MONITORING AND ANALYZING DISTRIBUTED CLUSTER PERFORMANCE AND STATISTICS OF ATLAS JOB FLOW

by

SREERANJANI RAMPRAKASH

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

August 2005

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to Mr. David Levine who has been constantly encouraging and directing me throughout my graduate studies and specifically through my thesis.

I am extremely grateful to Dr. Kaushik De whose ideas and suggestions have been the biggest motivation for my thesis work.

I would like to express my most heartfelt thanks to Dr. Mark Sosebee who has constantly helped in getting us all that we have needed to implement and guided us to achieve what we have.

I would like to thank Dr. Ramez Elmasri for his interest in my Research and thesis defense.

I am very thankful to Aaron Thor who has helped me in innumerable ways with his ideas as well as resourcefulness. It was indeed a very good experience working with him as a team in ultimately trying to build a scheduling algorithm for ATLAS.

I would like to also mention that my efforts were supported by Nurcan Ozturk, Sudhamsh Reddy and Nirmal Ranganathan and I would like to express my appreciation for their support.

I would like to thank my parents, my close family, my fiancé and my close friends with whose constant understanding and support I have completed my thesis.

July 22, 2005

ABSTRACT

MONITORING AND ANALYZING DISTRIBUTED CLUSTER PERFORMANCE AND STATISTICS OF ATLAS JOB FLOW

Publication No. ______

Sreeranjani Ramprakash, M.S.

The University of Texas at Arlington, 2005

Supervising Professor: David Levine

Grid3 is a Grid facility used by many High Energy Physics experiments to enable physicists to process data intensive and CPU intensive jobs more effectively as well as more efficiently. The worldwide High Energy Physics experiment, ATLAS collaboration, works on several Grid facilities in many countries, Grid3 being the US ATLAS Grid facility. The European DataTag and NorduGrid are other Grid facilities used by the ATLAS experiment. Amongst other things, the highlights of Grid3 are participation by more than 25 sites across the U.S. and Korea which collectively provide more than 2000 CPU's, resources used by seven different scientific applications, including three high energy physics simulations and four data analyses in high energy physics, bio-chemistry, astrophysics and astronomy, more than 100 individuals are currently registered with access to the Grid, a peak throughput of 500-900 jobs running concurrently with a completion efficiency of approximately 75%.

Since each application and organization utilizing Grids has different measures for efficiency and different parameters such as number of successfully completed jobs, turnaround time, number of idle processors, etc., to be considered for scheduling, scheduling on any Grid still needs to be tailored for individual cases.

The ATLAS experiment is a High Energy Physics experiment that utilizes the services of Grid3 now migrating to the Open Science Grid (OSG). This thesis provides monitoring and analysis of performance and statistical data from individual distributed clusters that combine to form the ATLAS Grid and will ultimately be used to make scheduling decisions on this Grid.

The system developed in this thesis uses a layered architecture such that predicted future developments or changes brought to the existing Grid infrastructure can easily utilize this work with minimum or no changes. The starting point of the system is based on the existing scheduling that is being done manually for ATLAS job flow. We have provided additional functionality based on the requirements of the High Energy Physics ATLAS team of physicists at UTA. The system developed in this thesis has successfully monitored and analyzed distributed cluster performance at three sites and is waiting for access to monitor data from three more sites.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 What is a Grid?

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [1]. Each one of the terms used above describes a particular aspect of a Grid. The *infrastructure* is in reference to the pool of resources with the hardware creating the necessary connections between these resources along with the software used to monitor and control this hardware. The *dependable* aspect assures users that they will receive a predictable and sustained level of performance from the Grid.

Computational infrastructure, like other infrastructures, is fractal, or self-similar at different scales [1]. We have networks between countries, organizations, clusters, and computers; between components of a computer; and even within a single component. A cluster or network of workstations is a collection of computers connected by a high-speed local area network and designed to be used as an integrated computing or data processing resource. A cluster, like an individual end system, is a homogeneous entity with its constituent systems differing in configuration and not hardware or software architecture, and is controlled by a single administrative entity that has complete control over each end system. However, at different scales, we often operate in different physical and often political regimes. For example, the access control solutions used for a laptop computer's system bus are probably not appropriate for a trans-Pacific cable [1].

A defining feature of computational Grids is that they involve sharing of networks, computers, and other resources. This sharing introduces challenging resource management problems in a variety of areas. Many of the applications running on Grids need to meet stringent end-to-end performance requirements such as maximum allowed turnaround time, amount of storage space consumed, reliability of the system, across multiple computational resources connected by heterogeneous, shared networks. To meet these requirements, we must provide improved methods for specifying application level requirements, for translating these requirements into computational resources and network-level quality-of-service parameters, and for arbitrating between conflicting demands.

1.2 Understanding Grids

To further understand these requirements, the authors in [2] have discussed the requirements of cross-organizational operation of Grids in terms of protocol architecture as depicted in Figure 1. The layers in this protocol architecture are then mapped to the more common Internet Protocol architecture. This architecture helps identify the protocols and services required to create a usable Grid. Each layer in the protocol architecture is briefly explained in the following section. Our focus in this document will be in the *Resource* and *Collective* layers.

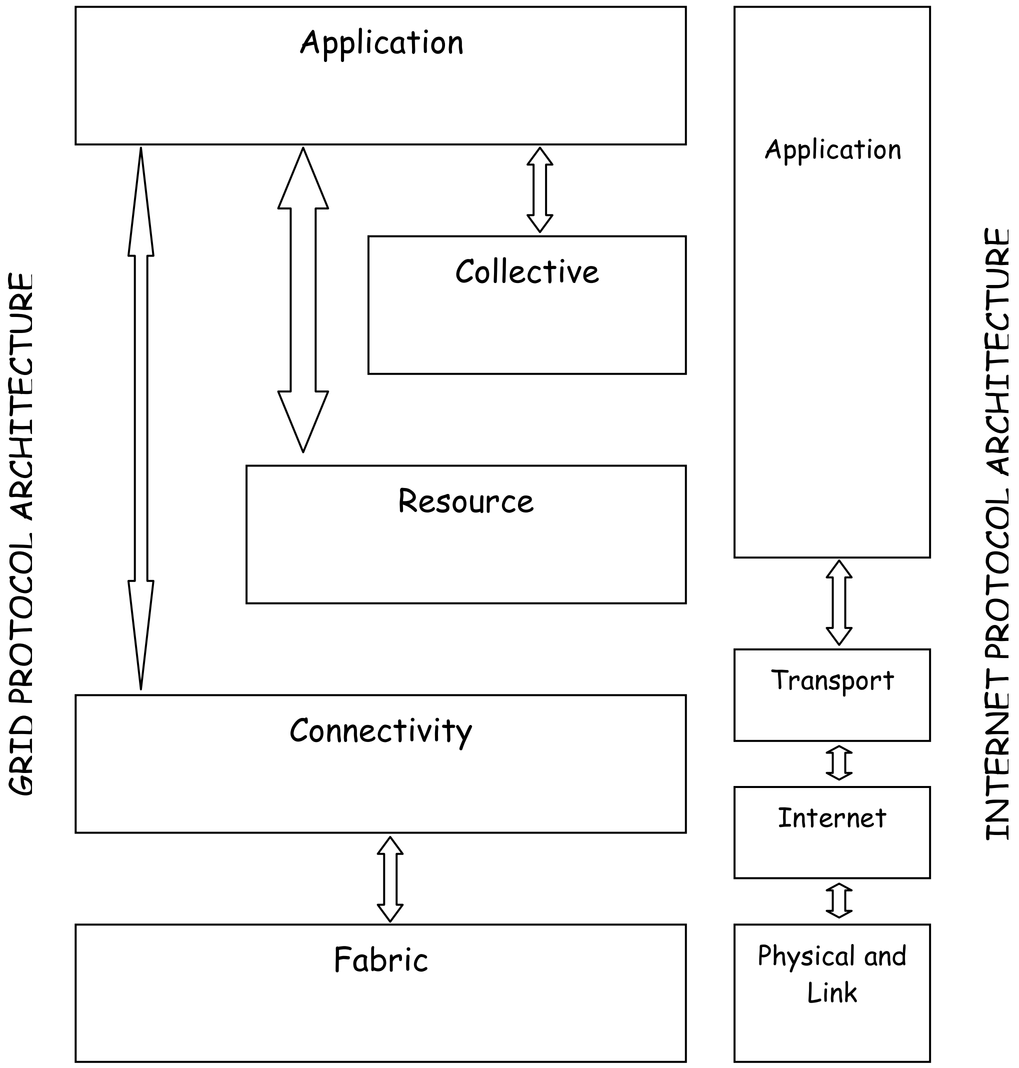


Figure 1 Layered Grid architecture and its corresponding protocols in the Internet architecture.

Fabric – This layer provides the actual resources that require shared access in a Grid. This includes computational resources, storage systems, catalog, network resources, sensors, etc. This could also mean resources like network file systems and clusters of computers, in which case this layer deals with internal protocols within a local domain. At the resource level enquiry or discovery mechanisms should be implemented along with resource management mechanisms to provide a minimum level of quality of service.

Connectivity – This layer defines communication and authentication protocols that are required for network transactions over Grids.

Resource – The main functions of this layer is to access and control local resources. This means that they do not deal with issues of global state across distributed collections of resources. Information protocols and Management protocols are the two primary classes of resource layer protocols. Information protocols obtain structure and state information, for example, configuration, current load, usage policy, etc. Management protocols negotiate access to shared resources, for example, specifying resource requirements in terms of advanced reservation and quality of service, process creation and data access.

Collective – This layer deals with interactions across collections of resources rather than any one specific or local resource. They can implement many sharing behaviors, for example, directory services, co-allocation, scheduling and brokering services, monitoring and diagnostic services, data replication services, workload management systems, software discovery services, community authorization servers, community accounting and payment services, collaborator services, etc.

Application – This layer comprises the user applications which are constructed in terms of the services defined in the layers below this one in the protocol stack.

CHAPTER 2

PREVIOUS WORK IN GRID SCHEDULING

2.1 Distributed job scheduling on computational Grids using multiple simultaneous requests

In [3] the authors propose a meta scheduling scheme that is based on a distributed scheduling model.

Scheduling can be on varying levels and can be categorized in a hierarchical fashion. At the lowest, we have CPU scheduling which is done at the operating system level. Next in the chain we have the batch scheduling done at a cluster level. Here, the scheduling involves assigning jobs to individual nodes. From there, the OS scheduler takes over the scheduling of the job. Next up we have the concept of meta scheduler. This is what is aware of the Grid and accepts jobs from other, similar meta schedulers from other sites and then submits the jobs to the batch scheduler at the cluster.

The distributed scheme puts a meta scheduler at every site and jobs are submitted to the local meta scheduler where the job originates. Each of these meta schedulers query each other for load information periodically, and if any of the other sites has a lower load then the job is transferred to the site with the lowest load. The proposed scheme starts by keeping the total number of sites as N, where N can be varied. The scheduler then is a K-distributed model where the job submitted to the local meta scheduler is sent to each of the K least loaded sites out of the N total number of sites. When a job is able to start at any site, the meta scheduler at that site informs the meta scheduler at the originating site which in turn informs the other K-1 sites requesting them to cancel the job. The constraint on this is that the notification needs to be atomic to ensure that processor cycles are not wasted. Also, a high degree of overbooking is done at each site which in turn increases the work done at each local scheduler. The amount of inter-site communication also increases with K.

This scheme would not work in the ATLAS Grid scenario due to several reasons. Apart from the drawbacks in the algorithm itself, ATLAS jobs are too large to be transferred to more locations than one only to be dropped when one of the sites processes them. Also the global state that needs to be maintained across sites increases overhead. The authors have used a system with four sites to demonstrate and with the number of sites in ATLAS steadily increasing this algorithm would soon not scale.

2.2 Multisite resource selection and scheduling algorithm on computational Grid

In [4] the authors propose a clustering-based Grid resource selection algorithm that proposes a multi site resource selection scheme for synchronous iterative application. These applications are iterative in nature, with each iteration separated from the previous and subsequent iterations by a synchronization operation. Examples of synchronous iterative algorithms include simulations and many image processing and data classification algorithms.

Here they cluster resources in a computational pool based on network delays and descending order of computational capacity. Then candidate schedules are generated and evaluated from the Grid computing resources, which can be non-dedicated, homogenous or heterogeneous workstations or personal computers. In the end, a single resource selection is suggested in the case of invalidation of multisite resource selection algorithm.

This algorithm will not work with ATLAS for two reasons. First, clustering Grid resources on the basis of computational capacity and network delays will not be possible since sites in the ATLAS Grid are not pooled resources but distributed and already geographically and politically clustered resources. Second, ATLAS applications are simulations based on specific input data sets, with these data sets typically in the order of a Gigabyte in size. In such a scenario, the simulation, even if it is synchronous iterative, cannot be run independently on different clusters. This property can be used within a cluster to schedule on different nodes simultaneously but due to the huge overhead of moving large input files cannot be used on the scale of a Grid to schedule on different clusters simultaneously.

2.3 A comparison among Grid scheduling algorithms for independent coarse-grained tasks

In [5] the authors propose an algorithm called RR (named after its Round Robin influence). At the beginning of the scheduling, every processor is assigned exactly one task. If some task is completed then RR receives the result of the task and assigns a yet unassigned task to the processor. With *m* tasks remaining uncompleted, this is how RR manages these tasks: if some task *v* is completed on a processor, then RR receives the result of *v* and kills all task instances of *v* running on processors except the one processing *v* currently, selects a task *u* in a round robin fashion and replicates *u* onto the processor that just processed *v*.

Coarse grained tasks can be defined as larger tasks that can be partitioned further but by doing so, we will be making the task finer grained than the original. Round Robin schemes work well with coarse grained tasks.

The reason RR would not work on the ATLAS Grid is because the task lengths and the number of processors required to execute the tasks is information that is needed before hand. Also, this algorithm works on the assumption that all tasks (jobs in our case) are known before scheduling decisions are made.

2.4 The Globus Project: A status report

In order to better understand this scheduler, we will have to further explore the Globus Toolkit. The open source Globus Toolkit is fundamental enabling technologies for the Grid, letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy. The toolkit includes software services and libraries for resource monitoring, discovery, and management, plus security and file management[23]. The various components of the Globus Toolkit can be broadly classified into security, data management, execution management, information services and common runtime.

In [6] the component of interest in our case from The Globus Toolkit is execution management provided by GRAM – Globus Resource Allocation Manager. Here, the idea is that there are several GRAM's running, each responsible for local sites, interfacing with an underlying resource management tool such as LSF (Load Sharing Facility) or Condor. GRAM provides a standard network-enabled interface to local resource management systems [23]. Grid tools can therefore manage resource allocation using a standard API, expressed with the use of a resource specification language (RSL). At the higher level, resource brokers use RSL to specify requirements such as MFLOP's (Millions of Floating point Operations), and even the hardware resources required.

The Globus Resource Allocation and Management services of the Globus Toolkit cannot be used for scheduling on the ATLAS Grid since it does not provide scheduling or resource brokering capabilities, but merely acts as an API between meta-schedulers or brokers and local resource management mechanisms.

Condor-G is another system that uses GRAM from the Globus Toolkit built on top of Condor running locally on each site to schedule on the Grid. Since it is fundamentally using the GRAM component of the Globus Toolkit, it carries the above mentioned constraints and hence does not meet the requirements for an ATLAS scheduler.

2.5 Chameleon: A resource scheduler in a data Grid environment

In [7] the authors propose a detailed scheduling model for a data Grid environment considering the following parameters: network bandwidth, number of available nodes, system attributes at each site, size of input data, size of application code, and size of produced output data. The cost models are calculated for five different scenarios: local data with local execution, local data with remote execution, remote data with local execution, remote data with same remote execution and remote data with different remote execution. The response time to the users' job request is calculated in each scenario factoring each of the parameters outlined before along with expected execution times.

This particular scheduler, though it is the closest to what ATLAS needs, still does not consider some of the prime factors in job scheduling in ATLAS. These factors are number of failed jobs at a site and utilization at any site. It also doesn’t factor contributions to load at a site from other job flows apart from ATLAS job flow.

CHAPTER 3

ATLAS AND THE UNDERLYING PROBLEM

The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [2]. The sharing that we are concerned with here is not just file exchange, but also direct access to computers, software, data, and other resources, as is required by the collaborative problem-solving environment of ATLAS, a High Energy Physics collaboration. This sharing is controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization (VO) [2].

The ATLAS collaboration forms one such VO. The *A* *T*orroidal *L*HC *A*pparatu*S*, ATLAS, experiment is a $500 million plus High Energy Physics experiment that is due to launch in the year 2007. It is a collaboration of about 2000 physicists participating from 150 universities and laboratories in 34 countries. At its peak operation, the ATLAS experiment is expected to generate one Petabyte of “raw” data per year which is used in chunks of up to Gigabytes of inputs to execution jobs. Several clusters from universities and laboratories combine their computing, storage and networking resources during the analysis phase to form a Grid that will be used to reconstruct and analyze this data.

ATLAS is currently supporting the worldwide *L*HC *C*omputing *G*rid (LCG) project [28] and two main Regional Grid projects, NorduGrid [27] and the US Grid (also called Grid3). The LCG project is the main Grid project supported by all LHC experiments. It aims to build a worldwide, production-class computing Grid. It is built upon the developments of the European Data Grid middleware, the US Virtual Data Toolkit project and European DataTag monitoring tools [29]. The NorduGrid project is established mainly across Nordic countries (Denmark, Norway, Sweden, Finland) and continues to expand. The Grid3 collaboration is a data Grid with dozens of sites and thousands of processors. The facility is operated jointly by the U.S. Grid projects iVDGL, GriPhyN and PPDG, and the U.S. participants in the LHC experiments ATLAS and CMS.

Challenging issues in data-intensive applications are the scheduling and configuration of complex, high-volume data and job flows through multiple levels of hierarchy. Poor performance, as perceived by a user, can be due to an inappropriate algorithm, poor load balancing, inappropriate choice of communication protocol, contention for resources, or a faulty router.

The challenge in any resource allocation algorithm or methodology is to determine which one among several conflicting parameter needs to be focused on in any given application environment. For example, concentrating on maximizing throughput along with minimizing turnaround time will result in a bias favoring shorter duration jobs, since these will utilize resources to a fine amount of granularity and being of shorter duration will reduce turnaround time. For example, ATLAS jobs can vary in size of input data, execution time caused by different reconstruction jobs or simulation jobs, and can result in jobs that have smaller input files being transferred or those that are of shorter duration being preferred so as to improve turnaround time at any site. As a result, the resource allocation algorithm becomes unfair to longer running jobs.

To resolve resource sharing issues, Grid schedulers are used in such Grid environments. There are several Grid schedulers that can be used to schedule jobs in a Grid environment. The ATLAS experiment requires that scheduling be done based on the turnaround time of jobs combined with effective CPU and memory utilization at individual sites and percentage of failed jobs at a site. Current Grid schedulers do not meet this requirement. For example, the Globus Resource Allocation and Management services of the Globus Toolkit cannot be used for this purpose since it does not provide scheduling or resource brokering capabilities, but merely acts as an API between meta-schedulers or brokers and local management mechanisms. Another example is the Chameleon resource scheduler in a data Grid environment described in section 2.5 above that considers both the computational and data storage resources but does not have provisions to consider turnaround time or number of failed jobs at a site. A site can be defined as a cluster of nodes or processors or a set of resources that are set aside by a laboratory or an educational institution to contribute to the computing power of a Grid.

The goal of a Grid scheduler for the ATLAS experiment would be to minimize turnaround time while ensuring optimum utilization at all sites that form the Grid.

CHAPTER 4

MONITORING AND ANALYZING DISTRIBUTED CLUSTER PERFORMANCE AND STATISTICS OF ATLAS JOB FLOW

4.1 <u>Introducing Capone</u>

The ATLAS Grid Computing Environment is a set of different tools designed to provide a simple and uniform computing environment for ATLAS scientists using the Grid for their production and analysis activity. In the U.S Grid3, this Grid Computing Environment or Grid Component Environment (GCE) is also called Capone.

In this section we will explore the architecture and design of Capone so as to establish the need for the scheduling provided by this thesis. In order to understand the design of Capone we will step slightly away from Capone to an important element that needs to be defined, the RLS client utilities of The Globus Toolkit.

The Replica Location Service (RLS) [19] maintains and provides access to mapping information from logical names for data items to target names. Replication of data items can reduce access latency, improve data locality, and increase robustness, scalability and performance for distributed applications. An RLS typically does not operate in isolation, but functions as one component of a data Grid architecture. RLS implementation depends on a number of factors such as consistent local state being maintained in Local Replica Catalogs, Replica Location Indices, etc.

The Grid Component Environment comprises two main component modules, the Supervisor, called Windmill [8], and the Executor [8], which is Capone. The Capone Executor system is designed to implement job requests from the ATLAS production supervisor Windmill, to be executed on Grid3. The Grid Component Environment is a suite of client and server tools that provide a basis for the set of Python classes that form Capone. Capone is distributed using a packaging tool called Pacman [30]. The GCE include the following tools:

- GCE-Client
    - The Virtual Data Toolkit [20]
        - Chimera Virtual data system [21]
    - RLS client utilities and scripts
- GCE-Server
    - ATLAS software releases
    - Job transformation scripts
    - Kickstart – a Chimera supplied package which wraps the remote application and returns exit codes to the client (submit) host

To understand the working of Capone we will first discuss the design principles and then describe its various components as shown in Figure 2.

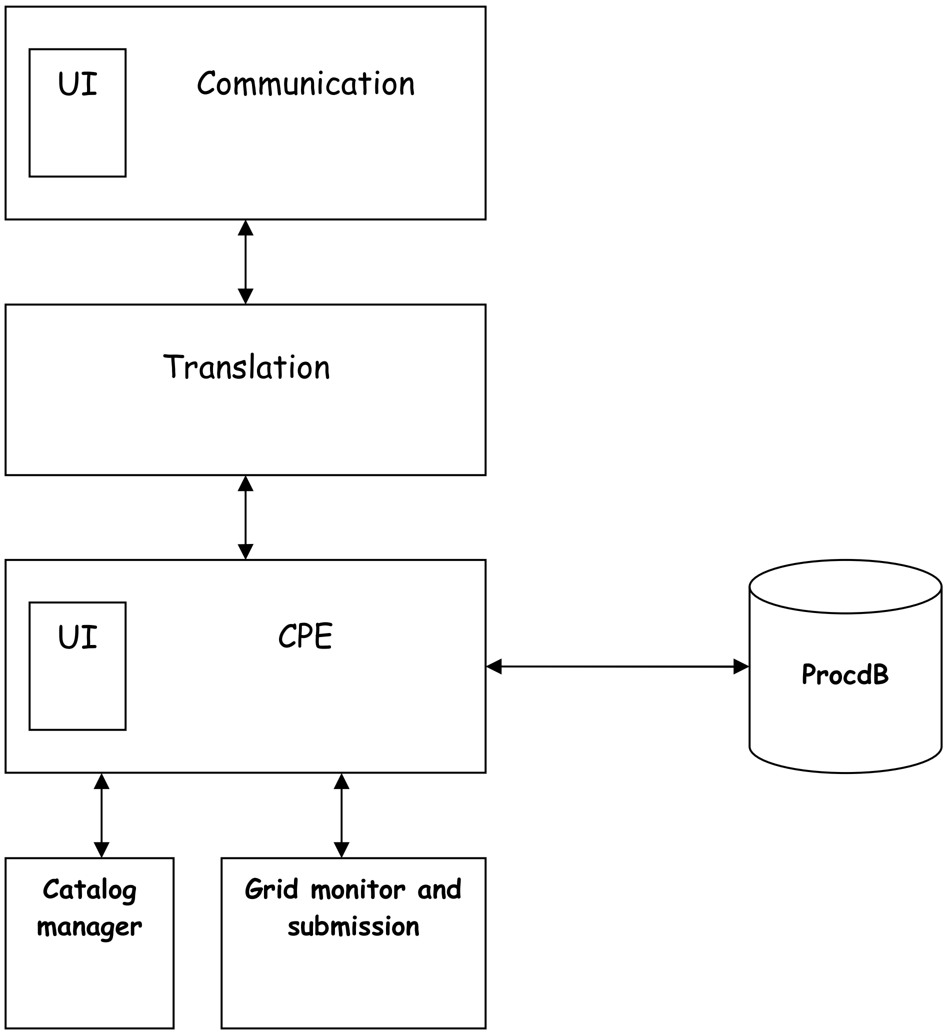


Figure 2 Various components of GCE Capone [8].

The main design principles are:

a) The Capone Process Engine, hereafter referred to as CPE, handles all communication. This means it separates the communication between the message passing interface and the lower level Grid communication protocols.
b) Grid-level process executions are initiated and handled by the CPE.
c) Grid-level error handling is done by the CPE. This includes querying all job monitors, queue monitors, etc., and logging these queries.
d) A database called ProcDB that stores the states of all processes currently managed by the CPE.
e) A query on the state of a process results in varying levels of detail depending on what information is requested.
f) The CPE should be able to handle interaction in the form of messages from external sources like the Supervisor or a direct user interface.

Figure 2 comprises these components:

a) CPE (main process engine) module – This is a finite state machine that performs the following actions:
    - At startup, it listens for messages from other modules
    - Receives messages from the communication module
    - Set of actions for each message received
    - Return to listening for messages
    - While exiting, cleanup and then exit gracefully

b) Communication module – This module again provides the following steps of action:

   - Request messages in terms of number of jobs requested, execute certain process, get data involved with a process execution, get the status of a process, exit and shutdown the module, etc.
   - Passing messages to external entities like the production Supervisor module, etc.
   - Passing messages to or through web services, Python scripts and graphical user interfaces, etc.

c) Translator module – This module translates from the external request language to the internal representation used in the CPE. There are two kinds of request languages used; they are: ATLAS supervisor requests, and the ADA Job Description Language known as AJDL.

d) Execution module – There are three main tasks of the executor module. They are:

   - Submitting jobs to the Grid
   - Registering the jobs into the catalog; and
   - Execution on the local machine

e) Catalog interface module – This module communicates with catalogs such as the component in Don Quixote that deals with cataloging [26] and the Globus RLS module

f) Process state database (ProcDB) module – This module records two kinds of information. They are:

- Information about the state of the currently executing module. This can be in one of many forms, like a text file, a MySQL database or memory structures with log files for error recovery.
- Information from the executing process about job related parameters and catalog related information.

g) User interface module to Capone's communication module (*eg*., template framework provided by the Supervisor team)

h) User interface to the CPE

i) User interface to the ProcDB

The working of Capone can be summarized using the following algorithm:

A. Supervisor communicates with Capone by
    i. asking the Executor module how many jobs it can execute
    ii. Executor replies with availability
    iii. Supervisor sends that many jobs to Executor

B. Executor module checks the input

C. Executor registers input with the catalogs

D. Executor translates input to required format

E. Proceed through execution steps by
    i. taking input in given format
    ii. generating output
    iii. registering output with the Virtual Data Toolkit (VDT) catalog

F. Verify output

G. Register the output with the ATLAS catalog, Don Quixote

H. Clean the VDT environment by

    i. freeing Grid resources utilized

    ii. removing registered entries from VDT catalog

I. Return to step A.

4.2 Design specifications and considerations

4.2.1 Overview

The purpose of this section is to familiarize the reader with the monitoring and analyzing of distributed cluster performance and statistics in ATLAS job flow in terms of design considerations, constraints, assumptions and dependencies. The intended audience is mainly developers who will extend the functionality of the monitoring system provided by this thesis. The scope of this document is extended to this thesis as well as work used to build this thesis. The following section discusses the architecture of the monitoring and analyzing system for ATLAS job flow implemented in this thesis.

4.2.2 Assumptions

To better define the system built in this thesis let us first look at the list of assumptions that have been made before building the system. A list of the assumptions, in no particular order, is:

i. Sources of information are limited to Ganglia [10], Monalisa [24] and Capone [8].
ii. Ganglia and Monalisa are configured correctly and transmit up-to-date information.
iii. Only error codes from Condor-G listed on the web site (Capone error codes) are considered in the code; all other error codes are treated as generic errors
iv. The system is built on the assumption that the current methodology of collection of information is the best method for ATLAS information collection
v. The system is built assuming that the Site where the system runs has access to all Ganglia information of other sites through the daemons gmond and gmetad (as XML data files)
vi. System has access to Capone log files from where information about jobs running on a submit host is collected
vii. Configuration file is updated every time a configuration change occurs at any site by the site administrator
viii. Capone updates to the log file accessed by system is accurate and complete

4.2.3 System architecture

The high level architecture of the system is shown in Figure 3. In order to fully understand the architecture, let us explore an element in the architecture, *Ganglia,* in a little more detail. *Ganglia* is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids [10]. It is based on a hierarchical design targeted at organizations of clusters such as the Virtual Organizations mentioned earlier in this document.

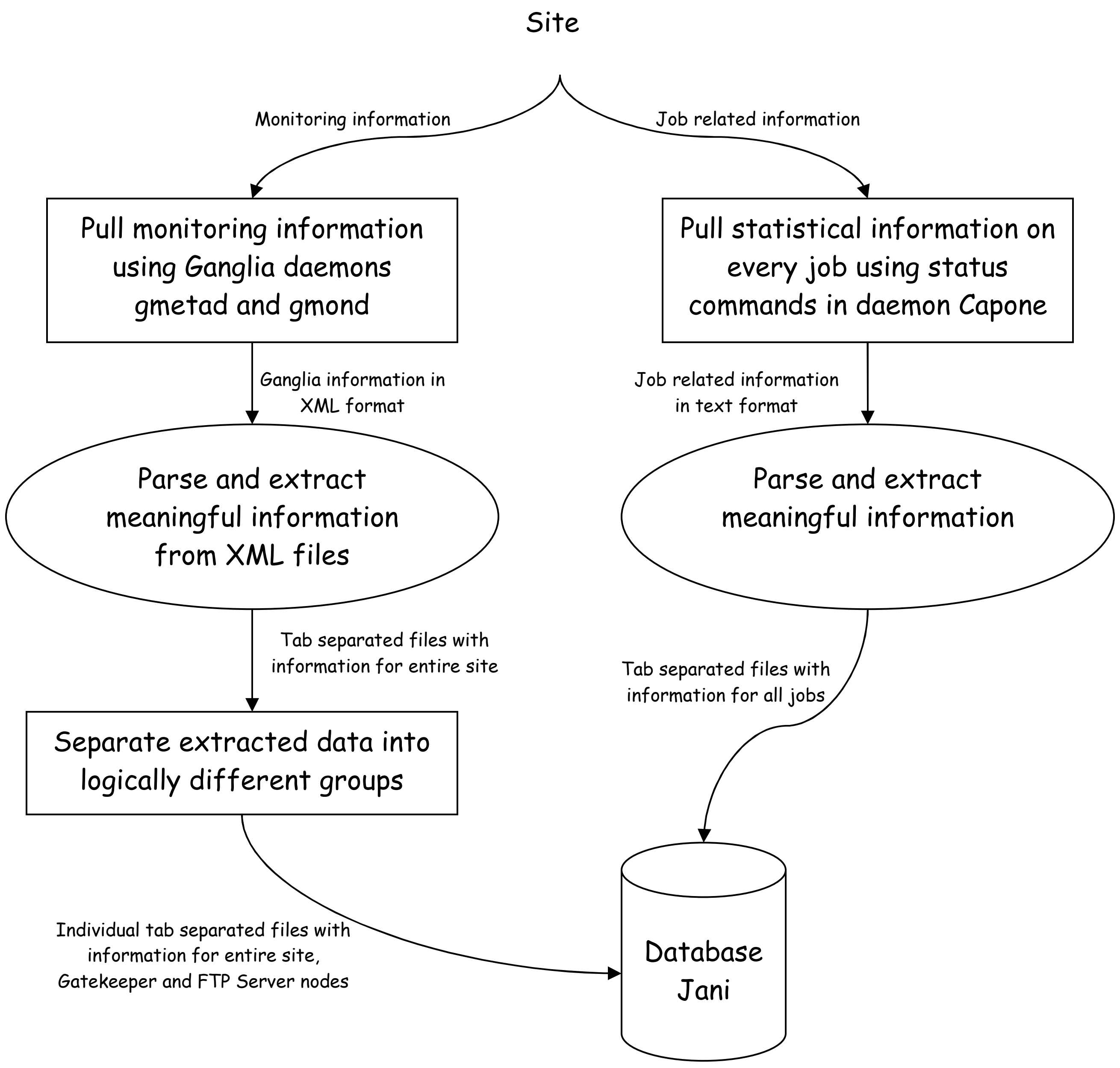

Site
Monitoring information
Job related information
Pull monitoring information using Ganglia daemons gmetad and gmond
Pull statistical information on every job using status commands in daemon Capone
Ganglia information in XML format
Job related information in text format
Parse and extract meaningful information from XML files
Parse and extract meaningful information
Tab separated files with information for entire site
Tab separated files with information for all jobs
Separate extracted data into logically different groups
Individual tab separated files with information for entire site, Gatekeeper and FTP Server nodes
Database Jani

Figure 3 High level architecture of the monitoring and analyzing distributed cluster performance and statistics of ATLAS job flow.

It leverages widely used technologies such as XML (eXtended Markup Language) for data representation, XDR (eXternal Data Representation – a data representation standard) for compact, portable data transport, and RRDtool (Round Robin Database tool) for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on over 500 clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes [10].

Another monitoring tool that needs to be discussed in a little more detail is *MonALISA* – MONitoring Agent using a Large Integrated Services Architecture. The *MonALISA* framework provides a distributed monitoring service system using JINI/JAVA and WSDL/SOAP technologies. The goal is to provide the monitoring information from large and distributed systems to a set of loosely coupled "higher level services" in a flexible, self describing way. This is part of a loosely coupled service architectural model to perform effective resource utilization in large, heterogeneous distributed centers. The framework can integrate existing monitoring tools and procedures to collect parameters describing computational nodes, applications and network performance. [24]

In Figure 3, we start from a site and gather data using two different means. The first, immediately below the site and to its left in the figure, is data from the site using the Ganglia daemon gmond. Here, the data is pulled from the site using telnet, and is obtained in the form of an XML dump file. The second, immediately below the site and to its right in the figure, is data from the site using the Capone daemon. Here, the data is pulled from the site using Capone commands, and is piped into a text file.

Another important source of information and its role in the architecture of this system is shown in Figure 4. This is a configuration file describing all available ATLAS Grid3 sites in the form of an XML document named “pool.config”. This file updates only when there is a change in the way resources are handled at a site.

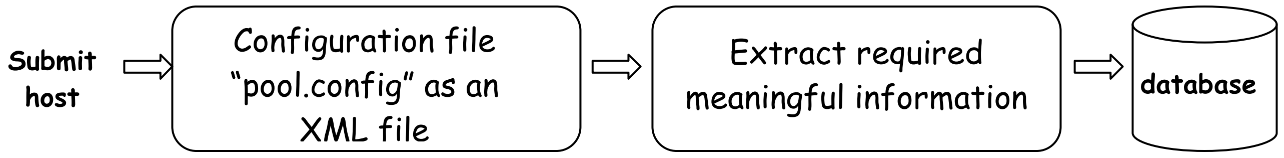

Submit host
Configuration file "pool.config" as an XML file
Extract required meaningful information
database

Figure 4 Information flow from configuration file "pool.config" XML file to the database.

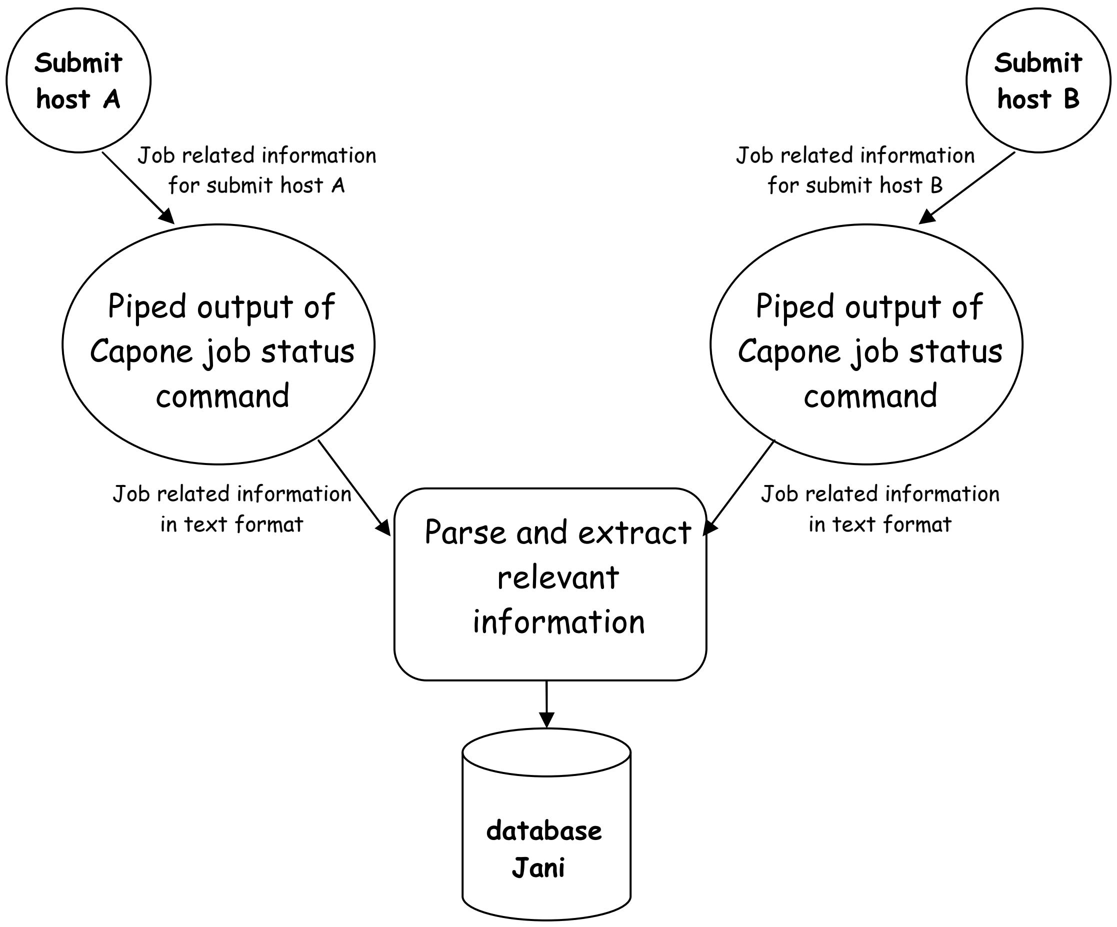

Submit host A
Submit host B
Job related information for submit host A
Job related information for submit host B
Piped output of Capone job status command
Piped output of Capone job status command
Job related information in text format
Job related information in text format
Parse and extract relevant information
database Jani

Figure 5 Information flow from different submit hosts to the database through Capone "job status" commands.

For example, if a site moves its entry node (known as the gatekeeper node) then this configuration file would be updated. This implies that this update to the database is manual and that it occurs when a user is aware of a change in the configuration file.

To get to the next level of architecture, consider the left half of Figure 3. This is depicted as Figure 5. In Figure 5, there are multiple submit hosts from which statistical information about jobs are pulled separately using commands in the Capone daemon such as “job status” commands. These are piped to text files and then parsed to extract the information that we require. This information is then stored in the database in the form of a table.

Next, let us consider the right half of Figure 3. This is depicted as Figure 6. In Figure 6, the Gatekeeper at a Site is queried using telnet to report Ganglia resource information in the form of an XML file. There are two parts to this XML file that interest us. The first, shown on the left side in Figure 6, is the almost static information like the number of CPU’s, the average CPU speed, the total amount of memory, the total amount of disk space, etc. This information is pulled and extracted from each site manually and only when the configuration file mentioned previously shows a change in the amount of resources at a site.

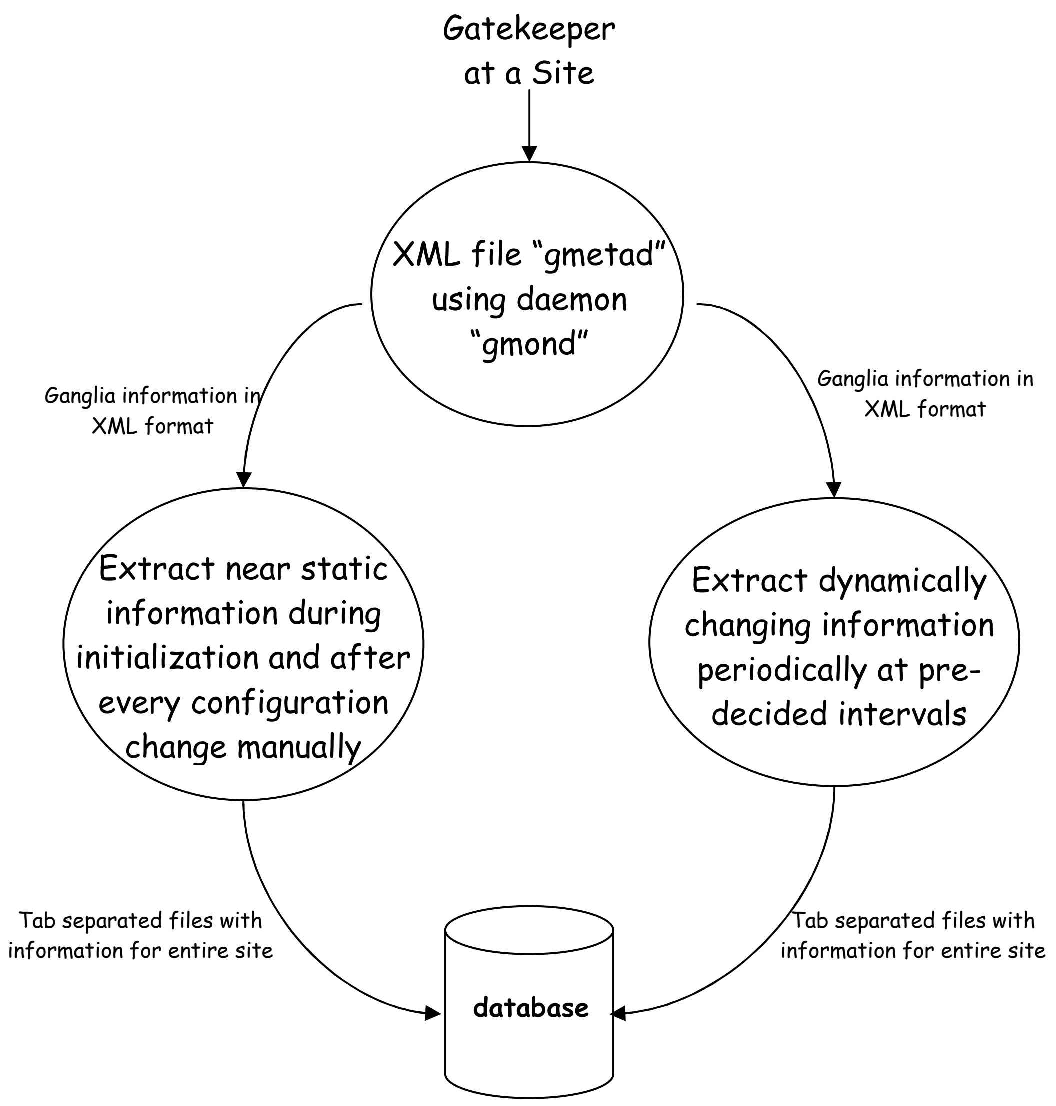


Figure 6 Separating and extracting information from Ganglia dump files into static and dynamic information.

The second, shown on the right side in Figure 6, is the dynamically changing information such that the CPU idle time, amount of disk space free at this time, amount of memory free at this time, average load over the past 15 minutes, average load over the past 5 minutes, the number of processes running, etc. This information is pulled and extracted at periodic intervals, the time between pulls being decided based on the frequency of change noticed in the information being extracted.

An important factor that we needed to factor into our design of the system is that the Gatekeeper node as well as the node running the FTP server being overloaded would slow down processing of jobs at any particular site. This results mainly due to the fact that we are considering the *average* amount of resources available at any site. If the Gatekeeper node was at its peak operation and its resources were almost completely taken then the number of jobs that can enter the site will be low whether or not resources are available within the site.

To better understand this concept let us first look at what a typical site in the Grid looks like. This is depicted in Figure 7. In Figure 7 we start first by entering a site from any other element on the Grid through the internet and intranet infrastructure. The entry point into any cluster is through either the Gatekeeper or through the FTP Server nodes. The Gatekeeper node deals with the scheduling of jobs at the cluster level. This also means that all jobs sent to execute at the cluster need to pass through this node. After this phase (also called Stage-In) the job is sent to the node that it needs to execute on by the gatekeeper node through the internal network in the cluster. The FTP Server node deals with obtaining the data sets from replication servers that store these data sets. The jobs that have been sent to these sites to be executed will have certain data sets that they will need to execute. These data sets are located and brought into the cluster or site through the FTP Server node.

As we can see in Figure 7, the Gatekeeper node and the FTP Server nodes become the bottlenecks in jobs being executed at any cluster or site. Keeping this in mind, scheduling decisions need to be taken after reviewing the load on each of these nodes at any site, before sending jobs or batches of jobs to the site.

Resource information for the Gatekeeper node as well as for the FTP Server node can be found in the Ganglia XML dump file “gmetad” for that site. To be able to extract this information we need to be aware of the name of the gatekeeper and FTP server nodes. This information can be obtained from the configuration file. For each site there will be an entry giving the Fully Qualified Domain Name (FQDN) of the Gatekeeper node and another entry with the FTP Server node.

Figure 8 shows how we can extract resource information for the gatekeeper node using Ganglia information and the configuration file “pool.config”. In Figure 8 we start from the XML file “gmetad” and obtain the site name from the “cluster” tag value in the XML file.

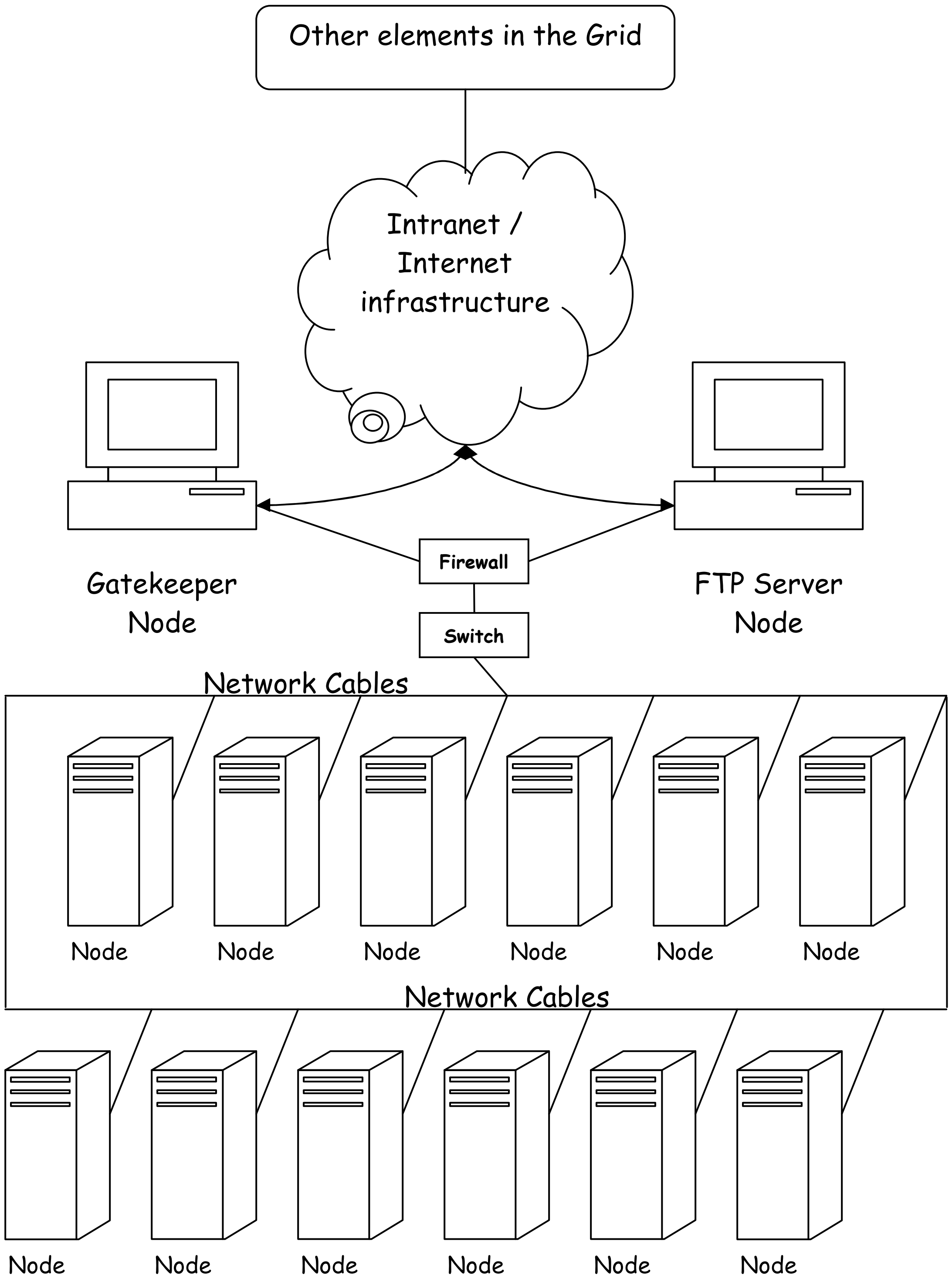


Figure 7 Internal topology of a typical site in the Grid.

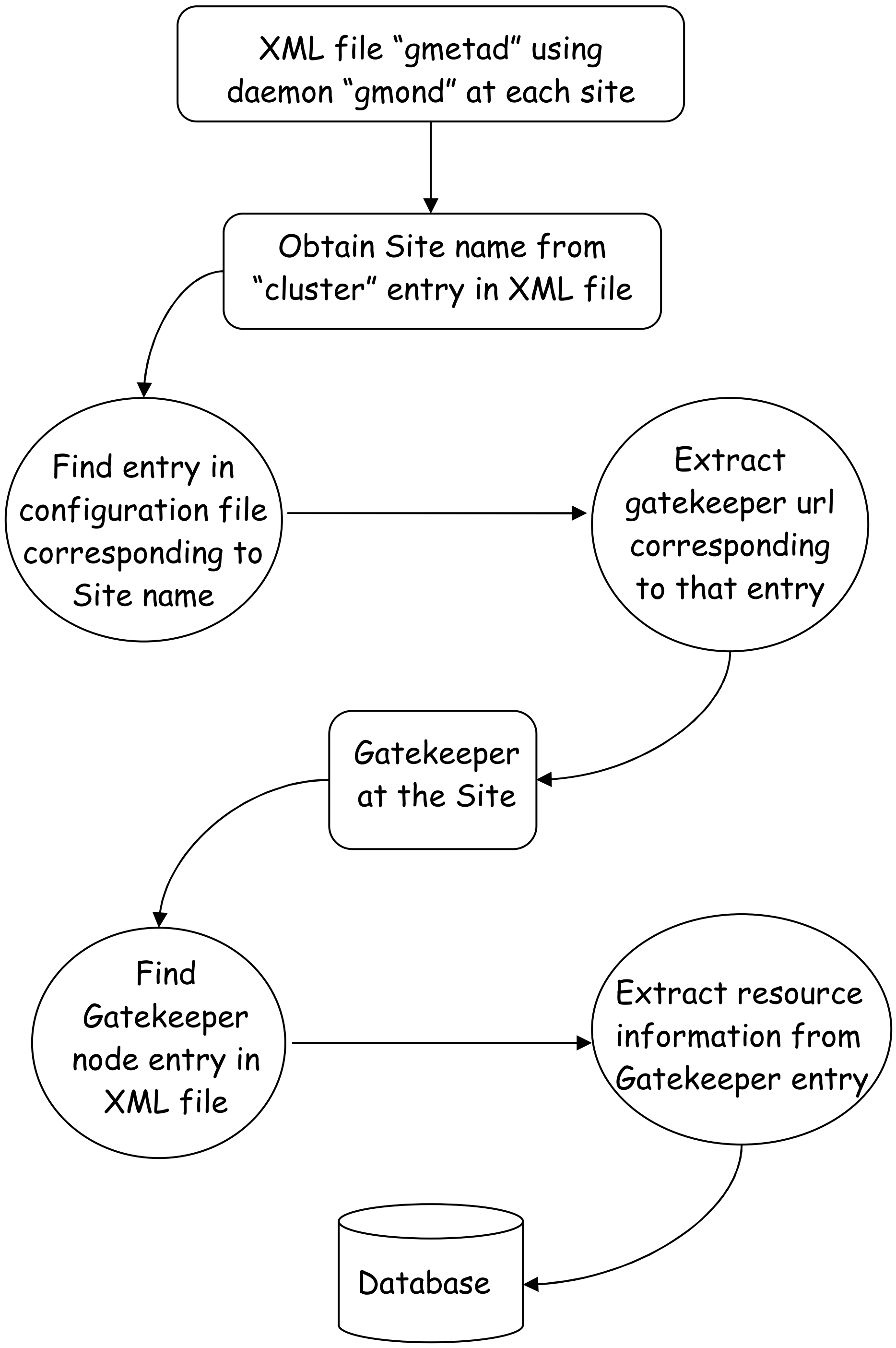

XML file "gmetad" using daemon "gmond" at each site
Obtain Site name from "cluster" entry in XML file
Find entry in configuration file corresponding to Site name
Extract gatekeeper url corresponding to that entry
Gatekeeper at the Site
Find Gatekeeper node entry in XML file
Extract resource information from Gatekeeper entry
Database

Figure 8 Extracting Gatekeeper dynamic information through Ganglia using configuration file "pool.config". Sequence of operations with passing time.

Next, we need to find the configuration file entry corresponding to this site name. As an aside note, we see that we will need a mapping between the site name according to Ganglia and the site name according to the configuration as they are not uniform across both locations. Once we find the site entry in the configuration file we will extract the gatekeeper URL from the "jobmanager universe" tag entry in the configuration file. Again, as an aside, since one URL can have several aliases we will need a mapping between the names obtained from the configuration file onto the "host name" used in the Ganglia dump file. Once we find the entry in the XML file corresponding to the gatekeeper node we can extract all the resource information associated with that node. The next step will be to input all these values into the table in the database that stores this information.

Though Figure 8 is the design for extracting Gatekeeper information from the Ganglia dump files, we will do the exact same thing to extract FTP Server node information. The only difference will be that we will look for FTP Server node address in the configuration file using the "jobmanager transfer" tag entry. Also, the resource information for the FTP Server node at each site is stored in a separate table in the database.

4.2 Another point of view

Currently Capone handles Grid scheduling by an operator modifying weights associated with various sites, in an initialization file in Capone. These changes are based on graphical information about these sites obtained from sources such as Ganglia / MonALISA. This is graphical display of information about parameters such as queue length, CPU utilization, memory utilization and related resource information at sites.

The goal of this scheduler is to automate the modification process based on a combination of turnaround time of jobs, CPU utilization at sites and number of failed jobs at a site. (Other related parameters will also be factored into the scheduler).

The Grid centric (information about the Grid elements) and job centric (information about the actual jobs being executed) monitors used in modifying weights in Capone by an operator are:

1. Job information obtained for a site for ATLAS jobs collectively from MonALISA (both CPU and data transfer information)
2. Overall view of job information for the Grid.

Other monitors that can be utilized in automating the modifications are:

1. Job information obtained directly from Capone regarding status of jobs
2. Information about load on gatekeeper and FTP server at any site. The gatekeeper and FTP server URL's can be obtained from configuration files.

The main factors in terms of ATLAS jobs considered in the design of this thesis can be listed as:

i. number of running jobs vs. number of idle jobs – for example in ATLAS job flow the important thing is to have at the least one idle job waiting in the queue for every job that is running. This ensures that CPUs are almost never idle.
ii. number of jobs run vs. total number of jobs present
iii. number of jobs run vs. number of failed jobs
iv. efficiency, in terms of successfully executed jobs, at a site vs. how many jobs have been run at the site

The goal of a scheduler for ATLAS would be to keep factors above such that the number of idle jobs at a site is at least equal to the number of running jobs at that site so that no CPU at that site is idle.

This leads to the requirement to build an automated system that takes scheduling decisions based on information currently being used, in combination with further resources, to minimize turnaround time of a job while optimizing CPU/memory utilization at a site, and modifying weights associated with sites in Capone based on this scheduling decision.

The main design considerations in building such a system involve

1. Designing a schema for a database that contains both static as well as dynamic information for Sites, their services, their resources, and jobs with their statuses, and the meaning of their statuses.
2. Populating the database with static and dynamic values as well as updating the database at decided intervals with dynamic information.
3. Analyzing key factors from the database to weight sites to be able to make scheduling decisions based on these weights.

In this section we will look at the architecture of the system from a slightly different perspective. Since the monitoring and analyzing is done using stored values in a database, we will look at the system from a database point of view.

The database is designed with a set of static (infrequently changing) tables and a set of dynamic (frequently changing) tables. The static tables serve as lookup tables used in comparing parameters from the dynamic tables. The schemas for the static and dynamic tables are shown diagrammatically in Tables 1 and Table 2, respectively. Each of the tables below is populated from different sources using scripts. The source for each of these tables' data is indicated below each table in the diagram.

The database is stored on a node accessible to the host running Capone. Each of these tables is updated periodically depending on how frequently the data in the table changes.

Table1. Static tables in database.

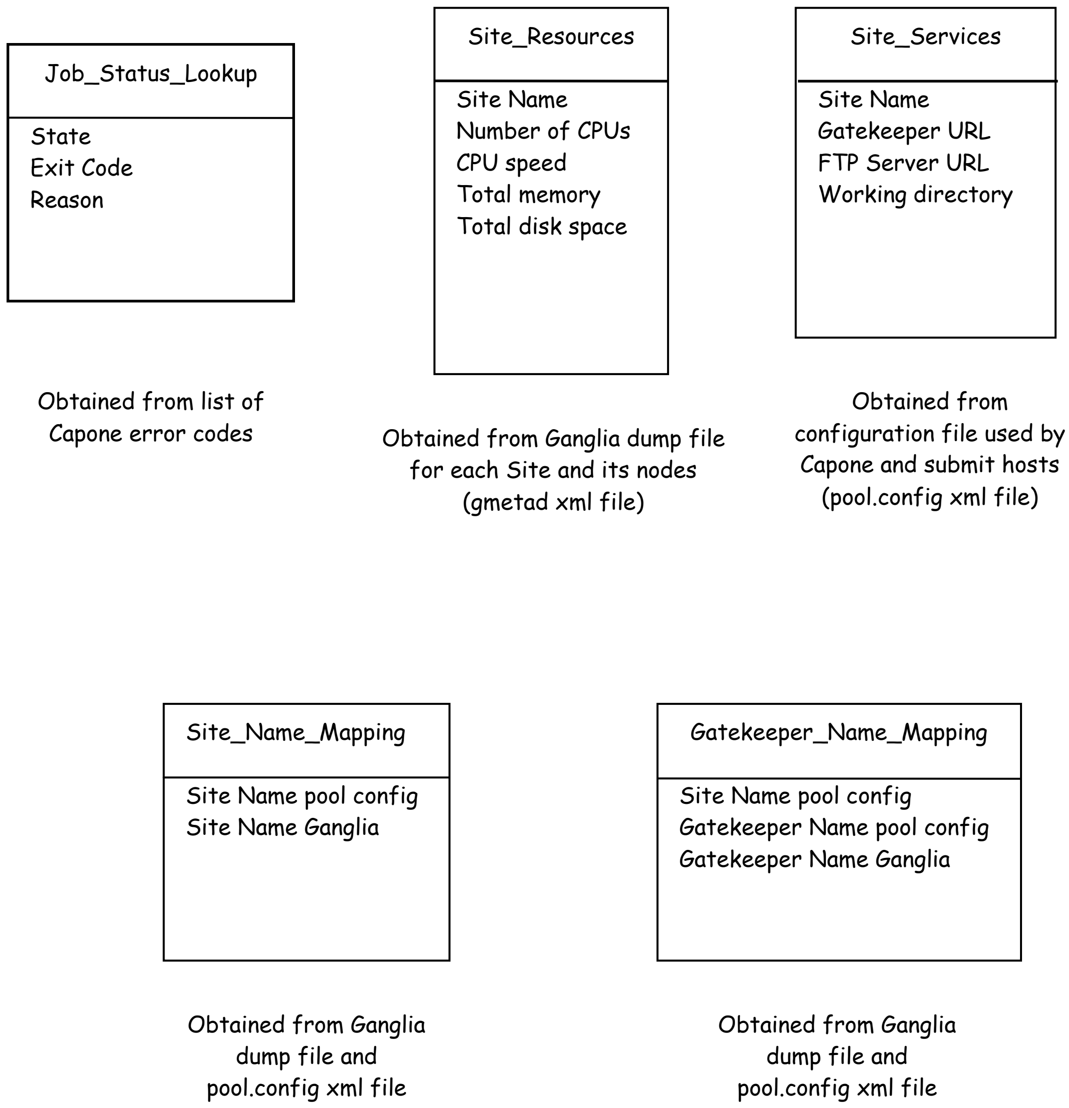
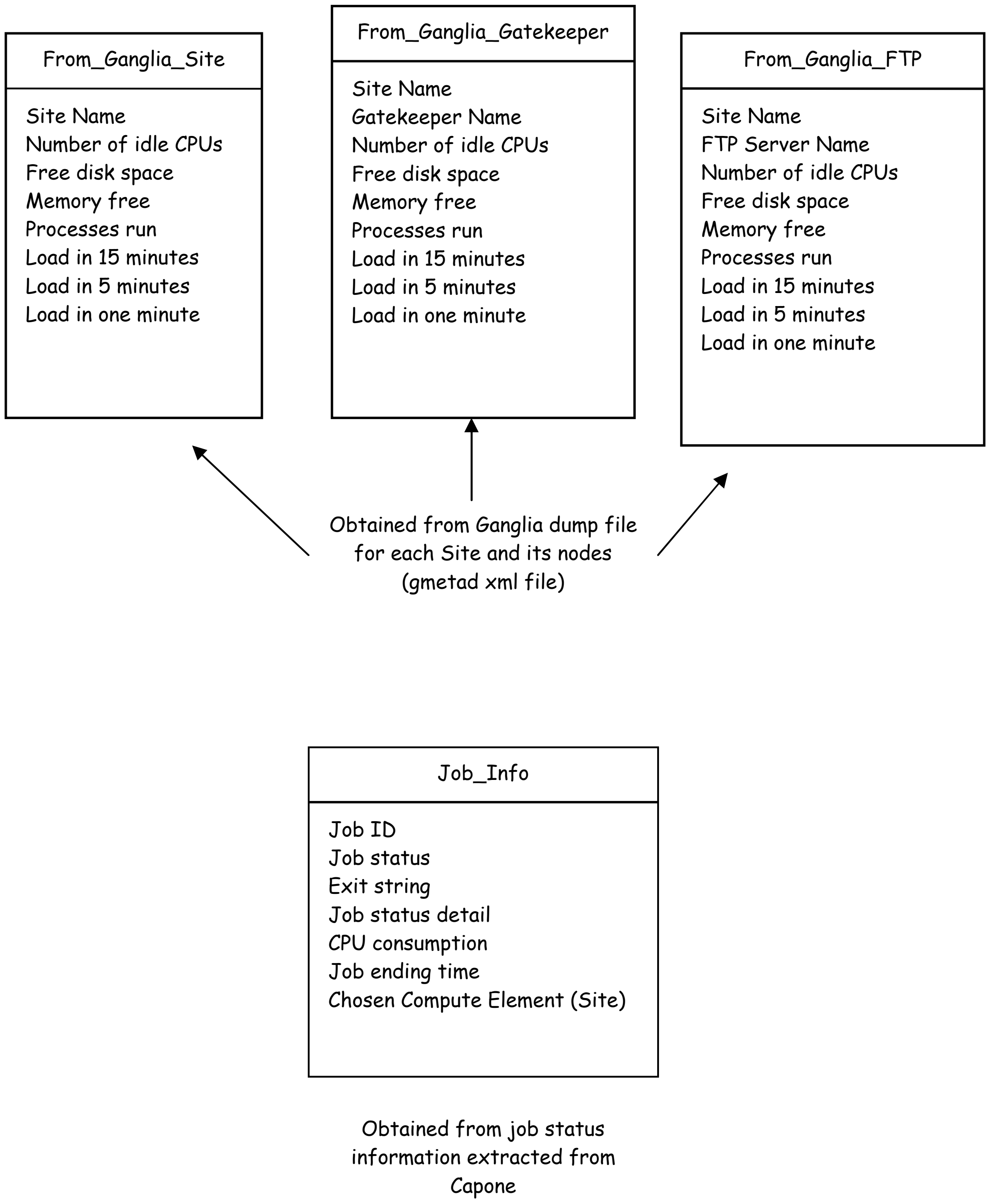

Job_Status_Lookup
State
Exit Code
Reason
Obtained from list of Capone error codes
Site_Resources
Site Name
Number of CPUs
CPU speed
Total memory
Total disk space
Obtained from Ganglia dump file for each Site and its nodes (gmetad xml file)
Site_Services
Site Name
Gatekeeper URL
FTP Server URL
Working directory
Obtained from configuration file used by Capone and submit hosts (pool.config xml file)
Site_Name_Mapping
Site Name pool config
Site Name Ganglia
Obtained from Ganglia dump file and pool.config xml file
Gatekeeper_Name_Mapping
Site Name pool config
Gatekeeper Name pool config
Gatekeeper Name Ganglia
Obtained from Ganglia dump file and pool.config xml file

Table2. Dynamic tables in database.



CHAPTER 5

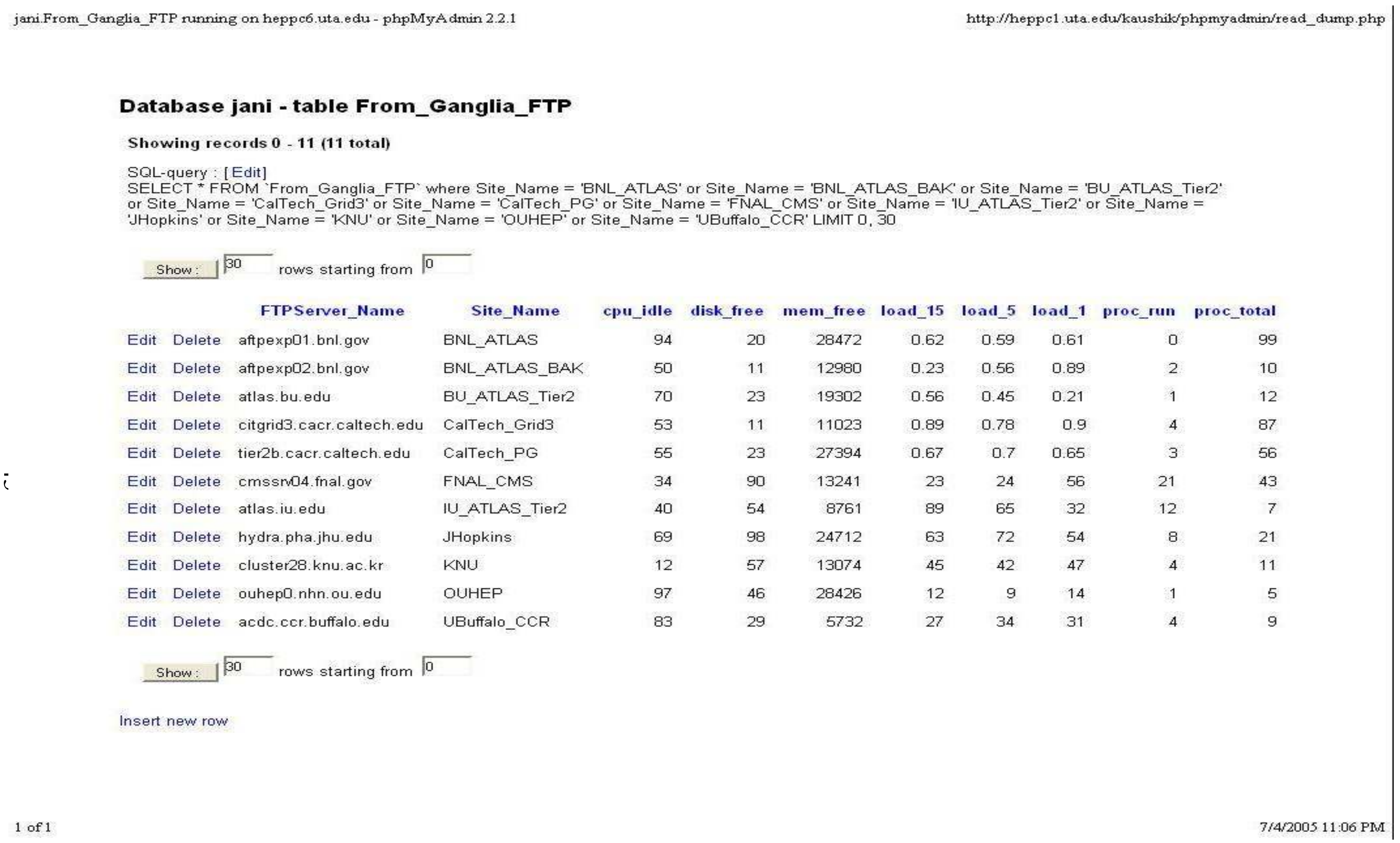
IMPLEMENTATION AND EXPERIMENTATION

5.1 Current system in utilization

The monitoring and analyzing of distributed cluster performance and statistics of ATLAS job flow has been implemented with the following programming paradigms and minimum requirements:

1. Python 2.2 and above with MySQLdb and XML DOM packages installed
2. MySQL version 3.23.49
3. Linux 2.4.20-28.7 Enterprise
4. Secure Shell 3.2.9

Figures 8 to 16 are snapshots of the current database which is stored on one of the nodes in the cluster. This section illustrates general "select" views of all the tables in the database currently being used in the site UTA-dpcc [31]. These databases are being populated from one of the hosts in the UTA cluster.



jani.From_Ganglia_FTP running on heppc6.uta.edu - phpMyAdmin 2.2.1   http://heppc1.uta.edu/kaushik/phpmyadmin/read_dump.php

Database jani - table From_Ganglia_FTP

Showing records 0 - 11 (11 total)

```
SQL-query : [Edit]
SELECT * FROM `From_Ganglia_FTP` where Site_Name = 'BNL_ATLAS' or Site_Name = 'BNL_ATLAS_BAK' or Site_Name = 'BU_ATLAS_Tier2'
or Site_Name = 'CalTech_Grid3' or Site_Name = 'CalTech_PG' or Site_Name = 'FNAL_CMS' or Site_Name = 'IU_ATLAS_Tier2' or Site_Name =
'JHopkins' or Site_Name = 'KNU' or Site_Name = 'OUHEP' or Site_Name = 'UBuffalo_CCR' LIMIT 0, 30
```

Show : 30 rows starting from 0

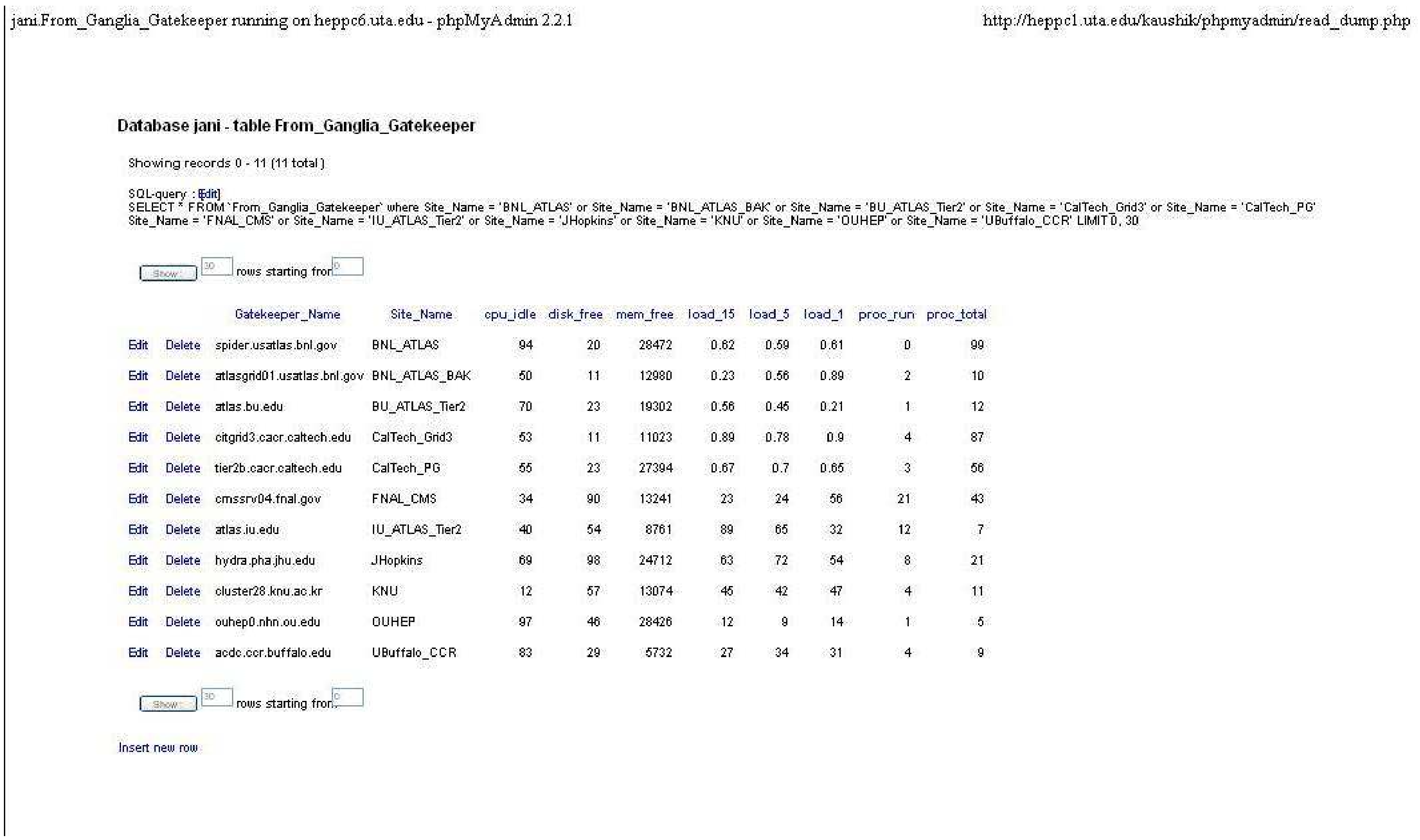
		FTPServer_Name	Site_Name	cpu_idle	disk_free	mem_free	load_15	load_5	load_1	proc_run	proc_total
Edit	Delete	aftpexp01.bnl.gov	BNL_ATLAS	94	20	28472	0.62	0.59	0.61	0	99
Edit	Delete	aftpexp02.bnl.gov	BNL_ATLAS_BAK	50	11	12980	0.23	0.56	0.89	2	10
Edit	Delete	atlas.bu.edu	BU_ATLAS_Tier2	70	23	19302	0.56	0.45	0.21	1	12
Edit	Delete	citgrid3.cacr.caltech.edu	CalTech_Grid3	53	11	11023	0.89	0.78	0.9	4	87
Edit	Delete	tier2b.cacr.caltech.edu	CalTech_PG	55	23	27394	0.67	0.7	0.65	3	56
Edit	Delete	cmssrv04.fnal.gov	FNAL_CMS	34	90	13241	23	24	56	21	43
Edit	Delete	atlas.iu.edu	IU_ATLAS_Tier2	40	54	8761	89	65	32	12	7
Edit	Delete	hydra.pha.jhu.edu	JHopkins	69	98	24712	63	72	54	8	21
Edit	Delete	cluster28.knu.ac.kr	KNU	12	57	13074	45	42	47	4	11
Edit	Delete	ouhep0.nhn.ou.edu	OUHEP	97	46	28426	12	9	14	1	5
Edit	Delete	acdc.ccr.buffalo.edu	UBuffalo_CCR	83	29	5732	27	34	31	4	9

Show : 30 rows starting from 0

Insert new row

1 of 1   7/4/2005 11:06 PM

Figure 9 Snapshot of the information stored in table From_Ganglia_FTP in the database. (Result of a select query from table From_Ganglia_FTP)



jani.From_Ganglia_Gatekeeper running on heppc6.uta.edu - phpMyAdmin 2.2.1

http://heppc1.uta.edu/kaushik/phpmyadmin/read_dump.php

Database jani - table From_Ganglia_Gatekeeper

Showing records 0 - 11 (11 total)

SQL-query : [Edit]

```
SELECT * FROM `From_Ganglia_Gatekeeper` where Site_Name = 'BNL_ATLAS' or Site_Name = 'BNL_ATLAS_BAK' or Site_Name = 'BU_ATLAS_Tier2' or Site_Name = 'CalTech_Grid3' or Site_Name = 'CalTech_PG'
Site_Name = 'FNAL_CMS' or Site_Name = 'IU_ATLAS_Tier2' or Site_Name = 'JHopkins' or Site_Name = 'KNU' or Site_Name = 'OUHEP' or Site_Name = 'UBuffalo_CCR' LIMIT 0, 30
```

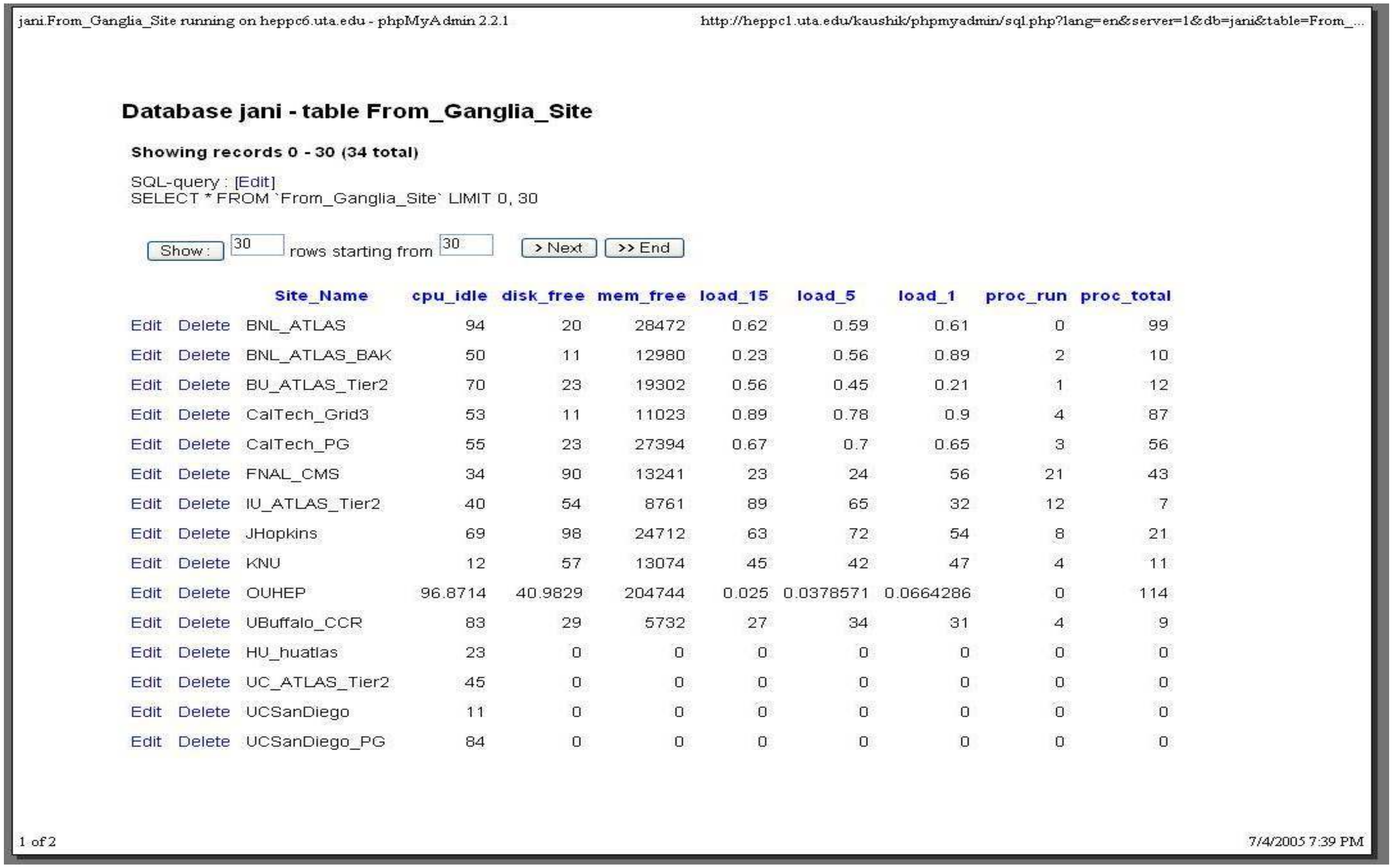
Show 30 rows starting from 0

		Gatekeeper_Name	Site_Name	cpu_idle	disk_free	mem_free	load_15	load_5	load_1	proc_run	proc_total
Edit	Delete	spider.usatlas.bnl.gov	BNL_ATLAS	94	20	28472	0.62	0.59	0.61	0	99
Edit	Delete	atlasgrid01.usatlas.bnl.gov	BNL_ATLAS_BAK	50	11	12980	0.23	0.56	0.89	2	10
Edit	Delete	atlas.bu.edu	BU_ATLAS_Tier2	70	23	19302	0.56	0.45	0.21	1	12
Edit	Delete	citgrid3.cacr.caltech.edu	CalTech_Grid3	53	11	11023	0.89	0.78	0.9	4	87
Edit	Delete	tier2b.cacr.caltech.edu	CalTech_PG	55	23	27394	0.67	0.7	0.65	3	56
Edit	Delete	cmssrv04.fnal.gov	FNAL_CMS	34	90	13241	23	24	56	21	43
Edit	Delete	atlas.iu.edu	IU_ATLAS_Tier2	40	54	8761	89	65	32	12	7
Edit	Delete	hydra.pha.jhu.edu	JHopkins	69	98	24712	63	72	54	8	21
Edit	Delete	cluster28.knu.ac.kr	KNU	12	57	13074	45	42	47	4	11
Edit	Delete	ouhep0.nhn.ou.edu	OUHEP	97	46	28426	12	9	14	1	5
Edit	Delete	acdc.ccr.buffalo.edu	UBuffalo_CCR	83	29	5732	27	34	31	4	9

Show 30 rows starting from 0

Insert new row

Figure 10 Snapshot of the information stored in table From_Ganglia_Gatekeeper in the database. (Result of a select query from table From_Ganglia_Gatekeeper)



jani.From_Ganglia_Site running on heppc6.uta.edu - phpMyAdmin 2.2.1   http://heppc1.uta.edu/kaushik/phpmyadmin/sql.php?lang=en&server=1&db=jani&table=From_...

Database jani - table From_Ganglia_Site

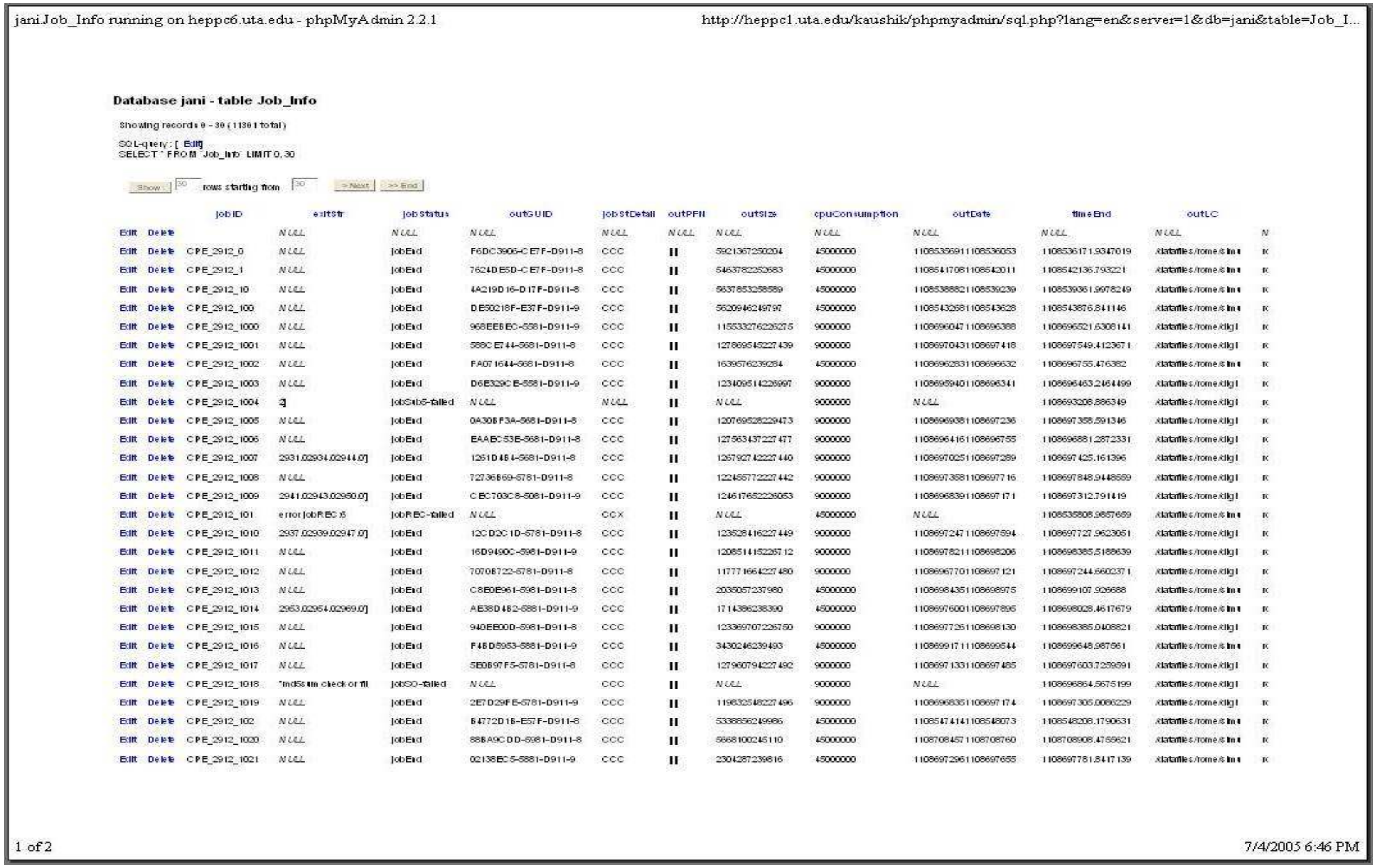
Showing records 0 - 30 (34 total)

SQL-query : [Edit]
```
SELECT * FROM `From_Ganglia_Site` LIMIT 0, 30
```

		Site_Name	cpu_idle	disk_free	mem_free	load_15	load_5	load_1	proc_run	proc_total
Edit	Delete	BNL_ATLAS	94	20	28472	0.62	0.59	0.61	0	99
Edit	Delete	BNL_ATLAS_BAK	50	11	12980	0.23	0.56	0.89	2	10
Edit	Delete	BU_ATLAS_Tier2	70	23	19302	0.56	0.45	0.21	1	12
Edit	Delete	CalTech_Grid3	53	11	11023	0.89	0.78	0.9	4	87
Edit	Delete	CalTech_PG	55	23	27394	0.67	0.7	0.65	3	56
Edit	Delete	FNAL_CMS	34	90	13241	23	24	56	21	43
Edit	Delete	IU_ATLAS_Tier2	40	54	8761	89	65	32	12	7
Edit	Delete	JHopkins	69	98	24712	63	72	54	8	21
Edit	Delete	KNU	12	57	13074	45	42	47	4	11
Edit	Delete	OUHEP	96.8714	40.9829	204744	0.025	0.0378571	0.0664286	0	114
Edit	Delete	UBuffalo_CCR	83	29	5732	27	34	31	4	9
Edit	Delete	HU_huatlas	23	0	0	0	0	0	0	0
Edit	Delete	UC_ATLAS_Tier2	45	0	0	0	0	0	0	0
Edit	Delete	UCSanDiego	11	0	0	0	0	0	0	0
Edit	Delete	UCSanDiego_PG	84	0	0	0	0	0	0	0

1 of 2   7/4/2005 7:39 PM

Figure 11 Snapshot of the information stored in table From_Ganglia_Site in the database. (Result of a select query from table From_Ganglia_Site)



jani.Job_Info running on heppc6.uta.edu - phpMyAdmin 2.2.1   http://heppc1.uta.edu/kaushik/phpmyadmin/sql.php?lang=en&server=1&db=jani&table=Job_I...

Database jani - table Job_Info

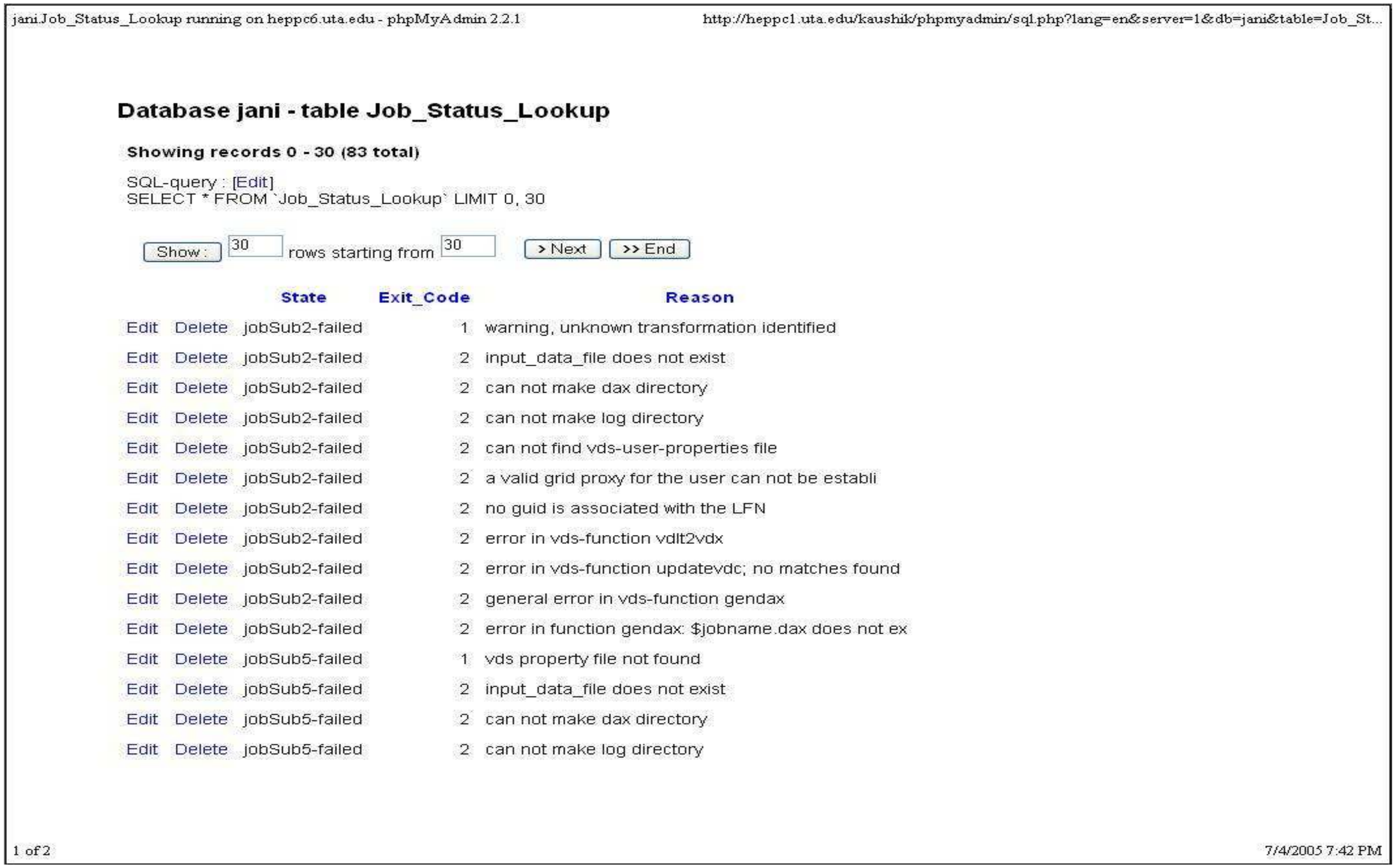
Showing records 0 - 30 (11301 total)

SQL-query : [Edit]
SELECT * FROM `Job_Info` LIMIT 0, 30

		jobID	exitStr	jobStatus	outGUID	jobStDetail	outPFN	outSize	cpuConsumption	outDate	timeEnd	outLC	
Edit	Delete		NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	N
Edit	Delete	CPE_2912_0	NULL	jobEnd	F6DC3906-CE7F-D911-8	CCC	II	5921367250204	45000000	11085356911108536053	1108536171.9347019	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1	NULL	jobEnd	7624DE5D-CE7F-D911-8	CCC	II	5463782252683	45000000	11085417081108542011	1108542136.793221	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_10	NULL	jobEnd	4A219D16-D17F-D911-8	CCC	II	5637853258589	45000000	11085388821108539239	1108539361.9978249	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_100	NULL	jobEnd	DE50218F-E37F-D911-9	CCC	II	5620946249797	45000000	11085432681108543628	1108543876.841146	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1000	NULL	jobEnd	968EEBEC-5581-D911-9	CCC	II	115533276226275	9000000	11086960471108696388	1108696521.6308141	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1001	NULL	jobEnd	588CE744-5681-D911-8	CCC	II	127869545227439	9000000	11086970431108697418	1108697549.4123671	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1002	NULL	jobEnd	FA071644-5681-D911-8	CCC	II	1639576239284	45000000	11086962831108696632	1108696755.476382	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1003	NULL	jobEnd	D6E329CE-5581-D911-9	CCC	II	123409514226997	9000000	11086959401108696341	1108696463.2464499	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1004	2]	jobSubS-failed	NULL	NULL	II	NULL	9000000	NULL	1108693208.886349	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1005	NULL	jobEnd	0A308F3A-5681-D911-8	CCC	II	120769528229473	9000000	11086969381108697236	1108697358.591346	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1006	NULL	jobEnd	EAABC53E-5681-D911-8	CCC	II	127563437227477	9000000	11086964161108696755	1108696881.2872331	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1007	2931.02934.02944.0]	jobEnd	1261D4B4-5681-D911-8	CCC	II	126792742227440	9000000	11086970251108697289	1108697425.161396	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1008	NULL	jobEnd	72736B69-5781-D911-8	CCC	II	122455772227442	9000000	11086973581108697716	1108697848.9448559	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1009	2941.02943.02950.0]	jobEnd	CEC703C8-5081-D911-9	CCC	II	124617652226053	9000000	11086968391108697171	1108697312.791419	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_101	error jobREC:6	jobREC-failed	NULL	CCX	II	NULL	45000000	NULL	1108535808.9857659	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1010	2937.02939.02947.0]	jobEnd	12CD2C1D-5781-D911-8	CCC	II	123528416227449	9000000	11086972471108697594	1108697727.9623051	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1011	NULL	jobEnd	16D9490C-5981-D911-9	CCC	II	120851415226712	9000000	11086978211108698206	1108698385.5188639	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1012	NULL	jobEnd	70708722-5781-D911-8	CCC	II	117771664227480	9000000	11086967701108697121	1108697244.6602371	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1013	NULL	jobEnd	C8E0E961-5981-D911-8	CCC	II	2035057237980	45000000	11086984351108698975	1108699107.926688	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1014	2953.02954.02969.0]	jobEnd	AE38D482-5881-D911-9	CCC	II	1714386238390	45000000	11086976001108697895	1108698028.4617679	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1015	NULL	jobEnd	940EE00D-5981-D911-8	CCC	II	123369707226750	9000000	11086977261108698130	1108698385.0408821	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1016	NULL	jobEnd	F48D5953-5881-D911-9	CCC	II	3430246239493	45000000	11086991711108699544	1108699648.987561	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1017	NULL	jobEnd	5E0B97F5-5781-D911-8	CCC	II	127960794227492	9000000	11086971331108697485	1108697603.7259591	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1018	"md5sum check or fil	jobSO-failed	NULL	CCC	II	NULL	9000000	NULL	1108696864.5675199	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_1019	NULL	jobEnd	2E7D29FE-5781-D911-9	CCC	II	119832548227496	9000000	11086968351108697174	1108697305.0086229	/datafiles/rome/dlgl	rc
Edit	Delete	CPE_2912_102	NULL	jobEnd	B4772D18-E57F-D911-8	CCC	II	5338856249986	45000000	11085474141108548073	1108548208.1790631	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1020	NULL	jobEnd	88BA9CDD-5981-D911-8	CCC	II	5668100245110	45000000	11087084571108708760	1108708908.4755621	/datafiles/rome/slmu	rc
Edit	Delete	CPE_2912_1021	NULL	jobEnd	02138EC5-5881-D911-9	CCC	II	2304287239816	45000000	11086972961108697655	1108697781.8417139	/datafiles/rome/slmu	rc

1 of 2   7/4/2005 6:46 PM

Figure 12 Snapshot of the information stored in table Job_Info in the database. (Result of a select * from Job_Info query)


jani.Job_Status_Lookup running on heppc6.uta.edu - phpMyAdmin 2.2.1 http://heppc1.uta.edu/kaushik/phpmyadmin/sql.php?lang=en&server=1&db=jani&table=Job_St...

Database jani - table Job_Status_Lookup

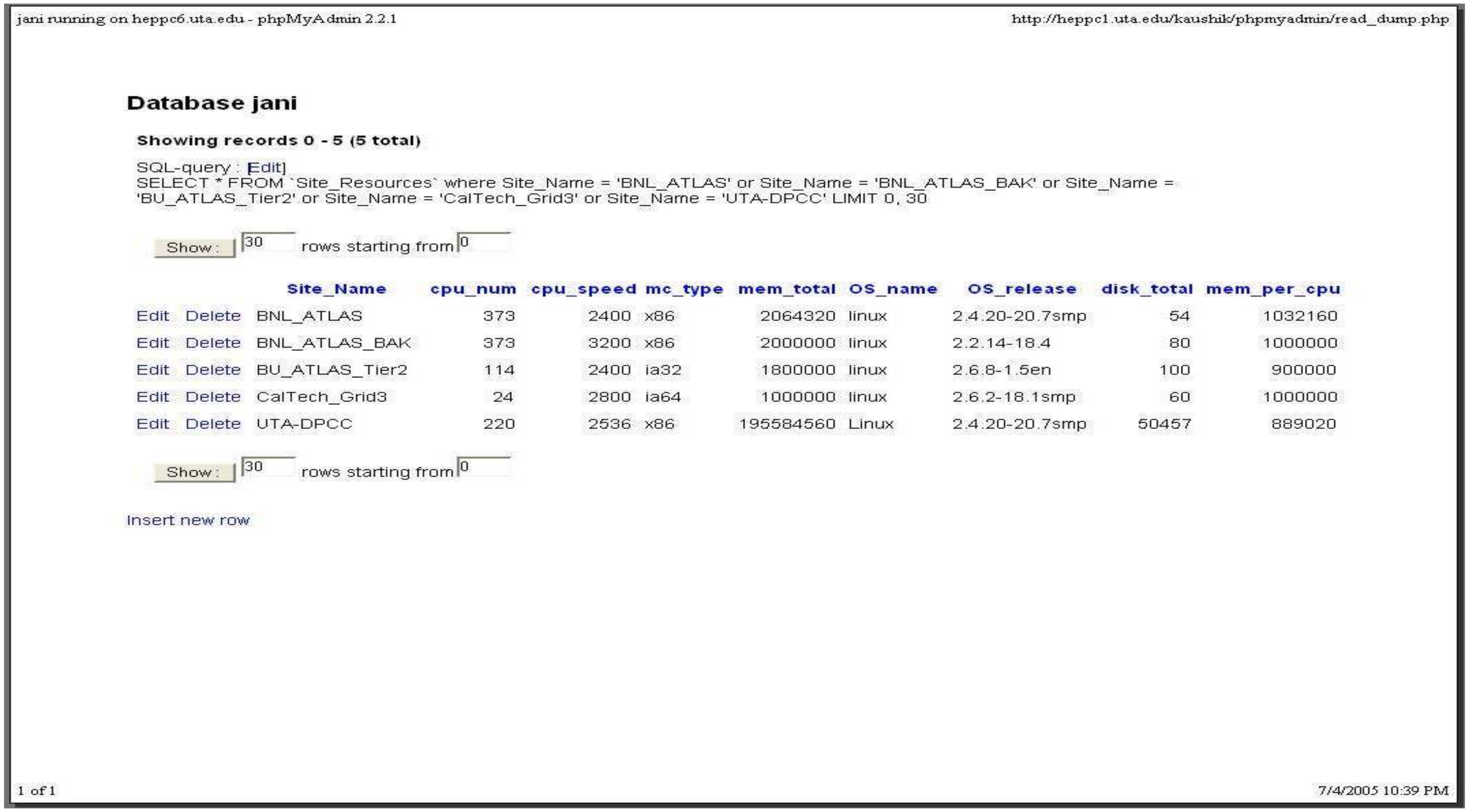
Showing records 0 - 30 (83 total)

```
SQL-query : [Edit]
SELECT * FROM `Job_Status_Lookup` LIMIT 0, 30
```

		State	Exit_Code	Reason
Edit	Delete	jobSub2-failed	1	warning, unknown transformation identified
Edit	Delete	jobSub2-failed	2	input_data_file does not exist
Edit	Delete	jobSub2-failed	2	can not make dax directory
Edit	Delete	jobSub2-failed	2	can not make log directory
Edit	Delete	jobSub2-failed	2	can not find vds-user-properties file
Edit	Delete	jobSub2-failed	2	a valid grid proxy for the user can not be establi
Edit	Delete	jobSub2-failed	2	no guid is associated with the LFN
Edit	Delete	jobSub2-failed	2	error in vds-function vdlt2vdx
Edit	Delete	jobSub2-failed	2	error in vds-function updatevdc; no matches found
Edit	Delete	jobSub2-failed	2	general error in vds-function gendax
Edit	Delete	jobSub2-failed	2	error in function gendax: $jobname.dax does not ex
Edit	Delete	jobSub5-failed	1	vds property file not found
Edit	Delete	jobSub5-failed	2	input_data_file does not exist
Edit	Delete	jobSub5-failed	2	can not make dax directory
Edit	Delete	jobSub5-failed	2	can not make log directory

1 of 2 7/4/2005 7:42 PM

Figure 13 Snapshot of the information stored in table Job_Status_Lookup in the database. (Result of a select query from table Job_Status_Lookup)


jani running on heppc6.uta.edu - phpMyAdmin 2.2.1 http://heppc1.uta.edu/kaushik/phpmyadmin/read_dump.php

Database jani

Showing records 0 - 5 (5 total)

```
SQL-query : [Edit]
SELECT * FROM `Site_Resources` where Site_Name = 'BNL_ATLAS' or Site_Name = 'BNL_ATLAS_BAK' or Site_Name =
'BU_ATLAS_Tier2' or Site_Name = 'CalTech_Grid3' or Site_Name = 'UTA-DPCC' LIMIT 0, 30
```

Show : 30 rows starting from 0

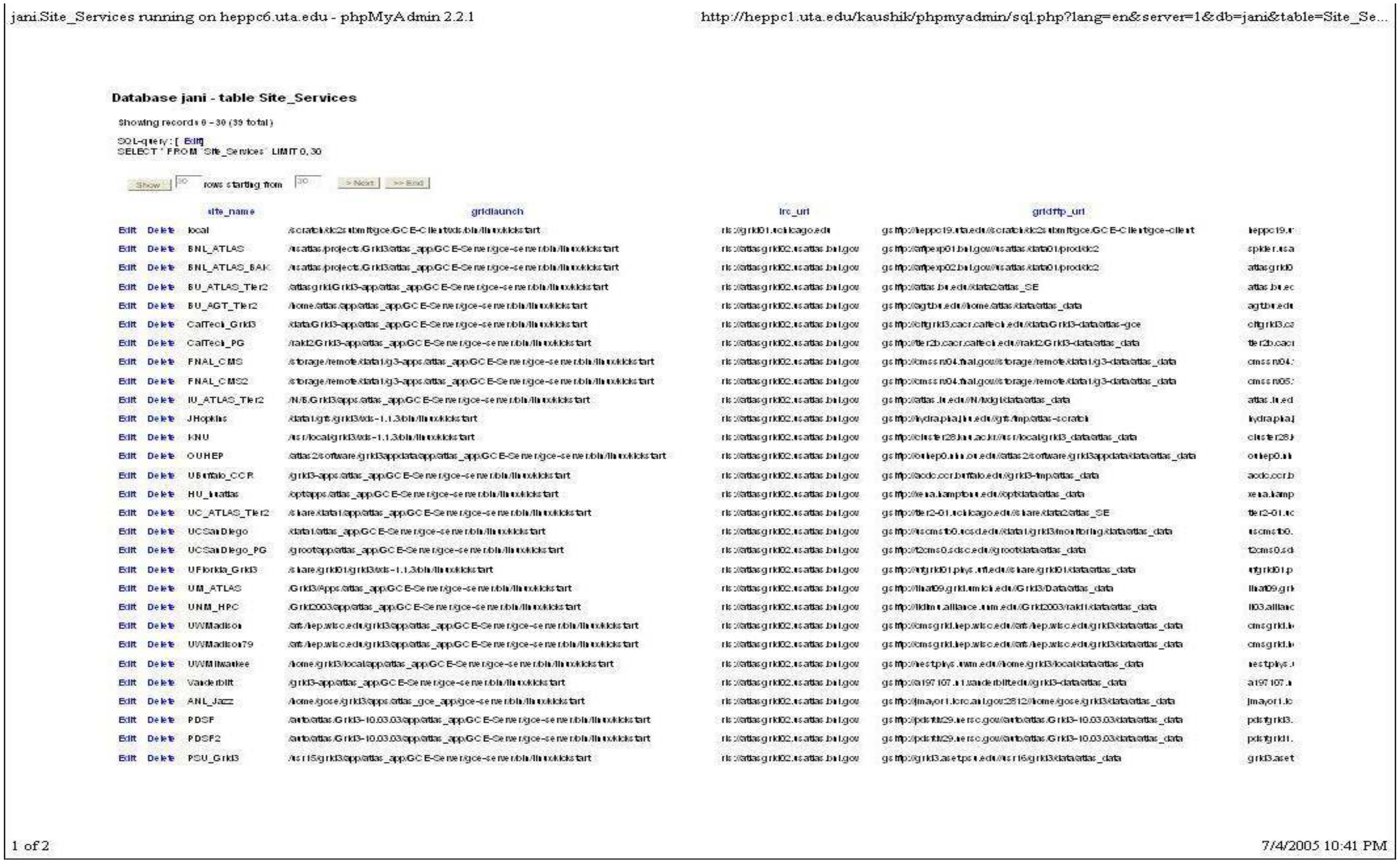
		Site_Name	cpu_num	cpu_speed	mc_type	mem_total	OS_name	OS_release	disk_total	mem_per_cpu
Edit	Delete	BNL_ATLAS	373	2400	x86	2064320	linux	2.4.20-20.7smp	54	1032160
Edit	Delete	BNL_ATLAS_BAK	373	3200	x86	2000000	linux	2.2.14-18.4	80	1000000
Edit	Delete	BU_ATLAS_Tier2	114	2400	ia32	1800000	linux	2.6.8-1.5en	100	900000
Edit	Delete	CalTech_Grid3	24	2800	ia64	1000000	linux	2.6.2-18.1smp	60	1000000
Edit	Delete	UTA-DPCC	220	2536	x86	195584560	Linux	2.4.20-20.7smp	50457	889020

Show : 30 rows starting from 0

Insert new row

1 of 1 7/4/2005 10:39 PM

Figure 14 Snapshot of the information stored in table Site_Resources in the database. (Result of a select query from Site_Resources)



jani.Site_Services running on heppc6.uta.edu - phpMyAdmin 2.2.1 http://heppc1.uta.edu/kaushik/phpmyadmin/sql.php?lang=en&server=1&db=jani&table=Site_Se...

Database jani - table Site_Services

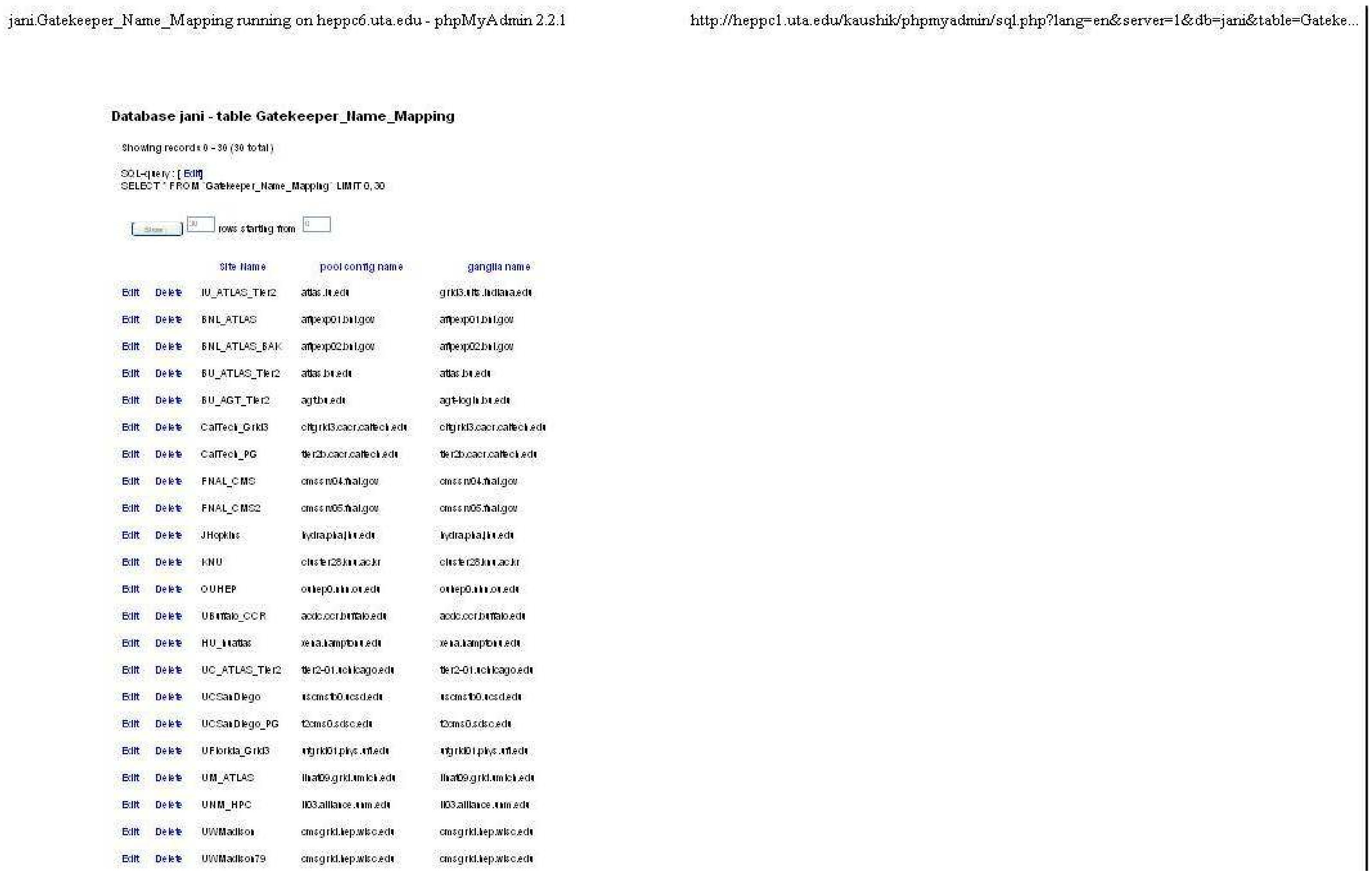
Showing records 0 - 30 (39 total)

SQL-query : [Edit]
SELECT * FROM `Site_Services` LIMIT 0, 30

		site_name	gridlaunch	lrc_url	gridftp_url	
Edit	Delete	local	/scratch/dc2submit/gce/GCE-Client/vds/bin/linux/kickstart	rls://grid01.uchicago.edu	gsiftp://heppc19.uta.edu//scratch/dc2submit/gce/GCE-Client/gce-client	heppc19.u
Edit	Delete	BNL_ATLAS	/usatlas/projects/Grid3/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://aftpexp01.bnl.gov//usatlas/data01/prod/dc2	spider.usa
Edit	Delete	BNL_ATLAS_BAK	/usatlas/projects/Grid3/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://aftpexp02.bnl.gov//usatlas/data01/prod/dc2	atlasgrid0
Edit	Delete	BU_ATLAS_Tier2	/atlasgrid/Grid3-app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://atlas.bu.edu//data2/atlas_SE	atlas.bu.ec
Edit	Delete	BU_AGT_Tier2	/home/atlas/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://agt.bu.edu//home/atlas/data/atlas_data	agt.bu.edu
Edit	Delete	CalTech_Grid3	/data/Grid3-app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://citgrid3.cacr.caltech.edu//data/Grid3-data/atlas-gce	citgrid3.ca
Edit	Delete	CalTech_PG	/raid2/Grid3-app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://tier2b.cacr.caltech.edu//raid2/Grid3-data/atlas_data	tier2b.cacr
Edit	Delete	FNAL_CMS	/storage/remote/data1/g3-apps/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://cmssrv04.fnal.gov//storage/remote/data1/g3-data/atlas_data	cmssrv04.
Edit	Delete	FNAL_CMS2	/storage/remote/data1/g3-apps/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://cmssrv04.fnal.gov//storage/remote/data1/g3-data/atlas_data	cmssrv05.
Edit	Delete	IU_ATLAS_Tier2	/N/B/Grid3/apps/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://atlas.iu.edu//N/lxdg1/data/atlas_data	atlas.iu.ed
Edit	Delete	JHopkins	/data1/gts/grid3/vds-1.1.3/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://hydra.pha.jhu.edu//gts/tmp/atlas-scratch	hydra.pha.j
Edit	Delete	KNU	/usr/local/grid3/vds-1.1.3/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://cluster28.knu.ac.kr//usr/local/grid3_data/atlas_data	cluster28.k
Edit	Delete	OUHEP	/atlas2/software/grid3appdata/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://ouhep0.nhn.ou.edu//atlas2/software/grid3appdata/data/atlas_data	ouhep0.nh
Edit	Delete	UBuffalo_CCR	/grid3-apps/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://acdc.ccr.buffalo.edu//grid3-tmp/atlas_data	acdc.ccr.b
Edit	Delete	HU_Hatlas	/opt/apps/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://xena.hamptonu.edu//opt/data/atlas_data	xena.hamp
Edit	Delete	UC_ATLAS_Tier2	/share/data1/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://tier2-01.uchicago.edu//share/data2/atlas_SE	tier2-01.uc
Edit	Delete	UCSanDiego	/data1/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://uscmstb0.ucsd.edu//data1/grid3/monitoring/data/atlas_data	uscmstb0.
Edit	Delete	UCSanDiego_PG	/groot/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://t2cms0.sdsc.edu//groot/data/atlas_data	t2cms0.sd
Edit	Delete	UFlorida_Grid3	/share/grid01/grid3/vds-1.1.3/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://ufgrid01.phys.ufl.edu//share/grid01/data/atlas_data	ufgrid01.p
Edit	Delete	UM_ATLAS	/Grid3/Apps/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://linat09.grid.umich.edu//Grid3/Data/atlas_data	linat09.gri
Edit	Delete	UNM_HPC	/Grid2003/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://lldimu.alliance.unm.edu//Grid2003/raid1/data/atlas_data	ll03.allianc
Edit	Delete	UWMadison	/afs/hep.wisc.edu/grid3/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://cmsgrid.hep.wisc.edu//afs/hep.wisc.edu/grid3/data/atlas_data	cmsgrid.he
Edit	Delete	UWMadison79	/afs/hep.wisc.edu/grid3/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://cmsgrid.hep.wisc.edu//afs/hep.wisc.edu/grid3/data/atlas_data	cmsgrid.he
Edit	Delete	UWMilwaukee	/home/grid3/local/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://nestphys.uwm.edu//home/grid3/local/data/atlas_data	nestphys.u
Edit	Delete	Vanderbilt	/grid3-app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://a197107.n1.vanderbilt.edu//grid3-data/atlas_data	a197107.n
Edit	Delete	ANL_Jazz	/home/gose/grid3/apps/atlas_gce_app/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://jmayor1.lcrc.anl.gov:2812//home/gose/grid3/data/atlas_data	jmayor1.lc
Edit	Delete	PDSF	/auto/atlas/Grid3-10.03.03/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://pdsfdtn29.nersc.gov//auto/atlas/Grid3-10.03.03/data/atlas_data	pdsfgrid3.
Edit	Delete	PDSF2	/auto/atlas/Grid3-10.03.03/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://pdsfdtn29.nersc.gov//auto/atlas/Grid3-10.03.03/data/atlas_data	pdsfgrid1.
Edit	Delete	PSU_Grid3	/usr15/grid3/app/atlas_app/GCE-Server/gce-server/bin/linux/kickstart	rls://atlasgrid02.usatlas.bnl.gov	gsiftp://grid3.aset.psu.edu//usr16/grid3/data/atlas_data	grid3.aset

1 of 2 7/4/2005 10:41 PM

Figure 15 Snapshot of the information stored in table Site_Services in the database. (Result of a select query from Site_Services)



jani.Gatekeeper_Name_Mapping running on heppc6.uta.edu - phpMyAdmin 2.2.1 http://heppc1.uta.edu/kaushik/phpmyadmin/sql.php?lang=en&server=1&db=jani&table=Gateke...

Database jani - table Gatekeeper_Name_Mapping

Showing records 0 - 30 (30 total)

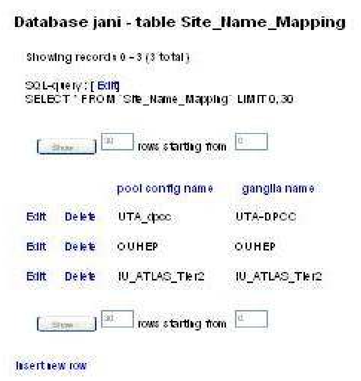
SQL-query : [Edit]
SELECT * FROM `Gatekeeper_Name_Mapping` LIMIT 0, 30

Show 30 rows starting from 0

		Site Name	pool config name	ganglia name
Edit	Delete	IU_ATLAS_Tier2	atlas.iu.edu	grid3.uits.indiana.edu
Edit	Delete	BNL_ATLAS	aftpexp01.bnl.gov	aftpexp01.bnl.gov
Edit	Delete	BNL_ATLAS_BAK	aftpexp02.bnl.gov	aftpexp02.bnl.gov
Edit	Delete	BU_ATLAS_Tier2	atlas.bu.edu	atlas.bu.edu
Edit	Delete	BU_AGT_Tier2	agt.bu.edu	agt-login.bu.edu
Edit	Delete	CalTech_Grid3	citgrid3.cacr.caltech.edu	citgrid3.cacr.caltech.edu
Edit	Delete	CalTech_PG	tier2b.cacr.caltech.edu	tier2b.cacr.caltech.edu
Edit	Delete	FNAL_CMS	cmssrv04.fnal.gov	cmssrv04.fnal.gov
Edit	Delete	FNAL_CMS2	cmssrv05.fnal.gov	cmssrv05.fnal.gov
Edit	Delete	JHopkins	hydra.pha.jhu.edu	hydra.pha.jhu.edu
Edit	Delete	KNU	cluster28.knu.ac.kr	cluster28.knu.ac.kr
Edit	Delete	OUHEP	ouhep0.nhn.ou.edu	ouhep0.nhn.ou.edu
Edit	Delete	UBuffalo_CCR	acdc.ccr.buffalo.edu	acdc.ccr.buffalo.edu
Edit	Delete	HU_Hatlas	xena.hamptonu.edu	xena.hamptonu.edu
Edit	Delete	UC_ATLAS_Tier2	tier2-01.uchicago.edu	tier2-01.uchicago.edu
Edit	Delete	UCSanDiego	uscmstb0.ucsd.edu	uscmstb0.ucsd.edu
Edit	Delete	UCSanDiego_PG	t2cms0.sdsc.edu	t2cms0.sdsc.edu
Edit	Delete	UFlorida_Grid3	ufgrid01.phys.ufl.edu	ufgrid01.phys.ufl.edu
Edit	Delete	UM_ATLAS	linat09.grid.umich.edu	linat09.grid.umich.edu
Edit	Delete	UNM_HPC	ll03.alliance.unm.edu	ll03.alliance.unm.edu
Edit	Delete	UWMadison	cmsgrid.hep.wisc.edu	cmsgrid.hep.wisc.edu
Edit	Delete	UWMadison79	cmsgrid.hep.wisc.edu	cmsgrid.hep.wisc.edu

Figure 16 Snapshot of the information stored in table Gatekeeper_Name_Mapping in the database. (Result of a select query from Gatekeeper_Name_Mapping)

jani.Site_Name_Mapping running on heppc6.uta.edu - phpMyAdmin 2.2.1 http://heppc1.uta.edu/kaushik/phpmyadmin/sql.php?lang=en&server=1&db=jani&table=Site_...



Database jani - table Site_Name_Mapping

Showing records 0 - 3 (3 total)

SQL-query : [Edit]
SELECT * FROM `Site_Name_Mapping` LIMIT 0, 30

		pool config name	ganglia name
Edit	Delete	UTA_dpcc	UTA-DPCC
Edit	Delete	OUHEP	OUHEP
Edit	Delete	IU_ATLAS_Tier2	IU_ATLAS_Tier2

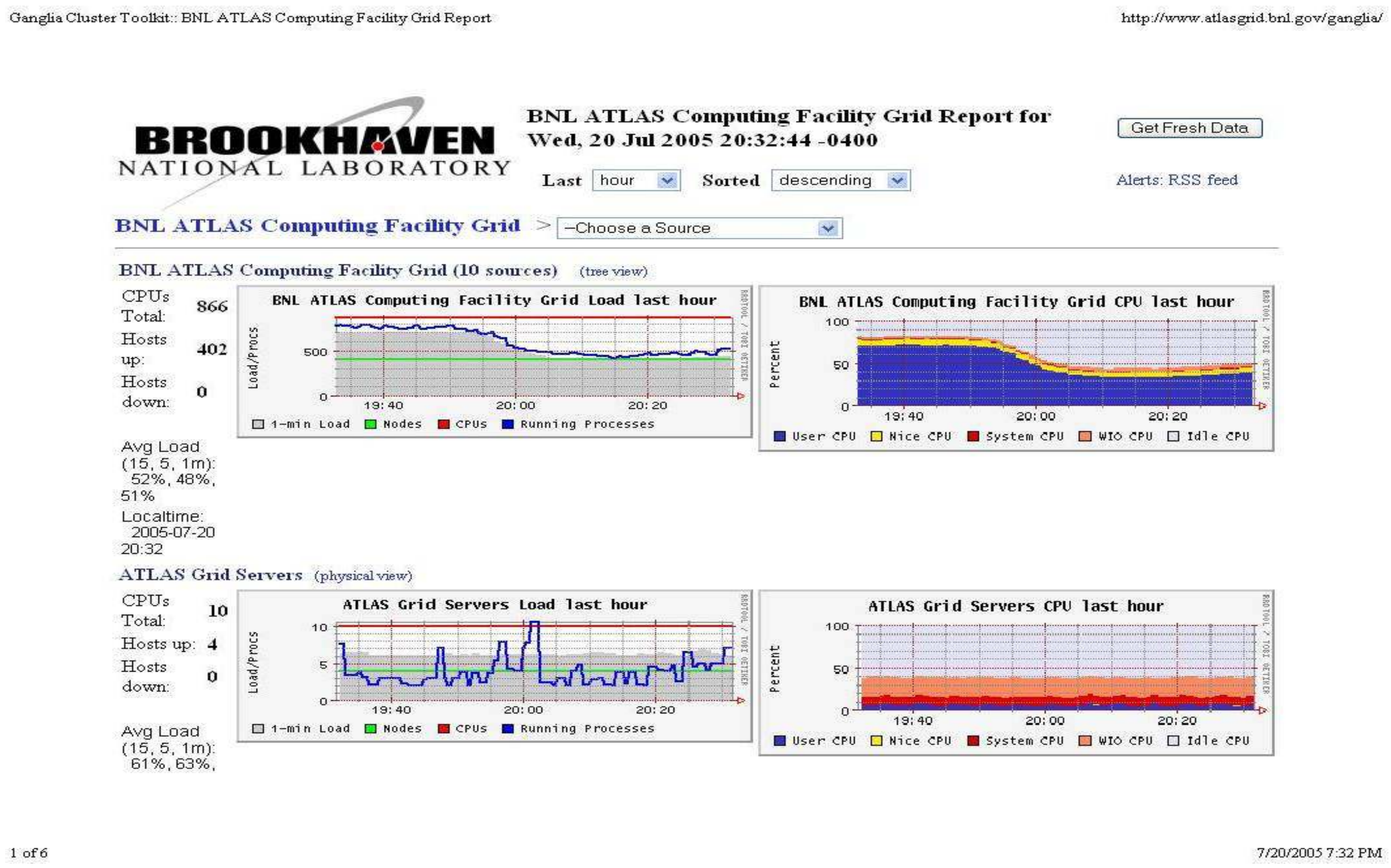
Insert new row

Figure 17 Snapshot of the information stored in table Site_Name_Mapping in the database. (Result of a select query from Site_Name_Mapping)

All scripts have been written with the design considerations from Chapter 4 working toward the database as presented in section 4.3. Ganglia information from other sites is being collected by telnet-ing on of three ports, port 8649, 8651 or 8652, depending on individual site settings on the gatekeeper node of the site.

Appendices A to E illustrate all the scripts written in this thesis as pseudo code converted straight from the Python scripts that run the monitoring and analyzing system.

Figures 18 and 19 show Ganglia information as represented on the web interface and in XML format respectively.



Ganglia Cluster Toolkit:: BNL ATLAS Computing Facility Grid Report http://www.atlasgrid.bnl.gov/ganglia/

BNL ATLAS Computing Facility Grid Report for Wed, 20 Jul 2005 20:32:44 -0400

Last hour Sorted descending

BNL ATLAS Computing Facility Grid > --Choose a Source

BNL ATLAS Computing Facility Grid (10 sources) (tree view)

CPUs Total: 866
Hosts up: 402
Hosts down: 0

Avg Load (15, 5, 1m): 52%, 48%, 51%
Localtime: 2005-07-20 20:32

ATLAS Grid Servers (physical view)

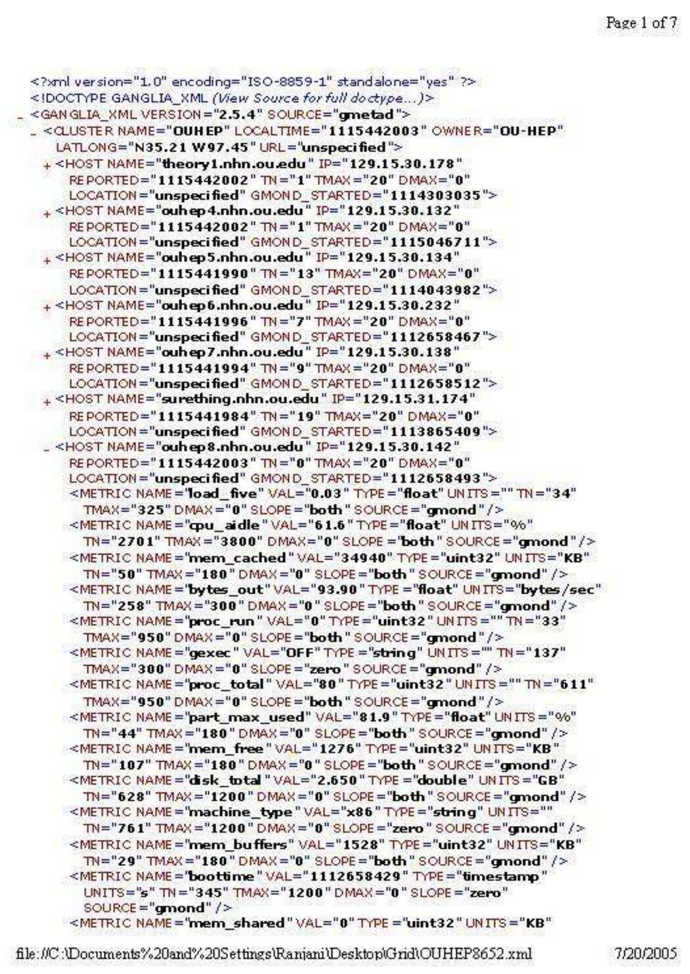
CPUs Total: 10
Hosts up: 4
Hosts down: 0

Avg Load (15, 5, 1m): 61%, 63%,

1 of 6 7/20/2005 7:32 PM

BROOKHAVEN NATIONAL LABORATORY
Get Fresh Data
Alerts: RSS feed
BNL ATLAS Computing Facility Grid Load last hour
Load/Procs
1-min Load
Nodes
CPUs
Running Processes
BNL ATLAS Computing Facility Grid CPU last hour
Percent
User CPU
Nice CPU
System CPU
WIO CPU
Idle CPU
ATLAS Grid Servers Load last hour
ATLAS Grid Servers CPU last hour
RRDTOOL / TOBI OETIKER

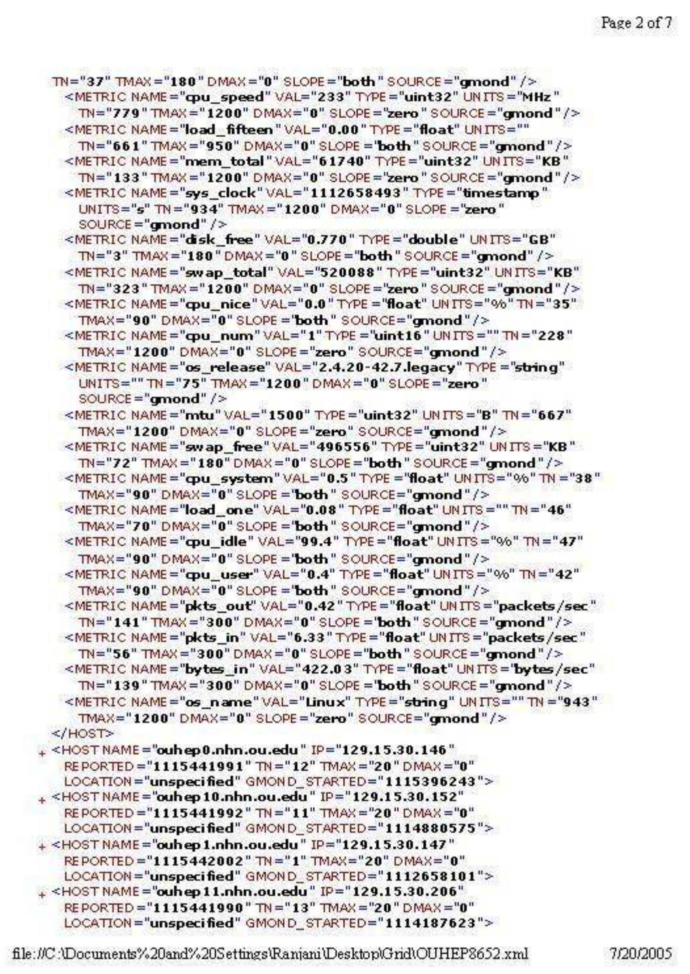
Figure 18 Snapshot of Ganglia web interface for Brookhaven National Laboratory cluster nodes.



Page 1 of 7

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<!DOCTYPE GANGLIA_XML (View Source for full doctype...)>
- <GANGLIA_XML VERSION="2.5.4" SOURCE="gmetad">
 - <CLUSTER NAME="OUHEP" LOCALTIME="1115442003" OWNER="OU-HEP"
     LATLONG="N35.21 W97.45" URL="unspecified">
   + <HOST NAME="theory1.nhn.ou.edu" IP="129.15.30.178"
       REPORTED="1115442002" TN="1" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1114303035">
   + <HOST NAME="ouhep4.nhn.ou.edu" IP="129.15.30.132"
       REPORTED="1115442002" TN="1" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1115046711">
   + <HOST NAME="ouhep5.nhn.ou.edu" IP="129.15.30.134"
       REPORTED="1115441990" TN="13" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1114043982">
   + <HOST NAME="ouhep6.nhn.ou.edu" IP="129.15.30.232"
       REPORTED="1115441996" TN="7" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1112658467">
   + <HOST NAME="ouhep7.nhn.ou.edu" IP="129.15.30.138"
       REPORTED="1115441994" TN="9" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1112658512">
   + <HOST NAME="surething.nhn.ou.edu" IP="129.15.31.174"
       REPORTED="1115441984" TN="19" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1113865409">
   - <HOST NAME="ouhep8.nhn.ou.edu" IP="129.15.30.142"
       REPORTED="1115442003" TN="0" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1112658493">
       <METRIC NAME="load_five" VAL="0.03" TYPE="float" UNITS="" TN="34"
         TMAX="325" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="cpu_aidle" VAL="61.6" TYPE="float" UNITS="%"
         TN="2701" TMAX="3800" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="mem_cached" VAL="34940" TYPE="uint32" UNITS="KB"
         TN="50" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="bytes_out" VAL="93.90" TYPE="float" UNITS="bytes/sec"
         TN="258" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="proc_run" VAL="0" TYPE="uint32" UNITS="" TN="33"
         TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="gexec" VAL="OFF" TYPE="string" UNITS="" TN="137"
         TMAX="300" DMAX="0" SLOPE="zero" SOURCE="gmond" />
       <METRIC NAME="proc_total" VAL="80" TYPE="uint32" UNITS="" TN="611"
         TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="part_max_used" VAL="81.9" TYPE="float" UNITS="%"
         TN="44" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="mem_free" VAL="1276" TYPE="uint32" UNITS="KB"
         TN="107" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="disk_total" VAL="2.650" TYPE="double" UNITS="GB"
         TN="628" TMAX="1200" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="machine_type" VAL="x86" TYPE="string" UNITS=""
         TN="761" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond" />
       <METRIC NAME="mem_buffers" VAL="1528" TYPE="uint32" UNITS="KB"
         TN="29" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="boottime" VAL="1112658429" TYPE="timestamp"
         UNITS="s" TN="345" TMAX="1200" DMAX="0" SLOPE="zero"
         SOURCE="gmond" />
       <METRIC NAME="mem_shared" VAL="0" TYPE="uint32" UNITS="KB"
```

file://C:\Documents%20and%20Settings\Ranjani\Desktop\Grid\OUHEP8652.xml 7/20/2005



Page 2 of 7

```
         TN="37" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="cpu_speed" VAL="233" TYPE="uint32" UNITS="MHz"
         TN="779" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond" />
       <METRIC NAME="load_fifteen" VAL="0.00" TYPE="float" UNITS=""
         TN="661" TMAX="950" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="mem_total" VAL="61740" TYPE="uint32" UNITS="KB"
         TN="133" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond" />
       <METRIC NAME="sys_clock" VAL="1112658493" TYPE="timestamp"
         UNITS="s" TN="934" TMAX="1200" DMAX="0" SLOPE="zero"
         SOURCE="gmond" />
       <METRIC NAME="disk_free" VAL="0.770" TYPE="double" UNITS="GB"
         TN="3" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="swap_total" VAL="520088" TYPE="uint32" UNITS="KB"
         TN="323" TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond" />
       <METRIC NAME="cpu_nice" VAL="0.0" TYPE="float" UNITS="%" TN="35"
         TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="cpu_num" VAL="1" TYPE="uint16" UNITS="" TN="228"
         TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond" />
       <METRIC NAME="os_release" VAL="2.4.20-42.7.legacy" TYPE="string"
         UNITS="" TN="75" TMAX="1200" DMAX="0" SLOPE="zero"
         SOURCE="gmond" />
       <METRIC NAME="mtu" VAL="1500" TYPE="uint32" UNITS="B" TN="667"
         TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond" />
       <METRIC NAME="swap_free" VAL="496556" TYPE="uint32" UNITS="KB"
         TN="72" TMAX="180" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="cpu_system" VAL="0.5" TYPE="float" UNITS="%" TN="38"
         TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="load_one" VAL="0.08" TYPE="float" UNITS="" TN="46"
         TMAX="70" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="cpu_idle" VAL="99.4" TYPE="float" UNITS="%" TN="47"
         TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="cpu_user" VAL="0.4" TYPE="float" UNITS="%" TN="42"
         TMAX="90" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="pkts_out" VAL="0.42" TYPE="float" UNITS="packets/sec"
         TN="141" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="pkts_in" VAL="6.33" TYPE="float" UNITS="packets/sec"
         TN="56" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="bytes_in" VAL="422.03" TYPE="float" UNITS="bytes/sec"
         TN="139" TMAX="300" DMAX="0" SLOPE="both" SOURCE="gmond" />
       <METRIC NAME="os_name" VAL="Linux" TYPE="string" UNITS="" TN="943"
         TMAX="1200" DMAX="0" SLOPE="zero" SOURCE="gmond" />
     </HOST>
   + <HOST NAME="ouhep0.nhn.ou.edu" IP="129.15.30.146"
       REPORTED="1115441991" TN="12" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1115396243">
   + <HOST NAME="ouhep10.nhn.ou.edu" IP="129.15.30.152"
       REPORTED="1115441992" TN="11" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1114880575">
   + <HOST NAME="ouhep1.nhn.ou.edu" IP="129.15.30.147"
       REPORTED="1115442002" TN="1" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1112658101">
   + <HOST NAME="ouhep11.nhn.ou.edu" IP="129.15.30.206"
       REPORTED="1115441990" TN="13" TMAX="20" DMAX="0"
       LOCATION="unspecified" GMOND_STARTED="1114187623">
```

file://C:\Documents%20and%20Settings\Ranjani\Desktop\Grid\OUHEP8652.xml 7/20/2005

Figure 19 Ganglia information from Gatekeeper node at site Oklahoma University in XML format. This is obtained using telnet on port 8651 on Gatekeeper node.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

Grid computing as we know it is a constantly developing field. Scheduling on the Grid is one of the many and most challenging aspects of Grid computing. The part of ATLAS related software designed and developed in this thesis worked towards gathering and analyzing information used in developing a scheduling algorithm that fits the needs of the ATLAS High Energy Physics experiment.

During the course of monitoring distributed clusters for the ATLAS experiment there are a few observations with regards to factors that affect any scheduling decision taken. Observing three sites over a period of 48 hours, we found that job execution times run into measures of hours, which means that job related information extracted from any one submit host can be updated anywhere between two and four hours and one will have a fairly good idea of the job flow. Again, observing three sites resource information over 48 hours, we found that site resource information is updated by Ganglia every 30 seconds, but since these job durations are long the state of the cluster changes only marginally in these 30 seconds. The actual implementation of how often resource related information is updated using Ganglia is entirely up to the local site administrator who deploys this system.

The ATLAS Grid Component Environment Capone is a continuously evolving process especially now that Grid3 is going through a metamorphosis into the Open Science Grid. With each evolving version and with the new standards being developed for the Open Science Grid the tools used for monitoring Clusters in the Grid will change. If any of the monitoring tools used in this thesis are no longer used for monitoring in the Open Science Grid, additional scripts need to be developed in order to continue information flow (preferably same or similar information) into the databases developed here. There is an additional parameter or data set that has been discussed as useful information in making a more informed scheduling decision and this is some form of job Transformation. This parameter has currently not been developed due to the lack of the existence of any job Transformation that will give us resource utilization based on the type of simulation or job being submitted. When this transformation becomes more readily available, its extraction and use in any scheduling decision will be helpful.

Another aspect of this system allows local site system administrators at any site to manage resources at their site such as temporary file spaces and old input files better in the case of jobs that have been executed at that site. The system provides a status check on all the jobs executed at the site which helps in making the decision of keeping or cleaning the temporary files and old input files.

Currently, a suitable scheduling algorithm that meets to the needs of the ATLAS experiment is being written. This is a venture that goes beyond the scope of this Masters thesis and hence would befit future work built on this thesis.

APPENDIX A

PSEUDO CODE FOR CREATING INFORMATION FOR TABLE FROM_GANGLIA_SITE

```
#! ***** file - creating_from_ganglia_site_info.py *****
#! ***** author - Sreeranjani Ramprakash *****
#! ***** version - 1.4 *****

def add_row(row,outputfile):
    Enter the row of average resources in the same order as how it appears in the table
in the database by using tab as a delimiter between fields
    Function called by main() below and takes the tab delimited file as input

def lookup(sitename)
    Establish connection with table Site_Name_Mapping in database

    Using the select command get the configuration Site name     corresponding to the
Ganglia Site name

Function called by main() below takes ganglia site name and returns configuration Site
name

def main()  :
    Accept output file name

    Open file to write into

    Accept xml file name to parse

    Open file to read

    Parse the XML document

Extract 'cluster' element which corresponds to the Site under    consideration

for each node in cluster:

        Insert Site_Name into dictionary after mapping Ganglia          Site name to
corresponding Site name in configuration           file using lookup()

        Extract all node elements within the Cluster in the           Site

        Count total number of nodes in cluster

        Extract sub-elements from node elements

        for each metric in each node:
            for each item in each metric:
                Extract the name of the metric and its value
                Add the value of the metric to its corresponding accumulator
        Calculate the average of each metric by dividing each accumulator by the total number of nodes
        Add each of these averaged metrics as a row corresponding to the average resources available at the Site using add_row() into output file
    Close the data files
```

APPENDIX B

PSEUDO CODE FOR CREATING INFORMATION FOR TABLE JOB_INFO

```
#! ***** file - creating_job_info.py *****
#! ***** author Sreeranjani Ramprakash *****
#! ***** version 1.8 *****

def create_dictionary(list):
    for each element in list

            Extract the value associated with key in the given list of strings using ":"
            as the delimiter between keys and values and adding them to a dictionary

    return dictionary

def add_row(dict,file):

        Enter the row of parameters associated with each job in the same order as it
        appears in the table Job_Info in the database using tab as the delimiter between
        fields

def main():
    Extract log file name from command line input

    Open log file with job status command piped output

    Accept output file name

    Open output file to write

    Create a list of strings of jobs in the log file

    Process one line(job) at a time

    for each job in list of jobs
        Extract individual job parameter values

        Split using "," ( doesn't take care of iterated lists within )

        Create a list of iterated lists

        for each parameter in the iterated list

                Check if the list has an iterated list within and extract if found
        Initialize and create dictionary using the list from above by invoking
        create_dictionary(list)

        Add one row of parameters associated with each job to output file

Close log file
```

APPENDIX C

PSEUDO CODE FOR CREATING INFORMATION FOR TABLE SITE_RESOURCES

```
#! ***** file - creating_site_resources_info.py
#! ***** author Sreeranjani Ramprakash *****
#! ***** version 1.1 *****


def add_row(row,outputfile):

     Enter the row of static resources in the same order as how it appears in the table
     in the database by using tab as a delimiter between fields


def main() :

   Accept output file name

   Open output file to write

   Accept xml file name to parse

   Open file to read

   Parse the XML document

     Extract 'cluster' element which corresponds to the Site under consideration

   for each node in cluster:

          Insert Site_Name into dictionary after mapping Ganglia Site name to
          corresponding Site name in configuration file using lookup()

      Extract all node elements within the Cluster in the Site

      Count total number of nodes in cluster

      Extract sub-elements from node elements

      for each metric in each node:

         for each item in each metric:

            Extract the name of the metric and its value

          Assign the static information as key value pairs to a dictionary
```

```
        Add each of these static metrics as a row corresponding to the static
        resources available at the Site using add_row() into output file

Close the data files
```

APPENDIX D

PSEUDO CODE FOR CREATING INFORMATION FOR TABLE SITE_SERVICES

```
#! ***** file - creating_site_services.py *****
#! ***** author Sreeranjani Ramprakash *****
#! ***** version 1.4 *****

def add_row(dict,file):

        Enter the row of site services information in the same order as how it appears in
        the table in the database by using tab as a delimiter between fields

def main() :

      Accept output file name

      Open output file to write

      Accept xml file name to parse which is the configuration file

      Open file to read

      Parse the XML document

      Extract 'pool' elements which corresponds to all the site entries in the configuration
      file

      Extract individual elements / attributes from 'pool' element
      for each tag in pool element

                Extract required information that includes site name, GridFTP server
                node address, gatekeeper node address, working directory at this cluster

                Add all extracted information to output file as a row using add_row()

      Close data file
```

APPENDIX E

PSEUDO CODE FOR POPULATING THE DATABASE WITH ALL THE DATA FILES CREATED

```
#! ***** file- populating_database *****
#! ***** author Sreeranjani Ramprakash *****
#! ***** version 1.0 *****

def add_info_to_db(filename):

     Connect to the MySQL server

     Create query string to load data from tab separated file into respective table in
database using carriage return as line separator

     Execute query

def main()

     Extract data file from command line and pass it to function add_info_to_db()
```

REFERENCES

[1] Foster, I., and Kesselman, C. (eds.). The Grid: Blueprint for a new computing infrastructure. Morgan Kaufmann, 1999.

[2] Foster, I., and Kesselman, C., Tuecke, S. The Anatomy of the Grid – Enabling Scalable Virtual Organizations.

[3] Subramani, V., Kettimuthu, R., Srinivasan, S., Sadayappan, P. Distributed job scheduling on computational Grids using multiple simultaneous Request. High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on 23 - 26 July 2002 Page(s):359 – 366.

[4] Zhang, W., Fang, B., He, H., Zhang, H., Hu, M. Multisite resource selectin and scheduling algorithm on computational Grid. Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International 26-30 April 2004 Page(s):105.

[5] Fujimoto, N., Hagihara, K. A comparison among Grid scheduling algorithms for independent coarse-grained tasks. SAINT 2004 Workshop on High Performance Grid Computing and Networking, IEEE Press, pp.674-680.

[6] Foster, I., Kesselman, C., The Globus Project: A status report. Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh 30 March 1998 Page(s):4 – 18.

[7] Park, S.M ., Kim, J-H., Chameleon: A resource scheduler in a data Grid environment. Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on 12-15 May 2003 Page(s):258 – 265.

[8] http://griddev.uchicago.edu/swhome/atgce/description.html referred on 18th June 2005.

[9] http://www-unix.globus.org/toolkit/docs/3.2/rls/key/index.html referred on 25th June 2005.

[10] http://ganglia.sourceforge.net/ referred on 26th June 2005.

[11] Foster, I., Kesselman, C., Nick, J.M., Tuecke, S. The Physiology of the Grid – An Open Grid Services architecture for distributed systems Integration.

[12] Roehrig, M., Ziegler, W. Grid scheduling Dictionary of Terms and Keywords.

[13] Foster, I., The Grid: A New Infrastructure For 21st Century Science , Article – Physics Today.

[14] Takefusa, A., Bricks: A Performance Evaluation system for scheduling algorithms on the Grids. http://ninf.is.titech.ac.jp/bricks .

[15] He, X., Sun, X-H., Laszewski, G.V., A QoS Guided scheduling algorithm for Grid Computing.

[16] Luo, J., Ji, P., Wang, X., Zhu, Y., Li, F., Ma, T., Wang, X. Resource management and task scheduling in Grid Computing.

[17] Foster, I. What is the Grid? A Three Point Checklist.

[18] Schopf, J.M., Nitzberg, B. Grids: The Top Ten Questions.

[19] http://www.globus.org/toolkit/data/rls/ referred on 18th June 2005.

[20] http://www.cs.wisc.edu/vdt//index.html referred on 18th June 2005.

[21] http://www.griphyn.org/chimera/ referred on 25th June 2005.

[22] http://www.ivdgl.org/grid3/ referred on 20th July 2005.

[23] http://globus.org/toolkit/about.html referred on 20th July 2005.

[24] http://monalisa.cacr.caltech.edu/ referred on 20th July 2005.

[25] Frey, J., Tannenbaum, T., Foster, I., Livny, M., Tuecke, S. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001.

[26] http://mbranco.home.cern.ch/mbranco/cern/donquijote/index.html referred on 20th July 2005.

[27] http://www.nordugrid.org/ referred on 20th July 2005.

[28] http://lcg.web.cern.ch/LCG/ referred on 20th July 2005.

[29] http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/grid/ referred on 20th July 2005.

[30] http://www.usatlas.bnl.gov/computing/software/pacman/ referred on 18th June 2005.

[31] http://heppc12.uta.edu/~thomas/hepweb/dpcc.html referred on 6th June 2005.

BIOGRAPHICAL INFORMATION

Sreeranjani Ramprakash graduated with a Bachelors degree in Computer Science and Engineering from Mangalore University, India in June 2001. After working for Dell International and Digital Global Solutions in India for a year she came to UT Arlington to pursue her Masters in Computer Science and Engineering. Being interested in networking in general she discovered Grid computing and pursued her interest further by completing a small project converting serialized job processing software to multiple job processing software. After this, she found what eventually went on to comprise her main research topic which is described in detail in this thesis. She graduated from University of Texas at Arlington with a Masters in Computer Science and Engineering in August 2005.