's perspective.

7.2 Future research

The Grid software development is getting oriented towards services architecture, the open Grid services architectural (OGSA [30] [31]) effort is working to take Grid technologies and integrate them into a Web services [32] framework. Efforts are underway in various standardization bodies, like Global Grid Forum (GGF) [83] to document "best practices", implementation guidelines and standards for Grid technologies. As Grid technologies mature, so will the Grid fabric. SAM-Grid is now concentrating on data reprocessing and analysis applications for the DØ collaboration. Significant challenges lie in utilizing the fabric for data reprocessing as the size of input sandboxes increase. Fault tolerance in the job management system is of prime importance, when thousands of local jobs are launched simultaneously. Fabric misconfigurations have been a major source of inefficiency, as sophisticated Grid aware fabric make inroads, some of the services in the middleware and the Grid-Fabric interface, which compensate for the fabric deficiencies can be removed, at the same time new sophisticated services like advance resource reservation can be added. The fabric management tools developed by the EU DataGrid project are steps in the right direction [28]. It should be noted that currently there is no such thing as "THE GRID" "THE GRID" is a vision, more likely to be a conglomeration of interoperable Grid.

REFERENCES

- [1] I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a new Computing Infrastructure*, (Morgan Kaufmann Publishers, 1998) ISBN 1-55860-475-8.
- [2] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal of Supercomputer Applications and High Performance Computing, January 2001.
- [3] 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.
- [4] I. Terekhov, A. Baranovski, G. Garzoglio, A. Kreymer, L. Lueking, S. Stonjek, F. Wuerthwein, A. Roy, T. Tannenbaum, P. Mhashilkar, V. Murthi, R. Walker, F. Ratnikov, T. Rockwell, *Grid Job and Information Management for the FNAL Run II Experiments*, in Proceedings of Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003, La Jolla, California, March 2003.
- [5] R. Walker, A. Baranovski, G. Garzoglio, L. Lueking, D. Skow, I. Terekhov, *SAM-GRID: A System Utilizing Grid Middleware and SAM to Enable Full Function Grid Computing*, in Proceedings of the 8th International Conference on B-Physics at Hadron Machines (Beauty 02), Santiago de Compostela, Spain, Jun. 2002
- [6] G. Garzoglio, I. Terekhov, J. Snow, A. Nishandar, S. Jain, *Experience producing simulated events for the DZero experiment on the SAM-Grid*, presented at Computing in High Energy and Nuclear Physics (CHEP 2004), Interlaken, Switzerland, Sep 2004.
- [7] M. Litzkow, M. Livny, and M. Mutka, *Condor – A Hunter of Idle Workstations*. Proc. 8th Intl Conf. on Distributed Computing Systems, 1988, pp. 104-111.
- [8] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, *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.

- [9] Tim Berners-Lee and R. Cailliau. “WorldWideWeb: Proposal for a HyperText project”, 1990 (<http://www.w3.org/Proposal.html>).
- [10] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke., *A Resource Management Architecture for Metacomputing Systems*, in Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.
- [11] Dan Farber, *Ten predictions to shake your world*, ZDnet, October 10, 2002.
- [12] Mark Baker, Rajkumar Buyya and Domenico Laforenza, *Grids and Grid technologies for wide-area distributed computing*, in Software – Practice and Experience 2002.
- [13] Zsolt N’emeth and Vaidy Sunderam, *Characterizing Grids: Attributes, Definitions, and Formalisms*, in Journal of Grid Computing 2003, pg 9-23.
- [14] J. M. Schopf, B. Nitzberg, *Grids: Top ten questions*, Scientific Programming, special issue on Grid Computing, Vol. 10, No. 2, pg. 103-11.
- [15] E Gallopoulos, EN Houstis, and JR Rice, *Computer as Thinker/Doer: Problem-Solving Environments for Computational Science*, IEEE Computational Science and Engineering, Vol 1, No 2, pages 11-23, Summer 1994.
- [16] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, *A Security Architecture for Computational Grids Proc. 5th ACM Conference on Computer and Communications Security Conference*, pp. 83-92, 1998.
- [17] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke, *GASS: A Data Movement and Access Service for Wide Area Computing Systems*, Sixth Workshop on I/O in Parallel and Distributed Systems, May 5, 1999.
- [18] Tobias Groothuyse, *Information and Monitoring Systems for Grids: Divide or Unify?*, B.Sc thesis, Vrije Universiteit Amsterdam.
- [19] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, *A Directory Service for Configuring High-Performance Distributed Computations*, Proc. 6th IEEE Symposium on High-Performance Distributed Computing, pp. 365-375, 1997.

- [20] G. Garzoglio, I. Terekhov, A. Baranovski, S. Veseli, L. Lueking, P. Mhashilkar, V. Murthi, *The SAM-Grid Fabric services*, talk at the IX International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT-03), Tsukuba, Japan, Dec 2003.
- [21] Baranovski, A., Bertram, I., Garzoglio, G., Lueking, L., Terekhov, I., Veseli, S., Walker, R., *SAM-Grid: Using SAM and Grid middleware to enable full function Grid Computing*.
- [22] Carpenter, L., Lueking, L., Moore, C., Pordes, R., Trumbo, J., Veseli, S., Terekhov, I., Vranicar, M., White, S., White, V. *SAM and the Particle Physics Data Grid*, White paper (<http://www.ppdg.net/docs/WhitePapers/SAMandPPDG.pdf>).
- [23] Paul D. Coddington Lici Lu Darren Webb Andrew L. Wendelborn, *Extensible Job Managers for Grid Computing*, 26th Australasian Computer Science Conference (ACSC2003), Adelaide, Australia.
- [24] J. Fromm, K. Genser, T. Levshina, I. Mandrichenko, *FBSNG- Batch System for Farm Architecture*, CHEP 2001, Sep 5th, Beijing, China.
- [25] Jean-Yves Girard, *How to manage executables and input and output data when submitting a Grid job*, IBM developer works article.
- [26] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*, IEEE Mass Storage Conference, 2001.
- [27] www.python.org.
- [28] A. Silverman, et al., *Towards automation of computing fabrics using tools from the fabric management workpackage of the EU DataGrid project*, in Proceedings of Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003, La Jolla, California, March 2003.
- [29] Ian Foster, "The Grid: A New Infrastructure for 21st Century Science (Reprint)", *Physics Today*, Vol. 55 #2, p. 42, 2002.

- [30] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration" Open Grid Service Infrastructure Working Group, Global Grid Forum, June 22, 2002.
- [31] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, "Grid Services for Distributed System Integration", IEEE Computer, June 2002.
- [32] David De Roure, Nicholas Jennings, Nigel Shadbolt and Mark Baker, "Research Agenda for the Semantic Grid: A Future e-Science Infrastructure", Technical Paper, 19th February 2002.
- [33] Madhu Chetty and Rajkumar Buyya, "Weaving Electrical and Computational Grids: How Analogous Are They?" Technical Report, Monash University, Nov. 2001.
- [34] Rajkumar Buyya, Steve Chapin, David DiNucci, "Architectural Models for Resource Management in the Grid", Grid 2000, Bangalore, India.
- [35] Klaus Krauter, Rajkumar Buyya, and Muthucumar Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems", Technical Report: Manitoba and Monash Universities, 2000.
- [36] Foster and K. Kesselman, "The Globus Project: a status report", IPPS/SPDP'98 Heterogeneous Computing Workshop S.4-18 (1998).
- [37] A.S. Grimshaw, W.A. Wulf, J.C. French, A.C. Weaver and P.F. Reynolds Jr., The Legion Vision of a Worldwide Virtual Computer, CACM 40 (1997)
- [38] K. Czajkowski et al., *A resource management architecture for meta-computing systems*, Technical Report, MCS D (Argonne National Lab, 1997) .
- [39] A.S. Grimshaw and W.A. Wulf, *Legion - a view from 20,000 feet*, Proc. 5th IEEE International Symposium on HPDC, IEEE Computer Society Press.
- [40] A.S. Grimshaw, W.A. Wulf, J.C. French, A.C. Weaver and P.F. Reynolds Jr., *A Synopsis of the Legion Project*, Technical Report CS-94-20 (University of Virginia, 1994)
- [41] A.S. Grimshaw, W.A. Wulf, J.C. French, A.C. Weaver and P.F. Reynolds, *Legion: the next Logical Step toward a Nationwide Virtual Computer*, Technical Report CS-94-21 (University of Virginia, 1994)

- [42] A. Nishandar, D. Levine, I. Terekhov, G. Garzoglio, S. Jain, *Black Hole Effect: Detection and Mitigation of Application Failures due to Incompatible Execution Environment in Computational Grids*, submitted in Cluster Computing and Grid 2005, Cardiff, UK.
- [43] UK Particle Physics and Astronomy Research Council (PPARC), Media Release-2nd April 2004.
- [44] Globus Project: *GridFTP: Universal data transfer for the Grid*, White paper.
- [45] Personal communication with Igor Terekhov.
- [46] DØ experiment: <http://www-d0.fnal.gov/>
- [47] CDF experiment: <http://www-cdf.fnal.gov>
- [48] US CMS: <http://www.uscms.org>
- [49] US ATLAS: <http://www.usatlas.bnl.gov>
- [50] FNAL Grid computing page: <http://Grid.fnal.gov>
- [51] SAM-Grid project: <http://www-d0.fnal.gov/computing/Grid>
- [52] CERN Home page: www.cern.ch
- [53] Kerberos security: <http://web.mit.edu/kerberos/www>
- [54] NSF Middleware Initiative (NMI): <http://www.nsf-middleware.org>
- [55] Globus toolkit: <http://www-unix.globus.org/toolkit>
- [56] Information Sciences Institute: www.isi.edu
- [57] Argonne National Laboratory: <http://www.anl.gov>
- [58] Moore's Law: <http://www.intel.com/research/silicon/mooreslaw.htm>
- [59] PARC: <http://www.parc.xerox.com>

- [60] MINOS: <http://www-numi.fnal.gov>
- [61] PHP: <http://www.php.net>
- [62] NorduGrid: <http://www.norduGrid.org>
- [63] Fermilab Mass Storage System (MSS): <http://www-isd.fnal.gov/enstore>
- [64] BQS batch system: <http://webcc.in2p3.fr/man/bqs/intro>
- [65] CORBA: <http://www.corba.org>
- [66] Gartner research: <http://www4.gartner.com>
- [67] LHC: <http://lhc-new-homepage.web.cern.ch/lhc-new-homepage>
- [68] PBS: <http://www.openpbs.com>
- [69] R.W. Watson and R.A. Coyne. *The parallel I/O architecture of the high performance storage system (HPSS)*, IEEE MSS Symposium, 1995.
- [70] Xindice: <http://xml.apache.org/xindice>
- [71] SAM projects: http://d0db.fnal.gov/sam/doc/userdocs/creating_sam_projects.html
- [72] RSL: http://www.globus.org/gram/gram_rsl_parameters.html
- [73] Interactions.org: www.interactions.org
- [74] Fermilab FAQ: <http://www.fnal.gov/pub/about/faqs>
- [75] Particle Physics Data Grid: <http://www.ppdg.net>
- [76] Grid PP: <http://www.Gridpp.ac.uk>
- [77] <http://www.kernelthread.com/publications/security/sandboxing.html>
- [78] Fermi National Accelerator Laboratory: <http://www.fnal.gov>

- [79] Dan Farber, *Ten predictions to shake your world* , ZDnet October 10, 2002
- [80] Buyya – FAQ: <http://Gridcomputing.com/Gridfaq.html>
- [81] Jonathan Ledlie, Jeff Shneidman, Margo Seltzer, John Huth, *Scooped, Again*.
- [82] ntpd: <http://ntp.isc.org/bin/view/Main/DocumentationIndex>
- [83] G. Garzoglio, *The SAM-Grid and the use of Condor-G as a Grid job management middleware*, presentation at the Condor-Paradyn week, April 2004, Madison-WI.
- [84] SAM: <http://d0db-prd.fnal.gov/sam/>
- [85] McFarm: <http://heppc12.uta.edu/~thomas/hepweb/mcdocumentation.html>
- [86] RFC 1094, *NFS: Network File System Protocol Specification*.
- [87] SAM-Grid web: <http://samgrid.fnal.gov:8080>

BIOGRAPHICAL INFORMATION

Aditya Nishandar joined *The University of Texas at Arlington* in the fall of 2002. He received his bachelor's degree in Computer Engineering from *Pune Institute of Computer Technology* affiliated to *University of Pune*, India. He worked at Fermi National Accelerator Laboratory in Batavia, IL as a software developer on the SAM-Grid project from August 2003 to August 2004. His academic interests include Grid computing, cluster computing and mobile computing.