

The DBC: Processing Scientific Data Over the Internet

Chungmin Chen
Dept. of Computer Science
University of Maryland
College Park, MD 20742 USA

Kenneth Salem
Dept. of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1 Canada

Miron Livny
Computer Sciences Dept.
University of Wisconsin-Madison
Madison, WI 53706 USA

Abstract

We present the Distributed Batch Controller (DBC), a system built to support batch processing of large scientific datasets. The DBC implements a federation of autonomous workstation pools, which may be widely distributed. Individual batch jobs are executed using idle workstations in these pools. Input data are staged to the pool before processing begins. We describe the architecture and implementation of the DBC, and present the results of experiments in which it is used to perform image compression.

1. Introduction

In this paper we present the DBC (Distributed Batch Controller), a system that processes data using widely-distributed computational resources. The DBC was built as a tool for *enriching* scientific data stored in two mass storage systems at NASA's Goddard Space Flight Center (GSFC). Enriching data means processing it to make it more useful. For example, satellite images may be classified according to some domain-specific criteria. These classifications can then be stored as meta-data to support content-based retrieval of the original images. Another possibility is to produce compressed, approximate versions of the images. These images could be retrieved very quickly because of their small size, and would be suitable for applications such as preliminary visual inspection of the data. In general, data enrichment tasks such as these are computationally intensive. Enrichment of large volumes of data may require enormous computational resources.

The DBC can be viewed as a global scientific data processing system that can utilize any resource that is willing to participate in a data enrichment effort. It draws its comput-

ing cycles from a collection of workstation clusters which may be distributed across the Internet. Each of these clusters uses the Condor resource management system to harness available cycles from workstations within the cluster. The DBC moves data from a mass storage system to local disk caches at each Condor pool, controls the local execution of data enrichment tasks, and moves the resulting data products to another storage system. Because a data enrichment effort may be very time consuming, the DBC is designed to operate continuously for extended periods of time with minimal manual interference.

We believe that large-scale data processing efforts like the one described above will become more common in the near future. As any World Wide Web surfer knows, very large repositories of valuable scientific data are scattered throughout the Internet and are gradually coming online. These repositories were established and are maintained by national agencies and universities in order to disseminate experimental data throughout the scientific community. Many repositories store their data in the form of files and provide data access through file-oriented retrieval protocols like FTP.

The users of these repositories will require mechanisms for pooling their computational resources to process the wealth of available data. Although the DBC was designed with a specific data-processing goal in mind, it can serve the needs of some of these users. The DBC has no built-in knowledge of the archived data, the enrichment tasks, or the resulting data products. Special attention has been devoted in the design of DBC to modularity and to well-defined interfaces and protocols.

The remainder of this paper is organized as follows. In the next two sections we discuss the architecture of the DBC and the implementation of our prototype system. Although we focus our attention on the DBC itself, we also

present a brief overview of Condor, which manages the individual pools of machines which make up the DBC. We then present the results of several experiments in which a DBC system consisting of two workstation pools was used to process satellite images from a NASA repository. We also present an analysis of the system and a discussion of how its performance might be improved. In Section 6 we discuss some other systems that are related to the DBC.

2. Global Data Processing System Architecture

Figure 1 illustrates an architecture for a global batch data processing system. The purpose of the system is to execute a batch of data processing jobs, e.g., to compress a set of images. The system is organized as a federation of autonomous resource pools. The pools provide the computational resources needed to execute data processing jobs. In our implementation of the architecture, discussed in Section 3, each pool is managed using the Condor resource manager. However, in general, a pool may be almost any collection of resources that is capable of processing jobs.

The data processing system itself consists of a set of agents, one per resource pool, and a coordinator. The agents interact with their pools to request resources for batch job execution. Each agent has access to a disk buffer for staging data. The centralized coordinator assigns work to the agents, and tracks the status of the entire batch computation. The coordinator also serves as a centralized site for user administration of a batch computation. Through the coordinator, the user can query the status of the computation and can modify it, e.g., by adding new jobs to the batch.

Data is stored in archives. Input files for the batch jobs reside initially in a source archive. Output files are collected at the result archive. Agents interact with these archives to arrange for data to be staged from the input archive to the agent's local buffer, and from the buffer to the result archive.

Since the pools are autonomous, the agents have no direct control of the computational resources at any pool. The system must presume that the number of resources allocated to its batch jobs by the pool will vary with time. In fact, it is possible for a pool to terminate its interaction with the system at any time, or for new pools to join the system at any time. Within this environment, the system must harness as many resources as possible for use in completing the batch.

The federated architecture has several important advantages. First, there is no need to modify a pool's resource manager in order to include that pool in a global system. In fact, to each pool's resource manager, an agent looks like any other client. There is also no need for a pool to surrender control of any of its resources in order to participate in the system. This makes the architecture particularly attractive for coordinating pools that reside in different administrative domains.

2.1. System Responsibilities

Within the system architecture, the pools provide the computational resources needed to perform data processing. However, reliable and efficient processing of large batch processing jobs requires that a number of other issues be addressed by the global data processing system itself. They include the following.

Data Staging: The resource pools used to process data may be spread over a wide area. The data to be processed is located in a repository which may not be directly accessible to programs running in the pools. Similarly, output data must be collected in a repository and not left scattered at the pools. The data processing system addresses this problem by staging data between the repositories and buffers located at each of the pools.

High-Level Scheduling: Each pool allocates and schedules its own resources. However, the data processing system must determine which pool will be used to process each of the jobs in a batch, a procedure we call high-level scheduling. The goal of the high-level scheduler is to maximize the throughput of batch jobs. To accomplish this, it must keep the resources available from each of the pools as busy as possible.

Reliability: A batch may consist of many jobs, and may require hours, days, or even weeks to complete. Failures of various kinds are likely to occur within this time frame. The data processing system must detect and recover from these failures so that each job in the batch will eventually be executed to completion by some pool.

3. The Distributed Batch Controller

The DBC is an implementation of the global data processing architecture described in Section 2. The DBC supports a simple batch model that is well-suited to tasks like data enrichment. The resource pools with which the DBC interacts are collections of workstations managed by the Condor resource management system, which is described at the end of this section.

A DBC batch consists of one or more jobs. In our current implementation, a job is the execution of a single program. Each job may require one or more input files and may produce one or more output files. All jobs in a batch are executions of the same program, although each job may use different input files and produce different outputs. All of jobs in a batch are assumed to be independent of one another. For example, a job's input file may not be produced by another job in the same batch. Although this model is

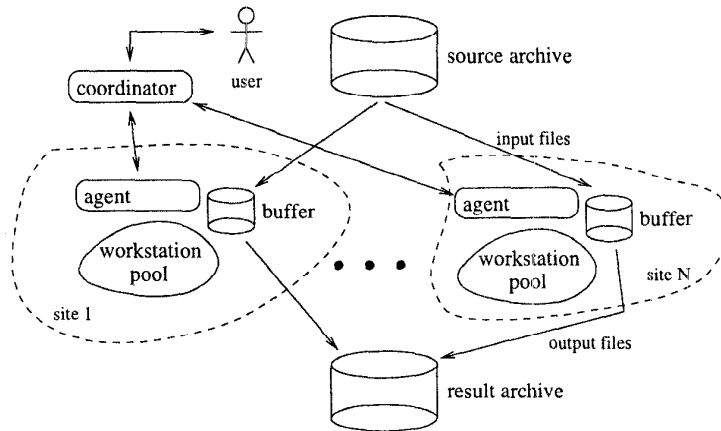


Figure 1. Architecture of the DBC

somewhat restrictive, it is well-suited to many data enrichment problems. For example, consider a large set of images that is to be classified before being stored in a database. Commonly, classification is performed by executing a classification program independently on each of the images.

Figure 2 shows the process architecture of the DBC prototype. The DBC consists of three types of processes: a master, workers, and job monitors. The master process implements the coordinator shown in Figure 1, and the worker and job monitor processes together implement the agents.

3.1. The Master and Worker Processes

The master performs high-level scheduling and keeps track of the execution status of each job in the batch. In part because of the batch model, high-level scheduling is very simple in the DBC. Each worker determines its willingness to execute additional batch jobs based on the availability of resources at its local pool. When resources are available, the worker requests work from the master. Since all jobs are independent, the master is free to assign any jobs that can be accommodated at the worker's site. When a worker completes a job, it notifies the master.

Each job request from a worker is accompanied by a parameter indicating the level of resource availability at the worker's pool. (In the prototype, the only resource monitored by a worker is the amount of local buffer space available for input and output files.) In response to the request, the master selects one or more unassigned jobs whose requirements can be accommodated by the worker's available resources and assigns them to the worker.

3.2. Job Monitor Process

A worker spawns a job monitor process for each job it is assigned. Processing a job involves transferring a data file from the data archive to the disk buffer at the worker's site, submitting a job to Condor for processing, and transferring an output file to the result archive. The job monitor initiates these steps and monitors their progress.

Transfer of input and output files is performed using the File Transfer Protocol (FTP) [8]. Both the source and result archives are assumed to be FTP servers. Interaction between the job monitors and the archives is complicated by the fact that there is no standard programming interface for FTP. The monitor transfers files by spawning executions of the UNIX FTP client program. For similar reasons, the job monitor submits its job request to Condor by spawning an instance of the *condor_submit* program.

An API for Condor, called CARMi [9], has recently become available. We intend to incorporate it into subsequent versions of the DBC. Unfortunately, we not aware of an existing API for FTP. To implement direct control of FTP file transfers, we can provide an implementation of the FTP protocol within the job monitor itself. Alternatively, we could make use of a wide-area file system such as Jade[10] or Prospero[7]. These systems would allow the monitor to transfer files to and from remote FTP servers using the UNIX file system program interface. We are studying these alternatives for the next version of the DBC.

3.3. Failure Detection and Recovery

The DBC is capable of detecting and recovering from several types of failures. Failures of the DBC worker processes are detected through a heartbeat mechanism. The DBC master maintains the status of the batch execution in

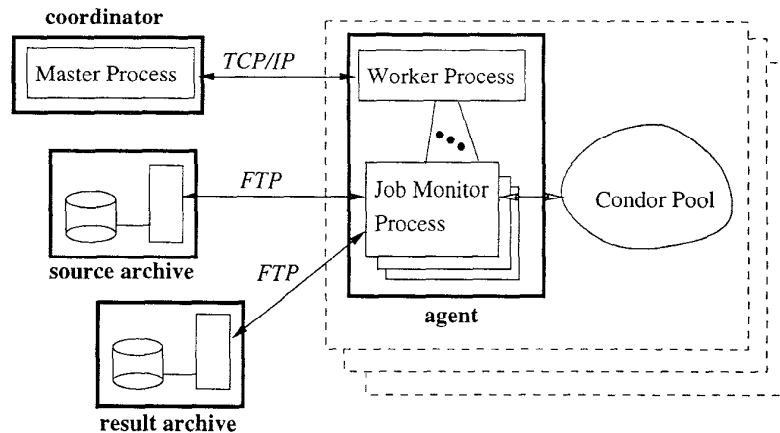


Figure 2. DBC Prototype Process Architecture

a file so that it can be restored after a failure. A failure of a job monitor is detected by the worker that created it. Condor itself provides a guarantee of reliable execution of jobs submitted to it. In addition, the DBC job monitor implements an optional timeout mechanism to detect hung jobs. Finally, file transfer failures are detected by monitoring the output of the FTP client program.

These mechanisms are intended to ensure that each job in a batch will appear to have executed exactly once, despite failures. If a worker fails while jobs are running in its pool, or if a pool refuses or is unable to process jobs that have been assigned to it, a high-level scheduler in the coordinator will assign them to another pool. The DBC ensures that multiple executions of the same job appear to be serialized, and that if a job is executed more than once, later executions overwrite the results of earlier executions. Thus, although a job may actually execute more than once, it will appear to have run once only.

The DBC is designed to allow workstation pools to join or leave the system at any time. Pools may leave the system because of failures, or simply because their resources are no longer available for DBC jobs. Since the DBC cannot assume that a lost pool will ever rejoin the system, it will simply reassign that pool's jobs to another pool. For the same reason, all critical state information is maintained by the DBC master, and not the workers.

3.4. Condor

The DBC assumes that each pool's resources are managed by Condor [6, 1]. Condor is a resource management system that runs on pools of UNIX workstations. Condor harnesses the computational power of unused workstations in the pool. Jobs submitted by Condor users are sent automatically to idle workstations in the pool for processing.

Should an idle workstation become busy, Condor jobs are moved from it to other idle machines.

Since the DBC uses Condor, it effectively draws its computational power from idle workstations in many pools, possibly distributed over a wide area. Jobs submitted to Condor by a DBC agent must compete with other Condor jobs for the available resources in a pool. A pool's resources are allocated to jobs (including DBC jobs) according to policies implemented by Condor. These policies may vary from pool to pool, according to the needs of the pools' owners.

4. Image Compression Experiments

We have used the DBC prototype as an engine for compressing a database of Landsat Thematic Mapper (TM) images. Compression was accomplished by a technique known as vector quantization (VQ). Vector quantization produces a very compact representation of an image which can be decompressed quickly. Compression is lossy. The decompressed images are useful for applications, such as database browsing or preliminary data analysis, which can tolerate approximate versions of the original images.

A program implementing VQ was provided to us by M. Manohar, from NASA's Goddard Space Flight Center (GSFC). This was the batch program used by the DBC. The program requires two input files, one holding the uncompressed TM image, and the other a small codebook used during compression. The same codebook file is used to compress all of the images. The program produces a single output file, which holds the compressed version of the input image. In our tests, we used a codebook with 128 entries and a four-by-four pixel vector size, which results in a 16-to-1 compression of the images.

For the purposes of our experiment, we selected from the database approximately one hundred images to be com-

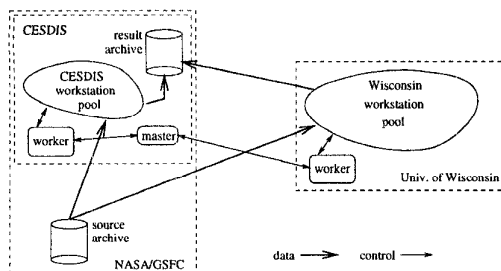


Figure 3. DBC Experimental Environment

pressed. Each image is a 2984 by 4320 array of pixels, with one byte used to represent each pixel. Thus, each image occupies slightly less than 13 megabytes. The single code-book file is only about 8 kilobytes in size.

Although we selected only a fraction of the available images, compressing even those requires a non-trivial amount of computation. Compression of a single 13 megabyte input image using the VQ program requires approximately 30 minutes on a Sparc10 workstation. Sequential processing of the one hundred selected TM images would occupy such a workstation for more than two days.

The DBC system we used for this experiment is illustrated in Figure 3. It consists of two workstation pools, one large and one small. The small pool is located at the Center of Excellence in Space Data and Information Systems (CESDIS), located at GSFC, near Washington, DC. The CESDIS pool includes nine Sparc10 workstations and a 140 megabyte disk buffer for the DBC. The other pool is located at the University of Wisconsin, in Madison, Wisconsin. The Wisconsin pool includes approximately one hundred Sun workstations of various types and a DBC disk buffer of 140 MB.

The DBC source archive, which holds the image database, is a mass storage system located at GSFC. Although both the mass storage system and the CESDIS workstation pool are located at GSFC, applications running at CESDIS do not have direct access to the mass storage system's files. The DBC result archive is located at CESDIS.

We initiated the image compression experiment at 7:00 pm EDT on a weeknight. It was complete approximately six hours later. Figure 4 shows the number of batch jobs completed by each of the pools as a function of time. The CESDIS pool was idle during much of the final hour of the experiment, while the last few jobs were being executed in the Wisconsin pool. Since jobs tended to complete more quickly a CESDIS, a more sophisticated high-level scheduler might have been able to minimize this boundary effect. The figure also shows bursts of job completions, particularly at the Wisconsin pool. This is also an artifact of the high-level scheduler.

It should be emphasized that the DBC's power to com-

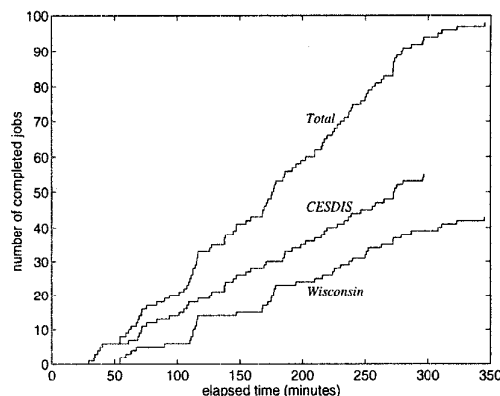


Figure 4. Progress of the VQ Experiment

press images is harnessed from existing, geographically-distributed, non-dedicated, computational resources. The amount of power that the DBC was able to deliver can be quantified by comparing it against a benchmark. As our benchmark, we choose an idealized, dedicated workstation that can execute the VQ program in time t_{ideal} , and that has direct, instantaneous access to the input and output archives. We define the *idealized processor equivalent* (IPE) rating, I , of the DBC system to be:

$$I = \frac{nt_{ideal}}{t_{DBC}}$$

where n is the number of jobs in the batch, and t_{DBC} is the batch completion time of the DBC system. For example, if our idealized workstation is about as fast as a Sparc10 then $t_{ideal} = 30$ minutes. Using this benchmark, the DBC achieved an IPE rating of approximately 8.5 during our experiment. Thus, the DBC system provided power equivalent to more than eight workstations having instantaneous access to the stored data.

5 Additional Experiments

By changing its batch program, the DBC may be used to support other data processing applications. We expect that the power that the DBC can deliver will depend on the application it is supporting. One important feature of an application is its *computational density*: the amount of computing required per megabyte of input and output data. We expect that the DBC will be most effective for applications with high computational densities. High densities allow the DBC to amortize the cost of staging data to be amortized over longer computing times.

To evaluate the impact of this application parameter, we wrote a artificial application, *ART*, whose density can be controlled. The ART program is modeled after VQ. It first

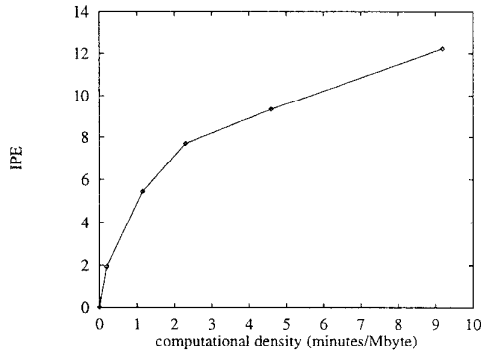


Figure 6. Effect of Computational Density on the IPE Rating

opens and reads a single input file sequentially. It then enters an empty loop, in which it spins for a number of iterations determined by a nominal computation time, t_{nom} , specified as a command line argument. After looping, ART writes an output file sequentially. By varying either the sizes of the input and output files, or the number of loop iterations, the computational density of ART can be controlled. We define the *nominal computational density* of the ART program, d , by

$$d = \frac{t_{nom}}{S_{in} + S_{out}}$$

where S_{in} and S_{out} are the sizes of the program's input and output files.

We ran a set of experiments with the DBC in which the nominal density of the ART program was varied. Each experiment consisted of the execution of a batch of ART jobs, with each jobs having the same nominal density. Figure 5 summarizes the ART program parameters we used. The input and output file sizes are identical to those used for the VQ experiments.

The experiments were run using the two-pool DBC system shown in Figure 3. This system was identical to the one used for the VQ experiments, except that the CESDIS pool had an additional workstation (for a total of ten) and an additional 20Mbytes of disk buffer space (for a total of 160 Mbytes.) Each experiment was started during off-peak hours because the processing power available from the Condor pool was the most stable then. This made it easier to compare the results of experiments run on different days.

Figure 6 summarizes the results of these experiments. The curve shows the IPE rating attained by the DBC system as a function of the computational density of the batch program. In all cases, the IPE rating is computed assuming an ideal processor than can process a single batch job in time t_{nom} , i.e., the ideal processor is about the speed of a SPARC10 workstation.

Our measurements show that the DBC can harness substantial power from geographically-distributed resources, provided that the batch jobs are computationally intensive. With d near two minutes per megabyte, the system was measured at approximately 8 IPEs. This is about the same as was measured for the VQ program, which has a comparable computational density. As d was increased to about nine minutes per megabyte, the IPE rating of the system increased by almost 60%. When d was reduced below one minute per megabyte, power dropped off quickly. Given the resources available at a pool, it is possible to derive some simple formulas that predict the amount of computational power the DBC will be able to harness from it. These calculations are not included here because of space limitations. However, they can be found in [2].

6. Related Work

There are several systems that support load sharing across pools of workstations. Besides Condor, which we discussed in Section 3, these include DQS [11], LoadLeveler, LoadBalancer, LSF (formerly Utopia [13, 12]), and Codine. A discussion and comparison of many of these systems can be found in [4]. The DBC differs from these systems in several ways. The DBC manages pools of workstations, rather than individual workstations. The DBC is intended to operate over wide area. It does not assume that the files required by batch jobs will be directly accessible by those jobs. Finally, an instance of the DBC exists to process a single batch of jobs. It is not concerned with allocating resources among multiple, competing batches. In contrast, systems such as the ones mentioned above accept jobs (or groups of jobs) from many users, and seek to allocate resources fairly among them.

Another related system is UFMulti [5], intended for data processing in high-energy physics. Like the DBC and the systems described above, UFMulti performs data processing using a pool of workstations. UFMulti focuses on multi-stage computations, in which jobs in one stage provide input for jobs at the next. The system's emphasis is on load balancing, so the more computationally demanding stages can be allocated additional workstations from the pool.

7. Conclusion

As the volume of on-line scientific data grows, so too will the need for resources to process and enrich that data. The DBC provides a flexible mechanism for applying a large, distributed collection of resources to a single data processing task. Because the DBC utilizes Condor-managed workstation pools, it can exploit the processing power of idle workstations. No dedicated computational resources

| Parameter | Symbol | Units | Values |
|-------------------------------|-----------|---------------|------------------------------|
| Nominal Computation Time | t_{nom} | minutes | 0.167,2.5,15,30,60,120 |
| Input File Size | S_{in} | megabytes | 12.29 |
| Output File Size | S_{out} | megabytes | 0.77 |
| Nominal Computational Density | d | minutes/Mbyte | $t_{nom}/(S_{in} + S_{out})$ |
| ART Jobs Per Batch | n | jobs | 100 |

Figure 5. ART Experiment Parameters

are required. Our experiments show that the DBC can focus substantial computing power on a data processing effort, even when many of the available computing resources are thousands of miles away from the data repository.

The DBC is still under development, and much remains to be done to improve the system's performance and functionality. A more general job model would increase the variety of applications that can be supported by the system. For example, it would be useful to support jobs consisting of program executions pipelines, in which the output of one job is the input to the next. Many scientific data processing activities are naturally expressed in such a model. We also need to provide modules that will allow the DBC to interact with resource managers other than Condor. We are currently developing a new, easier to administer, and more robust implementation of the DBC system. The new implementation uses PVM [3] to provide distributed process control and communication. We intend to make this implementation publicly available.

8 Acknowledgments

Support for this work has been provided by NASA through its Applied Information Systems research program. The authors are grateful to CESDIS for its cooperation and willingness to provide resources. We also wish to thank Robert Crompt, Nathan Netanyahu, Mareboyana Manohar at NASA/GSFC for their assistance and cooperation, and for providing access to software and data.

References

- [1] A. Bricker, M. Litzkow, and M. Livny. Condor technical summary. Technical Report TR 1069, Department of Computer Science, University of Wisconsin, Oct. 1991.
- [2] C. Chen, K. Salem, and M. Livny. The DBC: Processing scientific data over the internet. Technical Report CS-96-17, University of Waterloo, Dept. of Computer Science, Mar. 1996.
- [3] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: parallel virtual machine - a user's guide and tutorial for networked parallel computing*. The MIT Press, Cambridge, MA, 1994.
- [4] J. A. Kaplan and M. L. Nielson. A comparison of queuing, cluster, and distributed computing systems. Technical Report NASA Technical Memorandum 109025, NASA Langley Research Center, Oct. 1993.
- [5] J. Kasaraneni, T. Johnson, and P. Avery. Load balancing in a distributed processing system for high-energy physics (UF-Multi). Technical Report 95-002, Department of Computer and Information Science, University of Florida, 1995.
- [6] M. Litzkow and M. Livny. Experience with the Condor distributed batch system. In *Proc. of the IEEE Workshop on Experimental Distributed Systems*, pages 97–101, Oct. 1990.
- [7] B. C. Neuman. Prospero: A tool for organizing internet resources. *Electronic Networking: Research, Applications, and Policy*, 2(1), 1992.
- [8] J. Postel and J. Reynolds. File transfer protocol (FTP). Technical Report RFC-959, USC Information Sciences Institute, 1985.
- [9] J. Pruyne and M. Livny. Parallel processing on dynamic resources with CARM. In *Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '95*, Apr. 1995.
- [10] H. C. Rao and L. L. Peterson. Accessing files in an internet: The Jade file system. *IEEE Transactions on Software Engineering*, 19(6):613–624, June 1993.
- [11] Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida. *DQS User Manual*, DQS version 3.1.2.3 edition, June 1995.
- [12] J. Wang, S. Zhou, K. Ahmed, and W. Long. LSBATCH: A distributed load sharing batch system. Technical Report CSRI-286, Computer Systems Research Institute, University of Toronto, Apr. 1993.
- [13] S. Zhou, J. Wang, X. Zheng, and P. Delisle. UTOPIA: A load sharing facility for large, heterogeneous distributed computer systems. Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, Apr. 1992.