A Client-centric Grid Knowledgebase

George Kola, Tevfik Kosar and Miron Livny
Computer Sciences Department, University of Wisconsin-Madison
{kola,kosart,miron}@cs.wisc.edu

Abstract

Grid computing brings with it additional complexities and unexpected failures. Just keeping track of our jobs traversing different grid resources before completion can at times become tricky. In this paper, we introduce a client-centric grid knowledgebase that keeps track of the job performance and failure characteristics on different grid resources as observed by the client. We present the design and implementation of our prototype grid knowledgebase and evaluate its effectiveness on two real life grid data processing pipelines: NCSA image processing pipeline and WCER video processing pipeline. It enabled us to easily extract useful job and resource information and interpret them to make better scheduling decisions. Using it, we were able to understand failures better and were able to devise innovative methods to automatically avoid and recover from failures and dynamically adapt to grid environment improving fault-tolerance and performance.

1. Introduction

Grid computing [6] while enabling researchers to harness idle grid resources, creates difficulties because of the lack of guarantees. Scaling an application from the controlled well-understood cluster environment to a grid environment creates a plethora of problems.

Livny and Thain [12] [19] have pointed out that the only practical way of handling these problems is to make the client (submitting endpoint) responsible for the progress including failure handling. While client submit software like Condor [11] and Condor-G [9] address some of these problems, the presence of 'black holes', machines that accept jobs but never complete them, and machines with faulty hardware, buggy or misconfigured software impede the efficacy of using grid based resources.

We study the prevalence of black holes by analyzing the log files of two real-life grid applications. After detecting the presence of black holes, we investigate the reasons for their occurrence. As a viable solution, we introduce the concept of grid knowledgebase that keeps track of the job performance and failure characteristics on different grid resources as observed by the client.

Client middleware can use this knowledgebase transparently to improve performance and throughput of unmodified grid applications. Grid knowledgebase enables easy extraction of useful information simplifying a variety of tasks including bug finding, statistics collection and visualization.

In the spirit of grid computing, mutually trusting entities can share the knowledgebase. Grid sites can use it to detect misconfiguration, software bugs and hardware faults. We discuss the design and implementation of our prototype grid knowledgebase and evaluate its effectiveness on a real-life grid workload.

2. Why do we need a Grid Knowledgebase?

We performed an objective study to identify the presence of black holes by analyzing the log files of two real-life grid applications: NCSA DPOSS image processing pipeline [5] and WCER video processing pipeline [10]. Both the pipelines strive towards a fully automated fault-tolerant processing of terabytes of images and videos respectively.

In the WCER pipeline log files, we found the presence of three black holes that accepted a job each and did not seem to have done anything and scheduler was trying unsuccessfully to talk to the machine for over 62 hours. We also found a case where a machine caused an error because of a corrupted FPU. Through a careful analysis, we found that certain machines had problems with certain job classes while they executed others successfully. As a particular case, we found that the machine that had FPU corruption with MPEG-4 encoding had earlier successfully performed MPEG-1 encoding.

Detecting the above kinds of problems is difficult and the only party affected is the job that was unable to complete successfully. Further, in a grid environment, job submitter may not have control over the machine configuration. Following the 'dependability from client side' argument [12] [19], the job should be adapted to avoid those resources.

In an organization with thousands of compute nodes, it is a nightmare to ensure that all the different software are properly configured and working on all the nodes. While individual hardware and software are relatively easy to check, ensuring that different software work fine together is a non-trivial task. Most of the organizations at this point depend on user complaints to help them identify problems with such complex interactions. Many problems are not fixed because the users did not want to take the trouble of identifying the problem and reporting them.

In the WCER pipeline case, a couple of routine software and operating system upgrades fixed the problem. However, those upgrades took several months. The users did not try to find the cause of problem or report it because a few retries was probabilistically sufficient to get the jobs scheduled on a properly configured machine. A system capable of automatically identifying problem would greatly benefit site administrators. If site administrators use this information to fix the problems, it would result in better quality of service for jobs using that site.

To survive, organisms need to learn from experience and adapt themselves to changing environment. In a large-scale grid environment, it is imperative that jobs should adapt to ensure successful completion. Just as organisms pass the gained wisdom down the generations, grid jobs should pass down the gained wisdom to future grid jobs.

We need a mechanism to enable passing this information from current grid jobs to future ones. To enable this, we propose the concept of grid knowledgebase that aggregates the experience of the different jobs. It collects this information from the job log files produced by the batch scheduling systems like Condor/Condor-G. These log files are normally available to the client and are different from cluster/pool log files that many site administrators are unwilling to share. This log files essentially contain the view of the world as seen by the client.

We extract useful information from the log files and enter it into a database. We add an adaptation layer that uses this collected wisdom to adapt the failed job instances and future job instances of a grid application. This is similar to organisms learning from experience and works well because many grid applications consist of multiple instances of the same executable operating on different data.

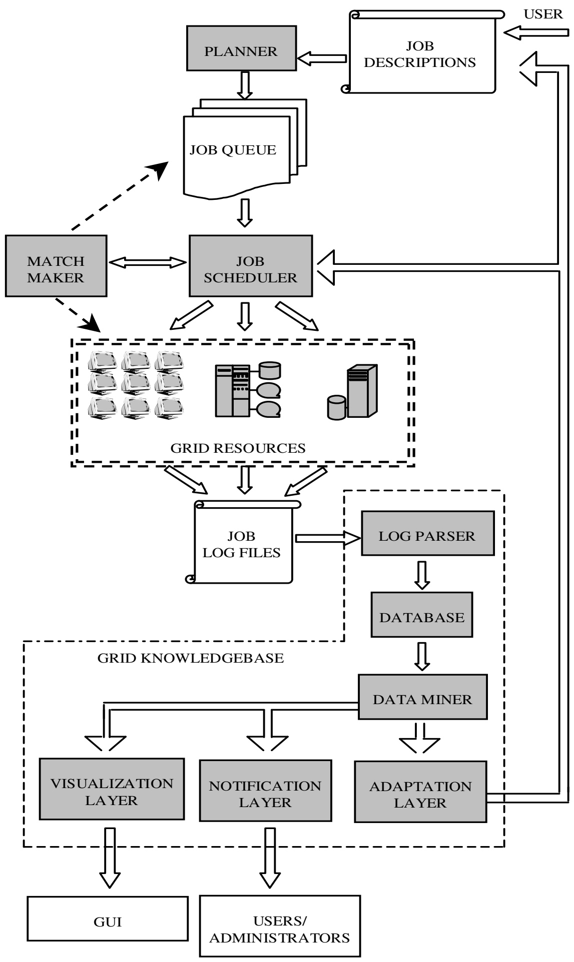

Figure 1. The main components of Grid Knowledge Base and their interaction with other parts of the system.

3. Grid Knowledgebase Framework

Grid Knowledgebase, as shown in figure 1, consists of six main components: log parser, database, data miner, adaptation layer, notification layer, and visualization layer.

The log parser extracts useful information from the log files of submitted jobs and enters it into a database. This information includes the list of events that occurred during a job's life, the timestamp of each event, list of compute nodes that executed this job, resource utilization statistics and error messages.

The data miner runs a set of pre-selected queries on the database periodically. It checks the event history of the job and compares it with its own templates; checks the state of the resources on which the job executed; compares expected job execution time with the observed execution time; and tries to infer the reasons for delays, failures and other problems. In cases where the pre-selected set of queries is not sufficient, the user can either modify the existing queries or add new ones. The users can tune how often to run each of the queries and can even make them event-based so that certain events trigger execution of certain queries.

The data miner queries help determine the problem and choose an appropriate course of action. At present, the data miner runs three sets of queries.

The first set of queries takes a job-centric view and tries to find out the jobs that failed and tries to find the reason for them and feeds this to the adaptation layer.

The second set of queries takes a resource-centric view. They determine the resources that failed to successfully execute jobs and feed this information to the notification layer.

The third set of queries takes a user-centric view and tries to get the information that users may be interested in observing and feeds this to the visualization layer. An example would be tracking the memory usage of an application.

The adaptation layer analyzes the information fed by the data miner and if possible, tries to adapt the job dynamically to prevent recurrence of encountered problems. In certain cases, simple changes in job requirements such as increasing the memory requirement and/or hard disk space requirements or avoiding failure-prone machines may solve the problem. The adaptation layer can also pass this information and the strategy it took to higher-level planners like Chimera, Pegasus or DAGMan.

The notification layer informs the user who submitted the jobs and execute-site administrators about possible problems such as misconfigured and faulty machines. Since it is more important for machines bought under a project funding to be able to run that projects applications successfully than it is for those machines to run some other job when idle, the notification layer allows users/administrators to attach weights to machine-application pair. The email sent to the administrator specifies the failures and sorts them by weight. Users/administrators can tune the frequency of email notifications.

The visualization layer generates the information necessary to visualize the inferred information. Figure 1 shows how all of the components of grid knowledgebase interact with each other and other entities in the overall scheduling system.

The different job classes using the framework can choose to share the data. Sharing this information between different organizations that submit the same classes of applications is very useful. In this case, the data miner can query remote data to get additional information. It is also possible to use a distributed database between different mutually trusted entities.

Organization may share this information even if they submit different application classes as the data miner can use the data to determine correlations between failures. For instance, a machine may fail when the input file is greater than 2 GB and we can use correlation to extract this information.

4. Implementation Insights

We have implemented a prototype of our grid knowledgebase framework and interfaced it with the Condor batch scheduling system. Condor/Condor-G jobs can choose to have the logs generated in either Condor specific user-log format or more generic XML format. Originally, our parser supported only the XML format hoping that we can make the jobs choose XML format logs. Further, we felt that by supporting XML, we could easily extend the parser to parse XML format logs produces by other batch scheduling systems. While interacting with users, we found that many of them had logs in the condor specific user log format and did not want to switch to XML. Because of this, we added support for condor user-log format and now our parser can parse both formats.

Condor-G uses Globus toolkit [7] functionality to schedule jobs on almost all grid-enabled resources. As Condor-G is being used by most of the Grid2003 [16] users to submit grid jobs, the ability to parse Condor-G job logs gives us the ability to parse most of the real-life grid job logs. Thus, most of the grid community can easily use the grid knowledgebase and benefit from it.

After designing the parser, we had to choose a database. At first glance, we thought a native XML database would be a good choice for storing event data . As we could not find a free native XML database with suitable performance and because we found it difficult to construct XML (XQuery [1]/XPath [20]) queries to extract the useful information, we decided to load job logs into relational database tables. The current system uses a *postgres* [18] database.

We faced several issues while designing the schema to represent job event history. The first was whether we should use a vertical schema or a horizontal schema. In vertical schema, the events are stored as job id, event pairs. The horizontal schema allocates a field in the relational table for each of the events that may occur. Vertical schema is faster for loading, but requires joins for querying. Horizontal schema requires some processing before loading but is faster for querying, as it does not require any joins. Horizontal schema may waste space if the tuple is very sparse i.e. if most of the events rarely occur. After careful analysis, we found that the space wastage depended on the database implementation and that most relational databases are optimized for horizontal schemas. Further, vertical schemas required complex queries. With that, we decided to go in for a horizontal schema.

We now had to convert each of the events into a field of a relational table. We encountered the following problem. Certain jobs were executed multiple times because of encountered failures resulting in multiple event sequences for the same job. Further, even in a single event sequence, certain events like exception occurred multiple times.

Our initial approach was to create multiple table entries for such repeated events. We soon realized that querying them and extracting useful information was not straightforward. After some research, we found that *postgres* was

Field	Type
JobId	int
JobName	string
State	int
SubmitHost	string
SubmitTime	int
ExecuteHost	string []
ExecuteTime	string []
ImageSize	int[]
ImageSizeTime	integer []
EvictTime	int []
Checkpointed	bool []
EvictReason	string
TerminateTime	integer []
TotalLocalUsage	string
TotalRemoteUsage	string
TerminateMessage	string
ExceptionTime	int []
ExceptionMessage	string []

Table 1. Relational database schema used to store job event history.

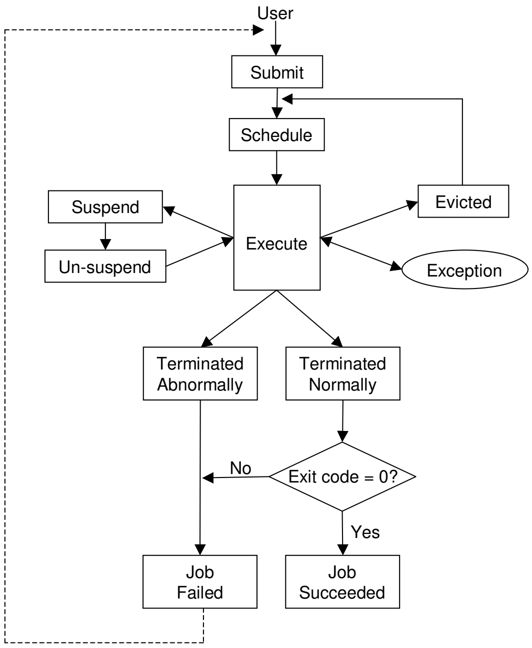

Figure 2. The state changes a job goes through.

an object-relational database and it supported 'array type', which essentially allows multiple entries in a single field. This addressed most of the issues we had. In the present form, each job maps into a single record in the database table.

Table 1 shows a simplified schema. To make it intuitive, we have simplified SQL varchar(n) to string and left out the bytes in the integer (we use int instead of int2, int4 and int8). The [] after a type makes it an array type. An array type can have a sequence of values.

A job goes through different states in the course of its life and the state field tracks that. Figure 2 shows the state change that a job typically goes through. The job enters the system when a user submits it. When the scheduler finds a suitable host, it assigns the job to that host and starts executing it there. During execution, the scheduler may periodically observe the job state like image size and log it. A number of exceptions may happen during job execution. For instance, the scheduler may be unable to talk to the execute machine because of a network outage.

An executing job may be evicted when the machine owner or a higher priority user wants to use the machine or when the job exceeds its resource usage limit. The evicted job is rescheduled. When a job is evicted, the job that has the ability to checkpoint may save its state and can resume from that state when it restarts. Finally, the job may terminate. If the job terminates abnormally, or it terminates normally with non-zero return value, the job is considered to have failed. If the job terminates normally with zero return value, it is considered successful.

The 'ExecuteHost' and 'ExecuteTime' are pairs in that first element of 'ExecuteTime' gives the time when the job started execution on the first 'ExecuteHost'. Other pairs are obvious from the first part of their field names.

5. Evaluation

Grid knowledgebase enabled us to extract useful information about jobs and resources and interpret them to gain a better understanding of failures. It helped us devise methods to avoid and recover from failures and helped us make better scheduling decisions. It helped us dynamically adapt our jobs to the ever-changing grid environment. We observed that using the grid knowledgebase in NCSA DPOSS image processing and WCER video processing pipelines increased their reliability and efficiency. Below, we list some of the contributions of grid knowledgebase to these real life data processing pipelines.

Collecting Job execution time statistics. We wanted to determine the average job execution time, its standard deviation, median, and fit a distribution to the execution time. Just the average and standard deviation is useful to benchmark two different clusters for this application. The grid knowledgebase allowed us to easily extract this information.

A simplified query to extract the average and standard deviation of MPEG1 encoder is shown below.

```
SELECT AVG(TerminateTime[index]-ExecuteTime[index]),
    STDDEV(TerminateTime[index]-ExecuteTime[index]),
FROM  WCER_VideoPipeline
WHERE TerminatedNormally[index] IS true
      AND JobName ILIKE('\%MPEG1-ENCODE\%') ;
```

Detecting and Avoiding Black Holes. During the processing of WCER video data, we detected the presence of some black holes. Jobs assigned to certain resources started execution but never completed. We called such machines black holes and decided to understand them and avoid them if possible.

To detect a black hole, we used the previously extracted run-time statistics. Our job execution times were normally distributed. So, we knew that 99.7% of the job execution times should lie between `average - 3*standard-deviation` and `average + 3*standard-deviation`.

Using the average and standard deviation calculated from grid knowledgebase, we set the threshold to kill a job to `average + 3*standard-deviation`. If a job does not complete within that time, we marked that execution node as a black hole for our job and rescheduled the job to execute elsewhere.

The standard deviation takes the performance difference between grid resources into account. If we want to detect sooner at the expense of false positives, we can decrease the threshold to `average + 2*standard-deviation`. Even with this threshold, we would only be rejecting around 4% of the machines, these would be the top 4% of slow machines, and this selective reject may considerably improve the throughput. Users can tweak this mechanism to improve throughput by avoiding a certain top fraction of the slow machines.

It is also possible to parameterize this threshold taking into account factors like input file size, machine characteristics and other factors. For this, we need to use regression techniques to estimate the mean and standard deviation of job-run-time for a certain input file size and machine characteristics. This parameterization would enable generation of tighter estimates and quicker detection of black holes.

6. Other Contributions

Grid knowledgebase has other useful contributions that we did not use during the execution of our two real life data processing pipelines but would be useful to the grid community. We list some of them below.

Identifying Misconfigured Machines. We can easily identify machines where our jobs failed to run. In our and other environments, we find that it is important for machines bought under a particular project grant to be able to successfully execute job classes belonging to that project. We also care about failures of other job classes that use our idle CPUs but they are not that important. To address this, we attach a weight to each of the different job classes. Extracting the failures observed by the different job classes and multiplying by the associated weight and sorting, we can get the list of machines ordered by priority that site administrator needs to look into.

We can also extract additional information by using correlation to help site administrator find the fault. For example, we can find that a certain set of machine fails for jobs that have an input file size that is larger than 2 GB. Site administrators can use this information to find the exact problem and fix it.

Identifying Factors affecting Job Run-Time. Some users may want to add more computing power to enable additional processing. They need to determine the suitable machine configuration. Using our system, they can extract the run-time on the different resource and try to extract the significant factors affecting the job performance. This would help them choose appropriate machine configuration.

Bug Hunting. Writing bug-free programs is difficult. Since the scheduler logs the image-size of the program and as the log-parser enters this into the database, the data miner can query this information and pass it to the visualization layer to graph the growth in memory size. A continuously growth may indicate the presence of a memory leak. Grid application developers can use this information to identify bugs. It is very difficult to find bugs that occur only on certain inputs. A simple query can find out the inputs that caused a high growth in memory size. Similarly, if a job fails on certain inputs, the grid knowledgebase can automatically derive this and email this information to the application developer.

Figure 3 shows grid knowledgebase finding a memory-leaking process. Condor preempted and rescheduled the job three time before the grid knowledgebase categorized it as a memory leaking process and notified the job submitter The submitter fixed the memory leaking code and resubmitted the job. After resubmission, we see that the image size of the job stabilized at a certain point and does not increase any more.

Application Optimization. Many grid applications contain a number of different processing steps executed in parallel. Grid application developers want to parallelize the steps depending on the time each takes to execute. While they try to do this by executing on a single machine, they would really like feedback on what happens in a real grid environment. At present, they find it difficult to extract this information. With the grid knowledgebase, application developer can write a simple two-line SQL query to extract this information and use it to redesign the application.

Adaptation. Smart grid application deployers can come up with easy triggers to adapt the jobs to improve through-

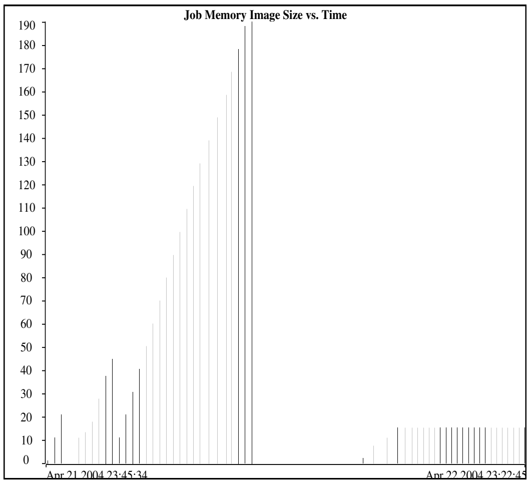

Figure 3. Grid Knowledgebase finds one of the jobs leaking memory and notifies the user. The job is resubmitting after fixing the memory leak. Memory image size of this job versus time is shown.

put. A simple example is to find the machines that take the least time to execute this job class and try to preferentially choose them over other machines if those machines are available.

Extensibility. The log parser parses XML logs. This is very extensible and users can even extend this mechanism to parse the logs produced by application itself. This may yield useful information.

7. Related Work

Parsing of log files to identify possible errors have been done both manually and in an automated manner for a long time. Our main contribution are: highlighting the fact that the logs have sufficient information to perform sophisticated operations like adaptation, extracting the log data, entering it into a well understood SQL database, and enabling people to perform sophisticated operations on this now easily queriable data.

Resource selection and work allocation strategies have been studied extensively in the domain of master-worker paradigm. Shao et al [17] discuss adaptive scheduling of master-worker applications on distributed computing resources. They build a model taking into account resource, work characteristics, and use it to choose an appropriate master and determine the amount of work to be distributed to each worker. Our work is different in that we adapt at job level while they adapt at work chunk level. Their adaptation is more fine grain, but it does not use previous history. Our work proposes a way to collect the history from a client perspective and uses that to adapt the failed job instances and future job instances of a job class. Our work is not specific to master-worker and is more general in nature. In addition, we use past experience gained from previous runs whereas their adaptation is based on a feedback loop at the time of execution and does not use previous history.

Application developers have been performing trace driven program optimization for a long time. Moe and Carr [13] explain how execution traces can be used to improve distributed systems. Their system captures traces at CORBA [15] RPC level and enables offline analysis that can be used to improve performance. Grid knowledgebase can be used to perform both online and offline analysis and optimization.

Chimera [8] is a virtual data system for representing, querying and automating data derivation. Pegasus [4] is a configurable system that uses artificial intelligence planning techniques to map complex workflows and execute them on the grid. Pegasus in combination with Chimera accepts abstract workflow and produces a concrete workflow that DAGMan [3] executes.

While Pegasus evaluates data generation versus reconstruction costs and makes appropriate decisions, it does not learn from experience of the job with the grid environment. Pegasus and grid knowledgebases adaptation layer are similar in that they help optimize the job execution using information not previously known to the job.

Our grid knowledgebase can work together with Pegasus and give it the ability to learn from experience. Pegasus can make better planning decisions and produce more optimized workflows taking into account the job characteristics on the different resources.

Our grid knowledgebase adaptation layer is at a lower level than Pegasus and it constantly monitors job progress and updates the job requirements accordingly. We believe that a feedback loop from Grid knowledgebase to Pegasus combined with Grid knowledgebase's job adaptation would result in superior throughput.

8. Future Work

We need to add a mechanism to remove a machine from the list of avoided machines when its problems have been resolved. While a simple white list may not suffice because we a fixed machine may have problems again, we need a white list with each machine entry having an associated time-stamp. For each machine in the white list, we do not use the query results that occurred before this time-stamp. To trim this white list, we retain only the latest entry for a machine. We need to experiment with this and see how well it works in practice.

The data in the grid knowledgebase at present keeps growing. This problem is not as bad as a typical job takes only a few hundred bytes. For the short term, creating new tables every month or whenever the application changes may be sufficient because we are likely to hit table size limits and not database size limits.

Database size growth may be a problem over a long period of time when hundreds of millions of jobs are executed. Just a log rotating works, we need to clear the old entries periodically. Instead of throwing away the old data, it may be better if we can apply some compression operation to retain only the useful information. For instance, if we buy two clusters from different vendors, we may want to check the failures that happened in each over a one-year period to determine if one is better than the other. Some logging systems smartly compress the old data and we need to look into this to see how we can apply similar techniques to the database data.

Another issue we need to look into is that we may want to attach more importance to recent data than old data. We need to come up with a way of attaching weight to results sorted by age to accomplish this.

We are working towards applying grid knowledgebase to other real life grid applications such as BLAST [14] and CMS [2].

9. Conclusions

We have introduced the concept of grid knowledgebase that keeps track of the job performance and failure characteristics on different resources as observed by the client side. We presented the design and implementation of our prototype grid knowledgebase and evaluated its effectiveness on two real life grid data processing pipelines. Grid knowledgebase helped us classify and characterize jobs by collecting job execution time statistics. It also enabled us to easily detect and avoid black holes.

Grid knowledgebase has a much wider application area and we believe it will be very useful to the whole grid community. It helps users identify misconfigured or faulty machines and aids in tracking buggy applications.

10. Acknowledgments

We would like to thank Robert Brunner and his group at NCSA, and Chris Thorn and his group at WCER for collaborating with us, letting us use their resources and making it possible to try the grid knowledgebase on real-life grid applications. We would also like to thank Mark Silberstein for useful feedback and suggestions.

References

[1] D. Chamberlin. XQuery: An xml query language. *IBM Systems Journal*, 41(4):597–615, 2002.

[2] CMS. The Compact Muon Solenoid Project. http://cmsinfo.cern.ch/.

[3] Condor. The Directed Acyclic Graph Manager. http://www.cs.wisc.edu/condor/dagman, 2003.

[4] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Pegasus: Planning for execution in grids. Technical Report 20, GriPhyN, 2002.

[5] S. G. Djorgovski, R. R. Gal, S. C. Odewahn, R. R. de Carvalho, R. Brunner, G. Longo, and R. Scaramella. The Digital Palomar Sky Survey (DPOSS). *Wide Field Surveys in Cosmology*, 1988.

[6] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, 2001.

[7] I. Foster and C. Kesselmann. Globus: A toolkit-based grid architecture. In *The Grid: Blueprints for a New Computing Infrastructure*, pages 259–278. Morgan Kaufmann, 1999.

[8] I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *14th International Conference on Scientific and Statistical Database Management (SSDBM 2002)*, Edinburgh, Scotland, July 2002.

[9] J. Frey, T. Tannenbaum, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grid. In *Tenth IEEE Symposium on High Performance Distributed Computing*, San Francisco, CA, August 2001.

[10] G. Kola, T. Kosar, and M. Livny. A fully automated fault-tolerant system for distributed video processing and off-site replication. In *Proceeding of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (Nossdav 2004)*, Kinsale, Ireland, June 2004.

[11] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[12] M. Livny and D. Thain. Caveat emptor: Making grid services dependable from the client side. In *2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02)*, Tsukuba, Japan, December 2002.

[13] J. Moe and D. A. Carr. Using execution trace data to improve distributed systems. In *Software Practice and Experience*, July 2002.

[14] NCBI. Blast project. http://www.ncbi.nlm.nih.gov/BLAST/.

[15] Object Management Group. The common object request broker: Architecture and specification revision 2.2, 1998.

[16] T. G. Project. The Grid2003 production grid:principles and practice. http://www.ivdgl.org/grid2003/, February 2004.

[17] G. Shao. *Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources*. PhD thesis, University of California at San Diego, May 2001.

[18] M. Stonebraker and L. A. Rowe. The design of Postgres. In *SIGMOD Conference*, pages 340–355, 1986.

[19] D. Thain and M. Livny. Building reliable clients and servers. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[20] World Wide Web Consortium. XML path language (XPath) version 1.0. w3c recommendation. http://www.w3.org/TR/xpath.html, November 1999.