

Optimizing Grid Scheduling Based on Local Cluster Scheduling Policies and Resource Availability

Tummalapalli Sudhamsh Reddy, David Levine, Farhad Kamangar and Nirmal Ranganathan
Computer Science and Engineering Department
The University of Texas at Arlington
Arlington, TX USA

***Abstract**—Current grid implementations are based on the common paradigm of treating clusters as separate administrative domains. Clusters are grouped to form grids which participate in Virtual Organizations (VO) and share resources to meet common computing goals. Each cluster is a separate administrative domain and can choose to be a part of many VOs. Because of the nature of a VO, a cluster typically gives a higher preference to scheduling jobs for one VO over other VOs. In this paper we present a grid scheduling architecture which takes into consideration the current resource availability and scheduling policies of clusters at various sites. This architecture considers both hardware and software resources. The primary goal of this scheduler is to reduce the completion time of jobs by reducing the queuing time of jobs, also providing an increase in CPU utilization. We show the effect of our scheduling technique using a prototype implementation and compare the results against a typical grid scheduler based on random allocation of resources.*

***Keywords:** Cluster policies, Grid Computing, Resource Description Language, Resource selection.*

1 Introduction

One significant goal of grid computing is to provide users with vast computational resources to solve large problems quickly [1],[2]. In most cases users want to get the results of their job back as soon as possible, and then run additional jobs to refine their results.

This work was supported, in part, by the NSF grant EIA-0216500. T.S.Reddy* is with the SAMGrid Team at Fermi National Accelerator Labs, on leave from the Computer Science Department, University of Texas at Arlington (CSE@UTA) reddy@fnal.gov . David Levine* is with (CSE@UTA) levine@cse.uta.edu Dr. Farhad Kamangar is with (CSE@UTA) kamangar@cse.uta.edu N. Ranganathan* is with (CSE@UTA) nrangana@cse.uta.edu

Clusters come together to form a grid by participating in Virtual Organizations (VO) [3]. The grids, by definition, consist of clusters [3], where each cluster is an administrative domain in itself. In order to do scheduling on the grid one needs a grid scheduler which interacts with the local cluster scheduler to schedule jobs on cluster nodes. But the grid scheduler does not have direct control over the local cluster schedulers, and each local cluster administrator may use different scheduling policies.

Because there is no control over an individual cluster's policy, any generalization of cluster scheduling policies is not possible. Yet, to efficiently schedule jobs on these clusters, one needs to understand or recognize the scheduling policies of these individual clusters. Some of the current VOs do their job scheduling by manually selecting which cluster receives which job or by randomly selecting a resource from a set of resources. In order to automate this procedure and to get significant improvement, there needs to be a mechanism to specify or express the cluster scheduling policies which can be modeled and automated.

In this paper we propose a policy description language (PDL) which is used to describe the policies of individual clusters. We also present a grid scheduling architecture that enables us to get an accurate picture of the current state and resource availability at each of the clusters, reducing the turn around time of the jobs and also increasing the resource utilization of the cluster.

The remainder of the paper is organized as follows: Section 2 contains a view of the related work. Section 3 contains the description of our Policy Description Language (PDL). Section 4 contains the proposed architecture. Section 5 contains the scheduling algorithm. Section 6 contains the implementation and results. Section 7 contains the conclusion and future work.

2 Related Work

The issue of Grid scheduling has been of major interest [4]-[9]. In [4] a conservative scheduling technique is used which is based on a predictor to predict variance to make scheduling decisions and adjust the algorithm based on a feedback mechanism. In [5] a compensation based algorithm is presented which also uses a feedback mechanism to provide a predictable execution time for jobs. In [6] a comparison of 11 static heuristic scheduling methods is discussed. In [7] a metascheduler for the grid is discussed with reference to the GrADS architecture. The goals of this approach was to minimize job execution time by using various methods such as stopping a bigger job so that a smaller job can proceed first, facilitate the execution of new jobs by stopping competing jobs, and minimizing the impact of new jobs on running jobs. In [8] the architecture proposed uses Globus [10],[11] as an underlying middleware to provide scheduling based on performance predictions.

The impact of cluster scheduling policies has been studied by using game theory [12] where the scheduling behavior of clusters has been termed ‘Selfish Behavior’ [12] as they give higher preference to jobs originating from the local site and lower priority to jobs originating from a remote site, this is observable since local users are given more importance than remote users. The impact of local cluster policies on grid scheduling has been studied in [13] where the simulated results show that local policies have a significant impact on response time of jobs. A greedy approach has also been proposed to overcome this behavior. This approach has been proposed based on assigning weights to different clusters based on how different their scheduling approach is from the approach of the local system. Though this technique gives good results, it is a very simplistic method which does not take into account how remote policies are different from local site policies. These remote site policies might be different than policies at a local site yet be advantageous for a user from one VO. In [14] the authors have proposed a prediction based system using historical information and local cluster scheduling policies. But, in this approach, the authors assume that the targeted site does not use dynamic policies such as dynamic prioritization and usage based throttling policies which is the case of Weighted Fair Sharing of resources, available with most Batch Systems like PBS, Torque and Condor. These assumptions are not realistic policies since weighted fair share is both popular and widely used.

A description language for resources, as well as jobs, has been discussed and successfully used in

Condor [15] and Globus. Examples of this type of language are ClassAds [16] in Condor, Resource Specification Language [17] in Globus and Glue Schema [18]. Condor uses the language to do matchmaking [19]. A constraint language for resource selection is discussed in [20] where multiple types of resources are considered. Glue Schema details can be sent as LDIF format or as Condor ClassAds. While Glue schema provides many of the details we are interested in, it lacks important custom parameters in its current version.

One weakness of the previously noted approaches is that, for the most part, they don’t consider the local policies of individual cluster sites while selecting the jobs, which leads to an increase in job queuing time. While [12] consider the selfish behavior, it has not presented any method to model the policies of clusters and in [13] a simplistic model of local jobs versus remote jobs has been presented, but no distinction has been made between types of remote jobs, where, in practice, a distinction is common and often some VOs are allowed to use more resources than the others.

3 Policy Description Language

The Policy Description Language (PDL) is used to express the current state of a cluster with respect to one, individual VO. Since the cluster policy may be changed by an administrator at anytime, there is a need for a PDL; for example, this is the case in our local cluster (DPCC). DPCC uses a weighted fair share policy where individual users belonging to a VO are grouped together and are given certain weight. The weight for each VO is assigned by the system administrator. The weight for each group is reduced when the amount of CPU usage by their VO is large, and the weight increases when the group has not used the system for some time (called “half life time” at which time their CPU usage hours are reduced in half). The current shared weight of the VO provides it with its priority such that when users submit jobs to the cluster - the jobs from users with higher priority are executed first. Since the weights of the groups change over time, one needs a method to accurately represent the current state of the system. The clusters which use simple policies such as First Come First Served, and Shortest Jobs First are easier to represent. Another issue is that all VOs are not treated in the same way when it comes to allocating resources. It is common that certain VOs are allowed to access more resources on a cluster than others, which are also decided by the system administrator; for example a high energy physics group may access up to 130 nodes simultaneously but a bio-informatics group may concurrently access only 30 nodes.

A system administrator may change policies at will, these policies frequently change over time and more often over a longer period of time, thus a mechanism is needed to represent the current policies at the current time of job scheduling. At the time of scheduling, the PDL of each cluster is needed, which will enable reducing the queuing time of jobs and hence reduce the overall execution of jobs. We propose a PDL, written in XML format, to facilitate creation and interpretation by all clusters and schedulers. A description of this PDL is given below:

As shown in Figure 1, the policy description language (in XML format) contains the following fields:

```

<?xml version="1.0" encoding="utf-8" ?>
<Policy_file>
  <cluster_name>DPCC</cluster_name>
  <Max_Nodes>10</Max_Nodes>
  <Scheduling_policy>WFS</Scheduling_policy>
  <Pre_empt_jobs>No</Pre_empt_jobs>
  <Security_Policy>
    <Certificate_Authority>DOEGRIDS</Certificate_Authority>
    <Certificate_Authority>DPCC</Certificate_Authority>
  </Security_Policy>
  <Max_Time_For_Starvation>48:00:00</Max_Time_For_Starvation>
  <Current_Group>
    <Group_name>scetest1</Group_name>
    <Current_Jobs>2</Current_Jobs>
    <Max_Jobs>10</Max_Jobs>
    <Current_Queue_Weight>1</Current_Queue_Weight>
    <Current_Queue_Size>0</Current_Queue_Size>
    <Max_Queue_Size>20</Max_Queue_Size>
    <Max_Runtime>10:00:00</Max_Runtime>
  </Current_Group>
  <Current_Group>
    <Group_name>scetest2</Group_name>
    :
    :
  </Policy_file>

```

Fig. 1. Policy Description File of DPCC cluster.

The PDL details the name of the cluster, the maximum nodes in the cluster, the scheduling policy of the cluster (such as FIFO, Shortest Job First, Weighted Fair Share, etc.). The PDL also includes a field representing whether the cluster can preempt a job, a list of certificate authorities that can issue a certificate to use the cluster resources and a field indicating the maximum time a job can be in a queue without executing, after which it must be executed before executing any other job. Once these parameters are supplied, we look at the parameters that are associated with an individual group: the name of the group, the number of jobs currently executing, maximum queue length for the group, the number of jobs that are currently queued, the maximum number of jobs that can be executing in the system at anytime from that group, the weight of the group, the maximum time for which any job can run before it is

stopped. These parameters are obtained by parsing the configuration files present on the cluster head node and expressed in XML. Figure 1 shows the state of the DPCC cluster.

4 Architecture

As shown in Figure 2, the proposed architecture contains four modules: a Cluster Hardware Monitor, a Cluster Software Monitor, a Cluster Policy Descriptor and Grid Scheduler. The first three modules collect information from each cluster and provide it to the scheduler. The scheduler presents a single point of access to the grid services. The scheduler then uses this information to schedule jobs on clusters based on the requirements of the jobs which are presented to the scheduler using a Resource Specification Language such as ClassAds or RSL. The Scheduler then does a scheduling process which uses the hardware resource availability information from the Cluster Hardware Monitor, software resource availability information from the Software Resource Monitor and the policy description of the clusters from the Cluster Policy Descriptor to satisfy the constraints provided by the RSL or ClassAds.

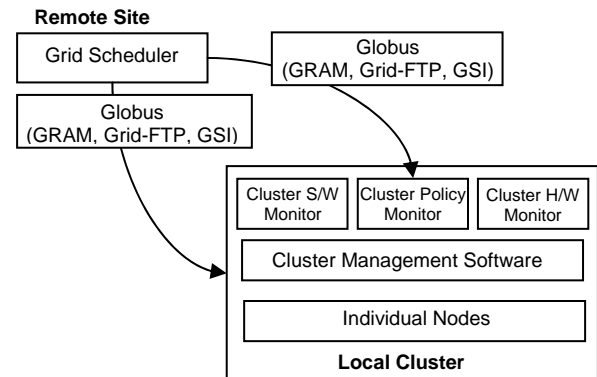


Fig. 2. Proposed Architecture

The Cluster Hardware Monitor collects information from the cluster using either the cluster monitoring systems like Ganglia [21] or MonALISA [22], or it can collect this information from the cluster management system like Condor or PBS by simple queries.

The software resource monitor collects information about installed software, for example, the databases available at a particular site or the versions of an operating system; this information can either be collected from the cluster management system directly, or every time a new resource is added the cluster administrator can update a file that is kept on

the master node which can be read by the software monitoring agent.

The policy description language files are obtained by the cluster policy monitor and are also kept at the cluster head node and can be read whenever they are required by the scheduler. The cluster policy monitor records the information from the cluster management systems and updates the information which is read by the grid scheduler during the scheduling process.

5 Scheduling

The input jobs are given to the scheduler and the scheduler then gets the required information from the hardware, software monitors and the policy descriptor. The scheduler process goes through the jobs one by one until all the jobs are processed. For each job, the list of clusters that can satisfy the software requirements are used and then the scheduler finds the hardware resources and policies of each of those clusters. From this information, it decides which cluster can give the best completion time, which is the sum of wait time and execution times. It will select the cluster and assign the job to the cluster. Figure 3 lists our scheduling algorithm.

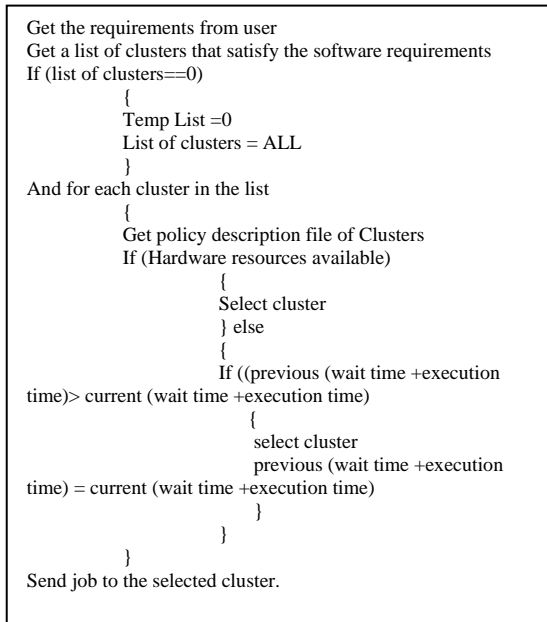


Fig. 3. The scheduling algorithm.

Since the clusters are selected dynamically during the execution of the scheduler, load balancing is also addressed if any cluster is heavily loaded then the scheduler will automatically not pick that cluster, since it will give us a longer wait time and the heavy load can be seen from the hardware resource monitor

as well as from the cluster policy description language.

6 Implementation and Results

We have implemented our scheduling algorithm using Barracuda, a 6 node cluster, Gecko, a 5 node cluster and a cluster of 3 nodes. The 3 node dual processor cluster is part of the DPCC cluster. The three clusters form a small grid for the purpose of our experiment. This setup was chosen because it is a typically configured small grid.

For the purpose of creating a representative scenario we have mis-configured one of the clusters. This has been done so that one cluster has a higher rate of failures as is normally seen in the case of such small grids. Four users are synthesized, members of three groups, such that two groups contain one member each and the third group contains two users. All the clusters run the PBS cluster management system and Globus software for communicating and management. The Barracuda and Gecko cluster have a simple FIFO scheduling policy such that jobs from all users are mapped into a single queue and then the order of execution is the order of the arrival of jobs into the queue. The DPCC cluster has a slightly more complex scheduling policy called Weighted Fair Share, where the users are mapped into different queues depending on their groups. Each queue has a different weight and attributes such as max queue length, max jobs, max runtime, and so forth. Depending on the value of these parameters, the scheduler picks up the job from the queue. Therefore, scheduling efficiently on such a cluster will require a better understanding of the local site status, for which we use the PDL. All modules of the proposed architecture are written in Perl and use Globus and PBS to communicate and collect information.

The Cluster Hardware Monitor was implemented by writing a wrapper around a PBS command called 'qstat'. The cluster hardware monitor collects the information from PBS and writes the information into a file. This file is then copied by the scheduler using the 'globus-url-copy' command, into the local drive and then using the contents of the file, giving the number of nodes that are currently free in that cluster.

The Cluster Software Monitor is implemented as a repository (file), where the System administrator enters the name of the software currently available, and when new software is installed, the file is modified to reflect the change in the system. The scheduler also copies this file into the local drive and uses its information while scheduling (matchmaking).

The Cluster Policy Monitor is implemented by parsing the 'server.conf' file on PBS head node and also by parsing the output of the PBS command 'qstat'. The information collected is the number of queues, and the queue attributes: max jobs and max queue size. The information obtained from parsing the 'qstat' output contains the number of jobs running from each queue and the number of jobs that are currently queued in the system for each queue. This information is written into an XML file, which is copied by the scheduler using 'globus-url-copy' during the scheduling process.

The scheduler first gets the software availability files from the clusters and then depending on the monitored values, decides from which clusters to get the remainder of the information. The scheduler then gets the hardware resource files and cluster policy files from these clusters. Finally the scheduler does an XPath query on the XML files and gets the required information: priority of the user, current number of jobs, and current queue size. Based on these parameters, it schedules the jobs.

We test our algorithm against a widely used approach: selecting sites based on randomness. In our implementation we randomly select any one of the three sites and submit our job.

To evaluate our proposed architecture, we synthesize and submit jobs to the grid scheduler. The inter-arrival time of jobs is a normal distribution of a mean of 20 seconds and standard deviation of 5 seconds, the running length of the jobs is also in a normal distribution of mean 180 and the standard deviation is 15 seconds.

From our experiments we observe that the number of job failures is higher in the case of random selection approach. This is mainly due to the fact that when the jobs start failing to be scheduled on to the worker nodes, they are reflected in the PDL as the jobs are queued, and while our approach takes that into consideration the random selection process does not and still selects that resource while ours stops using that resource and schedules to other resources.

From Figures 4 and 5 we also observe that the Queuing (or starting) time of jobs in the case where there were no job failures as well as in the case of job failures is lower by 25% - 30% in the case of the proposed approach over the random approach.

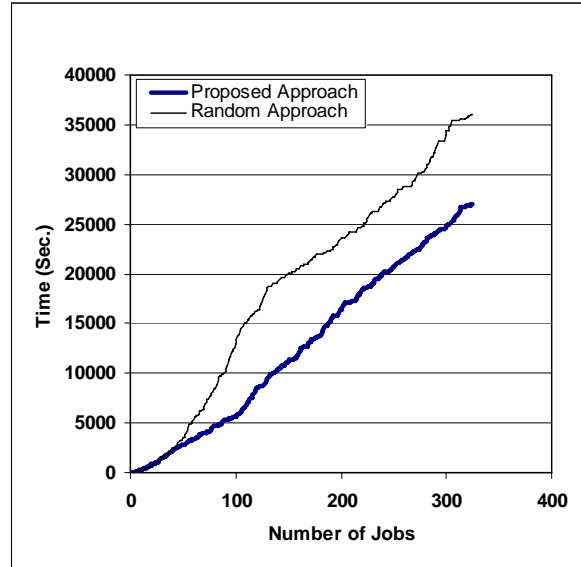


Fig. 4 Cumulative Queuing Time of Jobs during execution indicating job failures.

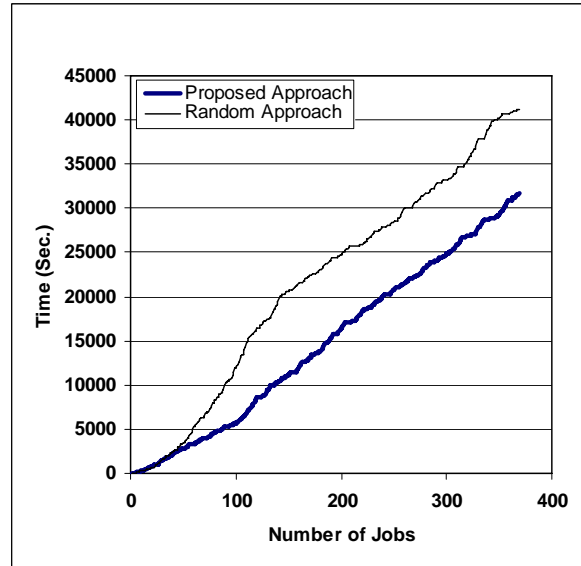


Fig. 5 Cumulative Queuing Time of Jobs during execution with extrapolated job queuing times for failed jobs.

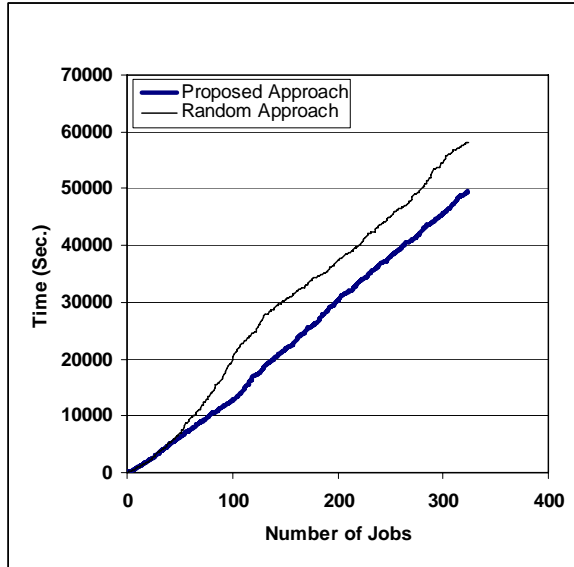


Fig. 6. Cumulative Execution Time of Jobs indicating job failures.

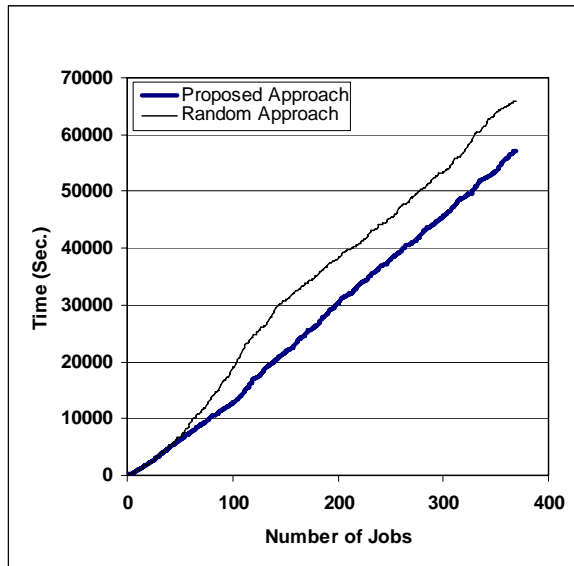


Fig. 7. Cumulative Execution Time of Jobs with extrapolated job execution times for failed jobs.

The total execution time, as shown in Figures 6 and 7, also follows a similar trend as this is dependent on the start time, since the execution time of each individual job is similar due to the job mix being similar.

Since the proposed approach is able to achieve faster start times and shorter total execution times for jobs, the resources are being better utilized. This increases the efficiency of the whole system since it

can execute more jobs using the proposed approach in the same amount of time as compared to the random selection process.

7 Conclusion

As may be seen from the results presented, the proposed approach performs much better than the random selection process. Since the system is dynamic, it also addresses basic load balancing. The theme of this work is based on the adage that the more information one has, the better decision one can make. The proposed approach has a significantly better performance because of the use of the policy description language during scheduling.

The algorithm that has been presented also lends itself to address various other issues such as memory usage of jobs on cluster nodes, the amount of I/O a job does, and other job variances. However, even the straightforward approach noted in this paper gives substantially improved performance, in several important metrics.

8 References

- [1] F. Berman, G. Fox, and T. Hey, "Grid Computing: Making the Global Infrastructure a Reality", John Wiley & Sons, 2003.
- [2] Neesgrid, <http://www.neesgrid.org/index.php>
- [3] Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid", International Journal of Supercomputer Applications, 2001.
- [4] L. Yang, J. Schopf, I. Foster, "Conservative Scheduling: Using Predicted Variance to improve Scheduling Decisions in Dynamic Environments", ACM/IEEE SC2003 Conference (SC'03).
- [5] Y. Tao, X. Wang and J. Gozali, "A Compensation-based Scheduling Scheme for Grid Computing", 7th International Conference on High Performance Computing and Grid in Asia Pacific Region, 2004.
- [6] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", Journal of Parallel and Distributed Computing 61, 810-837 (2001).
- [7] S. Vadhiyar, J. Dongarra, "A Metascheduler for the Grid", 11th IEEE international Symposium on High Performance Distributed Computing HPDC-2002.

- [8] D. Spooner, S. Jarvis, J. Cao, S. Saini and G. Nudd, "Local grid scheduling techniques using performance prediction", IEE proceedings:-Computer Digest Tech., Vol. 150, No. 2, March 2003.
- [9] M. Wu, X. Sun, "A General Self-adaptive task Scheduling System for Non-dedicate heterogeneous Computing", IEEE International Conference on Cluster Computing (CLUSTER'03).
- [10] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", Intl J. Supercomputer Applications, 11(2):115-128, 1997.
- [11] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems", IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.
- [12] Y. Kwok, S. Song, K. Hwang, "Selfish Grid Computing: Game-Theoretic Modeling and NAS Performance Results", CCGrid 2005.
- [13] S. Wiryaprasit and V. Muangsin, "The impact of Local Priority Policies on Grid Scheduling Performance and an Adaptive policy-based Grid Scheduling Algorithm.", 7th International Conference on High Performance Computing and Grid in Asia Pacific Region, 2004.
- [14] H. Li, D. Groep, J. Templon, L. Wolters, "Predicting Job Start Times on Clusters", In Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid Chicago, Illinois, USA, April 19-22, 2004.
- [15] Condor, <http://www.cs.wisc.edu/condor/>
- [16] R. Raman, M. Livny, and M. Solomon, "Resource management through multilateral matchmaking", IEEE International Symposium on High-Performance Distributed Computing (HPDC), 2000.
- [17] Globus, RSL document, http://www-fp.globus.org/gram/rsl_spec1.html
- [18] S. Androozzi. GLUE Schema Implementation for the LDAP Data Model. Technical Report INFN/TC-04/16. 30 September 2004.
- [19] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", IEEE International Symposium on High-Performance Distributed Computing (HPDC), 1998.
- [20] I. Foster, and C.Liu, "A Constraint Language Approach to Grid Resource Selection", Unpublished Manuscript.
- [21] Matthew L. Massie, Brent N. Chun, and David E. Culler, "The Ganga Distributed Monitoring System: Design, Implementation, and Experience", Parallel Computing, Vol. 30, Issue 7, July 2004
- [22] H.B. Newman, I.C.Legrand, P. Galvez, R. Voicu, C. Cirstoiu, "MonALISA : A Distributed Monitoring Service Architecture" CHEP03, La Jolla, California, March 24-28, 2003