Integration of the SAM-Grid Infrastructure to the DØ Data Reprocessing Effort

by

ANOOP RAJENDRA

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 2005

ACKNOWLEDGEMENTS

No written work is ever complete without the mention of the people who made it possible.

I would like to thank my supervising professor Mr. David Levine for his constant encouragement, support, and always pointing me in the right direction with respect to the research. I extend my sincerest gratitude to Mr. Gabriele Garzoglio, and Mr. Andreii Baranovski of the SAM-Grid team for patiently answering every question that I threw at them, for over an year. I would also like to thank Dr. Jaehoon Yu, who has provided valuable insight into the work.

The entire DØ team deserves special mention. This project would not be what it is today without their invaluable feedback.

Finally, my sincerest gratitude to Papa, Amma and Deepthi, who believe in me more than I believe in myself, and without whose constant and unwavering support, this would have been an unbearable and even impossible endeavor.

August 4 2005

ABSTRACT

Integration of the SAM-Grid Infrastructure to
the DØ Data Reprocessing Effort

Publication No. ______

Anoop Rajendra, M.S.
The University of Texas at Arlington, 2005

Supervising Professor: Mr. David Levine

The DØ experiment is one of the two high energy physics experiments currently being conducted at Fermi National Accelerator Laboratory, in Batavia, IL, on what is currently the world's highest energy particle accelerator, the Tevatron. The experiment produces vast amounts of raw data of the order of several hundred terabytes. This data needs to be converted from the raw format that comes from the detector, ie. digitized data, to a format that is close to the physics, ie. data that can be subjected to analysis. This process, called reconstruction, is done according to constantly evolving and improving reconstruction algorithms. This process of conversion from raw detector data to analyzable data is done periodically to obtain data of very high quality, and it is done over the entire existing dataset to obtain consistent results. The scale of computation that is necessary to process such a large dataset is enormous and presents a challenge. This thesis discusses the computation problem that needs to be solved to achieve such a high level of scalability, the design of a computational model that will solve this problem, the issues that were encountered during implementation of the design, and steps taken to resolve these issues.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Collaborate and Compute. This is the new *mantra* in the computing world, with the advent of a paradigm called ***Grid Computing***.

1.1 Grid Computing - Definition

Grid computing is a distributed computing paradigm that will enable users to access a vast pool of computing resources distributed across geographical and administrative boundaries.

Grid computing has been defined as a "distributed computing paradigm which makes it possible for scientific collaborations and other institutions to share their resources at an unprecedented scale and for geographically distributed groups to work together in ways that were previously impossible."[10]

Computational grids are implementations of the grid computing paradigm. Consider computing resources distributed between different administrative domains. These domains may be universities, research centers, and private institutions, even individual departments, in one of these institutions. Each of these domains may include computing resources such as computing clusters, storage systems, and high speed networks. The software interfaces that allow access to this pool of resources, are what make up the infrastructure of computational grids. The implementation of these interfaces, to build a distributed processing and data handling infrastructure, constitutes the development of the computing grid.

Another view of a computing grid is in the context of computing hierarchy. As shown in Figure 1.1 we can see that a collecting of computing nodes with glue software and network backbone form a compute cluster. A collection of computing clusters, again with glue software and network backbone, forms the basis of computing grid.

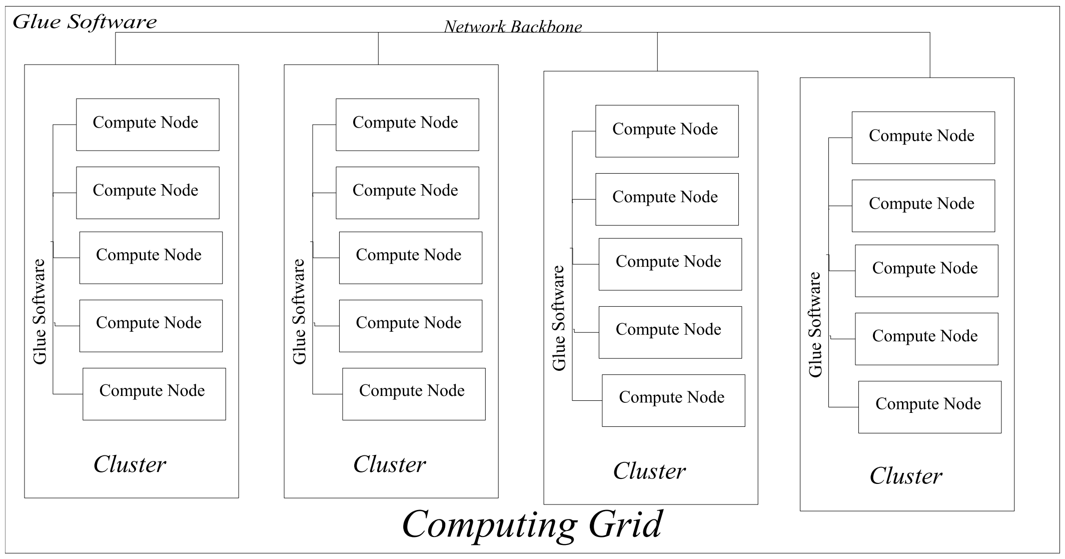

Figure 1.1. Computing Hierarchy.

1.1.1 Virtual Organisations

One of the most important features of the grid computing paradigm is the existence of collaborations between different organisations to solve large scale computation problems. For example, the High-Energy Physics experiment that this thesis discusses involves the participation of about 65 research institutions from across the world. These institutions are interested in the outcome of the experiment. The institutions collaborate to form an administrative unit called a Virtual Organisation(VO)[9]. Since all the members of the VO are interested in the results of the computation, they provide their computing resources to satisfy the computing needs of the VO.

There is usually a considerable overlap between different VOs. An example can be found in the High Energy Physics community. The institutions that are a part of the DØ experiment are a part of the VO called the Particle Physics Data Grid. Some of the member of this VO are based in Europe and are also a part of another virtual organisation called the European Data Grid. Different VOs set different goals for themselves, and the member organisations provide resources to achieve these goals. Since the members themselves may belong to multiple VOs, their resources are shared between these VOs. Another important feature of the Virtual Organisations is that membership to these VOs is not permanent. If the interests, goals and priorities of a member institution change, the member is free to withdraw from the VO, and the VO will not have access to the computing resources of the concerned member. Modern grid computing infrastructures provide extensive software support to Virtual Organisations with frameworks such as VOMS[1][22], that make management of Virtual Organisations very easy.

1.2 Grid Computing - Motivations

Two of the main reasons for the existence of grids today are physical limitations and economic necessities.

Physical limitations exist at the level of the computer chip. Gordon.E.Moore predicted that the number of transistors on a chip would double roughly every 24 months[19], which occurs mainly due to reduction in size of the transistor. However, we will eventually reach a critical density of transistors on a chip beyond which there can be no more increase in the number of transistors. Any increase in this density results in increased heat dissipation and power consumption[21]. Once this level is reached, one solution to increase computing power would be to parallelize execution of tasks on multiple chips.

This may be done by technologies such as multiple-core processors and multi-processor systems on a very low level where parallelism is achieved in the hardware layer. Examples of this are SMP (Symmetric Multiprocessor) Systems, and more recently, dual-core processor chips. On SMP systems, a single compute node has multiple processors that distribute processing between them. The distribution is usually done by the operating system, but specialized hardware is available to parallelize execution of tasks. One prerequisite of this is that the software must be able to take advantage of this feature, by being multi-threaded. If the computation that is to be run on an SMP system consists of a single thread of execution, then execution cannot be parallelized. The requirements on software are essentially the same for multiple-core systems.

Parallelism can also be achieved on a software layer. Examples of this include compute clusters and computing grids. The software framework that exists today to achieve this parallelism is varied. Some tools that achieve this parallelism are -

- **PVM** - PVM or Parallel Virtual Machine interface provides a virtual machine interface to a collection of computing elements. Here, the software that the users would like to run on the cluster are specially implemented using the PVM libraries. The PVM libraries provide mechanisms that allow the developers to develop programs that are inherently multi-threaded.
- **MPI** - MPI or Message Passing Interface is a library suite, which enables developers to develop programs that are multi-threaded in nature, where the inter-thread communication is achieved by message passing.
- **Batch Systems** - Batch Systems are tools that are deployed on clusters to enable parallel execution of jobs. The main difference between batch systems and libraries such as PVM and MPI, is that with the libraries, a single program is made multi-threaded, where each thread may be executed on a

different machine, whereas batch systems do not have a threaded view of software.

The economics of the grid are more evident. The concept of virtual organizations has been explained previously. The reasons for formation of such VO's can be extended to include economic reasons. When large scale computation problems, such as the one that this thesis discusses, are presented, it is usually prudent for the members of the VO to pool their computational resources to solve the problem. If the computation is distributed among the entire pool of resources, the time it would take for the computation to finish, is significantly lower than it would take to finish on the computing resources of one organization. It logically follows from this that by investing a fraction of the total cost of the grid, and by agreeing to contribute its computing resources to the grid, every member of the VO has now access to the entire pool of resources provided by the grid.

1.3 Computing Grids and the High Energy Physics Community

Currently the DØ and the CDF experiments at Fermi National Accelerator Laboratory use grid computing technologies extensively to solve their computational problems. Some of the other institutions involved in High Energy Physics, that are interested in grid development and deployment are Argonne National Laboratory, Brookhaven National Laboratory, University of Texas at Arlington, CERN (LHC Computing Grid)[6], University of Wisconsin, and a few others. These institutions are a part of the Particle Physics Data Grid VO.

1.4 The DØ Experiment

The DØ experiment[3] is one of the two high energy physics experiments currently being conducted at Fermilab. The experiment is being conducted on what is currently the highest-energy particle accelerator in the world. This particle accelerator is a synchrotron. It is capable of accelerating protons and anti-protons to energies of up to 1 TeV (tera electron volt)[1] each. The Tevatron is a proton-anti-proton collider[7]. The accelerator works by accelerating protons and anti-protons in the circular Tevatron in opposite directions, and getting them to collide. The DØ detector is built around the point of collision to detect the outcome of the collisions. Each of the collisions basically results in the break up of the protons and anti-protons into other particles. These particles then pass through the detector, which records the signature of these particles. The digital output of this detection denotes an event. Each of these events is processed through many levels of filters to weed out uninteresting events, and the remaining events are stored onto tapes.

1.5 DØ Computation

The Computation for the DØ experiment[4] involves mainly three categories.

- Monte-Carlo Simulations of the physics events in the DØ Detector.
- Conversion of raw detector data to a format that is ready for analysis. This stage is also called reconstruction or data processing.
- Analysis of the processed data.

In the first category of computation, ie. Monte Carlo Simulations, the detector is modelled in software and algorithms are used to analyze the behavior of the detector under different conditions and when different particles pass through it. These algorithms then process extremely large datasets of randomly selected events, so

[1]1 Tev = 10^{12} eV - 1 electon volt is energy gained/lost by an electron when it is accelerated by an electric potential of 1 volt.

that the performance of the algorithms and their correctness can be assessed. This iterative process results in algorithms that are much more refined than the initial ones. The new algorithms are also used to process randomly selected events and are subjected to further refinements. This results in a constantly evolving and improving algorithms which gives data of a much higher quality.

Once the algorithms for processing events have reached a satisfactorily mature stage, all the events that have been stored as detector data are subjected to another round of computation ie. reprocessing. The entire dataset is reprocessed so that we may obtain high quality of output. It is the reprocessing stage of computation, that this thesis addresses.

1.6 Goal of the Thesis

There exists a grid computing infrastructure at Fermilab today called the SAM-Grid infrastructure. This is a distributed computing framework currently capable of running DØ Monte-Carlo Simulations. A review of the SAM-Grid infrastructure is presented in the section titled “A Brief Review of the SAM-Grid Infrastructure” later in the thesis.

The DØ experiment currently has 250 TB of data that need to be reprocessed. This dataset must be made ready for analysis. Processing 250 terabytes of data requires a massive scale of computational resources that must be easily yet securely accessible in a transparent and consistent manner. The *computation* problem that must be solved to achieve this level of scalability is where computational grids come into the picture.

This effort, to use the SAM-Grid infrastructure to support data reprocessing over the grid, is the DØ Data Reprocessing Effort.

The goal of this thesis is to adapt the existing SAM-Grid infrastructure to support DØ Data Reprocessing Jobs, and deal with the issues of scalability that occur in such an implementation.

The thesis is ordered as follows. A brief review of SAM-Grid is given, followed by a description of the implementation of the Job and Information Management framework, adaptation of this framework to support data reprocessing, issues that have arisen during this implementation, steps taken to circumvent these issues, and finally measurements of system scalability and performance.

CHAPTER 2

A BRIEF REVIEW OF THE SAM-GRID INFRASTRUCTURE

There exists a grid computing infrastructure at Fermilab today called the SAM-Grid infrastructure. This is a distributed computing infrastructure developed at the Fermi National Accelerator Laboratory to handle the data processing needs of the DØ experiment. Originally, SAM-Grid was developed to support DØ Monte-Carlo simulations over the grid. This work has been presented in [12].

The purpose of the SAM-Grid infrastructure is data processing and data handling in a distributed fashion. To achieve this, the infrastructure is split into two parts.

- SAM, or Sequential Access to data via Meta-data[17, 18], is a distributed data management system, which manages the entire volume of data that is generated by the detector, as well as data that is processed and analyzed.
- JIM, or Job and Information Management system[11], is a distributed framework that manages execution of jobs over the grid and also provides monitoring information for these jobs.

2.1 SAM

The DØ experiment is, from a computing perspective, a data intensive experiment. This requires the use of a large scale data management system. The SAM system was designed at Fermilab to address these needs. The data handling system is designed as a

The system has four main functions.

- Store the raw detector data.

- Maintain a catalogue of all the data that is present in the system.
- Deliver raw data to requesting processes.
- Store processed data that are delivered by remote processes.

An overview of the SAM system is shown in Figure 2.1[17].

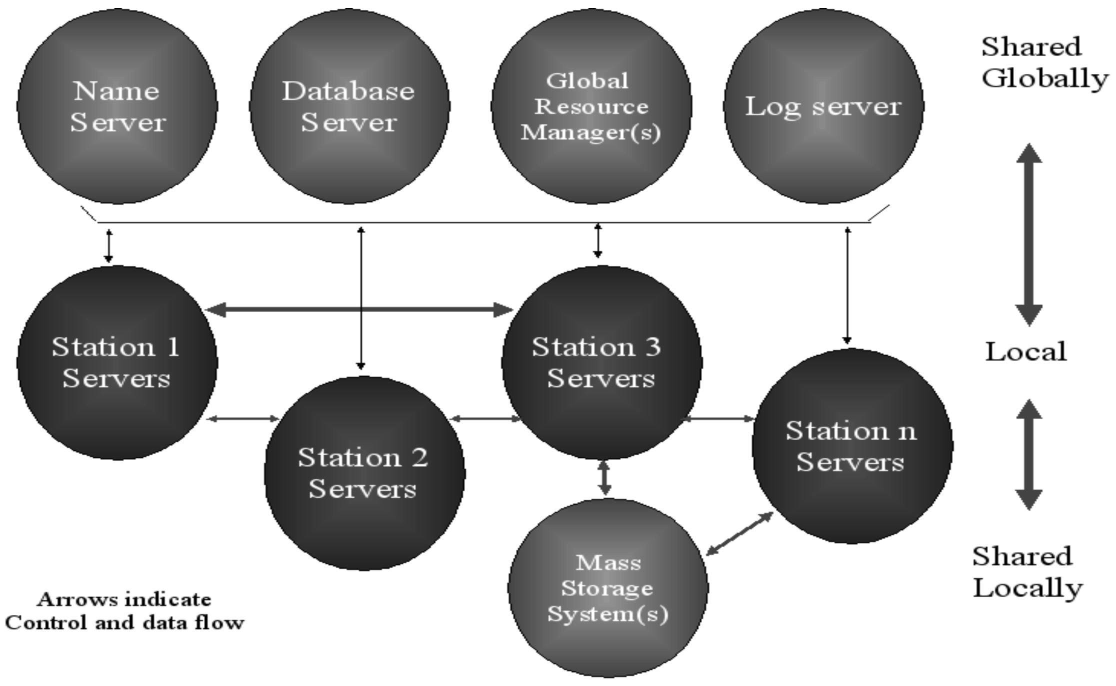

Figure 2.1. Overview of the SAM System.

The components of the SAM system come under two categories.

- Global Data Handling Services.
- Local Data Handling Services.

These components are described below. There is also a brief description about how these components interact with each other.

- **Global Data Handling Services** - These are services that are common to all the components of the system, and are accessible to all requesting clients, through software interfaces.

   1. *Database Server* - This maintains the entire meta-data catalogue necessary to locate any data in the entire SAM system. Every file in the SAM system has its properties described by meta-data. This meta-data is maintained in the database. The meta-data describes the properties of the files stored, such as size, run number, data type, parent files, date of storage, etc.
   2. *Naming Service* - This service allows all the components of the system to locate each other by name. This is done by CORBA tools, where each resource in the SAM system, such as components of the local data handling services and storage areas, is assigned a name. The service maintains a list of all the assigned names, and when a client requests to connect to a particular resource, the naming server resolves the name and sends it to the client.
   3. *Log Server* - This global service provides logging services to the SAM system. Every interaction between components of the SAM system, such as requests, start of processes, etc are logged with the SAM system using this service.
   4. *Resource Server* - This service basically manages resources that are available at the Mass Storage System and other storage locations. One of its primary functions is to multiplex store and retrieve requests so as to make optimal and efficient use of the system resources.

- **Local Data Handling Services** - These are services that are local to execution sites, and the resources they have control over are accessible to other components only through the interfaces provided by these services.
   1. *SAM Station* - The SAM station[17] is the primary entry point for all data on an execution site. The identity of an execution site is the name that the station is registered under, with the naming service. All

processes that are connected with the storage and retrieval of data are registered as processes running under the SAM station entity. The primary responsibility of the SAM station is to manage data on a disk area on the execution site called the SAM cache, and to manage transfers to the cache. All transfers of data occur between different stations, or between the station and the Mass Storage System. One important aspect of the station is that it is designed to only pull in data from outside. Another function of the station is to manage projects. When clients on an execution site need to access data, they inform the station master running at the site, of the data that they need, by starting a project. The station then arranges for all the data that is associated with the project to be downloaded into the cache. This data may then be given to the requesting clients.

2. *File Storage Server* - As mentioned above, the SAM station is mainly responsible for pulling data into its cache. However, when processed data is to be stored back into the SAM system, a File Storage Server[17] is used. This server's identity is coupled with that of the station. This means on an execution site, there may be a station and a file storage server(FSS) running, that would both be identified by the station name.
3. *SAM Stagers* - The SAM stager[17] is an entity which is responsible for the actual data transfers. The distinction between the station and a stager is that while the station decides where a particular volume of data is to be transfered from, the stager is responsible for transferring the data using lower level grid data transfer tools. Hence, a stager needs to be associated with a particular station, and when a station needs to pull data into its cache, it determines the source of the data,and tells the stager to retrieve data from the source. The same applies to

FSS also. When the FSS needs to store data, it only determines the destination for the data, and then requests the stager to do the transfer. The advantage of such modularization of the local data handling components is that this supports scalability, and distribution of components. For example, if there are multiple disk areas on an execution site that are designated as cache areas, then multiple stagers may be launched to manage transfer files to these disk areas. Part of the stagers may be associated with the station master and the rest, with the FSS.

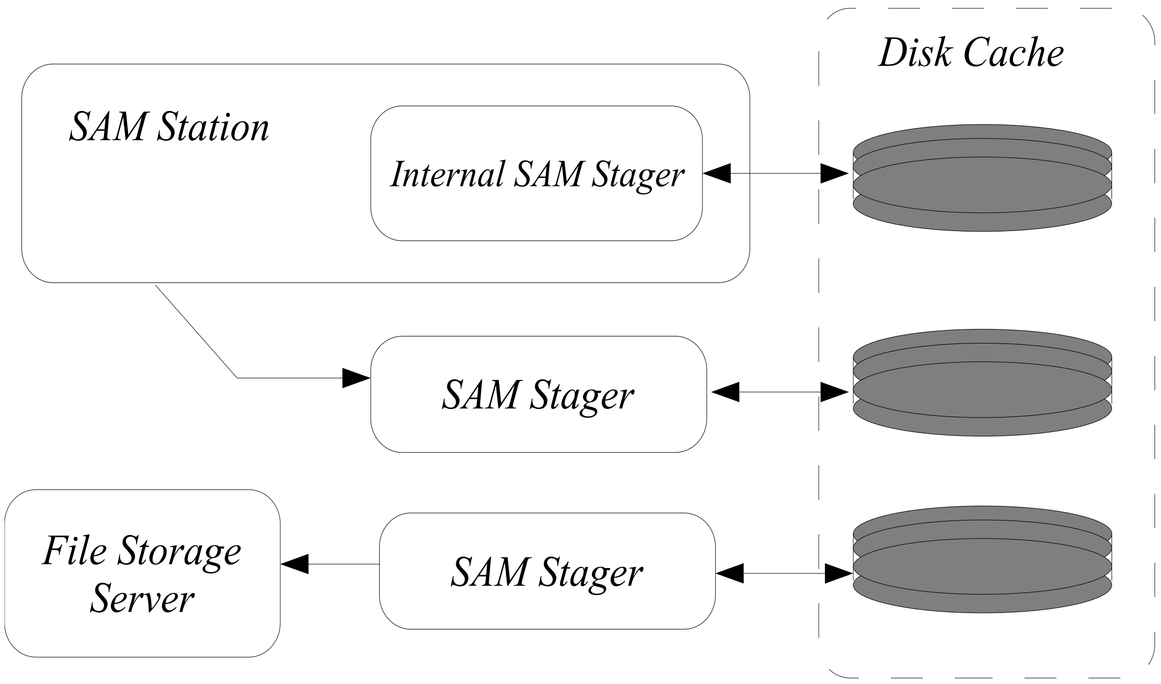

Figure 2.2. Local Data Handling Services.

A brief overview of the local data handling services and their operation is depicted in Figure 2.2.

2.2 JIM

The purpose of the JIM framework is to manage job execution on remote sites, and to collect monitoring information about these jobs.

The JIM framework divides the Grid infrastructure into the following components. These components may be directly mapped onto physical computing resources.

- *Client Sites* - These are sites from which a user may submit jobs to the Grid. These consist of a minimal installation of the framework.
- *Submission Sites* - These are sites where the user jobs are spooled before being sent for execution.
- *Execution Sites* - These are the sites where the submission site sends the jobs to. Execution sites are typically medium scale to large scale clusters, with extensive grid software, batch systems, large disk volumes, and high speed networks.

The JIM framework interfaces with the SAM system for its data handling needs. The operation of the JIM framework is explained in detail in the subsequent chapters.

2.3 Associated Tools and Utilities

The JIM framework makes use of various tools that provide core grid services and low-level job management utilities. Some of these tools are listed below.

2.3.1 Globus®

The Globus® toolkit is a collection of tools that can be used to build computing grids. It includes libraries and services that can be used for resource monitoring, discovery, security, and job and file management. Some of the tools that are present in the toolkit are

- GSI - Grid Security Infrastructure - This infrastructure is based on GSSAPI or Generic Security Services and Application Programming Interface. GSS-API provides for X509 certificate authentication and authorizations, and the mechanism is usually inhibited by a need for user intervention. GSI extends this mechanism by delegating credentials and creating proxy credentials that may be used without the need for user intervention.
- GridFTP - This is the transfer protocol that is commonly used on grid applications. The protocol specifications are an extension of the standard FTP specifications. GridFTP uses GSI as an authentication and authorization mechanism.
- GRAM - Globus® Resource Allocation Manager - This is used to manage and allocate remote resources to requesting processes. It is also used for remote job submissions and management. The GRAM protocol describes a Resource Specification Language that can be used to locate resources, submit jobs to these resources, and monitor jobs running at the resources. This too uses GSI for authentication and authorization.

2.3.2 Condor®

The Condor® high throughput computing tools are a suite of tools that are meant for workload management. They implement fully featured batch systems by providing job queuing mechanisms, scheduling policy, priority schemes, resource monitoring and management, and job management. An important feature of the Condor® system is the use of classified advertisements for advertising availability of resources. These are used extensively during resource matching, ie matching of jobs to resources.

Condor-G is a system that interfaces condor tools to the Globus® toolkit to provide grid level workload management. The principles of job and resource management are the same, but are at a much higher level in the hierarchy of computing.

The Condor® tools and the Globus® toolkit are extensively used in the SAM-Grid infrastructure to provide core grid services. The subsequent chapters explain in more detail the relevant tools from these toolkits.

CHAPTER 3

SAM-GRID AND DØ DATA REPROCESSING

The Job and Information Management system, also called the JIM framework is a distributed computing framework developed at Fermilab for management of job execution and monitoring. This chapter discusses the adaptation of the SAM-Grid infrastructure, specifically the JIM framework, to support the DØ Data Reprocessing effort.

The JIM framework follows a plug-in architecture. This makes adaptation of JIM for new types of computation a matter of writing new plug-ins, and subjecting them to rigorous testing and debugging.

The framework is designed around the Globus® Toolkit[2][15] and the Condor® [3] High-throughput Computing tools. These tools form the base of most grid computing infrastructure. The JIM framework essentially is a wrapper around these tools to provide access to the resources of the grid in a transparent manner.

Figure 3.1[13] shows a static view of the system. The components that form a part of this architecture are described below.

3.1 Components of the JIM Framework

- **Client Site**

  The Client Site is where a user may submit jobs from. This is usually a desktop or a laptop computer. This has a minimum installation of the JIM software and SAM software. The client software does not require the presence of any SAM or JIM services running on the client machine. The client software installed on the client site is essentially a wrapper around

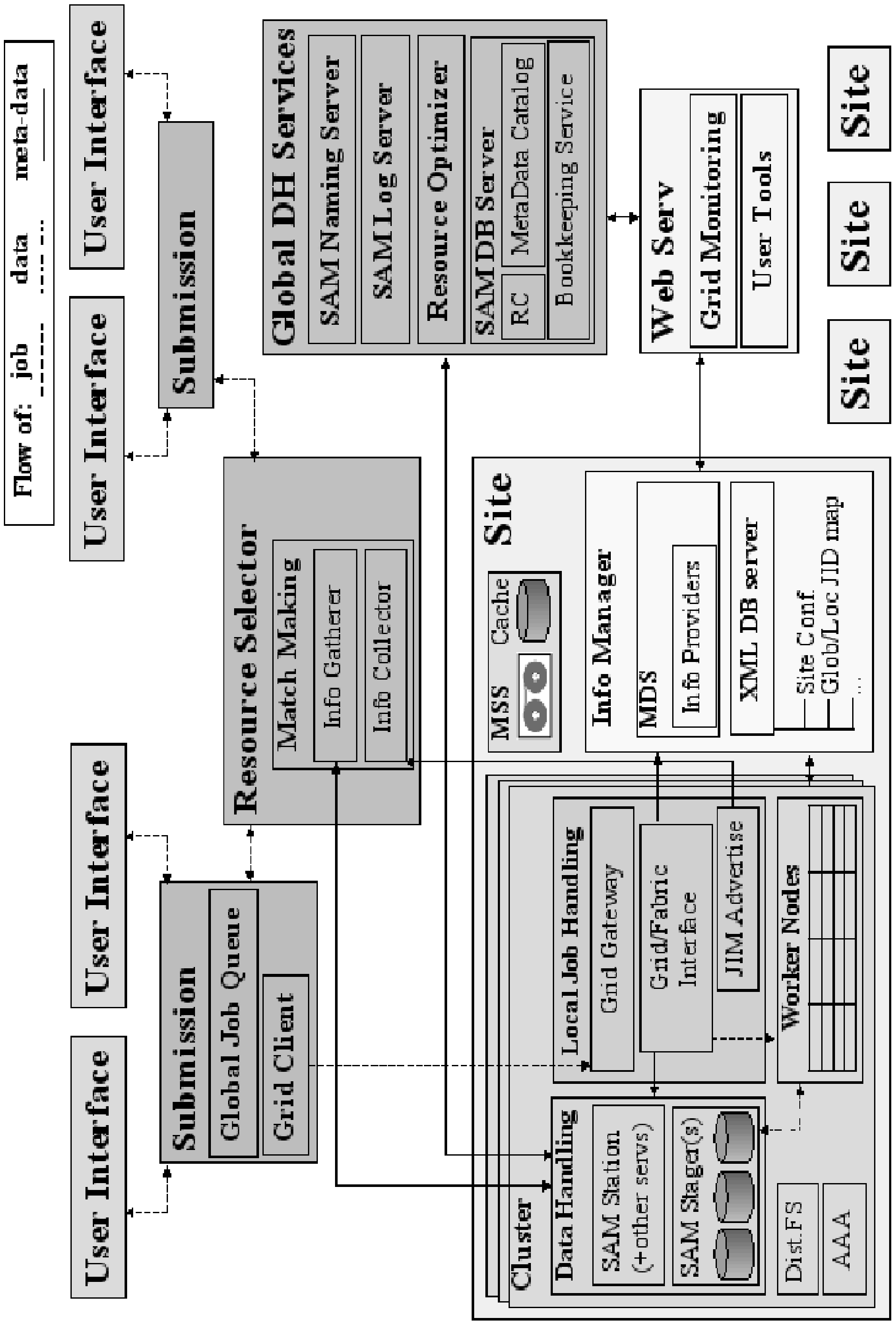

Figure 3.1. Static View of the SAM-Grid Architecture

the Globus® and Condor® job submission tools. This is depicted as "User Interface" in Figure 3.1.

- **Submission Site** The submission site is the component of JIM which is responsible for the actual submission of jobs to a computing resource. The servers on the submission site include the Scheduler daemon, the Negotiator daemon, and a Collector daemon. All these daemons are a part of the Condor® toolkit.
  - *Scheduler Daemon* - The scheduler is an entity which pools all the jobs that are submitted from the client sites and schedules them for execution. The "Job Queue" entity in Figure 3.1 depicts the queue maintained by the scheduler daemon.
  - *Collector Daemon* - The collector is the entity responsible for collecting information about the resources available in the grid. Each of the computing resources is a collection of communication, processing, and storage resources. These resources are advertised at the submission sites using classified advertisements[2]. These advertisements specify parameters such as storage space available on the site, the URL[4] that specifies the entry point of the execution site, the identity of the computing resource as specified for the SAM naming service, and a few other parameters. This is denoted by the "Information Collector" entity in Figure 3.1.
  - *Negotiator Daemon* - The negotiator daemon is the entity responsible for matchmaking of jobs to resources. When jobs are submitted from the client site, the requirements of the job are specified in the job description file that is created on the client site. The resources available at the execution sites is specified in the classified advertisements. This

daemon matches the resources requested by the job to the resources available. This is depicted by the entity named "Matchmaking" in Figure 3.1.

  - *Master Daemon* - There is also a Master Daemon on the submission site. Its principal function is to monitor the other daemons mentioned above. In the event a particular daemon crashes, it is the responsibility of the master daemon to restart the service. Though this service does not provide any core services to the SAM-Grid system, it is mentioned here for the sake of completeness. This entity is not depicted in the static view of the SAM-Grid system.

- **Execution Site**

  The execution site, is the most important site in the chain. This is the site where the actual execution of the job takes place. The execution site is made up of the following components. The components mentioned here augment the SAM components, which are already described in the previous chapters.

  - *Gatekeeper* - This entity is the entry point to the execution site that is presented to the grid. This means that any job that needs to be executed at this site must enter through the Gatekeeper. The "Grid Gateway" entity in the execution site depicts the Gatekeeper in Figure 3.1.
  - *Job Managers* - These are components that control the actual execution of the job. When a request for job execution comes from the grid, the job manager that is responsible for the particular type of job is activated and takes control of the job execution. Its responsibilities are to ensure that the environment required by the job is set up correctly, execute the job, and return the error code along with the output of the job. These are denoted by the "Grid-Fabric Interface" entity in Figure 3.1.

– *Batch System* - This is a component that must be present on every execution site with multiple machines set up as a clustered system. A clustered system has a hierarchical distribution of machines, with the head node at the top of the hierarchy and many worker nodes below it. The batch system essentially manages execution of jobs on the cluster by submitting jobs from the head node to the worker nodes, and then collecting the output and error logs of the jobs.

– *Batch Adapters* - There are many different batch systems in use today. Some of the examples are PBS®[5], Torque®[6], Condor® , SGE®[7], and others. Each of these systems has a different mechanism for job submissions. The job managers will need to interface with the batch systems using mechanisms specific to the batch system being used. Hence an abstraction layer is used to convert a generic job description to a description that the underlying batch system can understand. This abstraction layer is called the batch adapter. This also has the added functionality of emulating an ideal batch system interface. Thus these adapters are also called batch system idealizers. Batch adapters and their operations are described in more detail in [16] and [8]. The batch adapters are a part of the "Grid-fabric Interface" in Figure 3.1.

– *XML Database* - This database is mainly used to store monitoring information. When a job starts executing on a worker node of a cluster, it writes its state information to this XML database. The database is also used to store a lot of configuration options of many of the products installed on the execution sites. This way, the jobs in execution can query the configuration of the installed products.

The above description is a static description of the JIM framework. The operation of the framework and the interaction that occurs between different components of the framework is described next.

3.2 Operation of the JIM Framework

Before the description of the operation of the framework, we must describe the structure of a job that is to be executed.

The DØ Reprocessing Effort requires the processing of close to 250 TB of raw detector data. This data is split into files of much smaller sizes. The size of each file is dependant on the number of events present in the file. Since the number of events per file follows an almost random distribution, taking the average number of events to obtain an average file size will not be useful. The current file sizes range from 200 MB per file to 1.5 GB. These files are then grouped into datasets, whose sizes are limited to a 100 files. These datasets are called input datasets. Every grid job operates on a single input dataset. The DØ executables are also grouped into datasets called binary datasets. These are the executables that process files from the input dataset. Thus a job description includes the version of the binary dataset, the input dataset, and other job requirements such as the execution site to run on, number of events to process from each file of the input dataset, etc. The work flow of a grid job is depicted in Figure 3.2 [14].

3.2.1 Job Submission

As mentioned in the previous section, the client site is where a user submits her jobs from. The job that the user wants to run is described by a submission script. The submission script describes the characteristics and requirements of a job. An example submission script is shown in Figure 3.2.1.

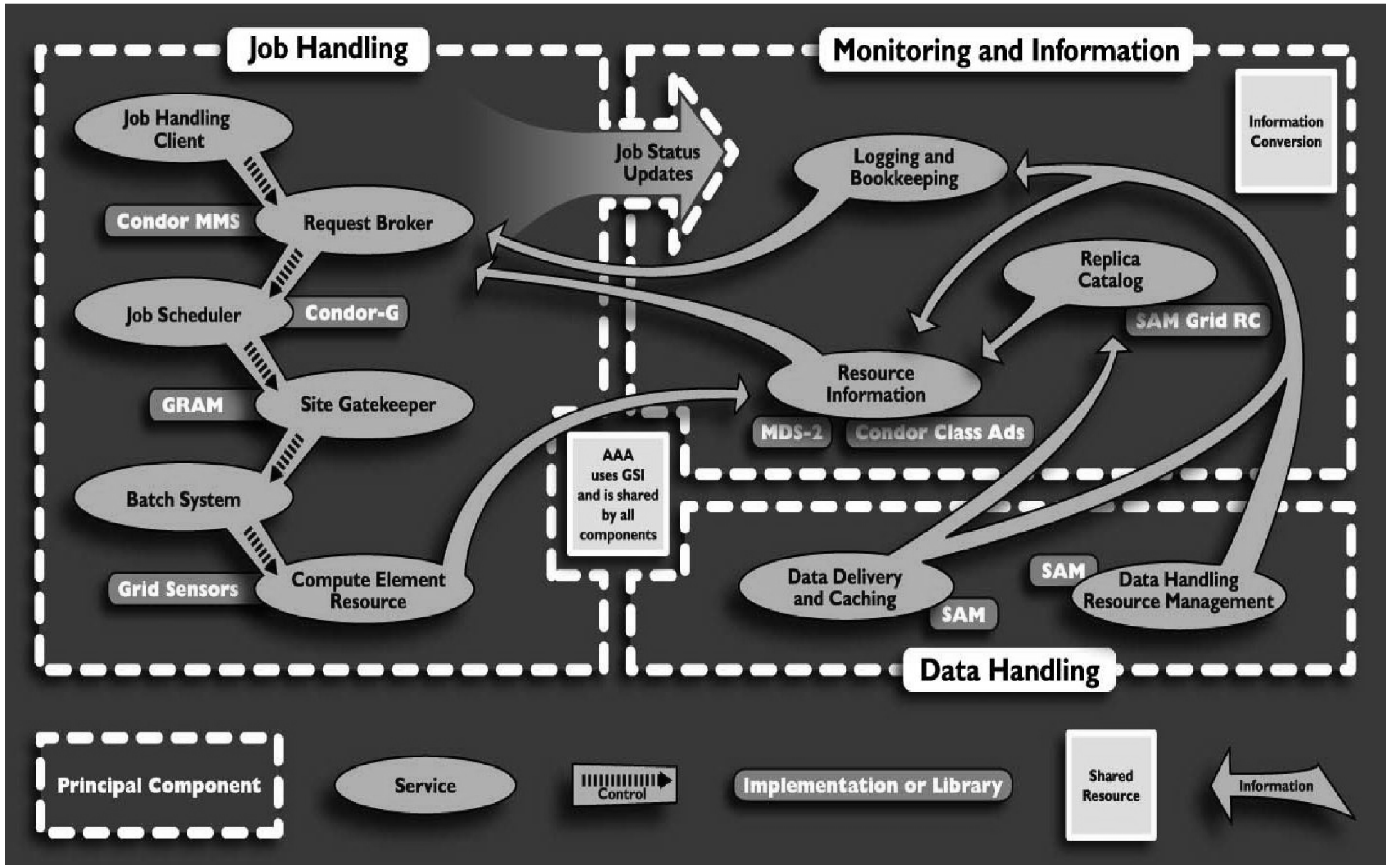

Figure 3.2. Work flow through the SAM-Grid Architecture

```
job_type = dzero_reconstruction
sam_experiment = d0
sam_universe = prd
group = dzero
station_name = samgfarm
d0_release_version= p17.03.03
jobfiles_dataset = d0repro_jobfiles_p17.03.03_2
input_dataset = dayset-2004-08-18-196489-0
test_run = true
check_consistency = true
instances = 1
```

Figure 3.3. Example Submission Script.

This script defines, the execution site to be used, the input dataset that must be processed, the version of the binary dataset that must be used for processing, and other parameters such as specific type of job that must be run. The types of jobs that the DØ Data Reprocessing effort deals with are Reprocessing jobs and Merging jobs. The client software takes this submission script as input, and converts it to a job description file. The client software is essentially a wrapper around Condor® submission tools. Thus the submission script needs to be converted to a format that the Condor® executables can interpret.

The job submission will also require grid security tools installed on the client site. The security infrastructure of SAM-Grid is based on GSSAPI[26][8] and GSI[9] [26, 25]that is provided by Globus® . The security infrastructure requires the identity of the user to be present at the client site, in the form of certificates that are issued by certificate authorities recognized by the relevant virtual organisation. The infrastructure generates proxies from the user credentials which are then used during job submission.

Once the user credentials are available and the submission script is converted to the Condor® Job Description File[10] format, the job is submitted through

Condor® mechanisms to a submission site. Currently there are six submission sites in operation in SAM-Grid. The client software can be configured to use any one of these sites provided the user is authorized to access the sites.

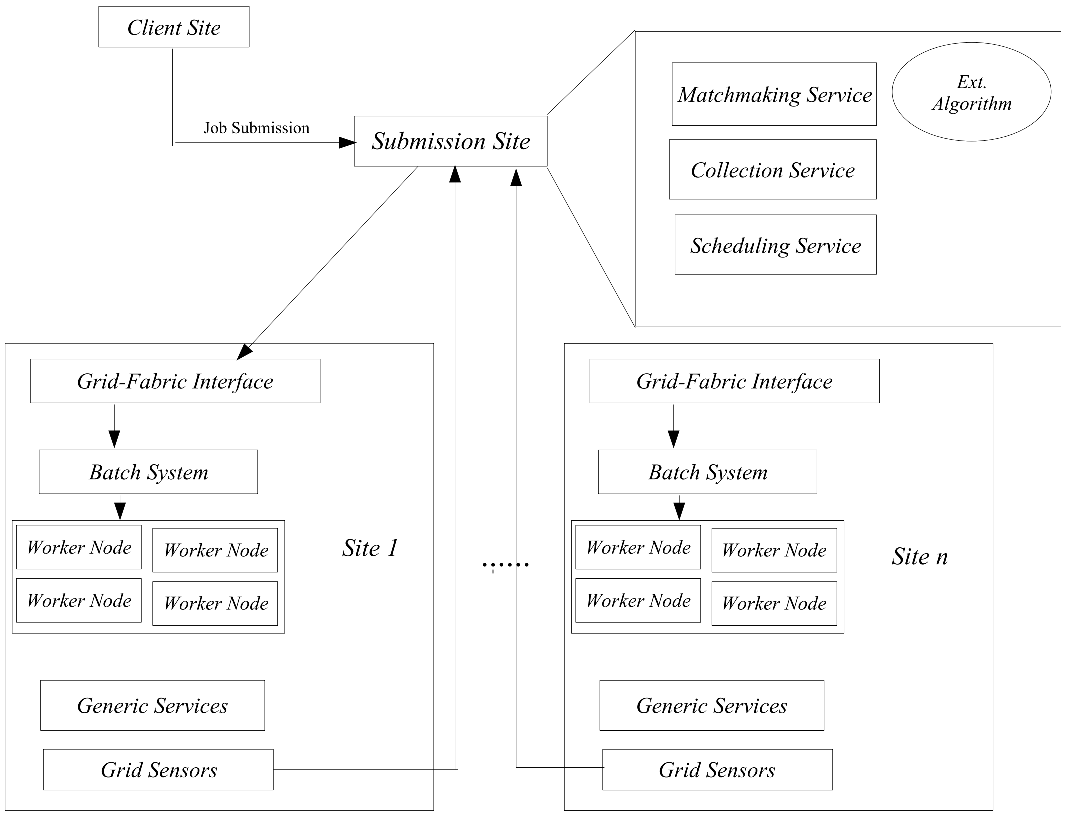

Figure 3.4. Flow of Control During Operation.

3.2.2 Matchmaking of Job Requirements

Once the job enters the submission site, it is put in the execution queue by the scheduler daemon that runs on the submission site. The submission site maintains a list of all the execution sites that are currently accepting jobs. The requirements of the job are matched to the resources provided by the execution sites to see which site would match the job requirements perfectly. Currently, the execution site on which the user wants to run her jobs is specified as a requirement. This means that the job will get matched to only one site. However, efforts are currently underway by the SAM-Grid development team to generalize the matchmaking of jobs to execution sites. This is being implemented by taking other parameters such as site rank[11] into consideration. The matchmaking of jobs is done by the Negotiator daemon. The mechanism of matchmaking is described in more detail in [23]

After a match has been made, the submission site ships the job over to the chosen execution site using Globus® mechanisms for job submission.

3.2.3 Job Execution

The grid job enters the execution site through a gatekeeper daemon. One of the parameters of the job specifies the job manager that the job requires. The gatekeeper determines this parameter and passes control of the job to the correct job manager.

When the job enters a cluster, it enters through the head node. The initial events before the job is sent to the worker node occur on the head node. This is depicted in Figure 3.5 The job manager software is a complex framework that manages the execution of grid jobs on the execution site. Currently all the jobs that are associated with the reprocessing effort use the SAM-Grid job manager. This job manager supports many types of grid jobs such as reprocessing jobs, merge jobs, Monte-Carlo jobs, and SAM analysis jobs. The support for each of this job type is implemented using specific job adapters. Once the type of the job is determined, control is passed to the job adapter associated with that job type. The job adapter then sets up the environment necessary for execution of the job. A brief note must be mentioned here about the need for different job adapters. Every job type, such as monte-carlo, reprocessing and SAM analysis, has a different execution mechanism, and requires a different execution environment. Each job type needs to be handled in a different fashin, and since the JIM framework is designed in a modularized fashion, supporting new job types is a matter of implementing a job adapter for that type of job.

The job needs to have the binary and input datasets present. For this, the job adapter contacts the local SAM station, and starts a project with it to get the input and binary datasets. The SAM station then looks into its cache to see if the files from the datasets are available.

If they are not available, then the system contacts the SAM meta-data catalogue to find out where the files of the datasets may be obtained. Once it establishes the most suitable source for the files, the SAM stager on the local machine starts retrieving these files into the SAM cache.

After the job adapter starts the SAM project, it prepares a temporary working directory called the sandbox are on the head node of the cluster. This directory contains all the required files that must be transported to the worker node of the cluster for the job to be executed smoothly.

The job manager software has multiple layers of wrappers that manage the execution of the job at the worker node. All these wrappers are put into the sandbox area to be transported to the worker node.

A mechanism is needed to bootstrap the job execution on the worker node. For this, a self extracting executable is created that contains tools that are necessary to retrieve all the scripts and software from the sandbox area. This executable contains the grid credentials, the data transport tool that implements the Grid-FTP protocol[1, 27], and a list of all the scripts in the sandbox area that are necessary for job execution.

Once the sandbox area is prepared with all the necessary scripts, the job adapters pass control over to the batch system. This is done through the batch adapters mentioned in the previous section. The self extracting executable is passed to the batch adapters to be run on the worker nodes.

Every grid job is divided into a number of batch jobs. When the job adapter starts the SAM project to retrieve all the input files, it determines the number of files in the input dataset. This number determines the number of batch jobs that will be created. This is the number of times the batch adapter submits the self extracting executable to the batch systems. When the batch adapter submits the jobs, it assigns a qualifier to all the batch jobs that are associated with this grid job. This is to aid during polling of jobs. The batch system then starts the execution of the bootstrapping script on the worker node.

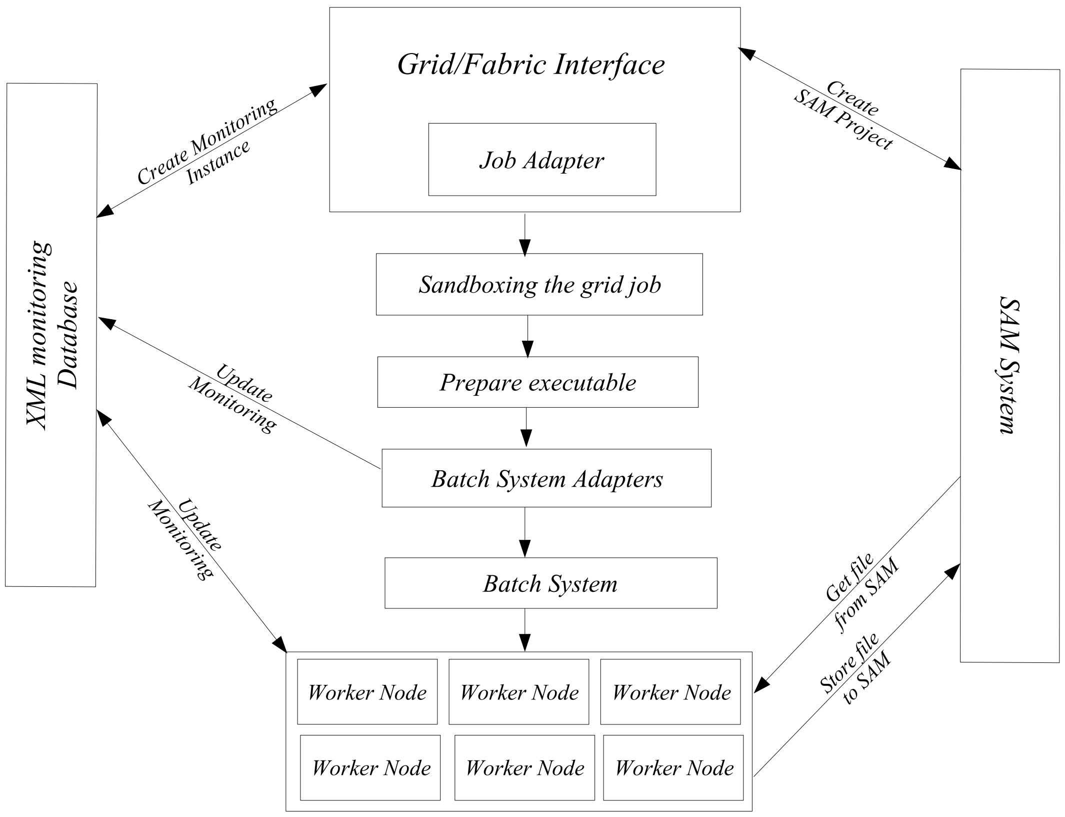

Figure 3.5. Flow of Control in the Cluster.

Since the execution is consistent over the entire sample of batch jobs, except for the input file that it gets, we describe the execution of one batch job at the worker node.

Once the batch system transports the bootstrapping executable to the worker node, the executable extracts itself and starts the necessary transport script. This script uses the grid credentials of the user and the tool that implements the grid-FTP protocol, and transfers all the necessary wrappers from the sandbox area on the head node.

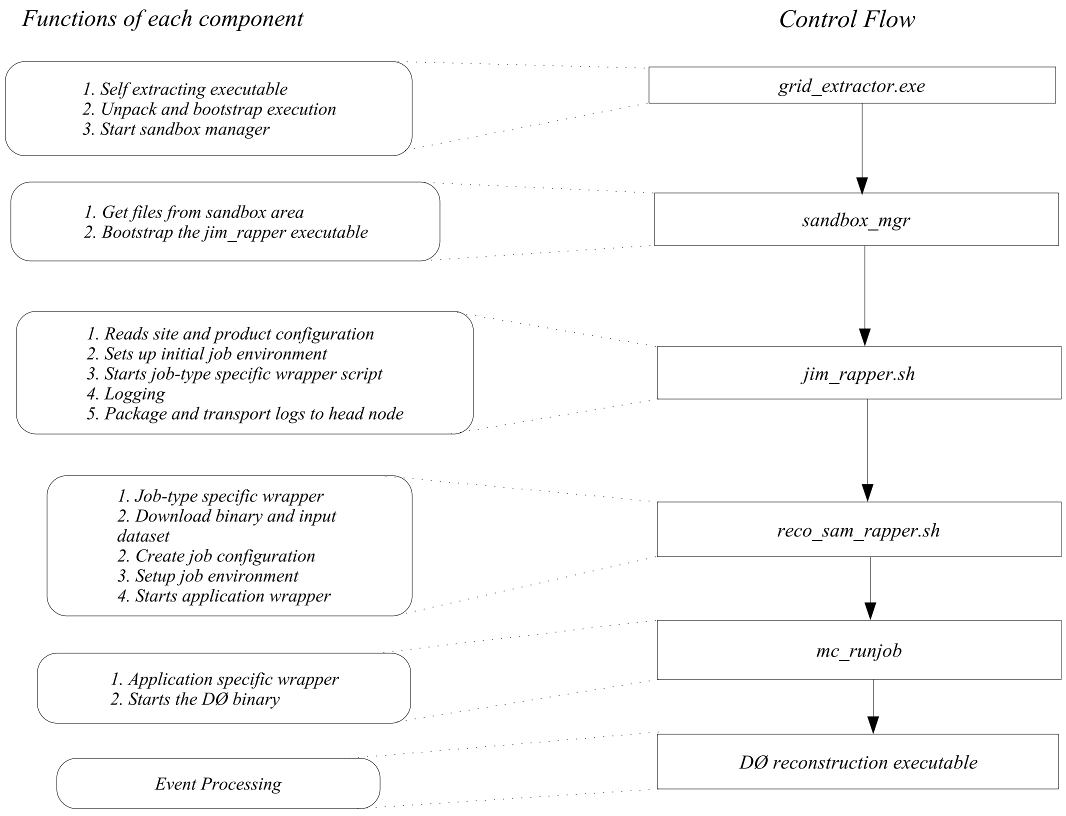
Functions of each component
Control Flow
1. Self extracting executable
2. Unpack and bootstrap execution
3. Start sandbox manager
grid_extractor.exe
1. Get files from sandbox area
2. Bootstrap the jim_rapper executable
sandbox_mgr
1. Reads site and product configuration
2. Sets up initial job environment
3. Starts job-type specific wrapper script
4. Logging
5. Package and transport logs to head node
jim_rapper.sh
1. Job-type specific wrapper
2. Download binary and input dataset
2. Create job configuration
3. Setup job environment
4. Starts application wrapper
reco_sam_rapper.sh
mc_runjob
1. Application specific wrapper
2. Starts the DØ binary
DØ reconstruction executable
Event Processing

Figure 3.6. Flow of Control on the Worker Node.

Setting up the environment for the reprocessing job to run smoothly is a non-trivial exercise. One of the important things that the bootstrapping script does is dynamic installation of requisite software on the worker node. This includes two layers of software installation.

- Installation of support software that provides for the necessary tools and environment for the DØ software to run.
- Installation of the DØ software itself, on the worker node, just before execution.

Apart from all the wrappers that are present in the sandbox, the sandbox contains a tar-ball that packages all the JIM and SAM software that will be required for the job to execute. The DØ software, ie. the executable in the binary dataset is designed to run in environment where it access to the SAM client software, which can be used to communicate with the SAM station. So the bootstrapping script transfers the tar-ball, and unpacks it, effectively installing the SAM environment on the worker node.This encompasses the installation of support software.

Control is then passed to the wrapper scripts. The wrapper scripts use the SAM software to get the input file meant for the job from SAM, along with the files in the binary dataset. The binary dataset which contains the DØ software is unpacked, and the DØ software is effectively installed on the worker node. This stage exposes one of the most important aspects of any grid computing environment. The purpose of any grid infrastructure is to be able to run jobs on remote computing elements that may be dynamically added to the infrastructure. Once a new element is added it may not have any of the requisite software installed. However the ability of the infrastructure to enable any virgin machine added to the system to run the DØ software by virtue of just being connected to the system, and without requiring prior installation of software, is an important feature of the SAM-Grid infrastructure.

After this, the environment necessary for DØ reprocessing is set up on the worker node and control of the job is passed to the DØ binary.

The DØ binary runs on the input file and produces the reconstructed output file which must be stored back into SAM. The execution of the binary itself is outside the scope of this document. The reprocessing job now transfers the output file back to the head node into the sandbox area. It then informs the File Storage Server about the file and requests that it be stored back into the SAM system.

There is disk area on the execution site registered with the SAM database called the durable location. This is used as a temporary location for storage of reprocessed files. This is another change made to SAM-Grid to support reprocessing. The reason for the presence of a temporary storage location is DØ merge jobs. This class of jobs and its operation is explained in section 3.3.

When the DØ executable requests the FSS to store the file back into SAM, it requests that the file be stored in the durable location.

Once the file store is complete, the control is returned back to the wrapper scripts. These scripts package all the logs and the output created by job into a tar-ball, and ship it back to the sandbox area.

After all the batch jobs have completed execution, the grid is notified of their completion. After this, the tarred output of all the batch jobs is packaged into an archive, compressed and shipped to the submission site. This effectively marks the completion of execution of the grid job.

3.2.4 Job Monitoring

One of the most important parts of job execution is monitoring[24]. The monitoring framework in SAM-Grid is divided into three levels.

- Grid level monitoring - Here monitoring is conducted by standard Condor-G®[12] mechanisms. The submission site polls the grid interfaces of the execution site for an update on the status of the grid job. The execution site returns this information, which is then displayed through the SAM-Grid submission site web interface.

- Cluster level monitoring - Here monitoring occurs at the batch system level. When the job adapters submit the jobs to the batch system, they make a list of all the batch jobs that are created. When the batch system is queried, all the batch jobs are checked and their status updated. One important feature is that this monitoring information is also written to the XML database present on the same system.
- Job level monitoring - As the batch job executes on the worker node, it is desirable to have the status information of these jobs updated and displayed in real time. To aid this, the wrappers which execute the DØ executable, write status and monitoring information to the XML database through the database's command line interface.

Monitoring at the job level is decoupled from the other two layers. The cluster level monitoring, however, depends on grid level monitoring. The grid polls the execution site for the status of the grid job. The job adapters then query the status of all the batch jobs associated with this grid job, and finally an aggregated status of all the jobs is returned back to the grid. The job adapters also update the status of the jobs in the XML database.

3.2.5 Job Identifiers

A brief note must be mentioned with regards to job identifiers. The identifier for the grid job is referred to as the Grid-ID, and is a unique identifier over the entire grid. This grid ID is created during initial job submission on the client site. When a grid job is assigned to an execution site, the Condor® daemons on the submission site create a unique identifier for the job called the Cluster-ID. This

is necessary to identify the job to the Condor® daemons. Once the job enters the execution site, and batch jobs are created, each batch job is assigned a batch system ID. A convenient way to identify all the batch jobs belonging to a certain grid job, is to assign a qualifier to the batch jobs, which is a direct mapping of the grid ID. This qualifier is called the section ID or project ID. The section ID is obtained by passing the grid-ID through a hash function. Most batch systems accept the qualifier in a certain format, such as fixed length formats, which do not allow the presence of symbols like underscore (_), hyphen(-), period(.), etc. However the grid ID contains all these characters and may be much longer than the format that batch systems can accept. This is why the grid ID needs to be hashed to obtain a section ID.
These identifiers are used extensively during the monitoring of the job.

3.3 Merging of Output

The sizes of outputs that the reprocessing jobs produce are proportional to the sizes of the input files that they process. Since the primary function of a reprocessing job is to act as a filter, the output files are smaller than the inputs.

Ideally once the output files are produced, they are supposed to be stored back to the Mass Storage System at Fermilab. However, it is more efficient to write large files of the size of about 5 GB to 10 GB to the robotic tape storage than files of smaller sizes that the reprocessing jobs produce, which are usually of the order of 100 MB to 700 MB. For this reason, it is a good idea to merge the output of reprocessing jobs, before storing the output into Enstore. This can be done by a class of jobs called Merging jobs.

Merging jobs are also submitted through the grid, and go through the same stages as a normal reprocessing job. Once the execution site is entered, the batch job starts on the worker node, the outputs of the reprocessing jobs that have been stored in the durable location, are retrieved and merged. The merged outputs are finally stored back into the mass storage system through SAM.

The above explanation shows the need for the durable location. It is mainly meant to provide a temporary location close to the execution sites.

Once merging jobs complete and store the output back into Enstore, a cleanup script is run to clean the durable location of the temporary files.

3.4 Work done as a part of the Thesis

The above explanation describes the working of the SAM-Grid infrastructure in detail. However, a brief note must be mentioned as to the parts of this implementation that were done as a part of this thesis.

- Implementation of a module in the user interface package that recognizes and parses the submission scripts for data reprocessing jobs. The main function of this script is to ensure that the required parameters are present in the submission script, and the submission script is parsed correctly.
- Modification of the user interface to reduce overhead during monitoring of jobs. The user interface polling mechanism would query the status of all the running jobs from Condor® and parse through the output of the poll to find the status of one job. The modification entailed querying the status of just one job from the Condor® system.
- Implementing the job adapter that would accept and process reprocessing jobs. This entailed
  - Parsing the parameters that come from the job manager, to make sure that all the necessary parameters are supplied.

  - Preparing sandbox area, and making sure the correct wrapper scripts are present.
  - Preparing the self-extracting bootstrapping executable to reflect the correct job type.
  - Implementing a mechanism to create and submit jobs to the batch system.

- Implementation of the wrapper scripts that would be used on the worker node to run reprocessing jobs. This entailed making sure that the input and binary datasets are correctly downloaded to the worker nodes to enable execution and studying the environment necessary for the DØ software to run and making sure that the wrapper scripts set this environment correctly.
- Identification of scalability and stability issues. This constituted of identification of problems during the testing stage as well as the production stage.
- Measures to alleviate the system failures caused by scalability issues.

The last two items along with their significance and effects are elaborated in subsequent chapters.

CHAPTER 4

SCALABILITY AND STABILITY ISSUES

A primary feature of a computational grid is its sheer size. The SAM-Grid infrastructure is currently deployed on fourteen high performance clusters, processing data at the rate of about one Terabyte per day. This presents huge challenges to the computational effort in terms of scalability and stability of the resources.

In this section we present the stability and scalability issues that were encountered during the adaptation of SAM-Grid for reprocessing, as well as the steps taken to overcome the issues or to minimize the chances of these problems occurring.

4.1 Increased Load due to Job Submission

- **Problem**

  Launching a grid job on an execution site is a computationally intensive activity. Every grid job that is launched has to start a project with the local SAM station, and retrieve all the files that are required for the job. This is a I/O intensive activity. The mechanisms of producing the self extracting executable, preparation of the sandbox area, and repeated submission of the jobs to the batch system are very computationally intensive and I/O intensive activities. Testing has shown that on a medium sized cluster, a maximum of 3 grid jobs can be running at any given point of time, and on a large sized cluster, a maximum of 6 grid jobs can be running at the same time without overloading the head node of the cluster. An important aspect of starting the grid job on the head node of a cluster is that the amount of computation necessary to setup the execution environment is not dependant on the size of the grid job, ie. the size of the dataset to be processed. The 3-6 job limit on the cluster essentially depends on how powerful the head node of the cluster is. The only part of the operation that depends on the size of the dataset is when the SAM station tries to get files in the input and binary datasets into the SAM cache. This problem is addressed in the next section.

When a job is submitted from the client site, it stays idle on the submission site for approximately five minutes before being shipped to the execution site. This is because the Condor® matchmaking cycle runs once every 200 seconds. The only time a job stays idle in the queue for more is when it is not matched to its desired site because the site's advertising capabilities are not operational. Here too, the job is shipped to the site as soon as the site's advertising capabilities are activated.

As soon as the job reaches the execution site, the gatekeeper starts to execute the job. Site operators submit a lot of jobs to keep their execution sites continuously busy. If all the jobs start at the same time on the cluster, then the head node of the cluster is easily overloaded, and may lead to the head node crashing.

- **Solution**

  One of the methods being used to prevent overloading the cluster due to job submissions is throttling. The throttling of job submissions can be controlled according to certain policies implemented on the submission site. The policies are currently being researched and the study of these policies is outside the scope of this document. Throttling essentially slows down job submission to the execution site based on the load on the execution site.

The execution site advertises the maximum permissible limit of running jobs on the site, by standard Condor® class-ad mechanisms. It also advertises its current load. The negotiator daemon looks at these two parameters and decides whether to ship the job to the execution site or to delay job submission.

This policy of job throttling essentially alleviates the burden of job execution on the execution sites.

4.2 Increased Load due to Data Transfer

- **Problem**

  Once a grid job starts executing on a cluster, the job starts to request files from the input and job datasets. As explained in the previous chapter, each grid job is divided into about a hundred batch jobs. Each of these batch jobs request files from the SAM system. Hence, there may be a hundred processes requesting files from the SAM system. This is under the assumption that there is only one grid job executing on the cluster. If there are more than one grid job on the cluster, the cluster may become overloaded quite quickly, leading to network and system failures. Usually the clogging of the data transfer pipes occurs mainly in intra-cluster transfers. The SAM system is robust enough to serialize data transfers from other SAM stations.

  Overloading the cluster due to file transfers is not restricted to transfers from the SAM system. When the self-extracting executable starts to execute on the worker node of the cluster, it tries to retrieve files from the sandbox area on the head node. For every grid job there are about a hundred batch jobs concurrently trying to transfer files from the sandbox area. As mentioned above there are usually 3 to 6 grid jobs executing on a cluster during peak usage. This means that there may be close to 300 - 600 jobs transferring files from the head node. Once again this may clog the data pipes, as the jobs waiting for files eventually time out. This results in the dataset not being fully processed and is categorized as failure of the grid job. It may also lead to the head node itself overloading and consequently crashing, due to the large number of data transfers that it might be trying to service.

- **Solution**

  One method that may be used for reduction of load is queuing. Currently, a Farm Copy[13] utility is used to queue transfers on the head node. Multiple queues are configured, where each queue can service a certain number of clients in parallel. The rest of the jobs in the queue are serviced in the order that they were received.

4.3 Increased Load due to Monitoring

- **Problem**

  Monitoring is one of the most important aspects in a computing infrastructure. The monitoring aspect of the SAM-Grid infrastructure is based on the Globus® monitoring rules. Globus® has a tendency to poll submitted jobs every 10 secs. When polling occurs from the grid, the job managers go through job adapters, down to the batch system to query the status of all the batch jobs, associated with the particular grid job that the grid is polling for.

  This chain of events is a very system intensive activity. There are three points of failure. The job adapters themselves may crash during retrieval of output, or the batch retrieval command may crash during execution if the batch system is servicing another request, or the batch system itself may crash. The last failure is very rare, and so will not be discussed. Each of these failures will be returned back to the previous layers. When a query does not return within a certain time limit, the grid assumes that the job has crashed, even though the job may still be running and only the retrieval of monitoring information has failed.

The grid decides its future action based on the current status of the job. If the grid thinks that the job has errored out, but in reality the job is still running, then the job is left in an inconsistent state. The grid would then proceed to clean out the sandbox area, and retrieve the job output. If there is any batch job in the batch system, corresponding to that particular grid job, that has yet to start execution on a worker node, the job will not be able to get any files from the sandbox area as it is already cleaned up and will surely fail.

At the batch system level, the occurrence of failure is quite frequent. When the batch system is queried for the status of the batch jobs, the batch system may overload and return nothing, or the requesting process above it may time out, waiting for the batch query command to return. The aftermath of this is the same as that of a grid poll crashing. Of all the batch systems that SAM-Grid is capable of interfacing, PBS, and Torque have proven to be the most unstable, with frequent timeouts during job status queries.

Another aspect of the stability of monitoring that must be mentioned is the presence of delays in jobs appearing on the batch system. After the grid job is divided into batch jobs and submitted to the batch system, there is usually a small time delay between submission and jobs showing up in the batch system queue. If the grid job is polled during this time, the batch system will not return the status of the jobs, and the grid will assume that the jobs are finished. Once again the effect of this is the same as that, when the monitoring queries fail. This problem has been addressed by the design of the batch adapters, also called batch system idealizers. This has been mentioned here for the sake of completeness, and more details about the problem as well as the solution may be found in [16, 8, 20].

- **Solution**

  From the above description, we may infer that the failure to retrieve monitoring information is very dangerous to job execution, because even a temporary failure of polling may result in failure of the grid job. Hence, this retrieval must be made completely robust and failsafe. This problem has been addressed with caching. There are two layers of caching. One is at the grid level and the other is at the batch system level. Both the layers are independent of each other.

  Ideally, every polling request that comes from the grid, goes all the way down to the batch system, and the output of the batch system is retrieved and packaged in the right format for the grid, and returned. However a policy is implemented where querying the batch system occurs at a much lower frequency than polling from the grid. Every time the batch system is queried, the results are cached. A lifetime is set for the cached results. Whenever the grid polls the status of a job, the cached results are checked. If the cache hasn't expired, then the cached results are returned. If it has, then the job adapter is invoked, the batch system is queried, and the cache is updated. Here the results that are returned to the grid are more often than not, outdated. However, it is better to return information to the grid that will not change the state of the job, than to not return anything, which may put the job in an inconsistent state.

At the batch system level, the caching mechanism is slightly different. Every successful query to the batch system is cached. Only when a query fails, the latest cache is retrieved and returned to the querying layer.

Caching at the grid level is done as a matter of policy, whereas the caching at the batch system level is done as a fall-back mechanism.

This chapter has described the scalability and stability issues that have come up during the implementation of the grid infrastructure and adaptation of the infrastructure to support data reprocessing. The solutions that have been presented here are not panacea for the problems, but offer significant improvements in performance and also the system scalable to a satisfactory magnitude.

CHAPTER 5

MEASUREMENTS

The implementation aspects of the SAM-Grid infrastructure are already discussed. In this section, we attempt to quantify the efforts in terms of computation time and efficieny.

The reprocessing effort is currently underway. The current statistics of the effort are as follows. Some parts of this data come from [5]. The rest have been arithmatically calculated.

Total physics events to be processed = 986189069 (˜ 1 billion events)

Total size of the dataset = 250 Terabytes of raw detector data

Start of the Reprocessing Effort = 25th March 2005

Date of Last status check = 27th July 2005

No. of events processed and stored = 548976450

Ratio of Events = 55.6 %

Size of the processed data = ˜ 139 TB

Time taken to process detector data = 124 days

Current Throughput = 1.1 TB/day or 4.4 million events/day Average

Total time to finish at the average rate of output = 227 days

The throughput mentioned above is average throughput. The graph shown in figure 5.1 shows the throughput of the grid over a period of one month. The graph is a logarithmic graph, and as we can see from it, the throughput for one month ranges between 1 million events/day to 10 million events/day The computing resources that SAM-Grid has access to, are very heterogenous in nature. There are currently 14 sites that are connected to SAM-Grid, of which 7 have been certified to run DØ data reprocessing jobs. The machines on these sites range from 1.1 GHz Pentium®[14] III single processor machines to 3.2 GHz Hyper-threaded Pentium®-4 Dual Processor machines, and from 1.3 GHz to 2.2 GHz AMD®[15] dual processor machines.

With such a vast variety of processors, it is rather difficult to assess the processing time. To do this we need to standardize the computing resource.
A brief aside about the DØ executable is that it is not multithreaded, and cannot take advantage of Symmetric Multi-Processing. So a dual processor system may be considered equal to two single processor systems.

For this purpose, we consider a standard 1 GHz Pentium®III processor as our base processing element. There are other parameters such as available memory, disk access speed, and network speed. The executable along with the input file uses approximately 1 GB of memory, which the worker nodes of all clusters have. The executable does not have any disk access other than writing logs during execution, and an occasional memory write,so disk access speed is not an important factor, and finally, the only network access during execution is internal to the cluster and hence almost no delay. Under these conditions, the processor speeds may be assumed to be linearly scalable.

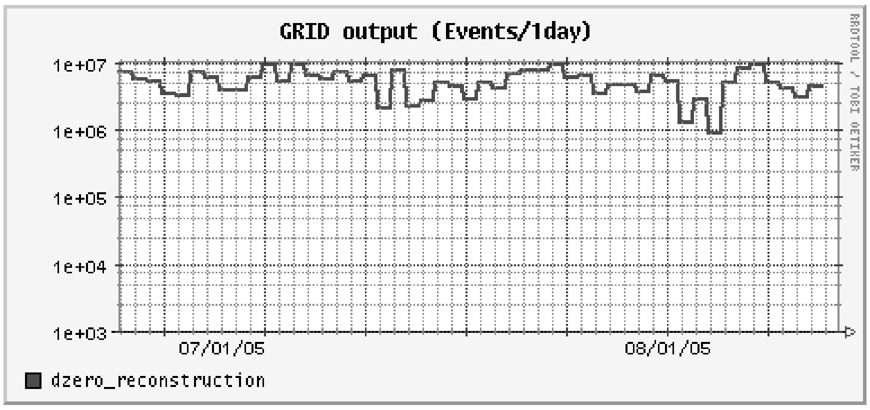

Figure 5.1. Throughput of the Grid over One month.

To prove this, a test reprocessing job was run with the same input dataset, and the same version of the binary executables on all the clusters. The result was that on a standard processing element , every event would take an average of 21.7 seconds to execute. On the 6 execution sites that were tested, with at least 8 different types of processors, the worker nodes exhibited similar or proportional execution times. Some of the example execution times are given in the Table 5.1.

Table 5.1. Execution times of the DØ jobs on different machines

Architecture	No. of Events	Execution Time(s)	Event Time(s)	Scaled time(s)
2.5 GHz	2433	22307	9.17	22.9
2.5 GHz	2770	22623	8.17	20.4
2.5 GHz	2773	24855	8.96	22.41
2.5 GHz	2746	21774	7.93	19.82
3 GHz	2811	20324	7.26	21.8
3 GHz	2773	20398	7.33	21.9
3 GHz	2815	20498	7.28	21.9
3 GHz	2776	19698	7.1	21.3
2 GHz	2770	27125	9.79	19.6
2 GHz	2773	26835	9.68	19.35
2 GHz	2746	26000	9.47	18.94
2 GHz	2817	27401	9.73	19.45

Thus on a single Pentium®III processor, we would take about 678 years to process the entire dataset of approx. 1 billion events.

Using these result, we may deduce the advantages of running reprocessing over a grid infrastructure.

Let us assume a single institution involved in reprocessing to have a medium scale cluster of 300 nodes. Each of these nodes has a standard processing element.

To have the best performace available, we assume ideal network access with no delay and no lag time for disk or memory access, and consider CPU processing to be the only time consuming activity.

Under such circumstances, the throughput of the cluster would be 300 events per 21.7 seconds. To process a total of close to 1 billion events, under ideal conditions would take over $2\frac{1}{4}$ years, with a maximum throughput of 1.1 million events per day.

However, the current statistics mentioned above take into consideration network delays and startup times which are non-trivial. This means that, the effort of running data reconstruction through the grid, is still 4 times faster than one cluster running the effort, with no scheduling or networking overheads .

It would be useful to calculate the average time a grid job takes to complete. The average time a grid job takes to complete will give us an indication as to, what the frequency of job submissions ought to be, to keep an execution site continuously busy.
The time it takes for a grid job to complete is calculated as follows.

$$\tau_{grid} = \tau_{overhead} + max(\tau_{exec})$$

where

τ_{grid} : Time the grid job takes from start to finish

τ_{exec} : Time a single instance of the DØexecutable runs

$\tau_{overhead}$: Time for the Overhead, such as setup of job and storing output

$AllTimementionedarewall - clocktime$

$\tau_{overhead}$ includes submission time to the cluster from the submission site, time for job setup on the execution site, time for the job and input datasets to be retrieved from SAM, and data tranfer delay for intra-cluster transfers.

We have already mentioned that the resources such as network speeds, disk access times, etc. are very similar on almost all execution sites. The $\tau_{overhead}$ term depends on mainly these factors, and hence can be taken to be a constant. This will simplify the calculation of the average time it takes for a grid job to execute.

Each grid job corresponds to processing one dataset. From the above explanation, we see that all other factors being equal, the time it takes for a grid job to execute depends on the number of events in the dataset. This means to find the average time of a grid job, we need the average size of the input dataset. However, of a random sample of approximately 3200 datasets, the distributed of the number of events is almost random. Figure 5.2 shows the distribution of events over the datasets. This sample of datasets corresponds to approximately 460 million events, ie. 45% of the entire sample to be reprocessed.

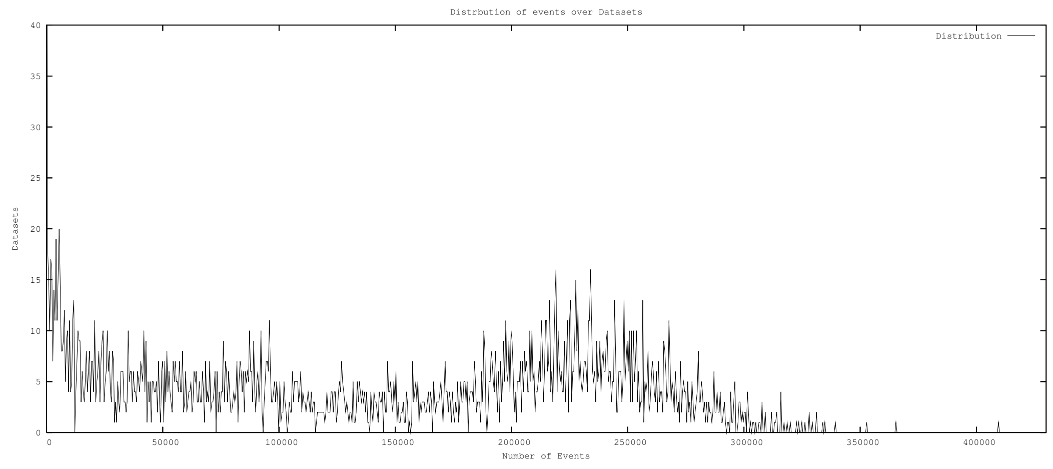

Figure 5.2. Distribution of the Events over a random sample of datasets.

With such a random distribution of events, there is a very large variance and determining the average time it takes for a grid job to execute does not yield any useful information.

A brief note must be made about the resources that are currently available to SAM-Grid. There are currently an equivalent of 2800 processing elements spread across 7 certified sites that are dedicated to the DØ Data Reprocessing Effort. Earlier, we gave statistics of the execution of jobs on a medium sized ideal cluster of 300 nodes. We now compare it with the grid containing 2800 nodes and network & scheduling overheads. With an increase of over 9 times the number of processing elements, we see an increase in the throughput by a factor of 4. Some of the reasons for this non-linear increase, are network delays and delayed job submissions. The solutions that have been implemented to make the system more robust have also contributed to slowing down production. For example, reducing transfer rates in intra-cluster slows down execution of jobs on the worker nodes, but leads to lower failure rates in grid jobs. The same goes for job throttling. An important point that needs to be stressed here is that the 4 times increase in throughput is an average increase. There are times when some execution sites are not operational due to failures in the systems themselves, or due to maintanance. Due to the many factors that affect the performance of jobs, it is very difficult to determine the effect each of the measures taken to solve scalability issues, have on the over performance of the system.

The solutions that have been implemented to make the system robust and scalable have decreased the failure rate significantly. When the SAM-Grid development team is alerted about unusually high failure rates, the problems are debugged and the systems patched. A brief note must be mentioned about the statistics given. Problems with scalability are not associated with all the execution sites. More often than not, it is usually just one execution site suffering from a particular issue. So the statistics mentioned below are only with respect to the site that was affected and only during the testing period. Patching and updating is not done incrementally but done during product updates. Only critical failures are patched incrementally and tested. These statistics are only from such testing, and are accurate only to a certain point. The effects of each of these patches on the other execution sites cannot be assessed, nor can their performance be assessed over a long period of time, as the number of patches increase, and may affect performance on sites where lack of patches hadn't shown any problems.

The statistics given below are from a time period of 2-3 days before patching to 1 day after patching.

- Failure rates due to Monitoring Problems

  N_{grid} = 20
  N_{fail} = 19
  Percentage of failure before patching = 95%
  $N_{grid.new}$ = 5
  $N_{fail.new}$ = 0
  Percentage of failure after patching = 0%
  N_{grid} - Number of grid jobs over a span of 2 days before patching.
  N_{fail} - Number of grid jobs that failed.
  $N_{grid.new}$ - Number of grid jobs tested after patching over a span of 1 day
  $N_{fail.new}$ - Number of grid jobs that failed after patching

  The failures mentioned here are with respect to only monitoring issues. This behaviour was observed only two sites out of the 8 currently certified sites.

- Failure rates due to Data transfer problems

  N_{batch} = 500
  N_{fail} = 240
  Percentage of Failures before patching = 48%
  $N_{batch.new}$ = 100
  $N_{fail.new}$ = 0
  Percentage of failure after patching = 0%
  N_{batch} - Number of batch jobs over a span of one day
  N_{fail} - Number of batch jobs that failed

These errors were noticed on one site only. The other sites were not under as heavy a load at the time this issue was raised, and so it is not possible to determine how the patch to alleviate data transfer problems affected other sites.

In both the above cases, only the number of jobs that failed due to explicitly the reasons that they were patched for are considered. Jobs that failed due to problems with the site and problems due to DØ binary failures are completely discounted.

Currently, there are no statistics regarding how job throttling affected the failure rates of the execution sites.

CHAPTER 6

CONCLUSION

Grid computing has proven to be a scalable, robust design for large scale computation. The design of the SAM-Grid distributed computing infrastructure(DCI) is a viable computing model for the DØ Data Reprocessing effort.

The JIM framework, which augments the SAM data handling system, to form the SAM-Grid DCI, provides extensive job execution and monitoring capabilities. Though the SAM-Grid infrastructure is currently designed to run only DØ application, the design can be easily modified to run any kind of High-Energy Physics application.

6.1 Future Work

The computing model presented here, though not primitive or prototypical, shows a lot of potential for extensibility and improvement.

- One of the possible areas of research in the future is automated job submission. Currently, site administrators are assigned datasets to process, which they then proceed to submit jobs from. However, this process will be automated in the future.
- Currently research is being conducted to adapt other experiments such as CDF and the CMS experiments to use the SAM-Grid DCI.
- There are also efforts to interface the SAM-Grid infrastructure with the LHC Computing Grid[6], so that the resources of the LCG may also be used by SAM-Grid.

Change and progress are but inevitable in computer science, and grid computing is no exception. The field shows a lot of promise, and the concept of computational grid is an area of extensive and exciting research. The work done in this thesis is but a humble contribution to the field.

APPENDIX 1

GLOSSARY OF TERMS

...

[1]VOMS - Virtual Organisation Management Systems

[2] Globus® : Globus is the registered trademark of the Globus Alliance

[3] Condor® : Condor is the registered trademark of the University of Wisconsin, Madison

[4]URL - Uniform Resource Locator. This describes the domain name of the head node of a cluster that is the entry point into the execution site. It also contains the job manager that accepts the connections.

[5]PBS® - Portable Batch System is a registered trademark of Altair Grid Technologies

[6]TORQUE® - Tera-scale Open source Resource and QUEue manager is a registered trademark of Cluster Resources Inc.

[7]SGE® - Sun Grid Engine is a registered trademark of Sun Technologies

[8]GSS-API – Generic Security Services - Application Programming Interface - This is the security infrastructure used by certain networking utilities such as Globus® and Kerberos.

[9] GSI - Grid Security Infrastructure. This infrastructure is based on GSSAPI, and supports, Proxy Authentication, X.509 Extension, Certificate Authority support, and a host of other features.

[10] Condor® JDF - The Condor® Job Description File is similar to the JIM submission script, uses the

```
key = value
```

type of description. The Condor® JDF describes the job in more detail that the submission script.

[11]Site Rank - This is the rank of a site based on the requirements of the job. When jobs request a certain minimum configuration, the execution sites are ranked depending on how well they can provide the requested resources.

[12]Condor-G® - is a registered Trademark of the University of Wisconsin, Madison
Condor-G is grid-enabled Condor® . Condor® is interfaced with the Globus® toolkit to form Condor-G

[13]Farm Copy or fcp - This is a utility developed at Fermilab, specifically to support scalable data transfers from slow or overloaded servants

[14]Pentium® - Pentium is a registered trademark of the Intel Corporation

[15]AMD® - AMD is the registered trademark of Advanced Micro Devices

APPENDIX 2

LIST OF PRODUCTS

Fermilab uses a Linux package manager called UPS/UPD - Unix Product System/Unix Product Distribution. All the packages that are deployed as a part of the SAM-Grid infrastructure are packaged in this format.

This chapter gives a list of all the products used for various components of the SAM-Grid infrastucture.

- *Client Site*:
  - Client package - jim_client
  - Creation of Authentication and Authorization Credentials - Globus®
  - Submission of Jobs to the scheduler - Condor®
- *Submission Site*:
  - Scheduling - jim_broker_client
  - Matchmaking of Jobs/Collection of resource class-ads - jim_broker
- *Execution Site*:
  - Jobmanager/Job Adapters/Job Wrappers - jim_job_managers
  - Sandboxing Utililty - jim_sandbox
  - Package for Dynamic installation - sam_client
  - Configuration of the JIM system - jim_config
  - Authentication and Authorization Framework - VDT(Virtual Data Toolkit, containing Globus® and Condor® tools), sam_gsi_config, sam_gsi_config_util
  - Batch Adapters/Idealizers - sam_batch_adapters
  - Batch_Systems - One of PBS, Torque, Condor, BQS, SGE or FBSNG.
- SAM System - The local data handling components of the SAM system consist of the Station, Stager and FSS, that are started as a part of the sam_bootstrap package.

- XMLDB Server - The XML Database server that is used to store monitoring and configuration information is Xindice developed by the Apache foundation.

REFERENCES

[1] W. Allcock, editor. *GridFTP Protocol Specification.* Global Grid Forum, 2003.

[2] Nicholas Coleman, Rajesh Raman, Miron Livny, and Marvin Solomon. Distributed policy management and comprehension with classified advertisements. Technical Report UW-CS-TR-1481, University of Wisconsin - Madison Computer Sciences Department, April 2003.

[3] The DØ collaboration. Online Resource. `www-d0.fnal.gov`.

[4] The DØ collaboration. Online Resource. `www-d0.fnal.gov/computing`.

[5] The DØ collaboration. Online Resource. `www-d0.fnal.gov/reprocessing/p17`.

[6] Miguel Sèrgio de Oliveira Branco. *Grid Tools for the ATLAS Experiment.* PhD thesis, Conseil Europeenne pour la Recherche Nucleaire, 2003.

[7] Fermilab. *The D0 Detector*, number 185 in A338. Nuclear Instruments and methods, 1994.

[8] Fermilab. *Extending the Cluster-Grid Interface Using Batch System Abstraction and Idealization.* Computing in High Energy and Nuclear Physics, 2005.

[9] Ian Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications and High Performance Computing*, January 2001.

[10] Ian Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, February 2004.

[11] G. Garzogli, I. Terekhov, A. Baranovski, S. Veseli, L. Lueking, P. Mhashilkar, V. Murthi. The SAMGrid Fabric Services. In *Proceedings of the Advanced*

Computing and Analysis Techniques in Physics Research. Fermilab, Advanced Computing and Analysis Techniques in Physics Research, 2003.

[12] G. Garzoglio, I. Terekhov, J. Snow, S. Jain, A. Nishandar. Experience producing simulated events for the DØexperiment on the SAM-Grid. In *Proceedings of Computing in High Energy and Nuclear Physics*. Fermilab, Computing in High Energy and Nuclear Physics, 2004.

[13] G. Garzoglio. Experience producing simulated events for the DØexperiment on the SAM-Grid. Presentation, 2004. Talk at CHEP, Interlaken, Switzerland, 2004.

[14] Gabriele Garzoglio. SAM-Grid Architecture. Poster, 2004. `http://projects.fnal.gov/samgrid/architecture.html`.

[15] Ian Foster, Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 1997.

[16] Sankalp Jain. Abstracting Heterogeneities of Computational Resources in SAM-Grid to Enable Execution of High Energy Physics Application. Master's thesis, The University of Texas at Arlington, 2004.

[17] Lee Lueking. SAM Overview and Operation at DØ. In *Proceedings of the Computing in High Energy Physics*. Fermilab, Computing in High Energy Physics, 2001.

[18] Lee Lueking. SAM Resource Management. In *Proceedings of the Computing in High Energy Physics*. Computing in High Energy Physics, 2001.

[19] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, January 1965.

[20] Aditya Nishandar. Grid-Fabric Interface for Job Management in SAM-Grid, A Distributed Data Handling and Job Management System for High Energy Physisc Experiments. Master's thesis, The University of Texas at Arlington, 2004.

[21] F. Pollack. New microarchitecture challenges in the coming generations of cmos process technologies. In *Proceedings of 32nd Annual International Symposium on Microarchitecture*. International Symposium on Microarchitecture, 1999.

[22] et. al. R. Alfieri. VOMS, an Authorization System for Virtual Organizations. *Proceedings of the 1st European Across Grids Conference*, 2003.

[23] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[24] A.S. Rana. A globally-distributed grid monitoring system to facilitate hpc at d0/sam-grid (design, development, implementation and deployment of a prototype). Master's thesis, The University of Texas at Arlington, 2002.

[25] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Security for Grid Services. *Twelfth International Symposium on High Performance Distributed Computing*, 2003.

[26] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. *3rd Annual PKI R&D Workshop*, 2004.

[27] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster. The Globus Striped GridFTP Framework and Server. In *Proceedings of Super Computing*, 2005.

BIOGRAPHICAL STATEMENT

Anoop Rajendra joined the University of Texas at Arlington in the Fall of 2003. He received his Bachelor's degree in Computer Science and Engineering from BMS College of Engineering, Bangalore, India, affiliated to the Vishweshvaraiah Technological University, India. He worked at the Fermi National Accelerator Laboratory in Batavia, IL as a software developer from September 2004 to August 2005. His academic interests include High Performance Computing and scientific computing.