# Management of Grid Jobs and Data within SAMGrid

Andrew Baranovski, Gabriele Garzoglio, Igor Terekhov
*Fermi National Accelerator Laboratory*
*Batavia, IL 60510 USA*

Alain Roy, Todd Tannenbaum
*Department of Computer Sciences*
*University of Wisconsin-Madison*
*Madison, WI 53706 USA*

## Abstract

*When designing SAMGrid, a project for distributing high-energy physics computations on a grid, we discovered that it was challenging to decide where to place user's jobs. Jobs typically need to access hundreds of files, and each site has a different subset of the files. Our data system SAM knows what portion of a user's data may be at each site, but does not know how to submit grid jobs. Our job submission system Condor-G knows how to submit grid jobs, but originally it required users to choose grid sites and gave them no assistance in choosing. This paper describes how we enhanced Condor-G to interact with SAM to make good decisions about where jobs should be executed, and thereby improve the performance of grid jobs that access large amounts of data. All these enhancements are general enough to be applicable to grid computing beyond the data-intensive computing with SAMGrid.* [1]

**Keywords.** SAMGrid, Condor-G, SAM, D0, grid computing, planning, middleware, data management, scheduling, data intensive computing.

## 1 Introduction

A key area in grid computing [6, 7] is job management, which typically includes planning a job's dependencies, selection of a cluster to execute the job, scheduling of the job at the cluster and ensuring reliable submission and execution. If data is partially replicated at different clusters, choosing the cluster where the job will run may have a large impact on the performance of the job, because a cluster with more of the necessary data will need to transfer less data in order for the job to run. In High Energy Physics (HEP) computing, which remains a principal application domain for grids as a whole, jobs are *data intensive* and therefore *data handling* is one of the most important factors. For HEP experiments such as D0 and CDF, data handling is the center of the grid system [19].

In this paper, we present results of a collaboration between the Condor and SAMGrid teams to develop a grid job system which uses a grid data handling system to assist in scheduling data-intensive jobs so as to achieve good performance. Our job management is based on Condor-G [9] and our data handling system uses SAM [4].

Condor-G is a is a fault-tolerant job submission system that can access computing resources via the Globus Resource Allocation Manager (GRAM) protocol. [1] GRAM is an attractive grid protocol for job submission because it is secure, uniform and can interact with a variety of underlying batch systems. However, GRAM does not provide a user-friendly interface and it is not a reliable protocol. Condor-G builds on top of GRAM and provides reliability, error recovery, monitoring, and a user-friendly interface to GRAM.

The SAM data handling system [17] has been developed at Fermilab over the last five years, for use primarily by the D0 and CDF HEP experiments. It is a globally distributed system, one of the first functioning data grids, which uses a centralized meta-data catalog and a fully distributed network of processing *stations*, each including a suite of servers. The system provides services for data provenance tracking, global on-demand data replication and routing among multiple storage systems, as well as coordination of access to various storage systems and other collective services. Presently, SAM is a production system serving petabyte-scale data to hundreds of D0 and CDF physicists scattered across dozens of institutions worldwide.

In the course of our collaboration between Condor-G and SAM, we designed and implemented mechanisms to allow Condor-G to interact with SAM to make good decisions about where to place jobs so that the least amount of data is transferred before the job can run, and therefore the job can complete more quickly. These improvements to Condor-G are used by the Jobs and Information Management (JIM) portion of the SAMGrid [10] application project whose primary purpose is to augment the SAM grid handling system

[1] Published in *The 2004 IEEE International Conference on Cluster Computing*, San Diego, CA, Sept 20-23 2004, pages 353- 360.

with job and information management services.

The rest of the paper is organized as follows. We discuss the relevant Condor-G enhancements in Section 2. We discuss the application of these technologies, the foremost being the interface of the job management with the data handling system, in Section 3. We discuss possible future research work in Section 4 and conclude in Section 5.

## 2 Condor-G Enhancements

For clients that desire access to grid resources accessible via GRAM, Condor-G represents a significant advance over the tools provided with the Globus toolkit. By providing client-side persistent state, durable distributed transactions, and network and end-point fault-tolerance, Condor-G works to prevent the loss or repetition of jobs [9].

But despite the benefits, Condor-G possessed limitations that stood in the way of our design goals for SAMGrid. Specifically, Condor-G required the user to specify at job submission time which grid site should run the job (or required this information indirectly, in the case of GlideIn[9]). That is, Condor-G did not aid users in selecting appropriate grid resources, and was not able to take advantage of the matchmaking framework from Condor [12] to select grid sites.

In addition, Condor-G lacked an appropriate mechanism to interface with the data placement knowledge and information contained within the SAM data handling system. This mechanism was essential so Condor-G could enrich job management decisions based upon an awareness of data placement in the grid.

In order to remove these limitations, we introduced several enhancements to Condor-G. In particular, we added matchmaking that allows Condor-G to select grid resources and we provided a mechanism for Condor-G to access information that is not easily advertised in a directory service.

### 2.1 Grid Matchmaking in Condor-G

Our first change was to introduce *planning* into Condor-G to automatically select the execution site for users. Planning is the process by which a client attempts to map logical requests onto physical resources, taking into consideration that these physical resources *are not under the control of the client*. To better understand planning, consider the analogy of an airline traveler. A traveler does not perform scheduling in the traditional sense, because he does not exert any direct control over the resources. The decisions about when and where the airplanes fly is left up to the resource owners, in this case the airlines. But a traveler usually can still identify a path from point A to point B in a manner mindful of independent constraints and preferences, even if no single airline has even identified the demand for travel along that path.

Several grid planners are under development. For example, the Chimera Virtual Data System [8] from the University of Chicago contains a mechanism to plan how to generate a logical file in the form of an abstract program execution graph. These abstract graphs are then turned into an executable DAG for the Condor DAGman [2] meta-scheduler by the Pegasus planner, which then can create the physical file that corresponds to the logical file. Pegasus is a grid planner developed by the Information Sciences Institute (ISI) at the University of Southern California [3].

For SAMGrid, we opted to enhance Condor-G with a planner designed around the ClassAd matchmaking framework [14]. In the subsequent sections we will show how this approach to planning meets the challenges outlined in Section 2 above. Other projects, such as the EU Datagrid [5], have also utilized matchmaking to perform planning with success. Since it is implemented *within* the Condor framework, the SAMGrid planner has several unique features, such as:

- flexibility of resource description, due to the *absence of a fixed schema*, and
- dramatically reduced need for an application-specific layer above the standard Condor layer in planning.

Before we proceed with the description of our enhancements, we give an overview of Condor's ClassAds and planning below. A reader who is familiar with Condor can skip to Section 2.1.2.

#### 2.1.1 The Basics of ClassAd Matching

A ClassAd is a set of uniquely named expressions, using a semi-structured data model so no specific schema is required. Each named expression is called an *attribute*. Each attribute has an *attribute name* and an *attribute value*. In our initial ClassAd implementation, the attribute value could be a simple integer, string, floating point value, or expression comprised of arithmetic and logical operators. A subsequent implementation introduced richer attribute value types and related operators for records, sets, and tertiary conditional operators similar to the C programming language. Figure 1 portrays an example of a simple ClassAd that could describe a cluster. In practice, ClassAds contain many more attributes.

The planner assigns significance to two special attributes: `Requirements` and `Rank`. `Requirements` indicates a constraint and `Rank` measures the desirability of a match. During evaluation, the scoping prefix `other` refers to the candidate matching ClassAd, and thus in Figure 1, `other.Name` represents the name attribute within a job ClassAd.

```
[
 Type         = "GridSite";
 Name         = "FermiComputeCluster";
 Gatekeeper   = "globus.fnal.gov/lsf";
 Load         = [
                  QueuedJobs  = 540;
                  RunningJobs = 200;
                ];
 Requirements = (   other.Type == "Job"
                  && Load.QueuedJobs < 100 );
 GoodPeople   = { "howard", "harry" };
 Rank         = member(other.Name,
                       GoodPeople) * 500
]
```

**Figure 1. Example of a ClassAd that could represent a compute cluster in a grid.**

The matchmaking algorithm requires that for two ClassAds to match, both of their corresponding `Requirements` must evaluate to `true`. The `Rank` attribute should evaluate to an arbitrary floating point number. `Rank` is used to choose among compatible matches: Among provider ads matching a given customer ad, the planner chooses the one for which the customer ad has the highest Rank value (noninteger values are treated as zero), breaking ties according to the provider's Rank value of the customer ad.

Interested readers can refer to [14] for more information about ClassAds and matchmaking.

#### 2.1.2 Matchmaking Planner Operation

The following steps are performed by the planner and depicted in Figure 2:

1. In the first step, Condor-G submission agents advertise job characteristics and requirements, and grid resources advertise their characteristics and requirements to the planner via ClassAds. This advertisement is performed periodically as a push model.

2. In the second step, the planner, also known as the matchmaker, scans the known ClassAds and creates pairs that satisfy each other's constraints and preferences.

3. In the third step, the planner informs the Condor-G submission agent of the match. At this point, Condor-G receives a copy of the resource ClassAd that has been matched to a specific job ClassAd. Condor-G then augments, or even changes, the attributes in the job ClassAd based upon information discovered in its matching resource ClassAd. In particular, the grid site that was chosen for the job is added to the ClassAd.

4. In the final step, Condor-G sends the job via the GRAM protocol to the grid resource now specified in the augmented job ClassAd.

### 2.2 Externally Supplied Function Evaluation in ClassAds Enhancement

ClassAds are an excellent data representation because they are both expressive enough to be useful but simple enough to be amenable to analysis. This simplicity has some drawbacks because users are unable to express arbitrary programs within ClassAds. For instance, users would find it very hard for a job ClassAd to state "I have the following files $x_1$ ... $x_{300}$ and I need to match with a resource whose list of files has the greatest overlap with my list of files." Not only is very unwieldy for users to specify, but it is likely to be information that is known elsewhere in a data handling system. Similarly, system administrators may not wish to publish information such as which users are allowed to access a system, but users may wish to select a system that they have access to.

While special-purpose hacks could be added to discover these sorts of information, these would not solve the general problem of accessing information that is not easily expressible in a directory service. Therefore, we have added the ability for users to write functions in another language that can be used in ClassAd expressions. These functions are evaluated at match time. This allows the matchmaker to base its decision not only on explicitly advertised properties but also on opaque logic that is not statically expressible in a ClassAd. Other uses include incorporation of information that is prohibitively expensive to publish in a ClassAd, such as local storage contents or lists of authorized users.

We show an example of such a user-defined function in Section 3.1. Adding such user-defined functions creates an interesting difficulty: as soon as functions are available for users to define, they are likely to do time-consuming computations, such as contacting remote databases to look up data needed for their calculations. Such operations can greatly slow down matchmaking. To some extent, this can be ameliorated by doing caching of the results of the functions, but this presents more complications: how long should the data be cached for? How do users interact with the cache? Should all functions be cached? We have solved this problem in our current implementation by implementing caching within the user-defined function, and we are investigating more general solutions.

The ability to call a user-defined function that can be used to decide which resource should be used is a significant enhancement to the matchmaking in Condor-G. It allows users to have more expressive requirements for their

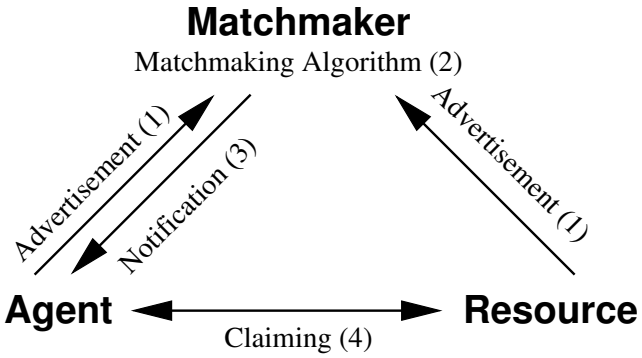


**Figure 2. Matchmaking on the Grid**

matches, and provides them with much greater control than they previously had.

### 2.3 Multi-tiered Architecture

In the past, Condor-G and the Globus Toolkit required that users submit their grid jobs from a computer that would remain available until the job has finished executing. This is inconvenient since many scientists would like to submit their jobs from laptops, or because their jobs may run long enough that they risk losing their connection to the grid during their jobs' execution.

Condor-G was extended to allow the jobs to be submitted to a second-tier machine that is more likely to remain available while the jobs are running. Users' jobs will be monitored and cared for on another computer, and they can return after a time, perhaps days, and retrieve the results.

## 3 Application to the SAMGrid

In this section, we discuss the materialization of our ideas in the grid Jobs and Information Management (JIM) part of the SAMGrid project. Figure 3 shows the principal job management architecture in our project. One interesting aspect of the design, discussed in Section 3.1, is that in addition to using the published information, the matchmaking service queries the resource providers, by means of evaluating the externally supplied function as described in Section 2.1.1.

### 3.1 Combination of the Advertised and Queried Information in the planner

The original matchmaking service gathered information about the resources in the form of ClassAds that completely described the available resources. This allows for a general and flexible framework for resource management such as matching jobs and computers, see [14]. This framework has been successfully implemented in the Condor batch system and has found other interesting uses, such as Hawkeye [11]. As described above, there is one limitation in that scheme, however, which is that the entities (jobs and resources) have to be able to express all their relevant properties in advance and cannot tailor the information to specific resources that they may be matched with.

Recall that our primary goal was to enable co-scheduling of jobs and data. In data-intensive computing, jobs are associated with long lists of data items (such as files) to be processed by the job. Similarly, resources are associated with long lists of data items located, in the network sense, near them. For example, jobs requesting thousands of files and sites having hundreds of thousands of files are not uncommon in production in the SAM system. Therefore, it would not be scalable to explicitly publish all the properties of jobs and resources in the ClassAds.

There are other examples of this sort of information that could not be included in a ClassAd prior to the work presented herein:

- A site may not want to advertise who is authorized to run jobs at the site, but the planner should not choose a site that the user is not authorized at. In simple situations, users can restrict the sites that are eligible, but this requires users to track and update them correctly. It may be more convenient for the matchmaker to check during matchmaking if the user is authorized.

- A site may prefer a job based that has similar data handling requests to jobs already scheduled at that site. Unlike the information about data already placed at sites, the information about scheduled data requests and their estimated time of completion is not described by any popular concept like replica catalogs.

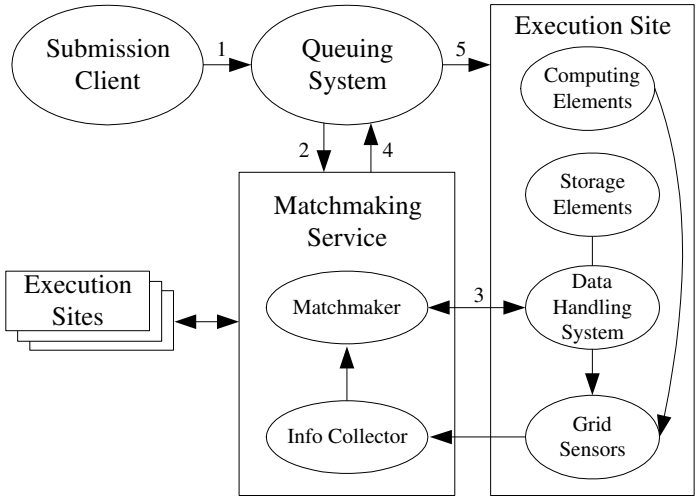


**Figure 3. SAMGrid's job management architecture. 1. Jobs are submitted to the job queue. 2. Job description is given to the matchmaker, which begins matching based on information from the grid sensors. 3. Matchmaker contacts the data handling system while matching. 4. Matchmaker provides the match to the queuing system. 5. The job is submitted across the grid to the execution site.**

In the SAMGrid design, we use the previously described user-defined functions to allow the planner to access information available in SAM about data. This is pictured in Figure 3 by arrows extending from the matchmaker to the local data handling systems, in the course of matching. It is implemented by means of externally supplied ranking functions. That is, users do not require a site to have a specific amount of their data on hand, but instead rank the sites by how much data is available, and the site with the most data is chosen. Specifically, the resource ClassAds in our design contain pointers to additional information providers (data handling servers called stations): as an example, the following is a resource ClassAd used in SAMGrid. The ClassAd is labeled "Machine" instead of "GridSite" for historical reasons.

```
[
  MyType               = "Machine";
  TargetType           = "Job";
  Software_CAF         = "Installed";
  site                 = "UToronto";
  schema               = "v0_8";
  SamStationName       = "cdf-toronto";
  SamStationUniverse   = "prd";
  SamStationExperiment = "cdf";
  Name            = "cdf-toronto.cdf.prd";
  architecture    = "Linux+2.4";
  gatekeeper_url  = "XXXXX:2119/jobmanager";
  nameservice = "IOR:000...44c3a...6500";
]
```

This is matched with the job ClassAd:

```
[
  MyType         = "Job";
  TargetType     = "Machine";
  Rank           = sam_rank_data_overlap(
                     other.SamStationName,
                     "jbot0h-361404",
                     other.nameservice);
  Requirements   =
    (  other.Software_CAF isnt Undefined
    && TARGET.SamStationUniverse=="prd"
    && TARGET.SamStationExperiment=="cdf");
  GlobusResource = "$$(gatekeeper_url)";
  Cmd = ...
  Args = ...
]
```

Note that the Rank expression invokes a user-defined function, `sam_rank_data_overlap`. This function will provide the amount of overlap between the user's data set and the files available at the grid site as a percentage. Because it is used in the Rank expression, jobs will prefer sites that have a greater percentage of the data that is needed, but do not require a site to have any of the data. This user-defined function includes logic similar to this pseudo-code:

```
station = resolve(Station_ID,...)
return station->get_preference(
                  job_dataset,...)
```

Design of other ClassAd functions is underway in the SAMGrid project.

### 3.2 Interfacing with the SAM Data Handling System

The co-scheduling of jobs and data has always been critical for the SAM system, where at least a subset of HEP analysis jobs (as of the time of writing, the dominant class) have their latencies dominated by data access. Note that the SAM system has already implemented the advanced feature of retrieving multi-file datasets asynchronously with respect to the user jobs [20, 18]—this was done initially at the cluster level rather than at the grid level, see also Section 4.

For jobs that are data-intensive, we attempt to minimize the time to retrieve any missing data and the time to store output data, as these times propagate into the job's overall latency. In order to minimize the grid job latency, we take the data handling latencies into account in the process of job matching. This is a principal point of this paper, and while we do not yet possess sufficient real statistics that would verify our design decisions, we stress that our system design enables the various strategies and supports considerations listed below.

In our initial implementation, we prefer sites that contain most of the job's data. Our design does not rely on a replica catalogue because in the general case, we need *local* metrics computed by and available from the data handling system:

- The network speeds for connections to the sources of any missing data;
- The depths of the queues of data requests for both input and output;
- The network speeds for connections to the nearest destination of the output files[2]

It is important that network speeds be provided by a high-level service in the data handling rather than by a low-level network sensor, for the same reason that having a 56Kbps connection to an ISP does not necessarily allow one to download files from the Internet at that speed.

We realize that in general, minimization of data handling latencies in job scheduling is insufficient and not always optimal. Data transfers may need to be explicitly scheduled in addition to jobs; see [15, 16] and references therein for some of the most interesting relevant work. The work of [16] has adopted a model with assumptions about file popularity distributions and where the jobs have single files as input. The work has shown by model-based simulations that sending the job to its data is not always best, due to the uneven growing of local schedulers' queues.

[2]In the SAM system the concept of data routing is implemented such that the first transfer of an output file is seldom done directly to the final destination.

## 4 Project Status and Future Work

In our first implementation of the job management in SAMGrid, we rank jobs by the amount of job's data cached at the sites (with rudimentary load-balancing based on randomization). The implementation of this first ranking function caches values of the known datasets for efficiency. We have deployed this system currently at 11 sites, which is a reasonable fraction of the SAM production system deployment. We have successfully submitted and brokered real user analysis jobs from CDF and D0 physicists, based on the logic described above.

Our Supercomputing 2002 demonstration was one of the first demos involving physics analysis on a grid. Two physicists from CDF have analyzed tens of gigabytes of real data [3] using a grid of 5 CDF sites: Fermilab (Illinois), University of Toronto (Canada), Rutgers (New Jersey), Rutherford Appleton Laboratory (UK), Kyungpook University (South Korea).

Referring to Figure 3, the user submits his job and specifies parameters like the dataset name, the full path to the directory containing the executable and other files. The job description is then translated into a ClassAd and submitted to the queuing system (step 1), which then sends the description of the job to the matchmaker (step 2). The execution sites advertise their resources to the matchmaking service by using the grid sensors. Section 3.1 shows job and resource description ClassAds used during the demo. The matchmaker considers each resource in turn, and queries the data handling system for information about the users data (step 3). The matchmaker communicates to the queuing system the best grid site(step 4) and the job is then submitted across the grid to be execute (step 5).

At the time of writing, we are deploying elements of SAMGrid onto other sites; more information will be available by the time of the Conference. Recall that the SAMGrid system will place terabyte scale jobs onto a grid consisting of dozens of sites. The Fermilab RunII experiments are expected to accumulate petabyte scale data within the next few years, likely before future HEP accelerators such as the CERN LHC turn on. The SAMGrid environment therefore gives us unique opportunities for the data-intensive job scheduling studies. What we have designed in the project and presented in this paper is a *design framework* for an organic integration of the job and data management, where the Condor-G planner will use a series of ranking functions of various complexity so as to make intelligent decisions based on the maximum of useful information, primarily from the data handling system (SAM). We plan research work in determining what information is most rele-

[3]The goal of the analysis was to determine the mass and the width of the J/Psi particle decaying in two muons and it served as a sanity check of the detector.

vant for these purposes, some ideas having been stated in Section 3.2. We plan to conduct experimental studies of the efficiencies of our scheduling strategies based on real system experience, including validation of and improvement on the model-based findings in [16].

For Condor-G, the main features that are working on are:

- Distributed, and possibly hierarchical, matchmakers;
- Updating resource ClassAds after matching in order to reflect changes that have occurred. For instance, a grid site may advertise the number of free nodes it has, and a match should reduce that number.

## 5 Summary

We have presented the design of a grid system to manage jobs and data, using the enhanced Condor-G technology for job management and SAM as the data handling system. We have dramatically improved Condor-G in several ways; most notably, we added matchmaking capabilities to select grid sites, and allowed user-defined ranking functions in matchmaking. We have used the new Condor-G to create a design framework for inclusion of data handling metrics into the matchmaker for better scheduling of data-intensive jobs, and have applied it to the SAM data handling system, in the framework of SAMGrid's Jobs and Information Management (JIM). Initially, our grid request broker prefers sites with most data cached, with more considerations being explored. Our work is implemented in real production system managing High Energy Physics analysis jobs and data.

## 6 Acknowledgements

This work is sponsored in part by DOE contract No. DE-AC02-76CH03000. Our collaboration takes place as part of the DOC Particle Physics Data Grid (PPDG), [13] Collaboratory SciDAC project. We would also like to express our gratitude for fruitful discussions with the members of the SAM and Condor teams.

## References

[1] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Proceedings of the IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1988.

[2] DAGMan, the directed acyclic graph manager. `http://www.cs.wisc.edu/condor/dagman`,.

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

[4] I. T. et al. Distributed data access and resource management in the D0 SAM System. In *Proceedings of the 10th International Symposium on High Performance Distributed Computing (HPDC-10)*, San Francisco, CA, July 2001. IEEE Computer Society Press.

[5] The European Union DataGrid Project. `http://www.eu-datagrid.org`.

[6] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[7] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[8] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proceedings of Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop*, May 1995.

[9] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.

[10] G. Garzoglio. The SAM-GRID Project: Architecture and plans. In *Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2002)*, June 2002.

[11] HawkEye: A monitoring tool for distributed systems. `http://www.cs.wisc.edu/condor/hawkeye`.

[12] M. Livny and R. Raman. High-throughput resource management. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[13] Particle Physics Data Grid (PPDG). `http://www.ppdg.net`, August 2002.

[14] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[15] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data intensive applications. In *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July 2002.

[16] K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[17] SAM. `http://runiicomputing.fnal.gov/sam`.

[18] I. Terekhov. Distributed processing and analysis of physics data in the dzero sam system at fermilab. Technical Report Fermilab-TM-2156, Fermi National Laboratory, Aug. 2001.

[19] I. Terekhov. Meta-computing at D0 (plenary talk). In *Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2002)*, June 2002.

[20] I. Terekhov and V. White. Distributed data access in the sequential access model in the D0 Run II data handling at Fermilab. In *Proceedings of the 9th IEEE Symposium on High Performance Distributed Computing (HPDC9)*, Pittsburgh, PA, August 2000.