SWARM: Scheduling Large-scale Jobs over the Loosely-Coupled HPC Clusters

Sangmi Lee Pallickara[1] and Marlon Pierce[2]

[1]University Information and Technology Service and [2]Community Grid Lab,

Indiana University, Bloomington, Indiana

{leesangm@cs.indiana.edu[1], mpierce@cs.indiana.edu[2]}

Abstract

Compute-intensive scientific applications are heavily reliant on the available quantity of computing resources. The Grid paradigm provides a large scale computing environment for scientific users. However, conventional Grid job submission tools do not provide a high-level job scheduling environment for these users across multiple institutions. For extremely large number of jobs, a more scalable job scheduling framework that can leverage highly distributed clusters and supercomputers is required. In this paper, we propose a high-level job scheduling Web service framework, Swarm. Swarm is developed for scientific applications that must submit massive number of high-throughput jobs or workflows to highly distributed computing clusters. The Swarm service itself is designed to be extensible, lightweight, and easily installable on a desktop or small server. As a Web service, derivative services based on Swarm can be straightforwardly integrated with Web portals and science gateways. This paper provides the motivation for this research, the architecture of the Swarm framework, and a performance evaluation of the system prototype.

1. Introduction

In response to the growing need for high-performance computing resources and data storage, several research centers have contributed to projects based on the Grid computing paradigm such as the TeraGrid project [1] and the Open Grid Science project [2]. The NSF funded TeraGrid project provides access to high performance computing (HPC) resources hosted by 11 institutions in the US. Cumulatively, the Teragrid project provides access to more than 110,000 CPUs that can deliver a peak computing throughput of 900 Teraflops. The size of the TeraGrid resources should increase dramatically over the next several years as large new resources (Big Blue at NCSA and Kraken at the University of Tennessee/Oak Ridge National Lab) come on line. Scientific users have obviously benefitted from this dramatic enhancement in the available computing resources. However, seamlessly adapting remote HPC resources continues to be a challenge. This is especially true for the Cyberinfrastructure or Gateway user community, where Grids resources need to be integrated within a larger group of users and research projects.

Although the Grid paradigm provides an excellent model for accessing remote resources, emerging scientific problems in high throughput workflows pose a challenge. We have encountered these computational challenges in our work on research projects in the bioinformatics and biochemistry domains. The PlantGDB project [3], in the bioinformatics domain, assembles unique transcripts from plant mRNA (messenger RNA)) sequences. Here, mRNA is a copy of the information carried by a gene on the DNA. Records downloaded from GenBank are processed through a data processing pipeline. This pipeline involves clustering and assembly sequences, which are highly compute intensive. The overlapping sequences among the parts of gene sequences are grouped by software, PaCE [4] that is often run in parallel. The results of this clustering process are then fed to the assembly process to generate contigs, which are the consensus sequences derived from multiple mRNA sequences. Depending on the species, the number of the clustered sequences varies from a few to several millions, in which case a large number of assembly processes are required.

To benefit from powerful computing resources, users login to remote machines or submit jobs through Grid toolkits such as the Globus Toolkits [5]. However, the sheer number of jobs easily exceeds one's ability to manage them manually. Finding the most efficient resource to submit, and monitoring such submitted jobs, is not feasible without intelligent support from the middleware. Furthermore, the Gateway-style applications also require interfaces to distributed resources on behalf of the users.

To cope with issues related to scale and management in such large-scale settings we have developed a high-level job scheduling framework, Swarm which provides the following features:

- Scheduling millions of jobs over distributed clusters
- A monitoring framework for large scale jobs
- User based job scheduling
- Ranking resources based on predicted wait times
- Standard Web Service interface for web applications

In grid environment, it has been realized that job scheduling is a fundamental issue to improve resource utilization and performance. GridWay[6], and PanDa[7] project provide job submission environment over the multi-site resources. On top of the scheduling functionality, Swarm provides the resource prioritizing feature which searches the batch queue system with the minimum wait time. Falkon[8], and myCluster[9] enable user to access provisioned resources to submit large-scale scientific jobs. Instead of provisioning resources, Swarm provides user-based resource pool which limits maximum number submissions to the batch queue system. It enables Swarm to incorporate with the policies from each batch queue systems more flexibly. In addition, the highly extensible design for the domain specific applications and lightweight software distinguish Swarm from other approaches.

The aims of this paper are threefold: first, to present the architecture of the Swarm job scheduling framework; second, to describe the scheme that prioritizes resources and distributes a massive number of jobs over distributed clusters; and finally, to evaluate performance at the client level of the standard Web Service.

The rest of the paper is organized as follows: Section 2 describes some of the scientific research projects that motivated our research. Related work is discussed in the section 3. In the section 4, we describe the architecture of Swarm. Performance evaluation of the Swarm system is presented in section 5. Conclusions and future work are discussed in section 6.

2. Motivations and Computational Challenges

For compute-intensive scientific applications, Grid enabled resources opened up the possibility of an on-demand experimental environment. Here we describe the computational challenges that we faced that in turn motivated us to develop the Swarm framework.

Challenge 1: Executing millions of jobs.

The PlantGDB project [3], in the bioinformatics domain, clusters and assembles mRNA sequences. To assemble the clustered sequences, the data processing pipeline runs from a few to millions of sequence assembling jobs, which are independent of each other. Similarly, the Meroueh's research [10, 11] in the biochemistry domain has similar issues in their scheme for drug discovery. Here, the compound discovery process launches millions of jobs identifying the low-energy binding modes of a small molecule within the active site of a receptor, whose structure is known. In both these projects, the number of jobs submitted by a single experiment could vary from a handful to several millions. Currently, job submission mechanism provided by Globus Toolkit which is de facto software in grid community does not allow the users to submit 1000s of jobs concurrently to the PBS like batch queue systems [12, 13].

Challenge 2: In different clusters, the same job can have different wait times in the batch queue.

Users also have to deal with the policies at these remote sites. These policies are applied to the batch system to which the jobs are submitted. Based on information provided by the users a given batch system prioritizes jobs within its queue. Queue wait-times typically have a significant effect on the total computing time for a given scientific experiment. Two other factors also play a very important role: the WallClockTime, which is the duration of the execution, and the number of available nodes for parallel jobs. To optimize the job execution times, users need to take into account the queue waits, the WallClockTime, and the number of available nodes.

Challenge 3: What if some of the jobs are incomplete or failed?

Fault handling is another critical challenge. There are potential failures in the resource side such as hardware failure or system shutdown. Some of the errors are program specific. For example, WallClockTime is specified based on the user's estimation. Therefore, there is a possibility to have incomplete executions due to WallClockTime violation. Fault detection and more intelligent reaction to the execution fault are required for the convenience.

Challenge 4: Monitoring millions of jobs running over several clusters.

After submitting millions of jobs, monitoring each of these jobs − based on their individual job IDs − is not practically feasible. Compounding these tracking problems is the fact that these jobs might be submitted to several different clusters. Traditional monitoring will not suffice: a more statistical approach is needed.

3. Related Works

There have been several approaches in the HPC community and the Grid community. Condor [14] is a well-known high-throughput resource management system that has been widely adopted in the scientific computing community. The Condor system provides advanced features including fault tolerance. The CondorG [15] release of Condor interoperates with other Grid computing resource management services such as the Globus toolkit. CondorG does not provide direct support for a scheduling policy for jobs submitted to Grid resources. However, it does supply mechanisms that may be useful for meta-schedulersat higher-levels in the stack; examples of such meta-schedulers include systems like ClassAd [16] and DAGMan [17].ClassAd allows users to specify the remote resources for jobs within the matchmaking process. DAGMan is a workflow manager where interdependencies between jobs or data can be specified. Swarm utilizes CondorG as the basic job submitter.

GridWay [6]is another metascheduling framework for grid resources. Besides the scheduling capability, GridWay provides other advanced features − such as fault tolerance, checkpoints, and process migration − that are not available to users who access Grid resources directly. Similarly, PanDa[7] provides large-scale job scheduling and analysis framework. PanDa is originally developed for a particle physics experiment at the Large Hadron Collider. For their large size datasets, PanDa interacts with own data management service to pre-place the dataset. CondorG, GridWay, and PanDa harness the Globus toolkit to cope with security and policy issues in Grid settings.

To utilize remote clusters managed by different institutions the glide-in style approach is useful. Glide-in style tools utilize computing nodes in the user's personal resource pool by submitting parallel jobs on these pools. CondorG provides a condor glide-in server [14], where users can submit jobs to these provisioned clusters as if they were the condor computing nodes.

myCluster [9] and Falkon[8] are built on top of the glide-in approach. myCluster provides the capability of provisioning a large number of distributed resources across the TeraGrid into personal clusters created on-demand. Falkon provides support for provisioning a large number of distributed resources and allows user groups access to resources via a Web service interface. It also factors in data management techniques to improve the performance. Glide-in style approach provides transparent access to remote resources.

At its lowest level of job management, Swarm also utilizes Globus technology along with the Grid's security scheme. As described in Section 5, we utilize Condor-G and BirdBath [18], Condor's Web Service interface. Unlike the glide-in style approaches, Swarm does not provide provisioning of resources. Swarm's resource pool is a set of tokens that can limit the utilization of resources. These tokens are not shared by users. Resources are allocated by the matchmaking process based on the predicted queue wait times. Large number of users can thus utilize Swarm for submitting jobs to the most efficient resources available in different grid clusters.

4. Managing Many Jobs with Swarm

4.1. Submitting Jobs

Users access the Swarm framework through a simple Web service client. This is useful for desktop users and Gateway style applications that need lightweight clients. This ease-of-use feature is also applied to file management. Swarm provides support for third party job submissions. Users or applications do not have to maintain input files allowing them to be lightweight: valid URLs to these input files can be specified instead. Similarly, the

generated output files are also provided as URLs. Swarm can thus satisfy requirements of high-throughput computing users who also prefer thin clients.

The need for launching millions of jobs is often associated with a single scientific experiment. To track large scale experiments, Swarm requires users to get a ticket before submitting millions of jobs. This ticket is randomly generated by Swarm and the user must provide this ticket for subsequent accesses related to submitting jobs, checking status, getting outputs, and canceling jobs.

Jobs are managed on a per user (user's account) basis. Since most supercomputing clusters manage their batch queue systems based on the users' accounts, Swarm provides queuing and submission mechanisms based on such individual accounts. Individual users have their own resource pool to track the usage of the resources. The tokens in the resource pool control the maximum number of jobs in the batch system.

4.2 Tracking the Status

Once the user submits a large group of jobs, tracking the status of these jobs is critical. Checking each of the jobs with millions of ids in several or more computing sites manually is not practical. Therefore, Swarm provides statistical status reports to the users. Each of the jobs maintains the status of:

- Requested: For jobs that stay in the backend Database
- Queued: For jobs in the Swarm user queue
- Submitted: For the jobs with available resources
- Idle: For the jobs waiting in batch queue system in the cluster
- Completed: For jobs that have completed
- Held: For jobs that been held
- Running: For the jobs being executed in the cluster

A status check provides the summary of the job status.

4.3 Flexible Resource Allocation

Although Swarm is client-side job submission middleware, Swarm is designed to work with a wide scale of computing resources. Unlike glide-in style solutions, Swarm does not require large time slot of a given computing resource. Instead of waiting for longer time in the queue to get a bigger chunk of the time slot from the batch queue system, Swarm targets any of the available resources for running the jobs. The scheduler evaluates the resources within the list of resources that the user has provided. This evaluation is based on the wait-time prediction by means of adapting the QBETS service [19]. This scheme provides better flexibility to utilize different clusters and their batch queue systems.

5. Architecture

Swarm is a set of Web services and local servers. Figure 1 depicts the architecture of the Swarm framework. From top to bottom, Swarm provides a standard set of Web service interfaces. Desktop users and gateway style applications can easily access Swarm via standard Web service interfaces. Each of the operations and parameters are defined in WSDL.

The requests from the user are delivered to the Request Manager. The Request Manager creates a ticket for the series of jobs, which is a 128 bit universally unique identifier. To provide the capability to track a large number of jobs, Swarm provides a simple structure to the submitted jobs. Jobs are identified with their ticket and internal ID. Here, internal ID is the identity of the job which is unique within the job group. This structure is especially useful for the web application, which deals with multiple experiments launched by multiple users.

As seen on the right hand side of the Request Manager in Figure 1, the Job submission process interacts with the Resource Ranking Manager, which prioritizes the resources over which the job is submitted to optimize the job execution process. This will be discussed in detail in the subsection 4.1.

Under the Request Manager and Resource Ranking Manager, there is a group of software components; referred to as Job Board. Swarm maintains a Job Board for each user. Each of the Job Board contains a Job Queue, Job Distributor, and Resource Pool. Users do not share any of these components. Matchmaking between the jobs and the resources are done in the user's Job Board. Subsection 5.2 will describe this in more detail.

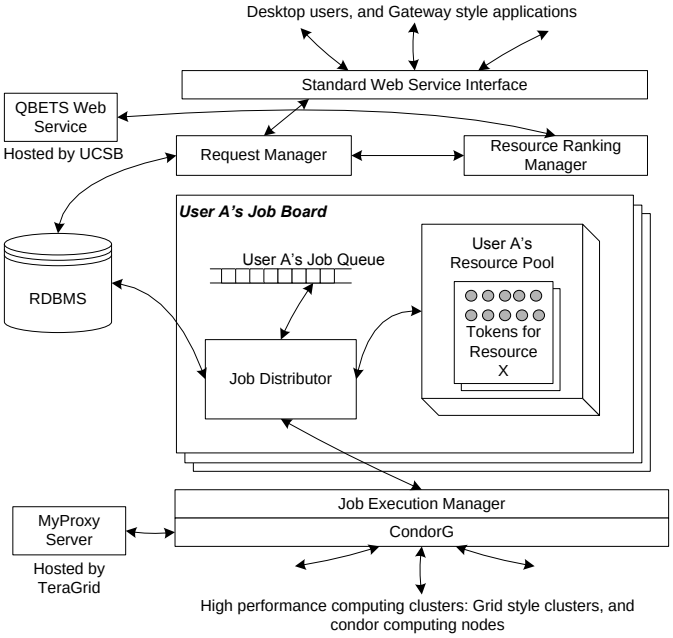

Figure 1. Swarm architecture. Client applications interact with the Swarm WSDL using standard Web service tools. In practice, we extend Swarm to make problem-specific services that inherit Swarm capabilities but provide a code-specific WSDL.

When the Job Distributor finds the available match of the remote resource, the Job Execution Manager will submit the job through CondorG's Web service APIs. The user's certificate (based on the X.509) is retrieved by means of interacting with MyProxy service and used to access to the Globus GRAM job manager. In addition, users are allowed to submit jobs to the ordinary Condor computing nodes through Swarm.

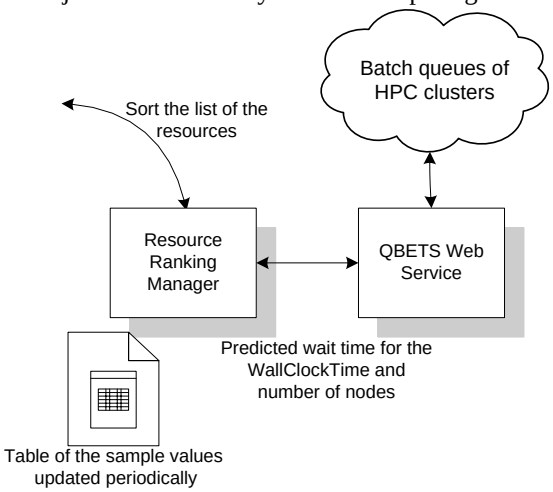

Figure 2. Interation between Swarm and QBETS webs services. Swarm uses QBETS information for resource matching.

5.1. Ranking the Resources for the Requested Job

In a lot of cases, a scientist has an account that is valid for multiple computing clusters. Popular scientific software is typically available from multiple clusters without the need to do any additional installations. Some scientists install their own software at multiple computing sites to achieve better performance. Swarm provides automated prioritizing process for each of the jobs.

Well-grouped and site-prioritized jobs are in the user's individual job queue awaiting available resources. Swarm provides matchmaking between the jobs in the queue and resources in the pool by using the following criteria:

- "First In First Out" internal queue for the jobs, and
- Available resources with the smallest wait-time first

If the user specifies priority or preference for the resources, Swarm utilizes that during the matchmaking process.

Users are allowed to specify multiple resources to submit a job. To prioritize the resources listed in the user's job description, Swarm interacts with the QBETS batch queue prediction service. Figure 2 depicts the interaction between Swarm and QBETS service. The QBETS service provides queue delay predictions. The WallClockTime and number of nodes are key factors to get the predicted delay. Here the WallClockTime is duration of the execution of the job and the number-of-nodes is number of the computing nodes for the parallel jobs. The WallClockTime and number of nodes are specified in the job description and Resource Ranking Manager passes that information to the QBETS Web service and gets the result of predicted wait time in the batch queue.

The wait time does not change gradually; it is also not very time sensitive. Therefore, we sample the range of key parameters and the predicted delay. The Resource Ranking Manager keeps a table, which is a set of combinations of {`batch queue, range of the number of node, range of the WallClockTime, predicted delay`} and refers to it when the resource list is required to be prioritized.

5.2. Matchmaking

The Job Distributor is the core component of the matchmaking between jobs and resources. As depicted in the Figure 1, each of the users has their own instance of the Job Board containing the Job Distributor. The Job Distributor scans the user's job queue in a FIFO fashion. For the each of the jobs, Job Distributor evaluates the resource pool to determine whether there is a token which is not taken. As soon as the Job Distributor finds an available resource, the Job Execution Manager submits the job to the relevant resource.

If the number of jobs exceeds 1000s, keeping all of the information about jobs in the memory and scanning the queue every time is not very efficient. Therefore, individual users keep a few hundred jobs in their queues and keep the rest in the backend database. Job updates are also synchronized between Swarm and the database. Keeping backend database provides fault tolerance by coping with hardware failures or a system shutdown.

The Job Board maintains a resource pool which is a set of tokens for the individual resources. The number of tokens is the maximum number of the jobs that can stay in the specific batch queue system concurrently.

Finally, if the Job Distributor finds the resource available for the job, the job is submitted to the remote resource. Each of the matchmaking process is based on the user's account that is often valid over multiple supercomputing clusters. Therefore, if an individual scientist has multiple accounts, the job submission will process jobs independently for each of these accounts. Similarly, in the case of educational software, if there is a need to provide the computing resource with a community-shared account, the matchmaking process will serve those users as single account.

6. Performance Evaluation

We have implemented a prototype of the Swarm framework, in Java, based on Apache Axis2[20]. The server was hosted on a machine with 3.40GHz Intel Pentium 4 CPUs and 1GB RAM. The client software was hosted on a machine with a 2.33 GHz Intel Xeon CPU and 8GB RAM. The machines involved in the benchmark were hosted on 1 Gbps network.

Our first benchmark measured the total turnaround time for operations in a single user environment. Table 1 depicts turnaround times for two operations: job submissions and status check. We compared the average turnaround time as we increased the number of jobs. This job submission time includes operations related to creating a group job and submitting jobs. For a given web service request, we batched 100 jobs for measurement.

Table 1. Total turnaround time for the job submission and status check with various job sizes in the single user environment

Type of Operation	Total turnaround time with 100 jobs	Total turnaround time with 1000 jobs Total	Total turnaround time with 10,000 jobs	Total turnaround time with 100,000 jobs	Total turnaround time with 1,000,000 jobs
Job Submission	184 msec	1,590 msec	15,993 msec	157,183 msec	1,581,947 msec
Status Query	19 msec	53 msec	375 msec	796 msec	7,878 msec

We then setup N clients, in a different JVM, that send requests actively. The behavior of the N clients mirrors that of a scientist who submits large-scale jobs and subsequently checks the status of these jobs periodically (we set this to one-per-minute). The size of the jobs submitted by the N clients and the distribution of the various job sizes for the clients are specified in the Table 2.

The total turnaround times from the Swarm client to the Swarm service for the job submission request are shown in Figure 3. As can be observed, the total turnaround time for the job group grows directly proportional to the number of jobs. Similarly the turnaround time to create a job group increased in proportion to the number of users who access the server concurrently. For N users, the Swarm service maintains N Job Boards.

Figure 4 provides the average turnaround time for the different operations. The job group creation operation registers information about the job group and creates the ticket identifier for the associated jobs. The average turnaround time for the job submission increased as the number of concurrent users increased. Meanwhile, the average turnaround time for the job status query was not significantly affected by the number of concurrent users unless it was a single-user environment.

Table 2. Test scenario for the multi-users environment

Size of the Job Group	Job submission (creating group, submitting jobs)	Job management (status, get output, etc)	Distribution
Small group (<1,000 jobs)	30 % of total duration	70 % of total duration	40 % of total groups
Medium group (1,000~100,000 jobs)	20 % of total duration	80 % of total duration	50 % of total groups
Large group (> 100,000 jobs)	10 % of total duration	90 % of total duration	10 % of total groups

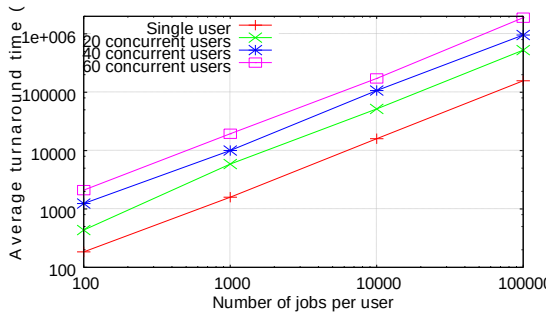

Figure 3. Average turnaround time for the various job size with various number of concurrent users

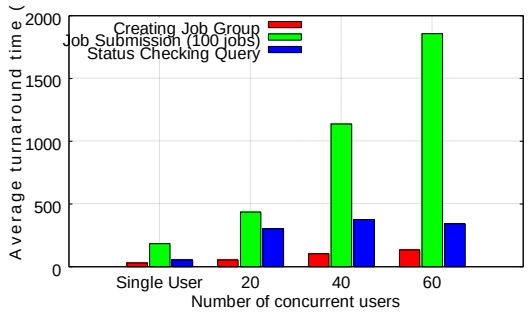

Figure 4. Average turnaround time per operations with various number of concurrent users

7. Conclusions and Future Work

In this paper, we have introduced a high-level job scheduling framework, Swarm. The Swarm service is designed to be extensible and lightweight so that users working on desktops or small servers can easily install and host it. Since it is Web service based, derivative services based on Swarm can be integrated in Web portals and science gateways. In this paper, we have discussed the motivation for this research, the architecture of the Swarm framework, and a performance evaluation of the system prototype.

As part of our future work, we plan to enhance our service with an intelligent error-handling scheme. On top of the potential system failure, current job submission mechanisms are often prone to have the task specific errors. The job scheduler should provide the scheme to cope with various task specific errors. We also plan to implement a batch job submission that can optimize the possible overhead of the batch queuing process in the remote cluster. We expect that proactive batch job mechanism can offer significant performance improvements for applications that submit a large number of small-scale jobs.

References

[1] Catlett, C. et al. 2007. TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications. HPC and Grids in Action, Ed. Lucio Grandinetti, IOS Press 'Advances in Parallel Computing' series, Amsterdam, 2007.
[2] Pordes, R., et al. 2007. The Open Science Grid. Journal of Physics: Conference Series 78:012057.
[3] Qunfeng Dong, Shannon D. Schlueter, Volker Brendel PlantGDB, plant genome database and analysis tools. Nucleic Acids Research 32(Database-Issue): 354-359
[4] Anantharaman Kalyanaraman, Srinivas Aluru, Suresh C. Kothari. Space and Time Efficient Parallel Algorithms and Software for EST Clustering. Proceedings of the ICPP 2002: 331-338
[5] Ian T. Foster.2006. Globus Toolkit Version 4. Software for Service-Oriented Systems. Journal of Computter Science Technology. 21(4): 513-520

[6] Eduardo Huedo, Rubén S. Montero, Ignacio Martín Llorente. 2004. A framework for adaptive execution in grids. Soft., Pract. Exper. 34(7): 631-651
[7] Tadashi Maeno. 2008. PanDA: distributed production and distributed analysis system for ATLAS, Journal of Physics: Conference Series, **119** 062036.
[8] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, Mike Wilde. 2007. Falkon: a Fast and Light-weight tasK executiON framework. Proceedings of the IEEE/ACM SuperComputing
[9] Edward Walker, J. P. Gardner, V. Litvin, and E. P. Turner. 2007. Personal Adaptive Clusters as Containers for Scientific Jobs. Cluster Computing, vol. 10(3), September
[10] Li, Liwei; O'Callaghan, B. J.; Dantzer, J. J. and Meroueh S. O. 2008. PDBcal: A Comprehensive Dataset for Receptor-Ligand Interactions with Three-Dimensional structures and Binding Thermodynamics from Isothermal Titration Calorimetry. Chem. Biol. & Drug Design. 71:529-32
[11] Li, Liwei; Uversky N. V.; Dunker, K. A.; Meroueh, S. O. 2007. A Computational Investigation of Allostery in the Catabolite Activator Protein. J. Am. Chem. Soc. 129(50), 15668-76
[12] D. Angulo, I. Foster, C. Liu, and L. Yang, 2002. Design and Evaluation of a Resource Selection Framework for Grid Applications. Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July.
[13] Globus Toolkit, Overview and Status of Current GT Performance Studies. [Internet].[cited 2008 August 9]. Available from http://www.globus.org/toolkit/docs/4.0/perf_overview.html
[14] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. 2002. Condor - A Distributed Job Scheduler. Ed. Thomas Sterling, Beowulf Cluster Computing with Linux, The MIT Press, ISBN: 0-262-69274-0
[15] Alexandru Iosup, Dick H.J. Epema, Todd Tannenbaum, Matthew Farrellee, Miron Livny. 2007. Inter-Operating Grids through Delegated MatchMaking", in proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC07), Reno, Nevada, November.
[16] Alain Roy and Miron Livny. 2003. Condor and Preemptive Resume Scheduling. *Grid Resource Management: State of the Art and Future Trends*, Fall 2003, pages 135-144, Fall 2003, Ed. Jarek Nabrzyski, Jennifer M. Schopf and Jan Weglarz, published by Kluwer Academic Publishers.
[17] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2005. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005
[18] Clovis Chapman, Charaka Goonatilake, Wolfgang Emmerich, Matthew Farrellee, Todd Tannenbaum, Miron Livny, Mark Calleja, and Martin Dove. 2005. Condor BirdBath: Web Service interfaces to Condor. *Proceedings of the 2005 UK e-Science All Hands Meeting*, ISBN 1-904425-53-4, pages 737-744, Nottingham, UK, September 2005.
[19] Daniel Nurmi, John Brevik, Richard Wolski. 2007. QBETS: queue bounds estimation from time series. SIGMETRICS 2007: 379-380
[20] Srinath Perera, Chathura Herath, Jaliya Ekanayake, Eran Chinthaka, Ajith Ranabahu, Deepal Jayasinghe, Sanjiva Weerawarana, Glen Daniels. 2006. Axis2, Middleware for Next Generation Web Services. Proceedings of the ICWS 2006: 833-840